Volume 32 Number 4 November 2008 ISSN 0350-5596

Informatica

An International Journal of Computing and Informatics

Special Issue:

Intelligent Systems

Guest Editors:

Costin Badica
Maria Ganzha
Marcin Paprzycki

1977

EDITORIAL BOARDS, PUBLISHING COUNCIL

Informatica is a journal primarily covering the European computer science and informatics community; scientific and educational as well as technical, commercial and industrial. Its basic aim is to enhance communications between different European structures on the basis of equal rights and international refereeing. It publishes scientific papers accepted by at least two referees outside the author's country. In addition, it contains information about conferences, opinions, critical examinations of existing publications and news. Finally, major practical achievements and innovations in the computer and information industry are presented through commercial publications as well as through independent evaluations.

Editing and refereeing are distributed. Each editor from the Editorial Board can conduct the refereeing process by appointing two new referees or referees from the Board of Referees or Editorial Board. Referees should not be from the author's country. If new referees are appointed, their names will appear in the list of referees. Each paper bears the name of the editor who appointed the referees. Each editor can propose new members for the Editorial Board or referees. Editors and referees inactive for a longer period can be automatically replaced. Changes in the Editorial Board are confirmed by the Executive Editors.

The coordination necessary is made through the Executive Editors who examine the reviews, sort the accepted articles and maintain appropriate international distribution. The Executive Board is appointed by the Society Informatika. Informatica is partially supported by the Slovenian Ministry of Higher Education, Science and Technology.

Each author is guaranteed to receive the reviews of his article. When accepted, publication in Informatica is guaranteed in less than one year after the Executive Editors receive the corrected version of the article.

80th birthday of Prof. Anton P. Železnikar – congratulations to the founder of *Informatica*

This year passes by 80 years from the birth of Prof. Železnikar, one of the pioneers of informatics and informational theory in Slovenia, as well as one of the initiators of Slovenian society Informatika, the founder and the Editor-in-chief of the journal Informatica which celebrates this year the 32nd years of uninterrupted publication. We are using this opportunity to present some excerpts from his CV.

Anton Pavel Železnikar was born in Slovenj Gradec, Slovenia on June 8, 1928, as the seventh child of Vinko Železnikar (1887), MD, and Pavla, (1898, born Rogina), a teacher. His father was an innovative surgeon and the head physician of the hospital in Slovenj Gradec. There were ten children in the family: two half-brothers, two half-sisters, three brothers, and two sisters.

In 1933, before the primary school, he began to learn German and piano. After five classes of the primary school in Slovenj Gradec (1934-1939) he became a student of a real-gymnasium (special type of a classical secondary school) in Maribor (1939-1941). During German occupation, first, he visited the third and the fourth class of Hauptschule in Slovenj Gradec, and then the fifth and the sixth class of the Tegetthoff Gymnasium in Maribor. The schooling in the German gymnasium was decisive for his later professional and value orientation. Majority of the teachers of the gymnasium held the doctor of science degree, and did excellent teaching in German, English, mathematics, and also in Latin.

In May, 1945, he was called in the State Security Troops of the National Liberation Army, serving as a soldier up to February, 1946. The experience in this service helped him develop a strong pro-human stand, based on realizing the most perverted human values, especially the cynicism of the then leading communists.

In 1946, he finished the so-called nostrification exams for the gymnasium classes three to six. Afterwards he decided to study classes seven, eight, and gymnasium leaving exam (matura in Slovenian language) on a private basis. In 1948, he finally passed the exams and matura.

In 1948, he became a student of the Technical Faculty of the University in Ljubljana and, later, of the Electronic Department (altogether 10 semesters). His distinguished teachers at that time were J. Plemelj (mathematics), A. Peterlin (physics), and V. Koželj (theoretical electro engineering). He defended his diploma work in 1956, entitled "Magnetostrictive memory loop", being a part of the amplitude analyzer.

From 1955 to 1980 he was employed at Jozef Stefan Institute, in the Electronics Department. His work was oriented into the then emerging digital engineering using vacuum tubes and transistors. On this path he became aware of the significance of the modern technology, proceeding deeply into the sophisticated computer and software engineering and research. Within this period, he received his M. Sc and Ph. D in Electrical Engineering from the University of Ljubljana in 1966 and 1967, respectively. From 1961 to 1978 he served as the head of the Digital Engineering Department and from 1968 to 1978 also as a head of Electronics division of Josef Stefan Institute. In 1968, he became the assistant professor and in 1972 the associated professor at the University of Ljubljana.

In 1980 he moved to a fast growing Slovenian company Iskra-Delta Computers where he stayed until his retirement in 1990. Within the company he held the position of the head of Microcomputer lab between the years 1980 and 1982. In 1982 he was promoted to a position of the advisor of the general manager of Iskra-Delta Computers and a member of the research and development strategy board of Iskra Corporation where he stayed until the end of his work career.

In his long and fruitful scientific career he published over 100 scientific and research papers in four languages and 2 books. Most of his life was devoted to the research of the informational theory, including philosophy of the informational, theory of informational phenomenalism, informational machines and informational operating systems, informational investigations in literature, media, communication by theory and machines, informational theory of consciousness and informational entity programming.

Prof. Železnikar was awarded many times for his work from the National science foundation and from different professional associations from ex-Yugoslavia between the years 1968 – 1990. He is a member of New York academy of science, International academy of science San Marino, Cybernetic Academy ``Stefan Odobleja", Lugano and International Association for Cybernetics, Belgium.

In 1971, he co-organized and was the program committee member of the IFIP '71 Congress in Ljubljana, one of the biggest congresses in the area of electrical and computer science organized ever in ex-Yugoslavia.

In the prime of his life, Prof. Železnikar is still following the progress in information sciences and continues his work. We wish him a happy anniversary and plenty more years among us.

Niko Schlamberger, president of the Slovenian society Informatika

Editor's Introduction to the Special Issue

Intelligent Systems

The area of intelligent systems is rapidly developing and reaching in multiple directions. In this mini-special issue we present four papers that show some of the area that current intelligent systems enclose.

The first paper, entitled "The Cross-Entropy Method for policy search in Decentralized POMDPs," was written by Frans A. Oliehoek, Julian F.P. Kooij, and Nikos Vlassis. It concerns possible approaches to multiagent planning under uncertainty. One of possible methods is utilization of Decentralized Partially Observable Markov Decision Processes (Dec-POMDPs). Unfortunately, solving a Dec-POMDP exactly is an intractable combinatorial optimization problem. In the paper, a new Cross-Entropy-based method is applied to the problem. It operates by sampling pure policies from an appropriately parameterized stochastic policy, and then evaluates them to define the next stochastic policy to sample from. This process is repeated until convergence is reached. Experimental results demonstrate that the proposed method can efficiently search large solution spaces.

In the second paper, "On the Compilation of Programs into their Equivalent Constraint Representation," Franz Wotawa and Mihai Nica present a novel approach to program analyzing and debugging. They convert programs into their loop-free equivalents and, next, into the static single assignment form. This allows them to derive a corresponding constraint satisfaction problem, which can be used directly for debugging. Specifically, they utilize the hyper-tree representation of the constraint satisfaction problem for debugging, while the width of the hyper-tree is an indicator of the debugging complexity.

The next paper, "Automatic Streaming Processing of XSLT Transformations Based on Tree Transducers," written by Jana Dvorakova, recognizes the fact that XML streaming has become an ubiquitous mode for information exchange over the Internet, and deals with an important issue of transforming large XML documents or XML data streams. Specifically, an automatic streaming processor for XSLT transformations, which guarantees bounds on resource usage, is presented. In the proposed approach, resource bounding is achieved by employing tree transducers associated with a set of streaming algorithms. The input XSLT stylesheet is analyzed in order to identify its transformation type, which in turn allows application of the lowest resource-consuming streaming algorithm.

Finally, Oana Nicolae, Adrian Giurca and Gerd Wagner, in their paper "On Interchange between Drools and Jess," approach IT and business solutions based on business rules and consider challenges related to the target Platform Specific Implementation Model. The paper discusses an example of the business rules translation from a particular object oriented rule-system (Drools), to another rule-system coming from the AI area (Jess), using the R2ML as the interchange language. The proposed transformation preserves the semantic equivalence for a given rule set, taking also into account the rules vocabulary.

Papers presented here are based on those that were presented at the first edition of the International Symposium on Intelligent and Distributed Computing (IDC'2007), which took place on October 18-20, 2007 in Craiova, Romania. Overall, out of 33 full papers published in Symposium Proceedings we have selected six. However, after further refereeing we have eliminated two of them and left four best contributions.

Finally, we would like to thank all of our colleagues, who acted as referees of the papers and made sure that material presented here is of highest quality.

Costin Badica
University of Craiova, Romania

Maria Ganzha and Marcin Paprzycki
Systems Research Institute
Polish Academy of Sciences, Warsaw, Poland

The Cross-Entropy Method for Policy Search in Decentralized POMDPs

Frans A. Oliehoek and Julian F.P. Kooij
Intelligent Systems Lab, University of Amsterdam
Amsterdam, The Netherlands
E-mail: {faolieho,jkooij}@science.uva.nl, www.science.uva.nl/~faolieho

Nikos Vlassis
Dept. of Production Engineering and Management
Technical University of Crete
Greece
E-mail: vlassis@dpem.tuc.gr, http://www.dpem.tuc.gr/vlassis

Keywords: multiagent planning, decentralized POMDPs, combinatorial optimization

Received: March 15, 2008

Decentralized POMDPs (Dec-POMDPs) are becoming increasingly popular as models for multiagent planning under uncertainty, but solving a Dec-POMDP exactly is known to be an intractable combinatorial optimization problem. In this paper we apply the Cross-Entropy (CE) method, a recently introduced method for combinatorial optimization, to Dec-POMDPs, resulting in a randomized (sampling-based) algorithm for approximately solving Dec-POMDPs. This algorithm operates by sampling pure policies from an appropriately parametrized stochastic policy, and then evaluates these policies either exactly or approximately in order to define the next stochastic policy to sample from, and so on until convergence. Experimental results demonstrate that the CE method can search huge spaces efficiently, supporting our claim that combinatorial optimization methods can bring leverage to the approximate solution of Dec-POMDPs.

Povzetek: Prispevek opisuje novo metodo multiagentnega načrtovanja.

1 Introduction

The construction of intelligent agents is one of the major goals in Artificial Intelligence. In the last two decades more and more research has concentrated on systems with multiple intelligent agents, or multiagent systems (MASs). In this article we focus on the recently proposed model of decentralized partially observable Markov decision processes (Dec-POMDPs) (7). A Dec-POMDP is a generalization to multiple agents of the well-known POMDP model for single-agent planning under uncertainty (19).

The Dec-POMDP model presents a decision theoretic formalization of multiagent planning under uncertainty. It models a team of cooperative agents that individually receive observations of their environment and cannot communicate. It is a very general model which has for instance been applied in the context of cooperative robotics (5; 14), communication networks (30), and sensor networks (26). The Dec-POMDP does not explicitly consider communication, but communication actions can be modeled as regular actions. Also, there are extensions that do explicitly incorporate communication (32), which can typically be used to share observations.

In this paper we focus on the regular Dec-POMDP setting in which the agents have to select their action based on their private observation histories. During an off-line planning phase we try to find a policy for a fixed number of time steps for each agent. The profile of all individual policies is collectively called a joint policy. The policies should be selected such that, when they are executed jointly during the on-line execution phase, the resulting behavior is (near-) optimal according to a predefined performance measure.

In the single-agent (i.e, POMDP) case, the history of observations results in a 'belief' (a probability distribution) over states, which forms a sufficient statistic for the history and can therefore be used to base the action choice on (19). In the multiagent case, however, no such simplification is possible, and the agents have to base their actions on their entire history. This means that the number of possible deterministic policies for a single agent is finite, but grows doubly exponentially with the planning horizon. Planning for a Dec-POMDP requires finding a profile of such policies, one for each agent, which defines a very challenging combinatorial optimization problem.

In this paper we examine the *Cross-Entropy (CE) method*, a recently introduced and promising method for combinatorial optimization problems (12) and its application to Dec-POMDPs. The CE method provides a framework for finding approximate solutions to combinatorial problems, and has gained popularity for various applications (24; 9; 1; 11), due to its ability to find near-optimal solutions in huge search spaces. Because of the combinatorial nature of the decentralized setting, the CE method may be a valuable tool in many algorithms for Dec-POMDPs.

In this paper we show how the CE-method can be applied in the setting of Dec-POMDPs using a relatively direct translation of the general CE optimization method. This, however, is not entirely trivial, so we discuss the difficulties and propose solutions. Our primary goal in this paper is not to provide a state-of-the art method for solving Dec-POMDPs. In fact, there are some algorithms which are more advanced than the one we present here (14; 35; 34; 3; 28). Rather, we show how the CE method can be use to tackle the combinatorial nature of the problem. This will allow for improvement of existing algorithms in two related ways. First, existing methods may rely on exhaustive evaluation of restricted sets of (partial) policies. The CE method may help to speed up these parts of algorithms. Second, such speed-up may allow the consideration of less restricted sets of policies, increasing the accuracy of the method.

1.1 Related work

In the last decade, planning for multiagent systems under uncertainty has received considerable attention. The Dec-POMDP model has been introduced by Bernstein et al. (6, 7), who also proved that finding a solution for a finite horizon Dec-POMDP is NEXP-complete. The infinite horizon problem is undecidable, as it is for the single agent case (i.e., solving a regular POMDP) (23). Pynadath and Tambe (32) introduced the multiagent team decision problem (MTDP), an equivalent model to the Dec-POMDP and a communicative extension, the COM-MTDP.

There are three main algorithms for the optimal solution of finite-horizon Dec-POMDPs apart from brute-force policy search. Hansen et al. (17) proposed dynamic programming (DP) for Dec-POMDPs, an algorithm that works by incrementally constructing policies for longer horizons. The second is multiagent A* (MAA*), a form of heuristic search (38). Finally Aras et al. (4) recently proposed a method based on mixed integer linear programming.

Due to the complexity results for Dec-POMDPs, however, the focus of much research has shifted to two other topics. First, people have tried to identify special instances of the problem that are less hard. For instance, a lot of research has focused on the case where there are special assumptions on observability and/or communication (10; 16; 21; 33; 36). Becker et al. (5) considered the special case of Dec-POMDPs were agents have their local state spaces and transitions and observations are independent. Similar assumptions are made by Nair et al. (26); Kim et al. (20); Varakantham et al. (39) who exploit resulting locality of interaction in the context of sensor networks. Other special cases are identified by Goldman and Zilberstein (15).

Second, research has focused on methods that find approximate solutions. For example, joint equilibrium search for policies (JESP) is a method that settles for a local optimum (25). Emery-Montemerlo et al. (13) proposed to construct a policy by approximating the Dec-POMDP with a series of compressed Bayesian games (BGs). Both Szer and Charpillet (37) and Seuken and Zilberstein (35, 34) proposed approximate extensions of DP for Dec-POMDPs.

Except for JESP, all these approximate methods in one way or another restrict the search space to provide leverage. This reduced search space is then searched exhaustively to find the best approximate solution. In this work, rather than searching a constrained policy space exhaustively, we examine ways to search the entire space approximately. In particular, we show how the CE method can be used to directly search spaces of up to 10^{243} joint policies fairly effectively.

1.2 Overview of article

Section 2 provides a background on decentralized decision making and formally introduces the Dec-POMDP framework. In section 3 we treat the background of the CE method. Section 4 introduces direct CE policy search for Dec-POMDPs, detailing how we apply the CE method to Dec-POMDPs. Section 5 introduces an extension of our algorithm with approximate evaluation. In section 6 we give an experimental evaluation and finally section 7 concludes and discusses future work.

2 Decentralized POMDPs

Here we provide a formal background of decentralized POMDPs. We start by introducing the decentralized tiger (DEC-TIGER) problem, which is a standard benchmark. Next, we introduce the formal Dec-POMDP model. Finally, we formalize histories, policies and the optimality criterion and we discuss naive (i.e., brute force) policy search.

2.1 The decentralized tiger problem

In the DEC-TIGER problem, two agents standing in a hallway are confronted with two doors. Behind one door lies a treasure, while behind the other there is a tiger. As such, there are two possible states the world can be in: s_l, the tiger is behind the left door, and s_r the tiger is behind the right door. Initially, these states have equal probability. The goal is to open the door that holds the treasure, which would result in a positive reward. But if at least one of them opens the other (wrong) door they receive a large penalty.

Both agents can take three actions, namely `OpenLeft` (a_{OL}) and `OpenRight` (a_{OR}) to open the left or right door, and `Listen` (a_{Li}) to try to observe carefully behind what door the tiger growls. The two possible observations `HearLeft` (o_{HL}) and `HearRight` (o_{HR}) indicate whether the agent heard the tiger behind the left or right door respectively. Observations are received at every time step but are random and uninformative unless *both* agents performed `Listen`, in which case they hear the tiger behind the correct door with high probability (each agent has a 85% chance of getting the correct observation). The action `Listen` has a small cost associated with it, but can decrease the agents' uncertainty about the state (which remains unchanged).

Furthermore, the rewards are specified such that it is always beneficial for the agents to act jointly: it is better to open the wrong door jointly than doing it alone. After one or both agents has opened a door, rewards are given and the situation is reset to a random state. More details on the DEC-TIGER problem are provided by Nair et al. (25).

2.2 The formal model

The *decentralized partially observable Markov decision process (Dec-POMDP)* describes a stochastic, partially observable environment for a set of cooperating agents.

Definition 2.1. A Dec-POMDP is a tuple $\langle \mathcal{A}g, \mathcal{S}, \mathcal{A}, T, R, \mathcal{O}, O\rangle$ where:

- $\mathcal{A}g = \{1, \ldots, n\}$ is the set of agents.
- $\mathcal{S}$ is a finite set of states.
- The set $\mathcal{A} = \times_i \mathcal{A}_i$ is the set of *joint actions*, where $\mathcal{A}_i$ is the set of actions available to agent i. Every time step one joint action $\mathbf{a} = \langle a_1, ..., a_n\rangle$ is taken.[1]
- T is the transition function, a mapping from states and joint actions to probability distributions over next states: $T : \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathcal{S})$.[2]
- R is the reward function, a mapping from states and joint actions to real numbers: $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$.
- $\mathcal{O} = \times_i \mathcal{O}_i$ is the set of *joint observations*, with $\mathcal{O}_i$ the set of observations available to agent i. Every time step one joint observation $\mathbf{o} = \langle o_1, ..., o_n\rangle$ is received.
- O is the observation function, a mapping from joint actions and successor states to probability distributions over joint observations: $O : \mathcal{A} \times \mathcal{S} \to \mathcal{P}(\mathcal{O})$.

A Dec-POMDP is considered at a number of discrete time steps, or *stages, t*. At every such stage each agent i takes an individual action a_i. As a result of the taken *joint* action $\mathbf{a}$, the state then stochastically transitions from s to a new state s' according to T. At that point, the environment emits a joint observation $\mathbf{o}$ with probability $P(\mathbf{o}|\mathbf{a}, s')$, as specified by O. From this $\mathbf{o}$ each agent i observes its individual component o_i, selects a new action, etc.

In this paper we are searching for plans that specify actions for a fixed number of stages h. That is, we assume a finite *planning horizon* of h time steps. Furthermore, we assume that there is a distribution over states at the initial stage $t = 0$, also called the initial 'belief' $b^0 \in \mathcal{P}(\mathcal{S})$.[3]

[1] Unless stated otherwise, subscripts denote agent indices.

[2] We use $\mathcal{P}(X)$ to denote the infinite set of probability distributions over the finite set X.

[3] Unless stated otherwise, superscripts denote time indices.

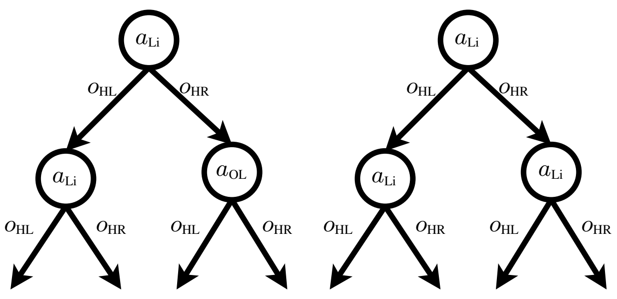


Figure 1: A part of an arbitrary joint policy for DEC-TIGER. Left shows to policy for agent 1, right for agent 2. The figure shows the actions taken at stages $0, 1$ and shows the observations received at stages $1, 2$.

2.3 Histories and policies

In a Dec-POMDP, an agent i only knows its own taken actions a_i and observations o_i. As a result it has to base its actions on the histories of those. Therefore, before we formalize the notion of a policy, we first formalize these histories.

Definition 2.2. The *action-observation history for agent i* $\vec{\theta}_i^t$ is the sequence of actions taken and observations received by agent i until time step t:

$$\vec{\theta}_i^t = \left(a_i^0, o_i^1, a_i^1, ..., a_i^{t-1}, o_i^t\right). \tag{2.1}$$

The *joint action-observation history* is a tuple with the action-observation history for all agents $\vec{\theta}^t = \langle \vec{\theta}_1^t, ..., \vec{\theta}_n^t\rangle$. The set of all action-observation histories for agent i at time t is denoted $\vec{\Theta}_i$.

Definition 2.3. The *observation history for agent* i is the sequence of observations an agent has received:

$$\vec{o}_i^t = \left(o_i^1, ..., o_i^t\right), \tag{2.2}$$

$\vec{\mathbf{o}}^t$ denotes a joint observation history and $\vec{\mathcal{O}}_i$ denotes the set of all observation histories for agent i.

Now we are ready to formalize policies.

Definition 2.4 (deterministic policy)**.** A *pure* or *deterministic policy* π_i for agent i in a Dec-POMDP is a mapping from observation histories to actions, $\pi_i : \vec{\mathcal{O}}_i \to \mathcal{A}_i$. A pure joint policy $\pi = \langle \pi_1 \ldots \pi_n\rangle$ is a tuple containing a pure policy for each agent. Π is the set of all pure joint policies.

One could expect that the most general definition of a policy is as a mapping from action-observation histories to actions. This is indeed the case for *stochastic policies* where an action is chosen at each action-observation history with a particular probability. For a deterministic policy, however, this is unnecessary because an agent can infer the actions it took from its observation history.

A pure policy can be visualized as a tree, as is illustrated in Figure 1, which shows a joint policy for the decentralized tiger problem. In this figure, each node marks a point where an agent has to decide upon an action. A path leading to such a node defines the observation history, and the depth corresponds to the stage at which the decision should be taken. A pure policy assigns to each node one action from $\mathcal{A}_i$, thereby defining what action the agent should take given some observation history.

Solving a Dec-POMDP amounts to finding an optimal joint policy π^* such that when the agents act accordingly, their expected shared reward is maximal. The quantity that we want the agents to maximize is the expected cumulative reward:

$$\pi^* = \arg\max_{\pi} E_\pi \left(\sum_{t=0}^{h-1} R(s, \mathbf{a}) \right). \tag{2.3}$$

Bernstein et al. (7) have shown that optimally solving a Dec-POMDP is NEXP-complete, implying that any optimal algorithm will be doubly exponential in the horizon. This becomes apparent when realizing that the number of pure joint policies is:

$$O\left[\left(|\mathcal{A}_*|^{\frac{(|\mathcal{O}_*|^h - 1)}{|\mathcal{O}_*| - 1}} \right)^n \right], \tag{2.4}$$

where $|\mathcal{A}_*|$ and $|\mathcal{O}_*|$ denote the largest individual action and observation sets.

The naive way of going about is to enumerate each of these joint policies and evaluate their expected cumulative reward, or *value*. The value of a specific (state, joint observation history) pair under a joint policy π is given by:

$$V_\pi(s^t, \vec{\mathbf{o}}^t) = R(s^t, \pi(\vec{\mathbf{o}}^t)) + \sum_{s^{t+1}} P(s^{t+1}|s^t, \pi(\vec{\mathbf{o}}^t)) \sum_{\mathbf{o}^{t+1} \in \mathcal{O}} P(\mathbf{o}^{t+1}|s^{t+1}, \pi(\vec{\mathbf{o}}^t)) V_\pi(s^{t+1}, \vec{\mathbf{o}}^{t+1}) \tag{2.5}$$

where $\vec{\mathbf{o}}^{t+1}$ is the new joint observation history at stage $t+1$: $\vec{\mathbf{o}}^{t+1} = (\vec{\mathbf{o}}^t, \mathbf{o}^{t+1})$. The total expected reward $V(\pi)$, with respect to the initial state distribution b^0 is then given by

$$V(\pi) = \sum_s V_\pi(s, \vec{\mathbf{o}}^0) b^0(s), \tag{2.6}$$

where $\vec{\mathbf{o}}^0$ is the initial (empty) joint observation history. For one joint policy this calculation requires evaluation of (2.5) for each of the $\sum_{t=0}^{h-1} |\mathcal{O}|^t = \frac{|\mathcal{O}|^h - 1}{|\mathcal{O}| - 1}$ joint observation histories and $|\mathcal{S}|$ states, leading to a total cost of:

$$O\left(|S| \cdot \frac{|\mathcal{O}|^h - 1}{|\mathcal{O}| - 1} \right). \tag{2.7}$$

3 Cross-entropy optimization

de Boer, Kroese, Mannor, and Rubinstein (12) described the *Cross-Entropy (CE)* method as a general framework to both rare event estimation and combinatorial optimization. We will focus only on the application to optimization. In particular, it has been illustrated how the CE method can be adapted to find good policies for a particular class of Markov Decision Processes (MDPs) (24; 12). In this section, we first provide a brief introduction to the CE method for optimization, followed by a description of the mentioned application to MDPs.

3.1 General CE Optimization

The cross entropy method can be used for optimization in cases where we want to find a—typically large—vector x from a hypothesis space $\mathcal{X}$ that maximizes some *performance function* $V : \mathcal{X} \to \mathbb{R}$. That is, when we are looking for

$$x^* = \arg\max_{x \in \mathcal{X}} V(x). \tag{3.1}$$

The CE method associates an estimation problem with this optimization. It maintains a probability distribution f_ξ over the hypothesis space, parametrized by a vector ξ. In particular, CE estimates the probability that the performance $V(\mathbf{x})$ of a sample $\mathbf{x}$ drawn according to f_ξ is higher than some γ:

$$P_\xi(V(\mathbf{x}) \geq \gamma) = \sum_{x \in \mathcal{X}} I(V(x), \gamma) f_\xi(x), \tag{3.2}$$

where $I(V(x), \gamma)$ is an indicator function:

$$I(V(x), \gamma) = \begin{cases} 1 & , V(x) \geq \gamma \\ 0 & , V(x) < \gamma. \end{cases} \tag{3.3}$$

Let γ^* be the optimal value, i.e., $\gamma^* \equiv V(x^*)$. Considering $\gamma = \gamma^*$ and a uniform distribution f_ξ in (3.2) gives the correspondence to the optimization problem (3.1). Also observe that $P_\xi(V(\mathbf{x}) \geq \gamma^*)$ is most likely very small under a uniform f_ξ, which explains the link to rare event estimation.

The core of the CE method is an iterative two-phase process:

1. Generate a set of samples $\mathbf{X}$ according to f_ξ.
2. Select the best N_b of samples $\mathbf{X}_b$, and use those to update the parameter vector ξ.[4]

The first step is rather trivial. The second step, however, deserves some explanation. Let $\gamma^{(j)}$ be defined as the minimum performance within the selected set of samples of the j-th iteration. I.e.,

$$\gamma^{(j)} \equiv \min_{\mathbf{x} \in \mathbf{X}_b} V(\mathbf{x}). \tag{3.4}$$

The CE method requires that this lower bound performance is not allowed to decrease over time: $\gamma^{(j+1)} \geq \gamma^{(j)}$. This implies that $\mathbf{X}_b$ can contain less than N_b samples because

$$\forall_{\mathbf{x} \in \mathbf{X}_b} V(\mathbf{x}) \geq \gamma^{(j)} \tag{3.5}$$

[4]The number of best samples is often characterized as a fraction $0 \leq \rho \leq 1$ of the set of samples $\mathbf{X}$. I.e., $N_b = \text{round}(\rho \cdot N)$.

is a hard requirement. The set $\mathbf{X}_b$ is then used to create $\xi^{(j+1)}$, a maximum-likelihood estimate of the parameters. These new parameters can be smoothed using a learning rate $0 \leq \alpha \leq 1$ by interpolating with $\xi^{(j)}$ the parameter vector of the previous iteration:

$$\xi^{(j+1)} = \alpha\xi^{(j+1)} + (1-\alpha)\xi^{(j)}. \tag{3.6}$$

This reduces the probability that some components of the parameter vector will be 0 or 1 early in the CE process, which could cause the method to get stuck in local optima.

Usually, the iterative process is stopped when $\gamma^{(j)}$ has not improved over some predefined number of steps. But other conditions such as a time limit or a fixed number of iterations can be used. When the stop condition is finally met, the best sample $\mathbf{x}$ found in the entire process is returned as an approximation of x^*.

3.2 The CE Method for MDPs

In their work, Mannor et al. (24) show how the CE method can be applied to shortest paths MDPs, a class of MDPs for which the optimal value function is stationary, i.e., the expected value of taking a particular action in a particular state is not dependent on the stage. The optimal policy for such a MDP is a mapping from states to actions $\pi_{\text{MDP}} : \mathcal{S} \to \mathcal{A}$, which can be represented as an $|\mathcal{S}|$-vector. As in section 3.1, the goal is to find the vector that maximizes a performance function, in this case the expected total reward. So rewriting (3.1), we are looking for

$$\pi^*_{\text{MDP}} = \arg\max_{\pi_{\text{MDP}}} V(\pi_{\text{MDP}}), \tag{3.7}$$

where the performance function now is the value of the MDP-policy π_{MDP}. The CE method tackles this problem by maintaining a parameter vector $\xi = \langle \xi_{s_1}, ..., \xi_{s_{|\mathcal{S}|}} \rangle$, where each ξ_s is a probability distribution over actions. Using these probabilities it is possible to sample N trajectories: starting from some start state actions are randomly selected according to the probabilities as described by ξ until the goal state is reached. Using the N_b best (highest total reward) trajectories $\mathbf{X}_b$, the parameter vector can be updated as follows:

$$P(a|s) = \frac{\sum_{\mathbf{x}\in\mathbf{X}_b} I(\mathbf{x}, s, a)}{\sum_{\mathbf{x}\in\mathbf{X}_b} I(\mathbf{x}, s)}, \tag{3.8}$$

where $I(\mathbf{x}, s, a)$ is an indicator function that indicates that action a was performed at state s in trajectory $\mathbf{x}$, and $I(\mathbf{x}, s)$ indicates whether s was visited in trajectory $\mathbf{x}$.

After updating the parameter vector ξ, a new set $\mathbf{X}$ of trajectories can be sampled, etc. Empirical evaluation shows that this process converges to (near-) optimal policy in only a few iterations (24).

4 Direct CE policy search for Dec-POMDPs

In this section we propose an adaptation of the CE method for Dec-POMDP policy search, which we dub direct CE (DICE) policy search for Dec-POMDPs because it directly searches the space of joint policies withouth first restricting it.

DICE is a direct translation of the ideas presented in the previous section. Still, there are some problems when trying to apply the procedure as outline in the previous section to Dec-POMDPs: First, because we consider finite horizon Dec-POMDPs, there is no stationary value function. Second, the policies of the agents are not defined over states, but over their individual observation histories $\vec{o}_i^{\,t}$, and these are not a Markovian signal. Third, there is no clear way to sample traces and use those to update the distribution.

In the Dec-POMDP case, the hypothesis space is the space of deterministic joint policies Π. In order to apply the CE method, we are required to define a distribution over this space and an evaluation function for sampled policies. Also, we show how the distribution can be adapted using the best policies found in each iteration. First, we will present two methods to define the distribution over the joint policy space. After that we describe how the parameter updates are performed. Finally, a we give a summary and complexity analysis of the DICE algorithm.

4.1 Policy distributions

In the case of Dec-POMDPs f_ξ denotes a probability distribution over pure joint policies, parametrized by ξ. We will represent this probability as the product of probability distributions over individual pure joint policies:

$$f_\xi(\pi) = \prod_{i=1}^{n} f_{\xi_i}(\pi_i). \tag{4.1}$$

Here ξ_i is the vector of parameters for agent i, i.e., $\xi = \langle \xi_1, ..., \xi_n \rangle$.

The question is how to represent the probability distributions over individual pure policies. One clear solution is to enumerate all the pure policies for an agent i and to maintain an explicit discrete probability distribution over this set of policies. I.e., the distribution is represented as a mixed policy (31). However, this approach suffers from two drawbacks. First, the number of pure individual policies π_i might be huge, making such an explicit distribution impractical to represent. Second, this representation is hard to parametrize in a meaningful way using some vector ξ_i, as it gives no access to the internals of the policies: parameters would specify probabilities for entire pure policies, rather than specifying behavior for particular observation histories as in figure 1.

Therefore, rather then using a mixed policy representation, we will use a behavioral- (31) or stochastic policy (22) description: a mapping from decision points to probability

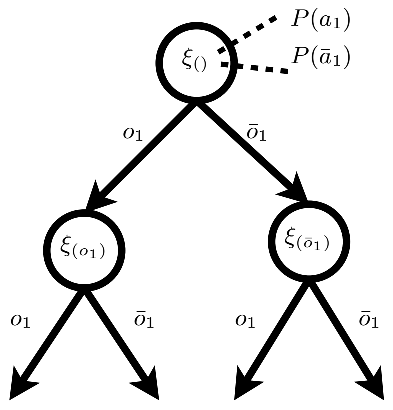


Figure 2: A part of a stochastic policy for an agent of a fictitious Dec-POMDP.

distributions over actions. Note that this is similar to the approach for MDPs, as described in section 3.2, were the states are the decision points.

4.1.1 Observation history based

The simplest way to represent a policy distribution is to make a direct translation from CE for MDPs: instead of maintaining a simple probability distribution over actions for each state, we now maintain one for each observation history (OH). Figure 2 illustrates such a representation of the policy distribution. It shows that for each observation history $\vec{o}_i^t$ a parameter $\xi_{\vec{o}_i^t}$, that specifies the distribution over actions, is maintained:

$$\forall_{a_i} \quad \xi_{\vec{o}_i^t}(a_i) \equiv P(a_i|\vec{o}_i^t). \tag{4.2}$$

Note that the distribution differs only from a pure policy (Figure 1) by keeping distributions over actions instead of a single action at the nodes of the tree. Consequently, the parameter vector for agent i is defined as $\xi_i \equiv \langle \xi_{\vec{o}_i^t} \rangle_{\vec{o}_i^t \in \vec{\mathcal{O}}_i}$, and the probability of a particular policy π_i for agent i as

$$f_{\xi_i}(\pi_i) = \prod_{\vec{o}_i^t \in \vec{\mathcal{O}}_i} \xi_{\vec{o}_i^t}(\pi_i(\vec{o}_i^t)). \tag{4.3}$$

We refer to this policy distribution representation as the OH-based representation.

4.1.2 Action-observation history based

Defining the parameters as in section 4.1.1 is the most straightforward approach, but does not take into account the action history: the choice for action $\pi_i(\vec{o}_i^t)$ has no influence on the choice for the action at the next time step $\pi_i(\vec{o}_i^{t+1})$. As explained in section 2, in general a stochastic policy does take into account the taken action. However, we know that there is at least one deterministic joint policy for a Dec-POMDP (which consists of individual policies that are a mapping from observation histories to actions). Moreover, in previous research we did investigate an action-observation history (AOH) based representation, but the influence appeared to be minor (27). Therefore we will not consider this further in this work.

4.2 Sampling and Evaluation

Unlike the MDP case, in the setting of Dec-POMDPs there is no trivial way to sample some trajectories given the joint policy distribution f_ξ and use that to update the distribution. Rather we propose to sample complete joint policies and use those for the parameter update.

Selecting a random sample of joint policies from the distribution is straightforward. For all the observation histories $\vec{o}_i^t$ of an agent i an action can be sampled from action distribution $\xi_{\vec{o}_i^t}$. The result of this process is a deterministic policy for agent i. Repeating this procedure for each agent samples a deterministic joint policy. The evaluation of a joint policy can be done using (2.6).

4.3 Parameter update

We described how to represent and sample from the probability distribution over policies. This section describes how the set of best policies $\mathbf{X}_b$ sampled from the previous distribution $f_{\xi^{(j)}}$, can be used to find new parameters $\xi^{(j+1)}$.

Let $I(\pi_i, \vec{o}_i^t, a)$ be an indicator function that indicates whether $\pi_i(\vec{o}_i^t) = a$. In the OH-based distribution the probability of agent i taking action $a^t \in \mathcal{A}_i$ after having observed $\vec{o}_i^t$ can be re-estimated as:

$$\xi_{\vec{o}_i^t}^{(j+1)}(a^t) = \frac{1}{|\mathbf{X}_b|} \sum_{\pi \in \mathbf{X}_b} I(\pi_i, \vec{o}_i^t, a^t), \tag{4.4}$$

where $|\mathbf{X}_b|$ normalizes the distribution since

$$\forall_{\vec{o}_i^t} \quad \sum_{a \in \mathcal{A}_i} \sum_{\pi \in \mathbf{X}_b} I(\pi_i, \vec{o}_i^t, a) = |\mathbf{X}_b|. \tag{4.5}$$

Note that the thus computed new parameter vector $\xi^{(j+1)}$ will afterward be smoothed using the learning rate α according to (3.6).

4.4 Summary and complexity analysis

Algorithm 1 summarizes the DICE policy search method. To start it needs I, the number of iterations, N, the number of samples taken at each iteration, N_b, the number of samples used to update ξ, and α, the learning rate. The outer loop of lines 3–17 covers one iteration. The inner loop of lines 5–13 covers sampling and evaluating one joint policy. Lines 14–16 perform the parameter update. Because the CE method can get stuck in local optima, one typically performs a number of restarts. We have not incorporated these in the algorithm itself, however.

Now we consider the complexity of this algorithm. For each iteration we draw N joint policies. The sampling of

Algorithm 1 The DICE policy search algorithm

```
Require: CE parameters: $I, N, N_b, \alpha$
 1: $V_b \leftarrow -\infty$
 2: initialize $\xi^{(0)}$ {typically uniform random}
 3: for $i \leftarrow 0$ to $I$ do
 4:     $\mathbf{X} \leftarrow \emptyset$
 5:     for $s \leftarrow 0$ to $N$ do
 6:         sample $\pi$ from $f_{\xi^{(i)}}$
 7:         $\mathbf{X} \leftarrow \mathbf{X} \cup \{\pi\}$
 8:         $V(\pi) \leftarrow \text{Evaluate}(\pi)$
 9:         if $V(\pi) > V_b$ then
10:             $V_b \leftarrow V(\pi)$
11:             $\pi_b \leftarrow \pi$
12:         end if
13:     end for
14:     $\mathbf{X}_b \leftarrow$ the set of $N_b$ best $\pi \in \mathbf{X}$
15:     Compute $\xi^{(i+1)}$ {using (4.4) }
16:     $\xi^{(i+1)} \leftarrow \alpha\xi^{(i+1)} + (1-\alpha)\xi^{(i)}$
17: end for
18: return $\pi_b$
```

such a joint policy involves, for each agent, selecting an action for each of its observation histories and has complexity

$$O(n \cdot |\mathcal{A}_*| \cdot |\vec{\mathcal{O}}_*|) = O\left(n \cdot |\mathcal{A}_*| \cdot \frac{|\mathcal{O}_*|^h - 1}{|\mathcal{O}_*| - 1}\right),$$

where $*$ denotes the agent index with the largest observation and action set respectively. The complexity of updating ξ is similar, but includes the term N_b (rather then being performed N times). Evaluation (i.e., computing the total expected reward) of a policy $V(\pi)$, is performed by evaluating equation (2.6) and (2.5) from the last stage $h-1$ up to the first 0. The complexity of this calculation for a single pure joint policy scales exponentially with the planning horizon, as explained in section 2. Because $|\mathcal{O}| = O(|\mathcal{O}_*|^n)$, we can rewrite (2.7) as

$$O\left(|S| \cdot \frac{|\mathcal{O}_*|^{nh} - 1}{|\mathcal{O}_*|^n - 1}\right),$$

This latter term typically dominates the complexity of sampling and updating ξ, therefore the total time complexity of DICE is given by

$$O\left[I \cdot N \cdot \left(|S| \frac{|\mathcal{O}_*|^{nh} - 1}{|\mathcal{O}_*|^n - 1}\right)\right].$$

5 Approximate Evaluation

The complexity analysis showed that the time required for DICE scales exponentially with the planning horizon, and that policy evaluation forms the bottleneck of our algorithm. In order to alleviate this bottleneck, we examine the application of approximate evaluation. The idea is that rather than computing the expected value, we sample this value by simulating r episodes, or *traces,* and using the average of outcomes as an estimate $\widetilde{V}(\pi)$ for the actual value $V(\pi)$. We will refer to the resulting method as DICE policy search with approximate evaluation (DICE-A).

Clearly, this approximation might introduce errors. Notice, however, that the CE method does not discriminate between policies within the set $\mathbf{X}_b$ of best samples. Therefore, as long as the relative ordering is preserved, the same policies are used to update the policy distribution, yielding the same results. In fact, only when the ranking of policies is disturbed near the cut-off threshold (around the N_b-th joint policy), will approximate evaluation influence the distribution updating process.

There is a second potential source of error, though. When the fraction of best samples $\mathbf{X}_b$ is used to update γ using (3.4), the new γ might in fact be an over-estimation. This could make it very difficult to sample new instances with a higher (approximate) value. In previous work, we therefore also considered a version of our algorithm that did not use the hard threshold γ, but rather always used the best N_b samples (27). The results, however, did not show a significant improvement, nor did we encounter any such difference in further experiments we performed. Therefore we will not consider this further in this paper.

5.1 Complexity

Simulating one trace of a joint policy involves looking up the actions for each of the n agents and sampling one of $|\mathcal{S}|$ successor states and one of $|\mathcal{O}| = O(|\mathcal{O}_*|^n)$ joint observations at each of the h stages. Such a simulation is performed r times, so the time complexity of performing approximate evaluation of a single joint policy is:

$$O\left(r \cdot h \cdot n \cdot |\mathcal{O}_*|^n \cdot |\mathcal{S}|\right). \quad (5.1)$$

DICE-A performs such an evaluation for each of the N sampled policies in each of the I iterations. Therefore, total time complexity of DICE with approximate evaluation is given by

$$O\left(I \cdot N \cdot r \cdot h \cdot n \cdot |\mathcal{O}_*|^n \cdot \mathcal{S}\right), \quad (5.2)$$

as long as approximate evaluation dominates the time needed to sample a policy and update the parameter vector ξ.

5.2 Error bounds

The estimated value $\widetilde{V}(\pi)$ is only an approximation of the true value $V(\pi)$. However, we are able to establish bounds on this error. In particular, we know that $V(\pi)$ is bounded when the immediate reward function is bounded. Let us write R_{min}, R_{max} for the lower and upper bound of the reward function, that is, $\forall_{s,\mathbf{a}}\ R(s,\mathbf{a}) \in [R_{min}, R_{max}]$. Then the value of a policy is bounded by

$$V(\pi) \in [hR_{min}, hR_{max}].$$

Wassily Hoeffding (18) proved that the probability that the sum of r independent random variables $X_1, ..., X_r$, each bounded $X_i \in [a_i, b_i]$, exceeds the expectation (of the sum) with $r\epsilon$ or more is bounded by

$$P\left((X_1 + ... + X_r) - E\left[X_1 + ... + X_r\right] \geq r\epsilon\right) \leq \exp\left(-\frac{2r^2\epsilon^2}{\sum_{i=1}^{r}(b_i - a_i)^2}\right),$$

for any $\epsilon > 0$.

In our setting, each X_i denotes the outcome of the simulation of the i-th episode, and is an unbiased estimate of $V(\pi)$. Also, in our setting we are interested in a two-sided error bound, and all X_i respect the same bound $X_i \in [hR_{min}, hR_{max}]$. Therefore we can write

$$\begin{aligned}
&P\left(\left|(X_1 + ... + X_r) - E\left[X_1 + ... + X_r\right]\right| \geq r\epsilon\right) \\
=&P\left(\left|\frac{(X_1 + ... + X_r)}{r} - \frac{E\left[X_1 + ... + X_r\right]}{r}\right| \geq \epsilon\right) \\
=&P\left(\left|\frac{1}{r}\sum_{i=1}^{r} X_i - E(X)\right| \geq \epsilon\right) \\
=&P\left(\left|\widetilde{V}(\pi) - V(\pi)\right| \geq \epsilon\right) \\
\leq&2\exp\left(-\frac{2r^2\epsilon^2}{r\left(hR_{max} - hR_{min}\right)^2}\right).
\end{aligned}$$

Using this result we can control the lower bound on the probability that an error of size less than ϵ is made. Suppose we want to approximate $V(\pi)$ with accuracy ϵ with at least probability δ. I.e., $\delta = \lfloor P(\text{error} < \epsilon)\rfloor$ is the lower bound on the probability of an error smaller than ϵ, yielding

$$\begin{aligned}
P(\text{error} \geq \epsilon) &= 1 - P(\text{error} < \epsilon) \\
P(\text{error} \geq \epsilon) &\leq 1 - \lfloor P(\text{error} < \epsilon)\rfloor \\
&= 1 - \delta.
\end{aligned}$$

Then we must have that

$$P\left(\left|\frac{1}{r}\sum_{i=1}^{r} X_i - E(X)\right| \geq \epsilon\right) \leq 2\exp\left(-\frac{2r^2\epsilon^2}{r\left(hR_{max} - hR_{min}\right)^2}\right) \leq 1 - \delta.$$

Solving the right-hand side for r goes as follows:

$$\begin{aligned}
-\frac{2r^2\epsilon^2}{r(b-a)^2} &\leq \ln\left(\frac{1-\delta}{2}\right) \\
2r\epsilon^2 &\geq -(b-a)^2\ln\left(\frac{1-\delta}{2}\right) \\
r &\geq \frac{(b-a)^2}{2\epsilon^2}\ln\frac{2}{1-\delta}.
\end{aligned}$$

with

$$(b-a)^2 = h^2\left(R_{max} - R_{min}\right)^2.$$

So, to guarantee an error smaller than ϵ (with probability δ), the required number of traces grows only quadratically with the horizon.

5.3 Post-evaluation

When using DICE with approximate evaluation, the end result is a joint policy and its estimated value. In order to know the true quality of the joint policy, an exact evaluation can be started at this point. However, due to the exponential complexity of such an exact evaluation, this is not always feasible. In settings where it is not, we propose to do a more accurate sample based estimation of the value of the joint policy.

Of course, it may happen that the new exact (or more accurately determined) value of the joint policy is less than the previous estimate, and perhaps also less than the estimated value of other joint policies encountered during DICE. To prevent this, one may keep a list of the best k joint policies encountered. At the end of the procedure, one can then exactly evaluate all these k joint policies and select the best one. Alternatively, the CE process can be augmented with one additional iteration, where all sampled policies are evaluated exactly (or more accurately).

6 Experiments

Here we give an empirical evaluation of the proposed algorithm. The implementation of DICE follows the description without any further remarks. In our DICE-A implementation we have not implemented an additional evaluation iteration or list of k best policies as suggested. We only apply post-evaluation to the best ranked joint policy. This post-evaluation consists of a more accurate evaluation of $20,000$ runs when the value function consists of more than $20,000$ $(s, \vec{\mathbf{o}})$-pairs, and exact evaluation otherwise.

First we examine the influence of all parameters of the CE optimization procedure. Next, we briefly discuss some benchmark problems and investigate the performance on these benchmark problems of different horizons and compare it to dynamic programming JESP. Finally, we investigate how DICE scales with respect to the number of agents, again comparing to JESP.

6.1 Different CE parameters

The CE method has quite a few configurable parameters: the learning rate α, the number of iterations I, the number of samples drawn per iteration N, the fraction of best samples kept for update ρ and the induced number of samples used for this update $N_b = \rho \cdot N$. We have empirically investigated different settings of these parameters for DICE policy search. Additionally, for DICE-A we investigate the parameter r for the number approximation simulations.

The results reported in this section are all obtained on the DEC-TIGER problem. The non-varying parameters in these experiments were set as follows $\alpha = 0.2,\ I = 30,\ N = 50,\ N_b = 5$.

First we examine the influence of the learning rate α. Figure 3 shows that low α (0.2) results in low variance, but if too low ($\alpha = 0.1$) the CE process cannot converge

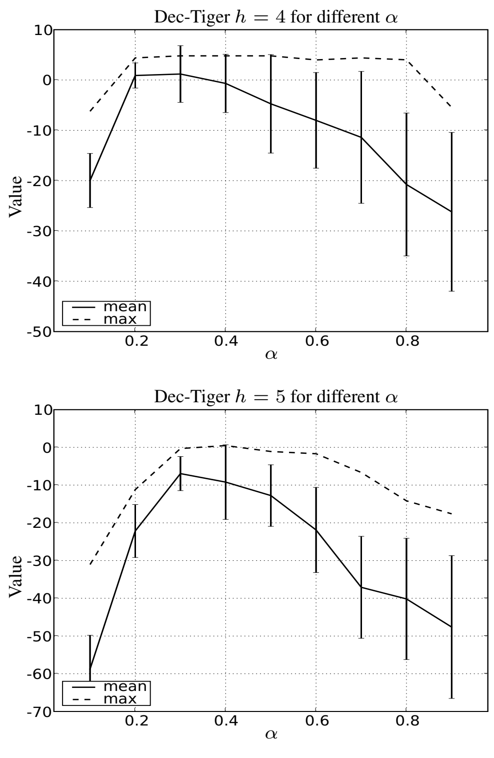


Figure 3: Performance for varying α. Top: horizon 4. Bottom: horizon 5.

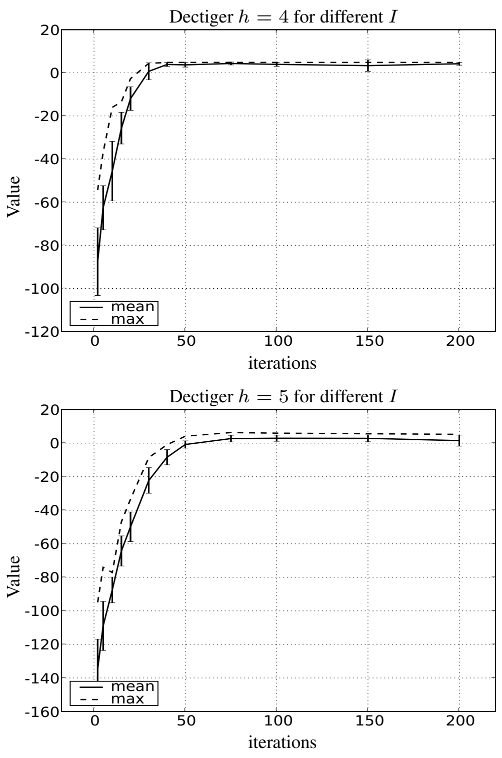


Figure 4: Performance for varying number of CE iterations. Top: horizon 4. Bottom: horizon 5.

to a good solution within a limited number of iterations. Also shown is that DICE on DEC-TIGER with 30 iterations $\alpha = 0.3 - 0.4$ gives the best trade-off between learning and convergence.

Next, we examine the number of CE iterations. The complexity analysis shows that this parameter should affect the running time linearly and this was experimentally confirmed. Figure 4 shows the value obtained by the algorithm for different numbers of iterations. It indicates that the algorithm converges after a fixed number of iterations. As we might expect, convergence requires less iterations for a smaller horizon (compare top and bottom figure). However, in both cases we see that the performance grows very rapidly when first increasing the number of iterations and then levels out. This indicates that even with a limited number of iterations, CE might be able to obtain fairly good results fast.

Figure 5 shows the values obtained by varying N, the number of joint policies sampled per iteration, which also increases the complexity linearly. The results were obtained using a fixed update fraction $\rho = 0.1$. Here too, we see that improvement is strong initially, and then flattens out later. Also note that the higher variance for $h = 5$ can be explained by looking at the performance for $I = 30$ in Figure 4.

The number of samples used to update the parameter vector ξ only marginally influences the run-time, and we were not able to determine this empirically. Also, using a larger fraction ρ decreases the performance. In this case, the quality of the sampled joint policies used to re-estimate the distribution parameters degenerates and CE will not converge towards good solutions. The results in figure 6 indicate optimal performance for ρ between 0.05 and 0.1. We omitted the nearly identical graph for $h = 5$.

In case DICE-A is used for policy search, the number of approximation runs influences the run time linearly. In figure 7 we can clearly see that the quality of the approximate evaluation converges quickly for horizon 4. For horizon 5 the size of the policies increases exponentially and more runs are needed to maintain approximation quality, but also here we see no improvement beyond $r = 1000$.

6.2 Results on different problems

Here we report the results of the performance of DICE on different Dec-POMDP problems and different horizon lengths. In particular we consider the following problems: BROADCAST CHANNEL, DEC-TIGER, MEETING ON A GRID and FACTORED FIREFIGHTING. The DEC-TIGER problem was discussed section 2.1. For the other problems we now provide a brief description.

The BROADCAST CHANNEL problem involves two agents that control a broadcast channel. Each agent decides at every stage whether or not to send a message across it. When both agents send a message at the same time a collision occurs. When a message is successfully transmitted, the agents get a reward of $+1$. More information can be found in (30; 17).

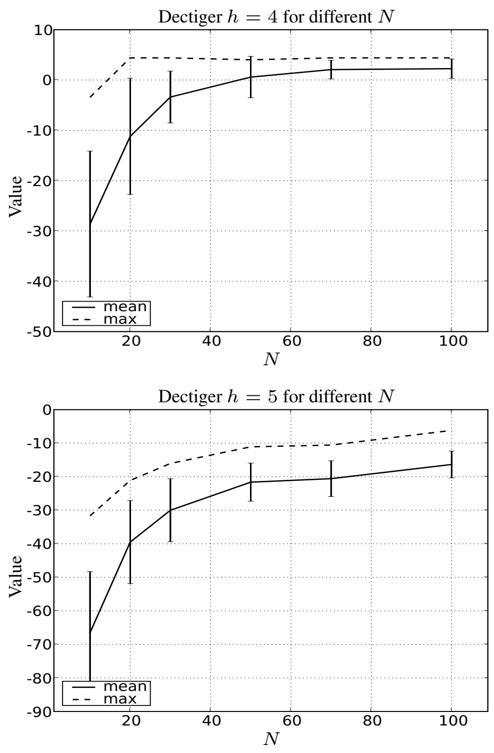


Figure 5: Performance under varying number of samples per CE iteration. Top: horizon 4. Bottom: horizon 5.

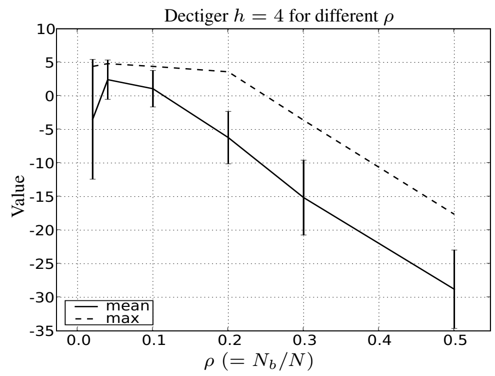


Figure 6: The fraction of samples ρ used to update the distribution f_ξ on horizon 4.

MEETING ON A GRID was introduced by Bernstein et al. (8) and considers two robots on a 2x2 grid. Their goal is to occupy the same square which gives them a $+1$ reward, but they have imperfect sensors and actuators which complicates the task. We consider the version with 2 observations per agent (2).

Finally, FACTORED FIREFIGHTING is a problem with 3 agents that have to fight fires at 4 houses (29). Each agent has 2 different houses it can go to (the sets are overlapping, but not equal). Every stage each agent should choose which of its 2 associated houses it wants to fight fire at. For each house that is burning, the agents receive a penalty of -1 or -2, depending on the level of fire.

Before showing results of the proposed CE approaches,

	Dec-Tiger	Broadcast	Grid	FFF
n	2	2	2	3
$\lvert\mathcal{S}\rvert$	2	4	16	81
$\lvert\mathcal{A}_i\rvert$	3	2	5	2
$\lvert\mathcal{O}_i\rvert$	2	2	2	2
2	7.290e02	6.400e01	1.563e04	5.120e02
3	4.783e06	1.638e04	6.104e09	2.097e06
4	2.059e14	1.074e09	9.313e20	3.518e13
5	3.815e29	4.612e18	2.168e43	9.904e27
6	1.310e60	8.507e37	1.175e88	7.846e56
7	1.545e121	2.895e76	3.454e177	4.925e114
8	2.147e243	3.352e153	Inf	1.941e230

Table 1: The number of joint policies for different problems and horizons. Inf denotes a value beyond double machine precision.

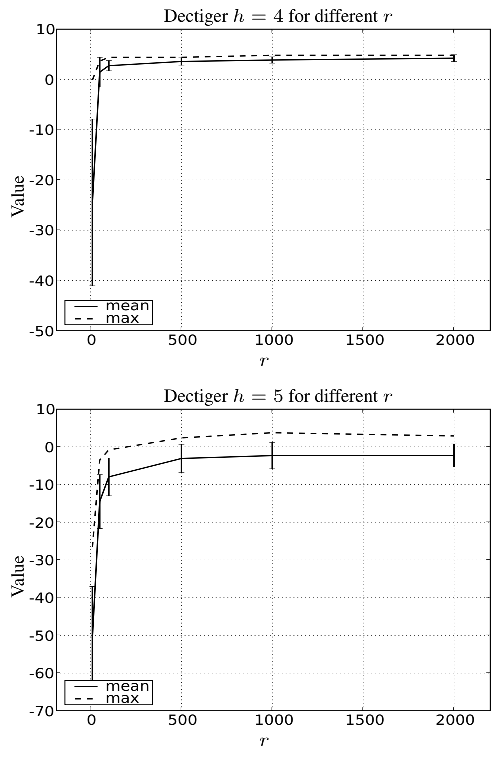


Figure 7: Performance for varying number of approximate evaluation runs. Top: horizon 4. Bottom: horizon 5.

we first report the size of the joint policy space for different considered problems in Table 1. Clearly we are dealing with huge search spaces here. In fact for $h = 8$ MEETING ON A GRID the number of joint policies was not representable by a double precision float (the maximum representable being 1.7977e+308). We emphasize that DICE directly searches these spaces, without first restricting them.

We report the results of DICE and DICE-A on these different test problems. We also compare against dynamic programming JESP which was proposed by Nair et al. (25). This method starts with a random joint policy and then iterates over the agents, computing a best-response policy for the selected agent while keeping the other agents fixed. The term “dynamic programming” indicates that this best-response is computed by solving an augmented POMDP for the selected agent. The reason to compare against JESP is that, as mentioned in the introduction, it is the only other approximate method that does not constrain the search space in any way.

The settings for DICE used in these experiments are: a learning rate of $\alpha = 0.2$, $N = 50$ joint policies per iteration, using the $N_b = 5$ best for update (i.e. $\rho = 0.1$). For DICE-A we used the same settings and the approximation was based on 1000 simulations. The results in this section are averaged over 100 random initializations, or restarts, of each solution method. However, due to the time needed for higher horizons, we have not always restarted 100 times for the highest horizon considered. The numerical results and the number of restarts over which they are obtained are listed in the appendix. The reported timing results are cpu-user times and obtained on an Intel Xeon 3.4 GHz machine with 2GB memory running Debian linux.

The results for the DEC-TIGER problem are shown in Figure 8. Note that the run-time results in the right figure use a log scale. Our evaluation shows that for low horizons ($h < 6$) DICE outperforms JESP but has taken, under these settings, more time to finish. However, the run-time of JESP increases much faster for larger horizons: for horizon 8 it is about ten times slower than DICE. The run-time of DICE-A is affected least by the value of the horizon.

In terms of quality of the found policy, DICE outperforms JESP for lower horizons: although all methods found (near-)optimal solutions for $h = 2, 3, 4$ within the 100 restarts, the variance of JESP is much higher. Unfortunately, the joint policies found by DICE for $h > 5$ are not very good, as the performance drops below JESP. However, this exposes the potential of approximate evaluation: which allows for much more iterations in less time than regular DICE. We also ran DICE-A with 200 iterations (20 restarts). While using approximation with the same settings does not seem to significantly change the results of the CE method by itself (DICE and DICE-A wth 50 iterations perform roughly equal), it does allow for more iterations, leading to good overall results while keeping the run-time acceptable. Only for $h = 8$, JESP really seems to achieve a better result, but with high variance, and high run-times.

For BROADCAST CHANNEL the results are shown in Figure 9. It shows that CE achieves a higher mean value and less variance with again little difference between DICE and DICE-A. The run-time of DICE-A on the other hand increases again much slower than the other two approaches, which eventually is more beneficial.

As indicated by Table 1, the MEETING ON A GRID problem is somewhat larger than the previous problems. This quickly makes exact evaluation problematic. For example, to compute the value of a horizon 6 joint policy 2.18e4 $(s, \vec{o})$-pairs have to be evaluated. Figure 10 shows that while JESP requires much more time than DICE, it does not result in better performance. The rightmost plot shows that the run-time of DICE-A is significantly lower from horizon 5 on. However, starting at $h = 6$ DICE-A seems to get trapped in a local optimum. Still, it is the only method to compute a policy for horizons 7 and 8.

The last problem, FACTORED FIREFIGHTING , is even larger. Because there are now 3 agents, the number of joint observation histories, and thus the number of entries to compute for exact evaluation and for JESP grows much faster. This is reflected by the results as the graphs in Figure 11 show. DICE and JESP can only find solutions for at maximum $h = 3$ and $h = 4$ respectively. Again this demonstrates the power of DICE-A, which does not encounter any problems computing results up to $h = 8$.

In general DICE and DICE-A seem to perform quite well in comparison to JESP. Although JESP's maximum value is usually equal or greater than the maximum value found by the DICE methods, its mean value is lower and the standard deviation is high. This indicates the need for many restarts in order to find a good solution and performing a lot of restarts becomes problematic for higher horizons, because of the exponential increase in run-time. The results of DICE, however, have a much lower standard deviation, indicating that less restarts are necessary. It still shares the burden of exponentially increasing run-times, though. DICE-A has proven to be a very powerful method. Even though the standard deviations are somewhat greater than regular DICE, it is able to trade off run-time and accuracy and thus achieves reasonable results even for higher horizons, when the other methods fail.

6.3 Varying the number of agents

In the previous section we investigated the ability of DICE to scale with respect to the horizon. Here we investigate the scaling behavior with respect to the number of agents.

So far, almost all available Dec-POMDP problems involve only two agents. Two exceptions are the FIREFIGHTING problem introduced by Oliehoek et al. (28), and the FACTORED FIREFIGHTING already mentioned. Here we will use the former (non-factored) version, since this allows us to vary the number of agents, while keeping the number of houses (and thus the number of states) constant.

Again we examined the performance of DICE, DICE-A and JESP for this problem, now varying the number of agents. We did this at two chosen horizons $h = 3$ and

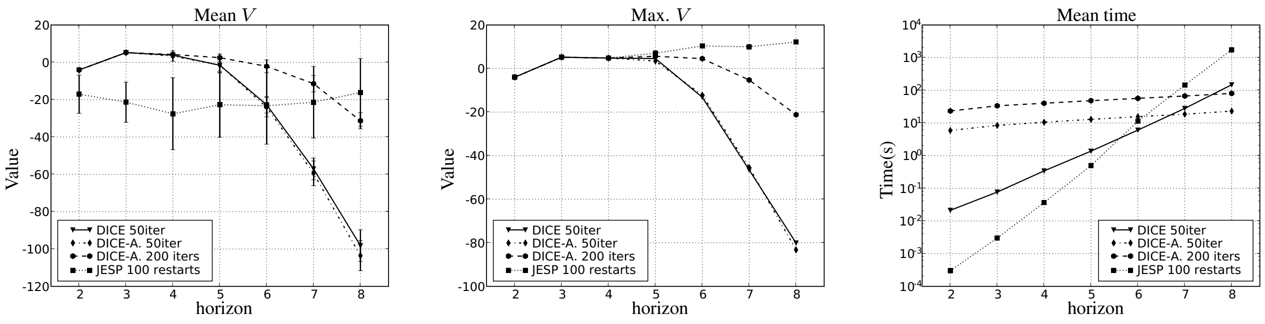


Figure 8: The DEC-TIGER problem. Left: average value. Middle: maximum value. Right: average run-time.

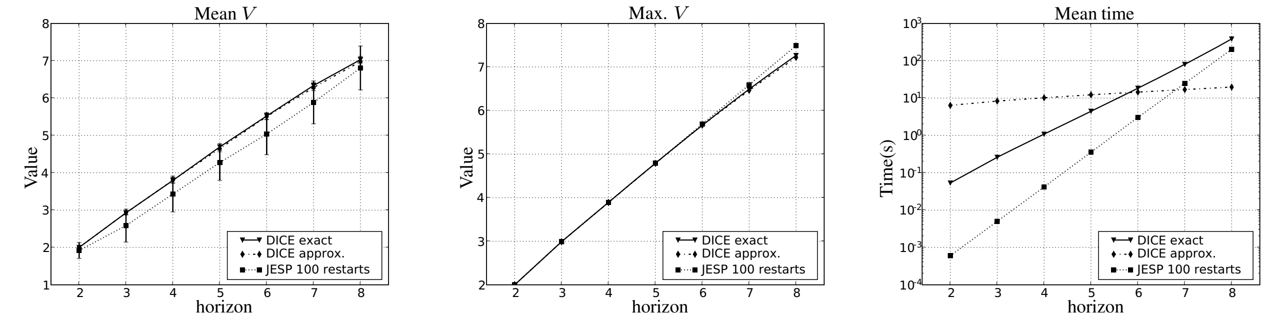


Figure 9: The BROADCAST CHANNEL problem. Left: average value. Middle: maximum value. Right: average run-time.

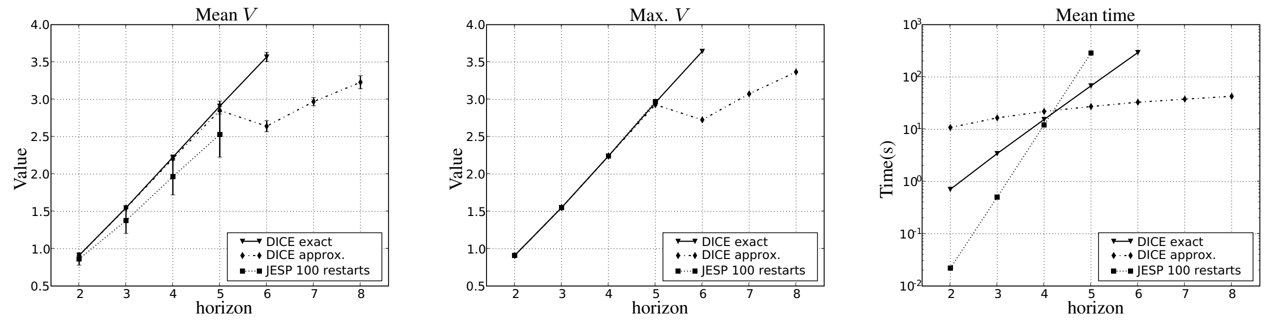


Figure 10: The MEETING ON A GRID problem. Left: average value. Middle: maximum value. Right: average time.

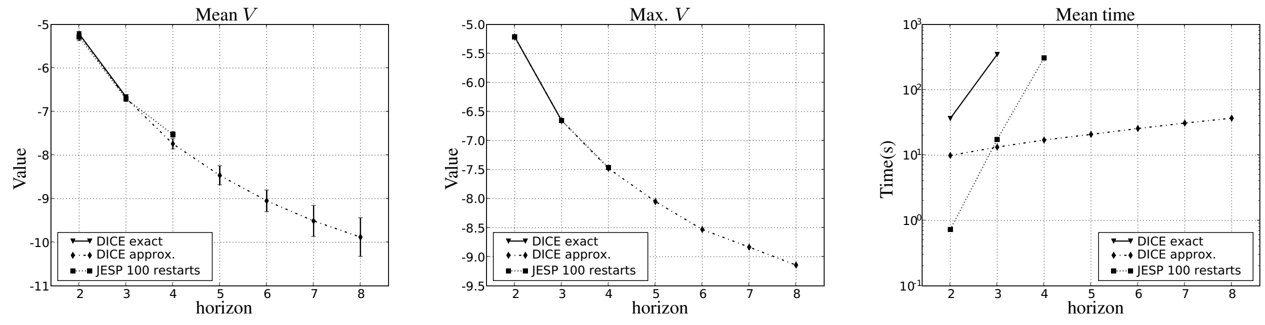


Figure 11: The FACTORED FIREFIGHTING problem with several restarts. Left: average value. Middle: maximum value. Right: average run-time.

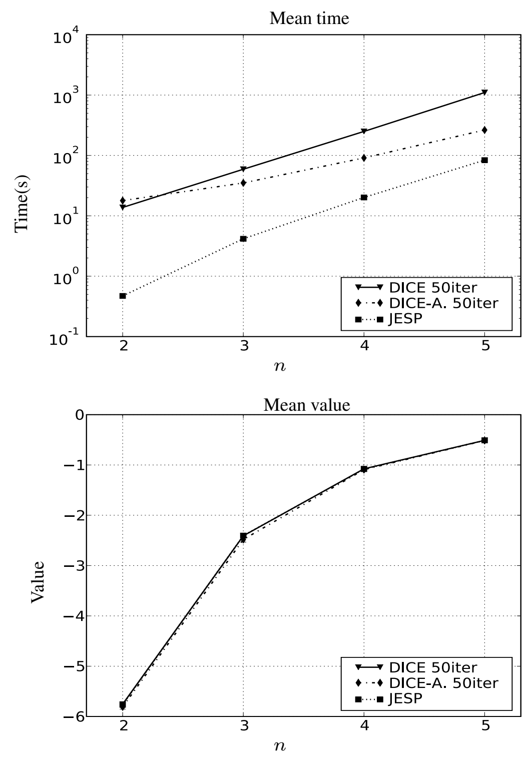


Figure 12: Performance for varying number of agents for $h = 3$ of the FIREFIGHTING problem.

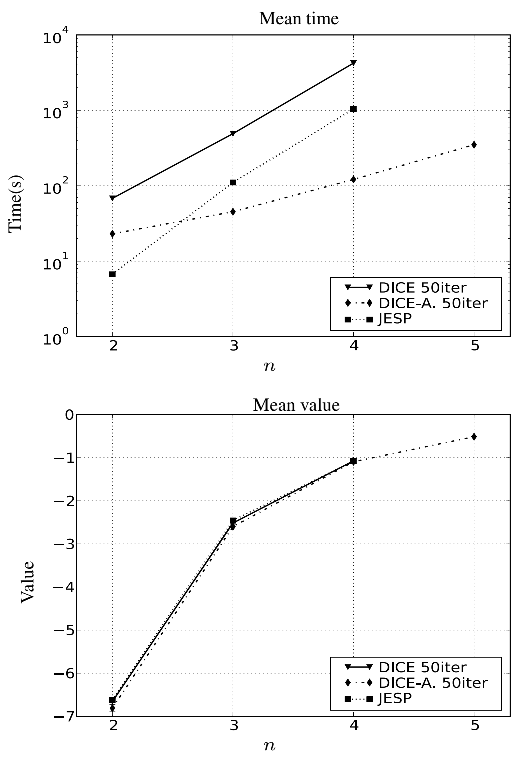


Figure 13: Performance for varying number of agents for $h = 4$ of the FIREFIGHTING problem.

$h = 4$, the results for which are shown in respectively Figure 12 and Figure 13. We used the same settings for the DICE algorithms as before. The number of restarts for all methods was set to 20.

The figures show that, for this problem, all methods performed very well in terms of achieved value. The maximum found value of all policies coincided (therefore these plots are omitted), which may indicate that these are true global optima. More interesting are the run time results. For $h = 3$, we see that JESP outperforms the DICE algorithm for all number of agents. However, its increase of run time when increasing the number of agents is higher than for DICE and particularly DICE-A. This is emphasized by the results for $h = 4$, that clearly show that run time for JESP, but also for exact DICE, grow so fast that they are unable to compute results for 5 agents within reasonable time.[5] The run times of the two different settings of DICE-A, however, grow much slower and as a consequence these methods are able to find a good solution for 5 agents. Note that in accordance with (5.2), the run time still increases exponentially in the number of agents, simply because simulating an episode (sampling joint observations) takes time exponential in the number of agents. In problems that exhibit observation independence (such as FIREFIGHTING), it is possible to work around this. We have not considered this efficiency increasing measure further, but stress that the efficiency of DICE-A could be further improved for such problems.

[5]Each algorithm was given 8 hours to compute all results for a given horizon.

7 Discussion and future work

This article has focused on decentralized decision making formalized in the Dec-POMDP framework. We argued that planning in such settings in principle is a complex combinatorial decision process. We demonstrated how to apply the CE-method, a recently introduced method for combinatorial optimization, for policy search in Dec-POMDPs.

We detailed the resulting algorithm, direct CE (DICE) policy search for Dec-POMDPs, and performed a complexity analysis, which identified the exact evaluation of sampled joint policies as a bottleneck. Consequently we proposed DICE-A which performs an approximate evaluation, and showed that, under some assumption, its time complexity is polynomial in the CE parameters. We also presented a formalization of the error bounds for this approximate evaluation.

We presented an empirical evaluation of the influence of the different CE parameters on policy search and also tested performance on different test problems from literature, over different horizons. In these latter experiments we compared against JESP, which, to our knowledge, is the only other approximate planning method for Dec-POMDPs that does not restrict the search space in any way. The results of this comparison were generally favorable for CE.

In particular, a nice feature of CE is that by adjusting the parameters, one is able to control the run-time. On the other hand, because JESP has no parameters, it is somewhat more easy to apply. In a final comparison we investigated how well the mentioned algorithms scale with respect to the number of agents. Although still exponential, DICE-A outperforms the other methods for larger problems.

However, this work does not intend to present a new state-of-the-art Dec-POMDP solver: we compare against JESP, which is one of the older Dec-POMDP algorithms, and more advanced methods have since been proposed. Rather our work shows that viewing these problems as combinatorial optimization problems and applying corresponding methods (such as CE optimization) can bring leverage to the planning process.

An interesting direction for future work is the application of the CE method (and potentially other methods for combinatorial optimization) in more recent algorithms. For instance, a Dec-POMDP can be approximated by solving for each stage t a pruned Bayesian game (BG) (13) or a compressed BG (14). The optimal solution of such BGs, however, involves enumeration of all the corresponding joint policies. CE might prove to be very effective to find approximate solutions for the BGs fast. In particular, it might turn out that the pruning/compression is no longer necessary for the same horizon when applying CE, and that when combining pruning/compression and CE, the algorithm can scale to higher h.

Another example is the method proposed by Seuken and Zilberstein (34). This method is an extension of dynamic programming for Dec-POMDPs that fixes the maximum number of policy trees that are retained in each iteration to a parameter *maxTrees*. The authors report that:

> "Even for a small problem 2 actions and 5 observations, setting *maxTrees=5* would be prohibitive because $(2 \cdot 5^2)^2 = 39,062,500$ policy tree pairs would have to be evaluated."

It might be possible to apply CE to this smaller policy search problem that occurs in each iteration of the DP process. This could lead to in improved efficiency, or the space could be less restricted in order to find a better approximate solution.

Other directions for future research would involve improving the efficiency of the CE method itself. One idea for this would be to use crude value approximation in the first iterations to quickly increase the probabilities of promising policies. In the course of the process, evaluation can be performed more accurately. Exact evaluation can most likely be accelerated by caching (intermediate) evaluation results of (parts of) joint policies. Also, the joint policies resulting from CE search might be improved by using those as a starting point for JESP, leading to a hybrid optimization scheme for multiagent settings.

Finally, and somewhat related, the success of approximate evaluation raises the question whether it is necessary to sample complete joint policies if they are only partially inspected during approximate evaluation. The CE approach could benefit from a construction that samples parts of (joint) policies.

Acknowledgments

We would like to thank Matthijs T.J. Spaan for his help and advice. The research reported here is part of the Interactive Collaborative Information Systems (ICIS) project, supported by the Dutch Ministry of Economic Affairs, grant nr: BSIK03024.

References

[1] G. Alon, D. Kroese, T. Raviv, and R. Rubinstein. Application of the cross-entropy method to the buffer allocation problem in a simulation-based environment. *Annals of Operations Research*, 134(1):137–151, 2005.

[2] C. Amato, D. S. Bernstein, and S. Zilberstein. Optimal fixed-size controllers for decentralized POMDPs. In *Proc. of the AAMAS Workshop on Multi-Agent Sequential Decision Making in Uncertain Domains (MSDM)*, May 2006.

[3] C. Amato, A. Carlin, and S. Zilberstein. Bounded dynamic programming for decentralized POMDPs. In *Proc. of the AAMAS Workshop on Multi-Agent Sequential Decision Making in Uncertain Domains (MSDM)*, May 2007.

[4] R. Aras, A. Dutech, and F. Charpillet. Mixed integer linear programming for exact finite-horizon planning in decentralized POMDPs. In *The International Conference on Automated Planning and Scheduling*, 2007.

[5] R. Becker, S. Zilberstein, V. Lesser, and C. V. Goldman. Solving transition independent decentralized Markov decision processes. *Journal of Artificial Intelligence Research (JAIR)*, 22:423–455, December 2004.

[6] D. S. Bernstein, S. Zilberstein, and N. Immerman. The complexity of decentralized control of Markov decision processes. In *Proc. of Uncertainty in Artificial Intelligence*, pages 32–37, 2000.

[7] D. S. Bernstein, R. Givan, N. Immerman, and S. Zilberstein. The complexity of decentralized control of Markov decision processes. *Math. Oper. Res.*, 27(4): 819–840, 2002.

[8] D. S. Bernstein, E. A. Hansen, and S. Zilberstein. Bounded policy iteration for decentralized POMDPs. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI)*, 2005.

[9] Z. Botev and D. P. Kroese. Global likelihood optimization via the cross-entropy method with an application to mixture models. In *WSC '04: Proceedings of the 36th conference on Winter simulation*, pages 529–535, 2004.

[10] C. Boutilier. Planning, learning and coordination in multiagent decision processes. In *TARK '96: Proceedings of the 6th conference on Theoretical aspects of rationality and knowledge*, pages 195–210, San Francisco, CA, USA, 1996. Morgan Kaufmann Publishers Inc. ISBN 1-55860-417-9.

[11] I. Cohen, B. Golany, and A. Shtub. Managing stochastic finite capacity multi-project systems through the cross-entropy method. *Annals of Operations Research*, 134(1):183–199, 2005.

[12] P.-T. de Boer, D. P. Kroese, S. Mannor, and R. Y. Rubinstein. A tutorial on the cross-entropy method. *Annals of Operations Research*, 134(1):19–67, 2005.

[13] R. Emery-Montemerlo, G. Gordon, J. Schneider, and S. Thrun. Approximate solutions for partially observable stochastic games with common payoffs. In *Proc. of Int. Joint Conference on Autonomous Agents and Multi Agent Systems*, pages 136–143, 2004.

[14] R. Emery-Montemerlo, G. Gordon, J. Schneider, and S. Thrun. Game theoretic control for robot teams. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1175–1181, 2005.

[15] C. V. Goldman and S. Zilberstein. Decentralized control of cooperative systems: Categorization and complexity analysis. *Journal of Artificial Intelligence Research (JAIR)*, 22:143–174, 2004.

[16] C. Guestrin, D. Koller, and R. Parr. Multiagent planning with factored MDPs. In *Advances in Neural Information Processing Systems 14*, pages 1523–1530, 2002.

[17] E. A. Hansen, D. S. Bernstein, and S. Zilberstein. Dynamic programming for partially observable stochastic games. In *Proc. of the National Conference on Artificial Intelligence*, pages 709–715, 2004.

[18] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, Mar. 1963.

[19] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2):99–134, 1998.

[20] Y. Kim, R. Nair, P. Varakantham, M. Tambe, and M. Yokoo. Exploiting locality of interaction in networked distributed POMDPs. In *Proceedings of the of the AAAI Spring Symposium on Distributed Plan and Schedule Management*, 2006.

[21] J. R. Kok and N. Vlassis. Using the max-plus algorithm for multiagent decision making in coordination graphs. In *RoboCup-2005: Robot Soccer World Cup IX*, Osaka, Japan, July 2005.

[22] D. Koller and A. Pfeffer. Representations and solutions for game-theoretic problems. *Artificial Intelligence*, 94(1-2):167–215, 1997.

[23] O. Madani, S. Hanks, and A. Condon. On the undecidability of probabilistic planning and infinite-horizon partially observable Markov decision problems. In *Proc. of the National Conference on Artificial Intelligence*, pages 541–548, 1999.

[24] S. Mannor, R. Rubinstein, and Y. Gat. The cross entropy method for fast policy search. In *Proc. of the International Conference on Machine Learning*, pages 512–519, 2003.

[25] R. Nair, M. Tambe, M. Yokoo, D. V. Pynadath, and S. Marsella. Taming decentralized POMDPs: Towards efficient policy computation for multiagent settings. In *Proc. of the Int. Joint Conf. on Artificial Intelligence*, pages 705–711, 2003.

[26] R. Nair, P. Varakantham, M. Tambe, and M. Yokoo. Networked distributed POMDPs: A synthesis of distributed constraint optimization and POMDPs. In *Proc. of the National Conference on Artificial Intelligence*, pages 133–139, 2005.

[27] F. A. Oliehoek, J. F. Kooij, and N. Vlassis. A cross-entropy approach to solving Dec-POMDPs. In *International Symposium on Intelligent and Distributed Computing*, pages 145–154, Oct. 2007.

[28] F. A. Oliehoek, M. T. J. Spaan, and N. Vlassis. Optimal and approximate Q-value functions for decentralized POMDPs. *Journal of Artificial Intelligence Research*, 32:289–353, 2008.

[29] F. A. Oliehoek, M. T. J. Spaan, S. Whiteson, and N. Vlassis. Exploiting locality of interaction in factored Dec-POMDPs. In *Proc. of Int. Joint Conference on Autonomous Agents and Multi Agent Systems*, pages 517–524, 2008.

[30] J. M. Ooi and G. W. Wornell. Decentralized control of a multiple access broadcast channel: Performance bounds. In *Proc. 35th Conf. on Decision and Control*, 1996.

[31] M. J. Osborne and A. Rubinstein. *A Course in Game Theory*. The MIT Press, July 1994.

[32] D. V. Pynadath and M. Tambe. The communicative multiagent team decision problem: Analyzing teamwork theories and models. *Journal of AI research (JAIR)*, 16:389–423, 2002.

[33] M. Roth, R. Simmons, and M. Veloso. Exploiting factored representations for decentralized execution in multi-agent teams. In *Proc. of Int. Joint Conference on Autonomous Agents and Multi Agent Systems*, pages 467–463, May 2007.

[34] S. Seuken and S. Zilberstein. Improved memory-bounded dynamic programming for decentralized POMDPs. In *Proc. of Uncertainty in Artificial Intelligence*, July 2007.

[35] S. Seuken and S. Zilberstein. Memory-bounded dynamic programming for DEC-POMDPs. In *Proc. of the Int. Joint Conf. on Artificial Intelligence*, pages 2009–2015, 2007.

[36] M. T. J. Spaan and F. S. Melo. Interaction-driven Markov games for decentralized multiagent planning under uncertainty. In *Proc. of Int. Joint Conference on Autonomous Agents and Multi Agent Systems*, pages 525–532, 2008.

[37] D. Szer and F. Charpillet. Point-based dynamic programming for DEC-POMDPs. In *Proc. of the National Conference on Artificial Intelligence*, 2006.

[38] D. Szer, F. Charpillet, and S. Zilberstein. MAA*: A heuristic search algorithm for solving decentralized POMDPs. In *Proc. of Uncertainty in Artificial Intelligence*, 2005.

[39] P. Varakantham, J. Marecki, Y. Yabu, M. Tambe, and M. Yokoo. Letting loose a SPIDER on a network of POMDPs: Generating quality guaranteed policies. In *Proc. of Int. Joint Conference on Autonomous Agents and Multi Agent Systems*, 2007.

Appendix

Tables 2 to 5 give a numerical overview of the results presented in section 6.2 and 6.3.

DEC-TIGER				
horizon	avg. value	std. dev. value	max. value	avg. time
DICE (50 iterations)				
2	-4.08	0.78	-4.00	0.02
3	5.19	0.00	5.19	0.08
4	3.81	1.28	4.80	0.34
5	-1.58	4.15	4.58	1.37
6	-22.75	4.40	-13.20	6.05
7	-57.11	5.85	-46.56	27.93
8	-98.23	8.47	-80.09	147.73
DICE-A (50 iterations)				
2	-4.19	0.70	-4.00	5.90
3	5.19	0.00	5.19	8.43
4	3.35	2.76	4.80	10.55
5	-1.56	3.11	3.45	12.90
6	-24.05	5.14	-12.22	15.60
7	-59.45	6.61	-45.43	18.58
8	-103.59	8.04	-83.27	23.02
DICE-A (200 iterations)				
2	-4.14	0.60	-4.00	23.26
3	5.19	0.00	5.19	33.54
4	4.16	0.87	4.80	40.20
5	2.50	1.93	5.63	48.10
6	-2.17	3.42	4.53	56.69
7	-11.46	4.39	-5.28	66.94
8	-31.30	4.38	-21.21	80.40
JESP				
2	-17.09	10.22	-4.00	0.00
3	-21.36	10.76	5.19	0.00
4	-27.58	19.26	4.80	0.04
5	-22.72	17.44	7.03	0.49
6	-23.28	20.57	10.38	11.32
7	-21.42	19.16	9.99	144.21
8	-16.20	18.26	12.22	1741.39

Table 2: Results for the DEC-TIGER problem. Statistics over 100 restarts, except for DICE-A with $I = 200$ (which we performed with 20 restarts) and horizon 8 JESP (which only completed 30 restarts).

FACTORED FIREFIGHTING				
horizon	avg. value	std. dev. value	max. value	avg. time
DICE (50 iterations)				
2	-5.21	0.00	-5.21	36.45
3	-6.66	0.01	-6.65	347.89
DICE-A (50 iterations)				
2	-5.22	0.02	-5.21	9.89
3	-6.69	0.03	-6.65	13.27
4	-7.74	0.12	-7.48	17.05
5	-8.46	0.22	-8.05	20.77
6	-9.05	0.25	-8.53	25.57
7	-9.51	0.35	-8.83	30.92
8	-9.88	0.44	-9.14	36.81
JESP				
2	-5.28	0.09	-5.21	0.73
3	-6.71	0.06	-6.65	17.41
4	-7.52	0.07	-7.46	308.36

Table 3: Results for the FACTORED FIREFIGHTING problem over 100 restarts.

BROADCAST CHANNEL				
horizon	avg. value	std. dev. value	max. value	avg. time
DICE (50 iterations)				
2	2.00	0.00	2.00	0.05
3	2.93	0.05	2.99	0.26
4	3.80	0.08	3.89	1.08
5	4.69	0.09	4.79	4.45
6	5.52	0.09	5.67	18.37
7	6.34	0.08	6.48	79.89
8	7.03	0.09	7.26	384.23
DICE-A (50 iterations)				
2	2.00	0.00	2.00	6.38
3	2.92	0.05	2.99	8.32
4	3.80	0.07	3.89	10.20
5	4.65	0.09	4.79	12.26
6	5.50	0.08	5.66	14.47
7	6.28	0.08	6.46	16.93
8	6.98	0.11	7.22	19.74
JESP				
2	1.92	0.21	2.00	0.00
3	2.59	0.44	2.99	0.01
4	3.43	0.48	3.89	0.04
5	4.27	0.47	4.79	0.36
6	5.04	0.55	5.69	3.03
7	5.88	0.57	6.59	24.71
8	6.81	0.59	7.49	202.46

Table 4: Results for the BROADCAST CHANNEL problem over 100 restarts.

MEETING ON A GRID				
horizon	avg. value	std. dev. value	max. value	avg. time
DICE (50 iterations)				
2	0.91	0.00	0.91	0.71
3	1.55	0.01	1.55	3.43
4	2.23	0.02	2.24	15.37
5	2.91	0.07	2.96	67.60
6	3.57	0.06	3.64	292.88
DICE-A (50 iterations)				
2	0.91	0.00	0.91	10.77
3	1.54	0.01	1.55	16.51
4	2.21	0.02	2.24	21.86
5	2.85	0.05	2.93	27.19
6	2.64	0.07	2.73	32.86
7	2.97	0.06	3.07	37.66
8	3.23	0.09	3.37	42.63
JESP				
2	0.86	0.08	0.91	0.02
3	1.38	0.17	1.55	0.50
4	1.97	0.24	2.24	12.11
5	2.53	0.30	2.97	287.02

Table 5: Results for the MEETING ON A GRID problem over 100 restarts.

FIREFIGHTING				
n	avg. value	std. dev. value	max. value	avg. time
DICE (50 iterations)				
2	-5.76	0.02	-5.74	13.77
3	-2.41	0.04	-2.39	59.07
4	-1.08	0.01	-1.07	249.88
5	-0.51	0.00	-0.51	1101.54
DICE-A (50 iterations)				
2	-5.80	0.04	-5.74	17.86
3	-2.48	0.05	-2.39	35.05
4	-1.09	0.02	-1.07	91.02
5	-0.51	0.01	-0.51	263.92
JESP				
2	-5.76	0.04	-5.74	0.47
3	-2.41	0.03	-2.39	4.17
4	-1.08	0.01	-1.07	20.15
5	-0.51	0.00	-0.51	83.61

Table 6: Results for the $h = 3$ FIREFIGHTING problem with varying number of agents over 20 restarts.

FIREFIGHTING				
n	avg. value	std. dev. value	max. value	avg. time
DICE (50 iterations)				
2	-6.66	0.06	-6.58	68.03
3	-2.52	0.02	-2.45	490.58
4	-1.07	0.00	-1.07	4227.08
DICE-A (50 iterations)				
2	-6.81	0.08	-6.67	23.07
3	-2.59	0.08	-2.45	45.22
4	-1.09	0.02	-1.07	121.37
5	-0.51	0.00	-0.50	350.04
JESP				
2	-6.62	0.03	-6.58	6.71
3	-2.45	0.03	-2.44	110.57
4	-1.08	0.01	-1.07	1040.75

Table 7: Results for the $h = 4$ FIREFIGHTING problem with varying number of agents over 20 restarts. For $n = 4$ is DICE shows an average over 4 completed runs and JESP over 18 completed runs.

On the Compilation of Programs into their Equivalent Constraint Representation

Franz Wotawa and Mihai Nica
Technische Universität Graz, Institute for Software Technology, Inffeldgasse 16b/2, 8010 Graz, Austria
E-mail: {wotawa,mihai.nica}@ist.tugraz.at

Keywords: constraint satisfaction problems, hyper-graph decomposition, debugging

Received: March 15, 2008

In this paper we introduce the basic methodology for analyzing and debugging programs. We first convert programs into their loop-free equivalents and from this into the static single assignment form. From the static single assignment form we derive a corresponding constraint satisfaction problem. The constraint representation can be directly used for debugging. From the corresponding hyper-tree representation of the constraint satisfaction problem we compute the hyper-tree width which characterizes the complexity of finding a solution for the constraint satisfaction problem. Since constraint satisfaction can be effectively used for diagnosis the conversion can be used for debugging and the obtained hyper-tree width is an indicator of the debugging complexity.

Povzetek: Članek opisuje analiziranje programov in iskanje napak v njih.

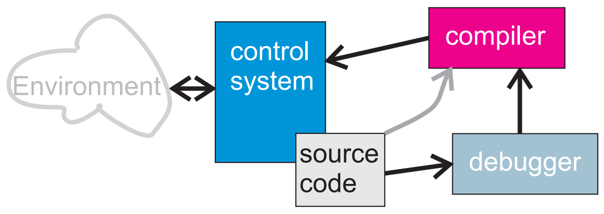


Figure 1: Interaction between the control and the debugging system

1 Introduction

Ideally intelligent systems should provide self-reasoning and reflection capabilities in order to react on internal faults as well as on unexpected interaction with their environment. Reflection capabilities are highly recommended for systems with strong robustness constraints, like space exploration probes or even mobile robots. A scenario, for example, is a robot that although having a broken engine, should reach a certain position. Without self-reasoning or reflection such a robot would simple fail to reach its goal. Another example would be a robot where the software fails because of a bug. In this situation a robot should recover and ideally repair itself. Note that even exhaustive testing does not prevent a program from containing bugs which might cause an unexpected behavior in certain situations.

In this paper we do not focus on whole systems which comprise hardware and software. Instead we are discussing how to represent programs to allow for reflection which can be used for enhancing the system with debugging functionality. In the context of this paper debugging is defined as fault localization given a certain test-case. We do not take care of verification and test-case generation which is used for fault detection and repair. In order to compute the fault location we follow the model-based diagnosis approach (22) but do not rely on logical models but use constraints instead for representing programs. The obtained constraint representation can be directly used for computing diagnosis, e.g., by using specialized diagnosis algorithms like the one described in (12; 25; 26).

Although, reflection and debugging capabilities are a desired functionality of a system they provoke additional computational complexity which can hardly be handled by the system directly because of lack of computational power. Note that model-based diagnosis is NP complete. Hence, a distributed architecture would be required which separates the running control program from the debugging capabilities. In Figure 1 we depict the proposed architecture. The debugging module takes the source code of the original system which is in this case a control system and a test-case to localize and repair the fault. The changed source code is compiled and transferred back to the original system.

In the proposed architecture there are several open issues which have to be solved. The first is regarding the test case. In particular one is interest in the origin of the test case. One way would be to have a monitoring system which checks the internal state of the control system. In case of a faulty behavior, the given inputs coming from the sensors, the internal state and the computed output together with the information regarding the expected output is used as a test case. The second issue is related to fault localization and repair. There is literature explaining fault localization and repair which can be used, e.g., (13; 24; 17; 23).

In this paper we focus on modeling for fault localization, i.e., for identifying the root cause of a detected misbehavior. This does not necessarily directly lead to good repair suggestions. Repair definitely requires knowledge about further specification details and incorporating different test-cases should help a lot. A detailed discussion of repair is outside the scope of this paper and still an important topic of research. The third and last issue is a more technical one. After recompiling the new program has to replace the old one on the fly which might cause additional problems because of ensuring integrity and consistency of the system's behavior.

Making systems self-aware and giving those self-healing capabilities increases their autonomy, which is important in missions like space exploration where the system cannot be directly controlled for whatever reason. The paper contributes to this research in providing a methodology for fault localization in programs. The methodology requires the availability of the source code and test cases. It compiles the program into their equivalent constraint representation and uses a failure-revealing test case to compute diagnosis candidates.

The paper is organized as follows. We first introduce the basic ideas behind our approach by means of an example. Afterwards, we go through all necessary steps of the conversion process, which finally leads to the constraint representation of programs. This section is followed by a presentation of experimental results we obtained from our implementation. The experiments focus on structural properties of the programs' constraint representation because the structural properties have an impact on the complexity of debugging. Finally, we discuss related research and conclude the paper.

2 Diagnosis/debugging

In this section we motivate the idea behind our approach by means of an example program that is expected to compute the area and circumference of a circle from a given diameter.

```
1.   r = d/2;
2.   c = r*pi;
3.   a = r*r*pi;
```

Obviously the statement in line 2 is faulty. A possible repair would be `c = 2*r*pi;` or `c = d*pi;`. We know this because of our knowledge in mathematics. However, in the more general case where large programs are involved, we have to rely on a process in order to detect, localize, and correct a fault. Such a process deals with the question: 'How can we find out what's wrong in the program' and is a standard in today's software engineering processes. In software engineering we first have to detect the fault. This is done in practice using test-cases or properties together with formal verification methods. In our case we rely on test cases. One test case which allow to detect the faulty behavior would be setting `d` to 2 and requiring `c` to be 2π and `a` to be π. The program computes the value π for both variables `c` and `a` which contradicts the test case.

This contradiction can be used to locate the fault. One way would be to have a look at the first definition of variable `c` which has assigned a wrong value and trace back using the dependencies between variables. In this case `r` is defined in line 1 and used in line 2. Hence, line 1 and 2 are possible fault candidates. This approach uses only structural information coded in the program and might overestimate the number of diagnosis candidates. Another approach would be to assume some line of the program to be faulty. In this case the statement corresponding to the line does not provide any known functionality. For example, when assuming line 2 to be faulty, we do not know how to compute a value for `c`. Therefore, we do not get any contradicting information and the assumption is consistent with the given test-case. Unfortunately, this is also the case when assuming line 1 to be faulty and only considering the computation of values like specified in the semantics of the programming language, i.e., from the begin of the program to its end.

The situation changes when considering the statement as equations where no direction of information flow is given. An equation like $r = d/2$ is a constraint which specifies a relationship between r and d. When we now assume the constraint corresponding to line 1 to be faulty we can compute a value for r using the given test-case. From the constraint corresponding to line 3 $a = r^2\pi$ and $a = \pi$ we derive $r = 1$ and finally using $c = r\pi$ leads to $c = \pi$ which contradicts the given expected value for c which is 2π. Hence, the assumption in this case contradicts the observations and cannot be a single-fault candidate. Only line 2 remains as single fault. The reason for this improvement is that when using constraints we have the capabilities for reasoning backwards.

Summarizing the above discussion, a solution to the debugging problem would be to convert programs into their constraint representation and use it for debugging. We choose a constraint representation because equations as well as logical sentences can be represented and integrated, and because of the availability of tools. However, there are still some challenges we have to solve.

First, variables might be defined more than once in a program. Every definition has to be considered separately. Programs comprise conditional and loop statements. How to handle them? For the first problem and the conditionals we propose the use of the static single assignment (SSA) form of programs which is used in compiler construction.

Another challenge is how to handle loops and recursive procedure calls? In order to represent loops we make use of the following observation. Loops can be represented by nested if-statements where the nesting depth is infinite. When restricting the nesting depth to a finite value the nested if-statements still behave the same as the loop statement when considering only inputs which do not cause the number of loop-iterations to exceed the nesting depth. Hence, under this assumption the nested if-statements are good enough to represent loops and we have reduced the problem of handling loops to the problem of handling con-

```
1.  if x < y {
2.    min = x;
    } else {
3.    min = y;
}
```

Figure 2: A program computing the minimum of two numbers

```
1.  i = 0;
2.  r = 0;
3.  while (i < x) {
4.    r = r + y;
5.    i = i + 1;
    }
```

Figure 3: A program for computing the product of two natural numbers

ditionals. A similar technique can be applied to solve the recursive procedure calls challenge.

Finally, we have to handle arrays and other programming language constructs like pointers or objects. In this paper, we present an approach for handling arrays. The other aspects of programming languages are ignored. This is due to the fact that our main application area is the embedded-systems domain. Programs used in embedded-systems usually are restricted and do not use of features like dynamic memory allocation because such features are error prone and likely lead to problems during operation.

3 Conversion

In this section, we describe the conversion process of programs into their equivalent CSP representation in detail. For more information regarding CSPs we refer the reader to Dechter (11). We start with converting programs into their loop-free equivalents which are used as basis for the conversion into the SSA form. Finally, we show how to extract the CSP from the SSA form. The complexity of the proposed compilation approach is composed of the complexity of each conversion step. While SSA construction and CSP extraction can be both handled in polynomial time computing a loop-free equivalent depends on the number of necessary iterations. This number depends on the program's complexity. In this section we use the programs given in Figure 2 and 3 as running examples. We further assume without restricting generality of the approach that the programs to be converted have a Java like syntax (ignoring object-oriented elements).

3.1 Loop-free programs

When executing while-statements they behave like a conditional statement in one step. If the condition is fulfilled the statements in the block are executed and the while-statement is executed again afterwards. Otherwise, the while-statement is not executed. Hence, it is semantically correct to represent while-statements using an infinite number of nested if-statements, i.e., `while ( C ) { B }` is equivalent to

```
if ( C ) {
    B if ( C ) {
        B if ( C ) {
            B if ...  } } }
```

C represents the condition, and B the statements in the sub-block of the while statement.

Of course it is not possible in practice to compile while-statements into an infinite number of conditionals. Instead we assume that the number of iterations of the while-statement never exceeds a certain limit say n. We argue that faults can be detected using test-cases which cause a small number of iteration and that it is therefore – for the purpose of debugging – not necessary to consider larger values of n. Moreover, we might introduce a procedure which is called whenever the limit is reached. This information would give us back additional information which we might use for increasing n in a further step. To set a small bound for the number of iterations is also used in a different context. Jackson (18) uses a similar idea which he called *small scope hypothesis* in his Alloy system for verification.

We formalize the bounded conversion from while-statements into nested if-statements by introducing the function $\Gamma : PL \times \mathbb{N} \mapsto PL$ where PL represents the programming language.

$$\Gamma(\texttt{while (} C \texttt{) \{ } B\,\texttt{\}}, n) = \\ = \begin{cases} \text{if}\,(C\,)\,\{\, B\Gamma(\texttt{while } (C\,)\,\{\,B\}, n-1)\} & \textbf{if}\; n > 0 \\ \text{if}\,(\,C\,)\,\{\,\text{too\_many\_iterations;}\,\} & \textbf{otherwise} \end{cases}$$

Considering the above discussion it is obvious that the following theorem which states the equivalence of the program and its loop-free variant with respect to their behavior has to hold.

Theorem 3.1. *Given a program $\Pi \in PL$ written in a programming language PL and a number $n \in \mathbb{N}$. Π behaves equivalent to $\Gamma(\Pi, n)$ for an input I iff the execution of $\Gamma(\Pi, n)$ on I does not reach* `too_many_iterations`.

We now use the program from Figure 3 and $n = 2$ to show the application of Γ which leads to the following program:

```
1.  i = 0;
2.  r = 0;
3.  if (i < x) {
4.    r = r + y;
5.    i = i + 1;
6.    if (i < x) {
7.      r = r + y;
8.      i = i + 1;
9.      if (i < x) {
10.       too_many_iterations;
      } } }
```

The loop-free variant can be used in all cases where $x = 0$ or $x = 1$ without causing a different behavior to the original program. It is interesting to note that the time complexity of a program which is corresponds to the number of iterations depending on the size of the inputs is now represented by the size of the converted program. Hence, we easily can give an estimate for the size of the converted programs (and also the number of iterations) when limiting the size of the input.

3.2 Static single assignment form

The SSA form is an intermediate representation of a program, which has the property that no two left-side variables share the same name, i.e., each left-side variable has a unique name. Because of this reason the SSA form is of great importance when building the CSP. Since all variables are only defined once the SSA form allows for a clear representation of the dependencies that are established between different variables inside the corresponding program. The SSA representation of a program is sometime also an intermediate step in the compiling process; basically before compiling a java file, first a transformation is undertaken and the SSA form is obtained. The obtained form will be the form used as input for the compiler.

In order to compile loop-free programs into SSA we have to analyze the program and rename all variables such that each variable is only defined once without changing the behavior of the program. Basically, this compilation is done by adding an unique index to every variable, which is defined in a statement. Every use of this variable in the succeeding statement is indexed using the same value. This is done until a new definition of the same variable occur.

For example, the following program fragment on the upper side can be converted into its SSA representation bellow.

```
1.   x = a + b;
2.   x = x - c;
```

The SSA representation:

```
1.   x_1 = a_0 + b_0;
2.   x_2 = x_1 - c_0;
```

Although, converting programs comprising only assignment statements is easy, it is more difficult for programs with loop or conditional statements. In our case we only need to consider conditional statements. The idea behind the conversion of conditional statement is as follows: The value of the condition is stored in a new unique variable. The if- and the else-block are converted separately. In both cases the conversion starts using the indices of the variables already computed. Both conversions deliver back new indices of variables. In order to get a value for a variable we have to select the last definition of a variable from the if- and else-part depending on the condition. This selection is done using a function Φ. Hence, for every variable which is defined in the if- or the else-branch we have to introduce a selecting assignment statement which calls the Φ function.

For example, corresponding SSA form of the program fragment

```
if (C) {
    ..  x = ..
} else {
    ..  x = ..  }
```

is given as follows:

```
var_C = C;
..  x_i = ..
..  x_j = ..
x_k = Φ(x_i,x_j,var_C);
```

The function Φ is defined as follows: $\Phi(x, y, b) = \begin{cases} x & \text{if } b \\ y & \text{otherwise} \end{cases}$

For algorithms for computing the SSA form and more information regarding the Φ function we refer the reader to (9; 7; 27). The SSA representation for the programs from Figure 2 and 3 are depicted in Figure 4 and 5 respectively. Note that we represent the Φ function as a function call `phi` in the program. It is possible to write a function body for `phi` that exactly represents the behavior of the Φ function. Hence, the program and its SSA representation can be executed on the same input even at the level of the programming language.

It is easy to see that the SSA form of a program always behaves equivalent to the original program, which we state in the following theorem.

Theorem 3.2. *Given a program $\Pi \in PL$. The SSA representation $\Pi' \in PL$ of Π is equivalent to Π with respect to the semantics of PL, i.e., for all inputs I both programs return the same output.*

For debugging purposes the input output equivalence, which is similar to the input output conformance (IOCO) used in testing, is sufficient. The SSA representation allows us to map, with little effort, the diagnosis set of program Π' to the original equivalent program Π.

In the following subsection we describe how arrays and function calls can be handled. Afterwards we discuss the compilation of programs into constraint systems.

3.3 Extensions

In the previously described conversion process we still face two important challenges as they are the conversion of arrays and procedure calls. In this section, we tackle both challenges and present conversion rules that have to be applied before converting the resulting source code in a SSA form.

We start with the array conversion. We assume that arrays are defined over a data type (although we do not consider this information in the conversion) and are of fixed length. Because of simplicity, we assume that we can only access one element of the array after the other. Note that in some languages there are constructs, which allow accessing an array partially, e.g., element 3 to 5.

```
1.  var_1 = ( x_0 < y_0 );
2.  min_1 = x_0;
3.  min_2 = y_0;
4.  min_3 = phi(min_1,min_2,var_1);
```

Figure 4: The SSA form of the program from Fig. 2

```
1.  i_1 = 0;
2.  r_1 = 0;
3.  var_3 = (i_1 < x_0);
4.  r_2 = r_1 + y_0;
5.  i_2 = i_1 + 1;
6.  var_6 = (i_2 < x_0);
7.  r_3 = r_2 + y_0;
8.  i_3 = i_2 + 1;
9.  r_4 = phi(r_3,r_2,var_6);
10. i_4 = phi(i_3,i_2,var_6);
11. r_5 = phi(r_4,r_1,var_3);
12. i_5 = phi(i_4,i_1,var_3);
```

Figure 5: The SSA form of the loop-free variant of the program from Fig. 3

Given an array A of length $n > 0$ with elements $\langle a_1...a_i...a_n\rangle$. The access to elements is assumed to be done using the [] operator, which maps from A and a given index i to the array element a_i. For example, z = A[1] gives you back the first element of array A. So far, arrays seem not to be very difficult to handle in our conversion process. If reaching a statement z = A[1] during SSA conversion A has only be represented as A_k where k is the currently given unique index k for A. But how to handle statements like A[m] = A[n] + 3? In this case, obviously A has to be assigned a new index, but how to handle the following program fragment?

```
...
A[1] = 2;
A[2] = 4;
...
```

A SSA translation into

```
...
A_1[1] = 2;
A_2[2] = 4;
...
```

would be wrong because this transformation does not respect previously done array changes in the appropriate manner. In order to respect the underlying semantics, we have a closer look at it. Assume a program fragment A[i] = f($\vec{x}$) where the i-th element of A is set to the outcome of function f given parameters $\vec{x}$. This statement only changes the i-th element but not the others. Formally, we define the semantics as follows:

```
{A}// A before the statement
A[i] = f(x⃗)
{A′}// A after the statement
```

with A′[i] = f($\vec{x}$) and $\forall j \in \{1,\ldots,n\}, i \neq j$: A′[$j$] = A[$j$]. As a consequence, we introduce a function Ψ that implements this semantics and replace the original statement with A = Ψ(A,i,f($\vec{x}$)). The function Ψ is similar to Φ and can be implemented such that the converted program is equivalent to the original one with respect to its semantics.

For example, consider the following program fragment:

```
1.  A[1] = 5;
2.  A[2] = A[1] + 5;
```

Accordingly to our conversion rule we obtain the new fragment:

```
1.  A = psi(A,1,5);
2.  A = psi(A,2,A[1] + 5);
```

The SSA representation is not based on the new fragment and captures the semantic in the appropriate way.

```
1.  A_1 = psi(A_0,1,5);
2.  A_2 = psi(A_1,2,A_1[1] + 5);
```

Note that Ψ or psi can be implemented as a function in order to ensure the equivalent behavior even in the context of program execution. Assume that psi has the formal arguments A, i, e and that the length of an array can be accessed via a function $length$, then the body of the function is given as follows:

```
j = 1;
while (j < length(A)) {
    if (j==i) {
      B[j] = e;
    } else {
      B[j] = A[j];
    }
    j = j + 1;
}
return B;
```

The function psi returns a new array B of A's size.

We now consider the last challenge we want to tackle, the procedure calls and in particular recursive procedure calls. Given a procedure M with its formal parameters $x_1,\ldots,x_n$, and the body δ(M), which itself is a program written in the same programming language. A procedure call M($a_1, \ldots, a_n$) with actual parameters $a_1, \ldots, a_n$ causes the execution of M's body where the formal parameters are assigned to their corresponding actual parameters, i.e., $x_i = a_i$. Hence, a transformation is easy. We only have to assign values to the formal parameter, which can be done using assignments, and use the body of the procedure instead of the procedure call.

In cases where the procedure returns a value, we have to introduce a new variable M_return. We further replace return e with M_return = e, and finally, we introduce a new assignment, where M_return is used appropriately. The described transformation of the return statement is only correct, whenever a procedure has only one of those statements. This is not a restriction because we always can modify a program to fulfill this requirement. For simplicity, we also assume that the variables used in bodies of procedures and the one used in the main program are different except in cases of global variables.

We now explain the idea behind the conversion using an example program where a procedure foo is called

```
...
x = foo(2,y,z-1);
...
```

Now assume that foo has three formal parameters x1,x2,x3 and the following body:

```
v = x1 + x2;
v = v - x3;
return v;
```

When applying the described conversion rule, we obtain the following program fragment:

```
x1 = 2;
x2 = y;
x3 = z-1;
v = x1 + x2;
v = v - x3;
foo_return = v;
x = foo_return;
```

This program can be easily compiled into its SSA form:

```
x1_1 = 2;
x2_1 = y_0;
x3_1 = z_0-1;
v_1 = x1_1 + x2_1;
v_2 = v_1 - x3;
foo_return_1 = v_2;
x_1 = foo_return_1;
```

We now extend the above idea to the general case, where we might face recursive procedure calls. The idea behind the conversion is the same, but similar to the handling of iterations of a while statement, we have to set a bound on the maximum number of recursive replacements during conversion. For this purpose we assume that there is such a bound MC given for a procedure M in a calling context. Similar to while statements we introduce a function $\hat{\Gamma} : PL \times \mathbb{N} \mapsto PL$ that defines the bounded compilation of a not necessarily recursive procedure call:

$$\hat{\Gamma}(\texttt{[x =]M(a}_1\texttt{,...,a}_n\texttt{)}, i) = \\ = \begin{cases} \texttt{x}_1\texttt{ = a}_1\texttt{;...x}_n\texttt{ = a}_n\texttt{;} \\ \hat{\Gamma}(\delta'(\texttt{M}), i-1) \\ \texttt{[x = M\_return;]} \\ \qquad \textbf{if } i \leq MC \\ \texttt{too\_many\_recursions}; \\ \qquad \textbf{otherwise} \end{cases}$$

Note that δ' denotes the body of method M, where the return statement return e has been changed to M_return = e. Moreover, $\hat{\Gamma}$ is assumed to be defined for all other statements where the statement itself is returned without any changes. Hence, when applying $\hat{\Gamma}$ to the body of a method, all statements are examined and left as they are with the exception of a new method call. With these additional assumptions the compilation function can be generally applied.

We illustrate the conversion using a program only comprising the call x = foo(1,2); where the body of foo is given as follows:

```
r = 0;
if (x1 > 0) {
  r= foo(x1-1,x2);
  r = r + x2;
}
return r;
```

The following program represents the recursion-free variant of the program calling foo with 1 allowed iteration, which is sufficient for specific procedure call foo(1,2) in order to return the correct result.

```
    x1 = 1;
    x2 = 2;
// 1. recursive call
    r = 0;
    if (x1 > 0) {
      x1 = x1 - 1;
      x2 = x2;
// 2. recursive call
      r = 0;
      if (x1 > 0) {
        too_many_iterations;
      }
      M_return = r;
// returning from 2. call
      r = M_return;
      r = r + x2;
    M_return = r;
// returning from 1. call
    x = M_return;
```

The converted program can be easily compiled in its SSA form.

3.4 Constraint representation

Constraint satisfaction problems (CSPs) have been introduced and used in Artificial Intelligence as a general knowledge representation paradigm of knowledge. A CSP (V, D, CO) is characterized by a set of variables V, each variable having a domain D, and a set of constraints CO which define a relation between variables. The variables in a relation $r \in CO$ are called the scope of the relation. An assignment of values from D to variables from V is called an instantiation. An instantiation is a solution for a CSP, iff it violates no constraint. A constraint is said to be violated by an instantiation, if the value assignment of the variables in the scope of the constraint are not represented in the relation of the constraint. There are many algorithms available for computing valid instantiations, i.e., solutions for a CSP. A straight-forward algorithm is backtrack search where variable values are assigned and consistency checks are performed until a valid solution is found. Moreover, it is well known that CSPs can be solved in polynomial time under some circumstances, i.e., the CSP must be acyclic, which we discuss later. Yannakakis (30) proposed such an algorithm. Based on this algorithm there are diagnosis algorithms (12; 25) available, which can be used for debugging directly providing there is a CSP representation for programs. For more information and details on CSPs we refer the reader to DechterŠs book (11).

The last step in the conversion process is to map programs in SSA form to their CSP representation. This extraction of the constraints from the statements of the SSA representation can be easily done. The algorithm is pretty straight forward and only implies analyzing the SSA representation line by line. In this conversion step, each line of the SSA representation is mapped directly to a constraint. Hence, all variables of a statement map directly to the scope of the constraint. The constraint relation is given by the statement itself by interpreting the assignment operator as an equivalence operator. For example, the statement `x_2 = x_1 + 2` is mapped to a constraint having 2 variables `x_2` and `x_1` and where the relation is stated as an equation of the form $\text{x\_2} = \text{x\_1} + 2$. Note that the latter representations is a more flexible one because it can also be read, for example, as $\text{x\_2} - \text{x\_1} = 2$.

The CSP representation of the program from Figure 2 is given as follows:

$$\begin{array}{rl}
\textbf{Variables:} & V = \left\{ \begin{array}{c} \{var\_1, x\_0, y\_0, \\ min\_1, min\_2, min\_3\} \end{array} \right\} \\
\textbf{Domains:} & D = \{D(x) = \mathbb{N} | x \in V\} \\
\textbf{Constraints:} & CO = \left\{ \begin{array}{c} var\_1 = (x\_0 < y\_0), \\ min\_1 = x\_0, \\ min\_2 = y\_0, \\ min\_3 = phi(min\_1, \\ min\_2, var\_1) \end{array} \right\}
\end{array}$$

When comparing the CSP representation of the program with its SSA form, which is depicted in Figure 4, we see that both representations are very close to each other. It is easy to prove that the SSA form of a program is equivalent to its CSP representation with respect to the behavior.

Theorem 3.3. *Given a program Π, its SSA representation Π', the corresponding CSP C_Π. The value assignments of the variables in Π', which are caused by executing Π' on an input I are a solution to the corresponding CSP C_Π and vice versa.*

From this theorem and the others we conclude that the transformation is behavior neutral and in this way the CSP representation captures the behavior of the program. As a consequence, the CSP representation can be directly used for debugging. As already mentioned there are circumstances under which the algorithm is more effectively and there are situations where the computation of solutions is hard. This holds now directly for debugging and we are interested in classifying programs regarding their debugging complexity. We define debugging complexity as a measure that corresponds to the complexity of computing a solution using CSP algorithms. In the following, we discuss structural properties of CSPs, which can be used for classification and which are based on the hyper-graph representation of programs.

In the hyper-graph representation of a CSP the variables of the CSP represent vertices, and the constraint scopes the hyper-edges. Thus hyper-edges connect possible more than one vertex. Hyper-graphs can be used to classify CSPs regarding their complexity of computing a solution. As already mentioned, solutions for CSPs with corresponding acyclic hyper-graphs can be computed in polynomial time (see (30)). Such hyper-graphs can be directly represented as hyper-trees. Unfortunately, not all CSPs are acyclic. But the good story is that cyclic CSPs can be converted into an equivalent CSP that is acyclic. What is needed is to join the right constraints. Joining constraints, however, is a drawback because it is time and space consuming. In the worst case all constraints have to be joined in order to finally receive the acyclic equivalent CSP representation, which is of course intractable.

As a consequence, one only gains computational advantages from the conversion of hyper-graphs into hyper-trees if the number of constraints to be joined is as minimal as possible. This number is referred to as hyper-tree width. More information regarding hyper-graphs and hyper-tree composition which allows to convert hyper-graphs into hyper-trees can be found in (14; 15). The hyper-graph and its corresponding hyper-tree for the CSP introduced above is depicted in Figure 6. The hyper-tree width for this example is 2.

Having a CSP representation of a program has the advantage of being able to use various algorithms for debugging purposes. However, the performance of debugging depends on the structure of the CSP. Hence, we are interested in the structural properties, i.e., the hyper-tree width, of the CSPs for various example programs. If the hyper-tree width for such examples is low, then computing diagnoses can be done effectively. In the next section we focus

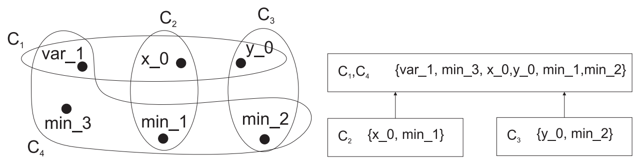


Figure 6: Hyper-graph (left) and corresponding hyper-tree (right) for the SSA form given in Fig. 4

therefore on the hyper-tree width of programs.

4 Experimental results on the hyper-tree width

As explained in the previous section the hyper-tree width has an impact on the complexity of debugging when using constraints a means for intermediate representation formalism. Gaining knowledge about the hyper-tree width of programs is therefore of importance. In this respect this section reports on the hyper-tree width of several programs. For this purpose, we implemented the conversion procedure in Java and used a constraint system that was implemented at out institute. For the hyper-tree decomposition we used an implementation provided by the TU Wien (see (16)). In particular, we made use of the Bucket Elimination algorithm based decomposition which is explained in (29). All experiments were carried out on a PC (Pentium 4, 3 GHz, 1 GB Ram).

The experiments are based on small programs that vary from 40 to 400 lines of code. The lines of code of the corresponding SSA forms and the size of constraint system in terms of number of constraints and variables are higher due to used while statement and their transformation to conditional statements. In particular, we wanted to give an empirical answer to the following research questions.

- Thorup (28) stated that there is limit of 6 for the hyper-tree width of structured programs. The theorem is based on using control dependence information and does not consider data dependences. Because the latter is of importance for debugging and our compilation respects those dependences, we wanted to know whether given limit still applies.
- The compilation of while statements and recursive procedure calls leads to an increase of statements and to a nested if-then-else structure. The question is how the nesting depth, i.e., the number of iterations for unrolling the while statements or recursive procedure calls, influences the hyper-tree width? Moreover, one might be interested whether there is a maximum bound on the hyper-tree with in such cases.

The environment for carrying out the empirical study is not the optimal for answering the above research question but the best one can expect today. The reason ist that the

ID	LOC	#W	#I	It	HW	T
1	70	1	6	3	5	1 s
1	70	1	6	50	5	364 s
2	110	0	11	-	5	1 s
3	70	4	5	3	5	11 s
3	70	4	5	20	6	2040 s
4	80	0	0	-	4	1 s
5	70	0	0	-	4	1 s
6	70	0	0	-	2	1 s
7	400	0	0	-	9	7 s
8	400	2	0	3	50	1000 s
8b	400	2	0	3	12	122 s
9	400	1	0	5	16	120 s
9	400	1	0	10	27	621 s
9	400	1	0	20	54	3959 s
9	400	1	0	35	51	16450 s
9	400	1	0	50	55	19245 s
10	400	1	0	10	20	80 s
10	400	1	0	20	23	274 s
10	400	1	0	50	25	2400 s
10	400	1	0	60	29	4120 s
10	400	1	0	70	25	4770 s
11	400	0	0	-	10	8 s
12	400	1	0	3	15	120 s
12	400	1	0	6	27	2580 s
12	400	1	0	10	43	3415 s
13	400	1	0	15	53	4010 s
14	60	0	0	-	2	1 s
15	50	1	4	3	6	1 s
15	50	1	4	10	6	5 s
15	50	1	4	100	6	456 s
16	40	7	0	1	2	1 s
16	40	7	0	10	3	600 s

Figure 7: Evolution of the hyper-tree width for different complexity programs

Bucket Elimination based decomposition algorithm is only an approximation algorithm and thus the solutions needs not to be minimal once. However, because of the size of the corresponding constraint systems other algorithms are hardly of use because they would take too much time and space.

The finally obtained results for the programs are depicted in Figure 7. There the programs are given a number (ID). The lines of codes (LOC) of the original program, the number of while statements (#W), the number of if-statements (#I), the number of iterations used to unroll the while-statements (I), the hyper-tree width (HW) obtained, and the time (T) required to compute the hyper-tree width are given.

In the case of programs 1 to 6 and 14 to 16, the hyper-tree width tends to be less influenced by the number of iteration of the while-structure. Moreover, for these programs the hyper-tree width reaches its maximal value after 2 to 3 iteration. This cannot be said for programs 7 to 13 were

```
x = 10;
y = 20;
while (x<100){
  x = x + y;
  y = y + 2;
```

Figure 8: Small example program `test`

the hyper-tree width ranges from 9 to 55. Even in the case where there is no unrolling of while statements (programs 7 and 11) the hyper-tree width ranges from 9 to 10. All of these programs represent digital circuits including some variants (like 8 and 8a) with a complex data flow.

Based on the obtained empirical results, we have to retract Thorup's theorem (28) because there are many programs that result in a constraint system of a larger hyper-tree width than 6. Note that it is not very likely to find a hyper-tree decomposition with a smaller width for the given programs. Moreover, the approximation algorithm seems to produce approximations, which are close to the optimum.

The second research question is harder to answer. In all experiments, we observed that after a certain number of iterations the hyper-tree width remains almost the same. For example take a look at program 9 and 10. In both cases it seems that the hyper-tree width reaches an upper bound. Of course because of the used approximation algorithm there is a variance in the obtained width. But it seems to be always around a certain value. More experiments have to be carried out in order to justify these findings.

For the purpose of motivating why the hyper-tree width stays constant after a certain number of considered iterations of the while statement, we use a small program `test`, which is given in Figure 8. For `test` we know that the maximum hyper-tree width is 3. This upper bound is reached after 2 times unrolling of the while statement, i.e., replacements of the while statement with nested if-statements.

In our example, we now compute the SSA form and the corresponding constraint systems for program `test` using 3 nested if-statements. The resulting SSA form and the constraint system are depicted in Figure 9 and Figure 10 respectively.

The hyper-graph corresponding to the constraint representation of `test` is given in Figure 11. Note that for the sake of clarity the graphical representation only comprises the constraints from the while structure in which the variable x is involved. From the hyper-graph it can be easily seen that the edges follow a certain pattern, which is repeated in every unrolling of the while statements. Hence, there is a possibility that the hyper-graph decomposition can be applied to these subparts of the hyper-graph separately and combined afterwards, which might lead to a constant hyper-tree width after a certain amount of unrolling steps.

In summary, we obtained the following results from the experimental study:

- The hyper-tree width of programs might become very large. Usually problems of hyper-tree width of 5 to 10 depending on the application domain are considered hard problems.
- In case of unrolling while-statements or recursive calls; it seems that the hyper-tree width reaches an upper bound. This would be an indication that debugging does not necessarily become more difficult, when the number of iterations increases. Note that the number of unrolling steps of while statements does depend on the considered test case, which is an independent criteria.
- The time for computing the hyper-tree width can be very large, which might be unacceptable in some circumstances. This can be for example the case when interactive debugging is a requirement. For automated debugging or off-line debugging the time for conversions is not a limiting factor. However, decreasing the overall analysis time is an open challenge.

What remains of interests is the question, why a certain program has a larger hyper-tree width than another similar program? Obviously the data and control dependences influence the overall hyper-tree width. But in case of similar programs the differences regarding the dependencies might not be large enough to justify high differences of the obtained hyper-tree width. This holds especially in case where a program comprises while statements. In the next section, we focus on this aspect and discuss one cause that leads to such an observation.

5 Discussion

In the performed experiments we observe that programs, which have a high hyper-tree width and where the number of iterations necessary to reach the maximum hyper-tree width is large, have less data dependences in the sub-block of the while statement. We illustrate these findings by means of two example programs `sum1` and `sum2`, which are depicted in Figure 12 and Figure 13 respectively. Both programs have about the same structure. In both programs a variable `i` is increased in every iteration until it reaches 100. Moreover, the hyper-tree width of both programs is about the same (2 and 1 for `sum1` and `sum2` respectively) when the number of considered unrolling steps is 1.

However, the situation changes, when considering more nested if-statements as a replacement for the while statement. For `sum1` the maximum hyper-tree width of 4 is reached after 3 iterations. For `sum2` the maximum hyper-tree width of 8 is reached after 12 unrolling steps. If considering the difference between the minimum and the maximum hyper-tree width, we obtain another different outcome. For `sum1` the difference is only 1, whereas for `sum2` the difference is 7.

```
x1_0 = 10;
y2_0 = 20;
cond_0=x1_0<100;
x1_1=x1_0+y2_0;
y2_1=y2_0+2;
cond_1=cond_0&&x1_1<100;
x1_2=x1_1+y2_1;
y2_2=y2_1+2;
cond_2=cond_1&&x1_2<100;
x1_3=x1_2+y2_2;
y2_3=y2_2+2;
x1_4=phi(x1_2,x1_3,cond_2);
y2_4=phi(y2_2,y2_3,cond_2);
x1_5=phi(x1_1,x1_4,cond_1);
y2_5=phi(y2_1,y2_4,cond_1);
x1_6=phi(x1_0,x1_5,cond_0);
y2_6=phi(y2_0,y2_5,cond_0);
```

Figure 9: The SSA form of program `test` (Fig. 8)

```
(x1_0 ),
(y2_0 ),
(X_cond_0 , x1_0 ),
(x1_1 , x1_0 , y2_0 ),
(y2_1 , y2_0 ),
(X_cond_1 , X_cond_0 , x1_1 ),
(x1_2 , x1_1 , y2_1 ),
(y2_2 , y2_1 ),
(X_cond_2 , X_cond_1 , x1_2 ),
(x1_3 , x1_2 , y2_2 ),
(y2_3 , y2_2 ),
(x1_4 , x1_2 , x1_3 , X_cond_2 ),
(y2_4 , y2_2 , y2_3 , X_cond_2 ),
(x1_5 , x1_1 , x1_4 , X_cond_1 ),
(y2_5 , y2_1 , y2_4 , X_cond_1 ),
(x1_6 , x1_0 , x1_5 , X_cond_0 ),
(y2_6 , y2_0 , y2_5 , X_cond_0 ).
```

Figure 10: The constraints for `test` (Fig. 8)

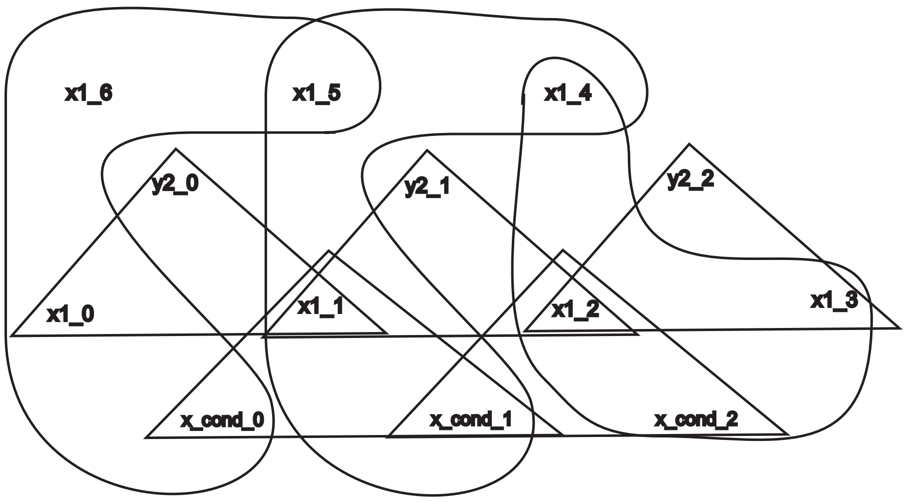


Figure 11: The hyper-graph of the constraint system from Fig. 10

```
a = f + i;
b = g + h;
while (i < 100) {
  x = x + a + b;
  i = x + i + 1;
}
```

Figure 12: Program `sum1`

```
a = f + i;
b = g + h;
while (i < 100) {
  x = a + b;
  i = i + 1;
  y = c + d;
}
```

Figure 13: Program `sum2`

By having a closer look at the programs we detect a major difference in their structure, which might explain this high difference in the hyper-tree width. For the variable `x` that is used in the block of program `sum1` a new value is computed in each iteration of the while statement. The outcome of variable `x` depends on the number of iterations and therefore on variable `i` (and the condition of the while statement). This is not the case for variable `x` and variable `y` in program `sum2`. Both variables are assigned the same value in each iteration step. For performance reason such assignment statements should be always placed outside a loop. In case of program `sum2` the values of variables in an iteration not necessarily depends on the previous iteration but on a variables that have been computed before calling the while statement. It seems that this difference makes the structure of the hyper-graph more complex, which leads finally to a higher hyper-tree width.

The given example shows that hyper-tree width may indicate an unwanted program structure. In program `sum2` either there are variables missing at the right side of statements 4 and 6, or both statements should be given outside the while statement because of performance reasons. Obviously, such problems can be detected using some rules like the following:

If a variable v is defined in the sub-block of a while statement, then there should be at least one not necessarily different statement in the same block that uses v.

Hyper-tree width offers another way for detecting such problematic cases occurring in programs.

6 Related research

Collavizza and Ruehner (8) discussed the conversion of programs into constraint systems and their use in software verification. Their work is very close to ours. The conversion steps are about the same and they also use the SSA form as an intermediate representation from which constraints can be obtained. However, there are some important differences. In their work the focus is on verification and not on debugging. They do not explain how to handle arrays and procedure calls as we did in the paper. Moreover, the structural analysis of programs using the concept of tree-decomposition methods and in particular hyper-tree decomposition is new. This holds also for the findings derived from the empirical analysis, which are of importance for automated debugging.

The authors of (31) proposed to diagnose errors in programs using constraint programming. Their approach requires that the programmer provides *contracts*, i.e., pre- and post-conditions, for every function. However, the authors do not investigate the complexity of solving the resulting problem and the scalability to larger programs. In particular, they do not consider structural decomposition or other methods which could make the approach feasible. Moreover, the practical applicability of their approach is also limited because it requires that contracts are specified for every function, which is very often not the case in real-world programs. Furthermore, their work does not properly handle recursive function calls.

Various authors, e.g., (13; 24; 20; 21), have described models to be used for fault localization using model-based diagnosis. Almost all of the work makes assumptions regarding the program's structure, uses abstractions which lead to the computation of too many diagnosis candidates, or does not handle all possible behaviors at once. The latter models, for example, consider one execution trace which prevent the diagnosis engine of remove some diagnosis candidates. In our representation we consider all possible behaviors up to a given boundary. This should lead to a reduction of the number of diagnosis candidates. In (19) Köb and Wotawa discussed the use of Hoare logic for model-based debugging which requires a Hoare logic calculus for computing diagnosis. Although, we share the same ideas on automated debugging, the described approach, which comprises the conversion to CSPs and their direct use, is new. From our point of view the described approach generalizes previous research directions.

Other work on debugging include (4; 5; 6). All of them are mainly based on program slicing (10). (6) integrates slicing and algorithmic program debugging (2) and (5) does the same for slicing and delta debugging (1). Critical slicing (4) which is an extension of dynamic slicing that avoids some of the pitfalls, can also be used in debugging. Because of the nature of slicing and the other techniques these approaches require more or less user interaction and cannot be used for really automated debugging. More information regarding other approaches to debugging and a good classification of debugging system is provided in (3).

7 Conclusion

In this paper we introduced a methodology for compiling programs into their equivalent CSP representation. The methodology comprises the conversion of while statement into their equivalent nested if-statement representation from which the SSA form is generated. From the SSA form itself we finally obtain the CSP representation. In the paper we argued that the CSP representation can be effectively used for debugging and allows for computing a complexity metrics for debugging, i.e., the hyper-tree width. This is due the fact that the complexity of debugging based on CSPs depends on the hyper-tree width. For the purpose of debugging we assume the existence of the source code and test cases, which reveal the fault. Another advantage of the CSP representation is that the available algorithms for constraint solving and in particular diagnosis can be directly used.

The presented empirical analysis showed that the hyper-tree width of programs varies a lot and can be more than 50. For the purpose of debugging this finding is not good, because usually constraint systems with a hyper-tree width of 5 to 10 are considered as complex. However, this fact shows that debugging itself is a complex task. From the empirical analysis we also obtain that previous work, which was mainly based on control dependences, on an upper bound of 6 for the hyper-tree width of programs cannot be justified in case of debugging in general. For a specific program there might be an upper bound even when compiling while statements in their equivalent nested if-statement representation.

Future research includes to solve the upper bound problem in the general case and to apply the debugging approach to smaller and medium size programs. The integration of the CSP representation and given program assertions like pre- and post-conditions is also of interest.

Acknowledgement

This work has been supported by the FIT-IT research project *Self Properties in Autonomous Systems project (SEPIAS)* which is funded by the Austrian Federal Ministry of Transport, Innovation and Technology and the FFG and by the Austrian Science Fund (FWF) within the project "Model-based Diagnosis and Reconfiguration in Mobile Autonomous Systems (MoDReMAS)" under grant P20199-N15. Moreover, we want to thank Nysret Musliu for providing us with an implementation of the hyper-graph decomposition.

References

[1] Andreas Zeller and Ralf Hildebrandt. Simplifying and isolating failure-inducing input. *IEEE Transactions on Software Engineering*, 28(2), feb 2002.

[2] Ehud Shapiro. Algorithmic Program Debugging. MIT Press, 1983.

[3] Nahid Shahmehri, Mariam Kamkar, and Peter Fritzson. Usability criteria for automated debugging systems. *J. Systems Software*, 31:55–70, 1995.

[4] Richard A. DeMillo, Hsin Pan, and Eugene H. Spafford. Critical slicing for software fault localization. In *International Symposium on Software Testing and Analysis (ISSTA)*, pages 121–134, 1996.

[5] Neelam Gupta, Haifeng He, Xiangyu Zhang, and Rajiv Gupta. Locating faulty code using failure-inducing chops. In *Automated Software Engineering (ASE)*, pages 263–272, November 2005.

[6] Mariam Kamkar. Application of program slicing in algorithmic debugging. *Information and Software Technology*, 40:637–645, 1998.

[7] Marc M. Brandis and H. Mössenböck. Single-pass generation of static assignment form for structured languages. *ACM TOPLAS*, 16(6):1684–1698, 1994.

[8] H. Collavizza and M. Rueher. Exploration of the capabilities of constraint programming for software verification. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 182–196, Vienna, Austria, 2006.

[9] Ron Cytron, Jeanne Ferrante, Barry K. Rosen, Mark N. Wegman, and F. Kenneth Zadeck. Efficiently computing static single assignment form and the control dependence graph. *ACM TOPLAS*, 13(4):451–490, 1991.

[10] Mark Weiser. Programmers use slices when debugging. *Communications of the ACM*, 25(7):446–452, July 1982.

[11] Rina Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.

[12] Yousri El Fattah and Rina Dechter. Diagnosing tree-decomposable circuits. In *Proceedings* 14^{th} *International Joint Conf. on Artificial Intelligence*, pages 1742 – 1748, 1995.

[13] Gerhard Friedrich, Markus Stumptner, and Franz Wotawa. Model-based diagnosis of hardware designs. *Artificial Intelligence*, 111(2):3–39, July 1999.

[14] Georg Gottlob, Nicola Leone, and Francesco Scarcello. On Tractable Queries and Constraints. In *Proc. 12th International Conference on Database and Expert Systems Applications DEXA 2001*, Florence, Italy, 1999.

[15] G. Gottlob, N. Leone, and F. Scarcello. A comparison of structural CSP decomposition methods. *Artificial Intelligence*, 124(2):243–282, 2000.

[16] http://www.dbai.tuwien.ac.at/proj/hypertree/ index.html

[17] A. Griesmayer, R. Bloem, and B. Cook. Repair of boolean programs with an application to C. In *Proc. 18th Conference on Computer Aided Verification (CAV'06)*, pages 358–371, Seattle, Washington, USA, August 2006.

[18] Daniel Jackson. *Software abstractions: logic, language, and analysis*. MIT Press, 2006.

[19] Daniel Köb and Franz Wotawa. Fundamentals of debugging using a resolution calculus. In Luciano Baresi and Reiko Heckel, editors, *Fundamental Approaches to Software Engineering (FASE'06)*, volume 3922 of *Lecture Notes in Computer Science*, pages 278–292, Vienna, Austria, March 2006. Springer.

[20] Cristinel Mateis, Markus Stumptner, and Franz Wotawa. Modeling Java Programs for Diagnosis. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, Berlin, Germany, August 2000.

[21] Wolfgang Mayer, Markus Stumptner, Dominik Wieland, and Franz Wotawa. Can ai help to improve debugging substantially? debugging experiences with value-based models. In *Proceedings of the European Conference on Artificial Intelligence*, pages 417–421, Lyon, France, 2002.

[22] Raymond Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, 1987.

[23] S. Staber, B. Jobstmann, and R. Bloem. Finding and fixing faults. In *Proc. 13th Conference on Correct Hardware Design and Verification Methods*, pages 35–49. Springer-Verlag, 2005. LNCS 3725.

[24] Markus Stumptner and Franz Wotawa. Debugging Functional Programs. In *Proceedings* 16^{th} *International Joint Conf. on Artificial Intelligence*, pages 1074–1079, Stockholm, Sweden, August 1999.

[25] Markus Stumptner and Franz Wotawa. Diagnosing tree-structured systems. *Artificial Intelligence*, 127(1):1–29, 2001.

[26] M. Stumptner and F. Wotawa. Coupling CSP decomposition methods and diagnosis algorithms for tree-structured systems. In *Proc. 18^{th} International Joint Conf. on Artificial Intelligence*, pages 388–393, Acapulco, Mexico, 2003.

[27] M.N. Wegman and F.K. Zadek. Constant propagation with conditional branches. *ACM Transactions on Programming Languages and Systems*, 13(2), April 1991.

[28] Mikkel Thorup.All Structured Programs have Small Tree-Width and Good Register Allocation,Information and Computation Journal, Volume 142,Number 2,Pages 159-181,1998.

[29] Artan Dermaku, Tobias Ganzow, Georg Gottlob, Ben McMahan, Nysret Musliu, Marko Samer. Heuristic Methods for hypertree Decompositions, DBAI-TR-2005-53, Technische Universität Wien, 2005.

[30] M. Yannakakis. Algorithms for acyclic database schemes. In C. Zaniolo and C. Delobel, editors, *Proceedings of the International Conference on Very Large Data Bases (VLDB-81)*, pages 82–94, Cannes, France, 1981.

[31] R. Ceballos and R. M. Gasca and C. Del Valle and D. Borrego Diagnosing Errors in DbC Programs Using Constraint Programming *Lecture Notes in Computer Science*, Vol. 4177, Pages 200-210, 2006.

Automatic Streaming Processing of XSLT Transformations Based on Tree Transducers

Jana Dvořáková
Department of Computer Science
Faculty of Mathematics, Physics and Informatics
Comenius University, Bratislava, Slovakia
E-mail: dvorakova@dcs.fmph.uniba.sk

Keywords: XML transformation, XSLT, streaming processing, tree transducer

Received: March 15, 2008

Streaming processing of XML transformations is practically needed especially when large XML documents or XML data streams are to be transformed. In this paper, the design of an automatic streaming processor for XSLT transformations is presented. Unlike other similar systems, our processor guarantees bounds on the resource usage for the processing of a particular type of transformation. This feature is achieved by employing tree transducers as the underlying formal base. The processor includes a set of streaming algorithms, each of them is associated with a tree transducer with specific resource usage (memory, number of passes), and thus captures different transformation subclass. The input XSLT stylesheet is analyzed in order to identify the transformation subclass to which it belongs. Then the lowest resource-consuming streaming algorithm capturing this subclass is applied.

Povzetek: Obravnavano je avtomatično pretakanje XSLT transformacij.

1 Introduction

XML (28) is a meta-language defined by W3 Consortium in order to store structured data. The initial W3C recommendation for XML was published in 1998 and since then XML has become a popular format. It is commonly used for data exchange among applications since it enables them to add semantics to data explicitly. Furthermore, XML is a suitable tool in every field where it is necessary to create document standards. XML usage is still growing fast and new technologies for processing XML documents are emerging.

Transformations of XML documents are needed in many situations. For instance, let us consider two applications exchanging data in XML format, each of them requiring different structure for the same content. Then a transformation must be performed while the data are being transferred between these applications. A typical XML transformation processor reads the whole input document into memory and then performs particular transformation steps according to the specification. References to any part of the input document are processed in a straightforward way by traversing the in-memory representation, and the extracted parts are combined to form the required output fragment. This approach is called tree-based processing of XML transformations. In early days of XML, this kind of processing was sufficient since the existing XML documents were small and stored in files. However, nowadays it is quite common to encounter extensive XML data (e.g., database exports) or XML data streams in practice. In both cases, the tree-based processing is not suitable - in the former case it is not acceptable or even possible to store the whole input document in the memory, while in the later one the XML data become available stepwise and need to be processed "on the fly".

In our case, the research on efficient processing of XML transformations was in part motivated by processing large XML data in the semantic repository Trisolda (9). The repository contains semantic annotation for various web resources. A standard format for specify such semantics is Resource Description Framework (RDF) which is an instance of XML. Since the amount of web resources annotated tends to grow very fast, the transformation of RDF data into other representations/views and vice versa cannot be performed in the tree-based manner. At the same time, it is not suitable to write the transformations by hand using a SAX parser. The transformations needed are not trivial, especially due to the complexity of RDF format, and along with adding new functions into Trisolda repository new transformations may become necessary. Therefore, a more flexible approach is employed and a processor is proposed such that the most efficient strategy for performing a given transformation is automatically chosen.

A natural alternative to the classical tree-based processing of XML transformations is the streaming (event-driven) processing. Here the input document is read sequentially, possibly in several passes; and the output document is generated sequentially in one pass. Only a part of the input document is available at a time, and thus advanced techniques must be used to process references to the input document and connect the extracted parts to the proper position within the output document.

Currently, the most frequently used XML transformation languages are XSLT (27) and XQuery (29), both general (Turing-complete) languages intended for tree-based processing. There is a great interest to identify XSLT and XQuery transformations which allow efficient streaming processing. When designing an XSLT/XQuery streaming processor, the key task is to find the way of handling the non-streaming constructs of the languages. The streaming algorithms for XSLT and XQuery transformations are however still under development and the complexity issues such as memory requirements and the number of passes needed for specific, clearly defined transformation classes have not yet been analyzed.

The main contribution of this work is the design of a system for automatic streaming processing of XSLT transformations yielding the following properties:

– The transformation classes captured are clearly characterized. Each such class contains transformations with common properties - it represents an XSLT subset obtained by restricting constructs used in the XSLT stylesheet.

– A streaming algorithm is designed for each transformation class. The main design goal is to minimize the upper bound of memory usage, i.e., to use optimal (or close to optimal) amount of memory. Such upper bound is explicitly stated for each algorithm.

These features are achieved by employing tree transducers as the underlying formal base. Specifically, the design of the processor is based on the formal framework for XML transformations introduced in (10). In this paper, the framework is simplified and customized in order to facilitate the implementation. It contains a general model – an abstract model of general, tree-based transformation languages, and a set of streaming models that differ in the kind of memory used and the number of passes over the input allowed. Each streaming model can simulate some restricted general model. The framework contains a simulation algorithm for each such pair *streaming model → restricted general model*. The framework is abstract, and thus can be used to develop automatic streaming processors for other general transformation languages as well (e.g., XQuery).

The implementation level of the framework for XSLT language includes the implementation of streaming models and two modules: (1) an analyzer that associates the input XSLT stylesheet with the lowest resource-consuming streaming model that is able to process it, and (2) the translator that automatically converts the XSLT stylesheet into the streaming model chosen according to the associated simulation algorithm. The processor based on the framework is easily extensible since new transducers and algorithms may be specified and implemented, as well as optimizable since the current algorithms may be replaced by the more efficient ones. Although there are some XML transformations such that their streaming processing is always high resource-consuming (e.g., complete reordering of element children), most of the practical transformations can be processed with reasonable bounds on the resource usage and thus, more effectively than when processed in the tree-based manner.

The rest of this paper is organized as follows: Section 2 contains description of both approaches to processing XML transformations and the complexity measures related to streaming processing. In Section 3, the customized formal framework for XML transformations is introduced and the underlying tree transducers are described. An example algorithm designed within the framework is presented in Section 4. In Section 5, the design of our automatic streaming processor for XSLT transformations is introduced. In Section 6, the relation to other work is discussed. Section 7 briefly introduces the implementation of the example streaming algorithm and Section 8 concludes with summary and comments on future work.

2 Complexity of streaming processing

In this section, the relevant complexity measures for the streaming algorithms for XML transformations are specified.

An XML document contains the following basic constructs: elements, element attributes, and text values. The document may be represented as a tree that is obtained by a natural one-to-one mapping between elements and internal nodes of the tree. The text values appear in the leaves of such tree. Reading a document in document order then exactly corresponds to the preorder traversal of the constructed tree.

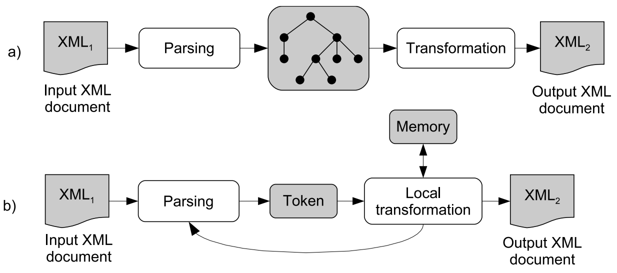


Figure 1: Two types of XML transformation processing: (**a**) tree-based processing, (**b**) streaming processing.

The tree-based processing of XML transformations (Fig. 1a) is flexible in the sense that the input document is stored in the memory as a tree and can be traversed in any direction. On the contrary, during the streaming processing (Fig. 1b) the elements of the input document become available stepwise in document order and similarly the output elements are generated in document order. The actual context is restricted to a single input node. Clearly, one-pass streaming processor without an additional memory is able to perform only simple transformations, such as renaming elements and attributes, changing attribute values, filtering. It must be extended to perform more complex restructuring. The common extensions are (1) allowing more passes over the input document, (2) adding an additional memory for storing temporary data. The extensions can be combined[1]. We obtain the corresponding complexity measures for streaming processing of XML transformations:

1. the number of the passes over the input tree,
2. the memory size.

It is reasonable to consider the complexity of the streaming processing in relation to the tree-based processing. As mentioned in Section 1, all XML transformations can be expressed in both XSLT and XQuery, and processed by their tree-based processors. Various transformation subclasses can be then characterized by putting restrictions on these general transformation languages, typically by excluding certain constructs.

When designing streaming algorithms, we have a choice regarding three settings – the type of the memory used (none, stack, buffers for storing XML fragments), and the values of the two complexity measures mentioned. Streaming algorithms with different settings may capture different transformation subclasses. Since the transformation subclasses are characterized as some subsets of the general transformation language considered, the key issue in the algorithms is to realize a streaming simulation of the non-streaming constructs included in the restricted language (see Fig. 2).

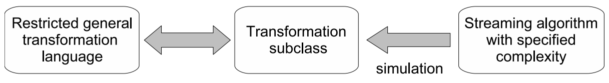

Restricted general transformation language
Transformation subclass
simulation
Streaming algorithm with specified complexity

Figure 2: The streaming simulation of subsets of a general transformation language.

We use tree transducers to design the streaming algorithms formally and to model transformation subclasses. They are included in the formal framework for streaming XML transformations that are described in the next section.

3 Formal framework

The framework is intended as a formal base for automatic streaming processors of the general transformation languages. It does not cover all XML transformations. In order to keep the models employed simple and comprehensible, it is restricted to model primarily the transformations that capture the relevant problems of streaming processing. In Section 5, a way how some of the restrictions on the transformation set can be overcome in the implementation is described.

The framework consists of the following formal models:

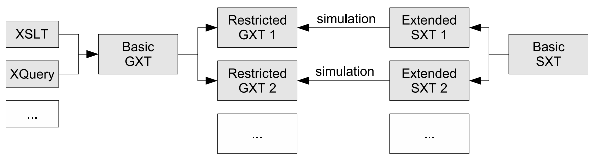


Figure 3: A schema of the formal framework.

1. a basic general model for tree-based processing of XML transformations and its restrictions,
2. a basic streaming model for streaming processing of XML transformations and its extensions.

The design of both models results from an analysis of various tree transformation models, XML transformation models as well as real-world XML transformation languages and systems. They are based on tree transducers, models for tree transformations (25) originated in the formal language theory. We introduce two novel models – a *general XML transducer (GXT)* used as the general model, and a *streaming XML transducer (SXT)* used as the streaming model. They are defined in common terms in order to facilitate development of the simulation algorithms.

The overall schema of the framework is shown in Fig. 3. The basic SXT represents a simple one-pass streaming model without an additional memory. Following the ideas from Section 2, it can be extended by memory for storing temporary data and by allowing more passes over the input document. The basic GXT represents the most powerful general model. As already mentioned, it does not capture all XML transformations, but only a subset significant in the case of streaming processing.

For each extended SXT, the transformation subclass captured is identified by imposing various restrictions on the basic GXT. The inclusion is proved by providing an algorithm for simulating this restricted GXT by the given extended SXT.

3.1 Notions and Notations

XML Document Abstraction. In what follows, element attributes and data values are not considered[2]. Let Σ be an alphabet of element names. The set of *XML trees* over Σ is denoted by $\mathcal{T}_\Sigma$, the empty XML tree is denoted by ε. An *indexed XML tree* may in addition have some leaves labeled by symbols from a given set X. A set of XML trees over Σ indexed by X is denoted by $\mathcal{T}_\Sigma(X)$. In the *rightmost indexed XML tree*, the element of the indexing set occurs only as the rightmost leaf. The set of rightmost indexed XML trees is denoted by $\mathcal{T}_\Sigma(X)_r$.

A particular XML tree $t \in \mathcal{T}_\Sigma(X)$ is uniquely specified as a triple (V_t, E_t, λ_t) where V_t is a set of nodes, $E_t \subseteq V_t \times V_t$ is a set of edges, and $\lambda_t : V_t \to \Sigma \cup X$ is a labeling function.

[1]More passes over the input tree are not possible for XML data streams that must be processed "on the fly".

[2]We refer the reader to (10) for the definition of the extended framework including both element attributes and data values.

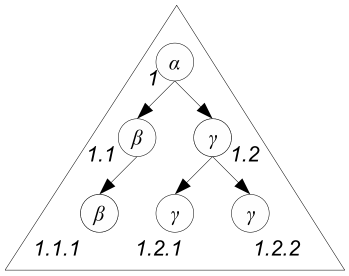


Figure 4: An example of the XML tree.

Example 3.1. An XML tree $t = (V_t, E_t, \lambda_t)$ over the alphabet $\Sigma = \{\alpha, \beta, \gamma\}$ and the empty indexing set $X = \emptyset$ is shown in Fig. 4. The nodes of t are uniquely identified by dynamic level numbering. The sets V_t, E_t and the labeling function λ_t are defined as follows:

$$\begin{array}{lllll} V_t & = & \{1, 1.1, 1.2, 1.1.1, 1.2.1, 1.2.2\}, & & \\ \lambda_t(1) & = \alpha, & \lambda_t(1.1.1) & = & \beta, \\ \lambda_t(1.1) & = \beta, & \lambda_t(1.2.1) & = & \gamma, \\ \lambda_t(1.2) & = \gamma, & \lambda_t(1.2.2) & = & \gamma. \end{array}$$

Selecting Expressions. Simple *selecting expressions*, derived from XPath expressions (26), are used to locate the nodes within the XML tree. The selecting expression is a *path* consisting of a sequence of *steps*. It can be either absolute (starting with /), or relative. The *step* consists of two components – an axis specifier $axis$ and a predicate $pred$. They are specified as outlined below. Comparing to the XPath language, the set of expressions is restricted and the syntax of some constructs is simplified – the meaning is explained in parentheses. The semantics of the selecting expressions follows the semantics of the equivalent XPath expressions.

$step$: $axis\,[pred]$
$axis$: $\times$ (self),
$\downarrow$ (child), $\downarrow*$ (descendant),
$\uparrow$ (parent), $\uparrow*$ (ancestor),
$\leftarrow$ (left sibling), $\overset{*}{\leftarrow}$ (preceding),
$\rightarrow$ (right sibling), $\overset{*}{\rightarrow}$ (following)

$pred$:	$*$	(select all elements)
	$name$	(select the elements named $name$)
	i	(select the element on i-th position within siblings)
	$step$	(select the elements having context specified by $step$)

The names of the elements are taken from an alphabet Σ. The set of selecting expressions over Σ is denoted by $\mathcal{S}_\Sigma$. The evaluation of a selecting expression in the context of some XML tree t and one of its nodes $u \in V_t$ returns the same set of nodes of t as the evaluation of the corresponding XPath expression. Note that the context set contains a single node only.

3.2 XML Transducers

General XML Transducer (Fig. 5a)**.** The input heads of GXT traverse the input tree in any direction and the output is generated from the root to the leaves. At the beginning of a transformation, the transducer has only one input head, which aims at the root of the input tree, and one output head, which aims at the root position of the empty output tree. During a single transformation step, the whole input tree is available as a context. One or more new computation branches can be spawned and the corresponding input control is moved to the input nodes specified by selecting expressions. At the same time, the output heads may generate a new part of the output.

Formally, the GXT is a 5-tuple $T = (Q, \Sigma, \Delta, q_0, R)$, where

– Q is a finite set of states,

– Σ is an input alphabet,

– Δ is an output alphabet,

– $q_0 \in Q$ is an initial state, and

– R is a set of rules of the form

$$Q \times \Sigma \rightarrow \mathcal{T}_\Delta(Q \times \mathcal{S}_\Sigma)\ .$$

For each $q \in Q$ and $\sigma \in \Sigma$, there is exactly one rhs such that $(q, \sigma) \rightarrow rhs \in Q$.

The right-hand side of a rule contains an XML tree over the output alphabet indexed by *rule calls* – pairs of the form (q, exp), where q is a state and exp is a selecting expression that returns a sequence of input nodes to be processed recursively. A simple example of a GXT transformation follows.

Example 3.2. Let $T = (Q, \Sigma, \Sigma, q_0, R)$ be a GXT where $Q = \{q_0\}$, $\Sigma = \{\alpha, \beta, \gamma\}$. and R consists of the rules

$$(q_0, \alpha) \rightarrow \varepsilon\ , \tag{3.1}$$
$$(q_0, \beta) \rightarrow \alpha((q_0, \downarrow[*]))\ , \tag{3.2}$$
$$(q_0, \gamma) \rightarrow \gamma((q_0, \downarrow[2]), (q_0, \downarrow[1]))\ . \tag{3.3}$$

The transducer processes the input trees over alphabet Σ. The subtrees at nodes named α are completely removed (rule 3.1), the nodes named β are renamed and get a new name α (rule 3.2), and at last, when encountering a node named γ, the first two children are processed in reversed order (rule 3.3).

The GXT is inspired mainly by the tree-walking tree transducer (TWR) (4) and data tree transducer (DTT) (23). It works on unranked trees, but does not handle data values. Similarly to TWR, the computation is high-level and based on rule calls. However, the XPath language is used for pattern matching on the paths of the input tree as it is in DTT. This choice is natural since XPath is used in the common general transformation languages (XSLT, XQuery). The GXT is tree-walking, i.e., the input tree is traversed in any direction. It allows more computation branches, but it is still sequential model in the sense that during a transformation step only a single rule call is processed.

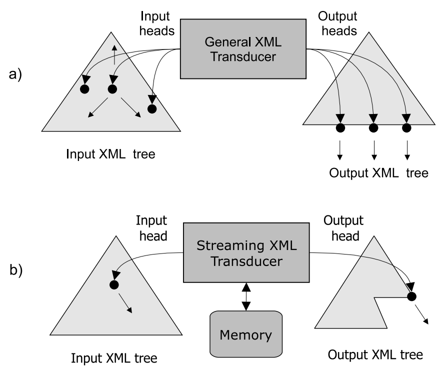


Figure 5: The processing model of the transducers: (**a**) the GXT; (**b**) the SXT.

Streaming XML Transducer (Fig. 5b). The SXT has a single input head that traverses the input tree in preorder, and a single output head that generates the output tree in preorder. Each node is visited twice during a single pass – once when moving top–down, and once when moving bottom–up. Thus, two types of SXT states are recognized (1) the states indicating the first visit of nodes and (2) the states indicating the second visit of nodes. During a single transformation step, the input head either moves one step in preorder or stays at the current position. At the same time, an output action is performed, depending on the type of rule applied. When applying a *generating rule*, a new part of the output is connected to the current position of the output head, and then the output head moves to the position under the rightmost leaf of the new part. When applying a *closing rule*, no output is generated, only the output head is moved one step upwards in preorder within the output tree.

Formally, the streaming XML transducer (SXT) is a 5-tuple $T = (Q, \Sigma, \Delta, q_0, R)$, where

– $Q = Q_1 \cup Q_2, Q_1 \cap Q_2 = \emptyset$ is a finite set of states,

– Σ, Δ are the same as in the case of GXT,

– $q_0 \in Q_1$ is the initial state, and

– $R = R_g \cup R_c, R_g \cap R_c = \emptyset$ is a finite set of rules of the form:

$$\begin{aligned} R_g : Q \times \Sigma \times Pos &\rightarrow \mathcal{T}_\Delta(Q \times \mathcal{S}_\Sigma)_r \\ R_c : Q \times \Sigma \times Pos &\rightarrow Q \times \mathcal{S}_\Sigma \end{aligned}$$

where $Pos = \{leaf, no\text{-}leaf\} \times \{last,\ no\text{-}last\}$[3]. For each $q \in Q$ and $\sigma \in \Sigma$ there is at most one rhs such that for each $pos \in Pos$ there is a rule $(q, \sigma, pos) \rightarrow rhs \in R$[4]. Furthermore, for each $(q, \sigma, pos) \rightarrow rhs \in R$, $rec(rhs) = (q', exp)$[5], one of the following preorder conditions holds:

1. *moving downwards*: $q \in Q_1$, and
– $pos[1] = no\text{-}leaf$, $q' \in Q_1$, $exp = \downarrow[1]$, or
– $pos[1] = leaf$, $q' \in Q_2$, $exp = \times[*]$,

2. *moving upwards*: $q \in Q_2$, and
– $pos[2] = no\text{-}last$, $q' \in Q_1$, $exp = \rightarrow[1]$, or
– $pos[2] = last$, $q' \in Q_2$, $exp = \uparrow[*]$,

3. *no input move*: q, q' are of the same kind, $exp = \times$.

The left-hand side of a rule consists of a state, an element name and a node position. The position is used to determine the preorder move within the input tree and it consists of two predicates – the first one indicating a leaf node, and the second one indicating a last node among the siblings. The right-hand side is an XML tree rightmost indexed by a rule call.

4 An example algorithm

In this section, a particular streaming simulation designed within our framework is presented. In particular, a top-down GXT is simulated by an SXT extended with stack of the size proportional to the height of the input tree.

The stack-based simulation is efficient - in order to evaluate simple top-down selecting expressions in the branches of the input XML tree the memory size proportional to the length of the branches, which equals height of the input tree, is needed. However, it is shown that a restriction to stack is sufficient. First, the models considered are described formally.

Restricted GXT. The restricted GXT, called the top-down GXT (TGXT) differs from GXT in the rule definition - R is a set of rules of the form

$$Q \times \Sigma \rightarrow \mathcal{T}_\Delta(Q \times top\text{-}\mathcal{S}_\Sigma)$$

where $top\text{-}\mathcal{S}_\Sigma$ is a set of simple top-down selecting expressions. It is a subset of selecting expressions such that only top-down axis ($child$ and $descendant$) and name predicates ($[name]$) are allowed. The simulated TGXT must in addition satisfy two input-dependent conditions:

1. The TGXT is *order-preserving* if and only if, for each of its rules, the input nodes returned by the selecting expressions in the rhs are in preorder for arbitrary input tree t and $u \in V_t$.

2. The TGXT is *branch-disjoint* if and only if, for each of its rules, the input nodes returned by the selecting expressions in the rhs are disjoint for arbitrary input tree t and $u \in V_t$.

[3]If $pos \in Pos$ is a node position, its first component is referred by $pos[1]$ and to its second component is referred by $pos[2]$.

[4]This condition is necessary to keep the model deterministic.

[5]If rhs is a particular right-hand side, its rule call is referred by $rec(rhs)$.

Intuitively, if any of the conditions is not satisfied, it may happen that a part of the input tree disproportional to the height of the input tree must be stored in the memory and thus the stack-based simulation is not applicable.

Extended SXT. The extended SXT, called the stack-based SXT (SSXT) is a 7-tuple $T = (Q, \Sigma, \Delta, \Gamma, q_0, z_0, R)$ where

– Q, Σ, Δ, q_0 are the same as in the case of SXT,

– Γ is a finite set of stack symbols,

– $z_0 \in \Gamma$ is the initial stack symbol, and

– R is a finite set of rules of the form:

$$\begin{aligned} R_g : Q \times \Sigma \times Pos \times \Gamma &\rightarrow \\ &\mathcal{T}_\Delta(Q \times \mathcal{S}_\Sigma \times \Gamma^*)_r \\ R_c : Q \times \Sigma \times Pos \times \Gamma &\rightarrow \quad Q \times \mathcal{S}_\Sigma \times \Gamma^* \end{aligned}$$

The lhs now contains, in addition, the current top stack symbol, and the rhs contains a sequence of stack symbols to be put on the top of the stack. All other symbols have the same meaning as in the SXT.

4.1 Construction of Simulating SSXT

The formal proposition follows. It says that, for each order-preserving and branch-disjoint TGXT, it is possible to construct an SSXT inducing equivalent translations.

Proposition 4.1. *Let $T = (Q, \Sigma, \Delta, q_0, R)$ be an order-preserving and branch-disjoint TGXT. Then an SSXT T' exists such that, for each $t_{in} \in \mathcal{T}_\Sigma$ and $t_{out} \in \mathcal{T}_\Delta$, if T translates t_{in} to t_{out} then T' translates t_{in} to t_{out}.*

The simulation proceeds in cycles. During a cycle, a single transformation step of T is simulated, called the *current transformation step*. Such simulation consists of several transformation steps of T'. A cycle is driven by the *cycle configuration* that consists of three items:

1. *current context node* - the current input node of T during the current transformation step,

2. *current rule* - the rule of T applied during the current transformation step,

3. *matched rule call* - a rule call of the current rule.

During the whole simulation, the matched rule call represents the leftmost[6] rule call, for which a match has been already found.

At the beginning of the simulation, the current context node is the root node of the input tree, the current rule is the rule of T of the form $(q_0, \sigma) \rightarrow rhs$ where q_0 is the initial state of T and σ is the name of the root of the input tree. The matched rule call is the left sentinel rule call which is a virtual rule call positioned to the left from all other rule calls. This special rule call is used to initialize a new cycle. In case no error is encountered, a cycle includes two phases - an *evaluation phase* and a *generation phase*.

[6]The positions of rule calls are always considered with respect to the preorder of the rhs of the rule.

Phase Alternation. During the evaluation phase the input head of T' traverses the subtree at the current context node in preorder, and at the same time it evaluates all selecting expressions in the rule calls of the current rule. The evaluation is accomplished in a standard way by means of finite automata[7]. Three cases are distinguished depending on the result of the evaluation phase:

1. *A matching node is found for exactly one rule call, and this rule call (newly-matched rule call) is either positioned to the right of the matched rule call or it equals the matched rule call.*

This type of cycle is called an *entering cycle* since it takes place when the input head of T' is moving downwards, and a new rule call of the current rule is matched and "recursively" processed. The generation phase follows: The output head generates a specific part of the output fragment of the current rule. The part is a set of nodes that appear between the matched rule call and the newly-matched rule call. After the generation, the current cycle configuration is stored in the stack. The matched node becomes the new current context node. The rule of the form $(q', \sigma') \rightarrow rhs$ where q' is the state in the newly-matched rule call and σ' is the name of the matched node becomes the new current rule, and the left sentinel rule call becomes the new current rule call. A new cycle starts driven by the new cycle configuration.

2. *A matching node is found for two or more rule calls, or a matching node is found for a rule call that is positioned to the left of the matched rule call.*

This situation occurs in case T is non-order-preserving and an error is reported.

3. *No matching node is found and the whole subtree at the current context node has been traversed.*

This type of cycle is called a *returning cycle* since it takes place when the input head of T' is moving upwards, the processing of some rule call is finished, and the control moves back to the processing of the rule containing this rule call. The current rule is denoted by r. The generation phase follows directly: The last part of the output fragment of r is generated. The top stack configuration becomes the new cycle configuration, and the new cycle starts.

5 Design of XSLT streaming processor

An automatic streaming processor for XSLT transformations is described which is based on the framework intro-

[7]This method was, for example, presented in the Y-Filter algorithm (5; 8).

duced. The models within the framework are abstract, and thus the framework provides means to develop efficient streaming algorithms for XML transformation subclasses at abstract level, and to adapt them to an arbitrary general transformation language. First, the general issues regarding the framework implementation are described, and then an adaptation for the XSLT transformation language is discussed in more detail.

5.1 Framework Restrictions

As mentioned in the previous section, the formal framework is restricted in several ways. Some of the restrictions can be easily overcome in the implementation, while others require more complex handling.

1. *Restrictions on the XML document.* Attributes and data values are associated with elements. They can be easily added to the implementation – if such construct needs to be processed, it is accessed using the same path like the parent element. On the other hand, if the construct needs to be generated in the output, the action is performed together with the generation of the parent element.

2. *Restrictions on the selecting expressions.* The simple selecting expressions used capture the typical problems that arise during the streaming location of the nodes in XML document (context references in predicates, backward axis). Other constructs must be handled separately – however, the techniques used for constructs included in our restricted set may be often exploited. Moreover, there has been already carried on a research on the streaming processing of large subsets of XPath language (see Section 6 for overview).

3. *Restrictions on the general transformation language.* A part of the restrictions in GXT results from the restrictions on selecting expressions, and others are caused by excluding certain general transformation constructs, such as loops, variables, functions. However, the GXT models transformations that reorder the nodes within an XML tree with respect to the document order, which is probably the most important issue in streaming processing of XML transformations if the specific issues concerning selecting expression evaluation are not considered.

5.2 Adaptation for XSLT

Let us now describe the design of the prototype XSLT streaming processor. The GXT represents an abstract model for general transformation languages. Since our intention is to adapt the framework for the XSLT language, it does not need to be implemented directly. Instead, we are looking for a correspondence between restricted GXTs and XSLT subsets. The GXT models the XSLT transformations driven by the structure of the input document. Thus, each XSLT stylesheet consisting of a list of simple templates activated by structure and mode can be directly converted to GXT. The matching element of such simple template is referenced by a name only and the body of the template may contain several output elements (possibly nested) and calls for applying another templates. The template is called by a selecting expression and a mode.

Specifically, an XSLT stylesheet xsl convertible to GXT consists of (1) an initializing template and (2) several rule templates. The *initializing template* sets the current mode to the initial state of the GXT.

```
<xsl:template match="/">
   <xsl:apply-templates
         select="child::*" mode="q0"/>
</xsl:template>
```

Each *rule template* can be directly translated to a single rule of GXT. It is of the following form.

```
<xsl:template match="name" mode="q">
    ...  template body ...
</xsl:template>
```

The resulting GXT rule r is of the form $(q, name) \to rhs$ Thus, the left-hand side consists of the element name in the `match` attribute and the state in the `mode` attribute. The rhs is created by translation of the template body as described below.

The template body contains a sequence of (possibly nested) output elements and `apply-templates` constructs. An output element named $name$ is specified directly as a pair of tags (alternatively, the `element` construct might be used):

```
<name>
    ...element content ...
</name>
```

The `apply-templates` construct has a `select` attribute that contains a selecting expression, and a `mode` attribute that represents a state of the resulting GXT.

```
<xsl:apply-templates
         select="selexp" mode="q'"/>
```

Each `apply-templates` construct can be translated to a single rule call. For the case above, a rule call of the form $(q', selexp)$ is obtained.

The rhs of the rule r is created from the template body so that each output element corresponds to a single node of rhs and each `apply-templates` construct corresponds to a single rule call of rhs. The structure of rhs is determined by nesting of the output elements and `apply-templates` constructs in the template body. The resulting GXT is of the form $T = (Q, \Sigma, \Delta, q_0, R)$ where

– Q contains the modes appearing in xsl,

– R contains the rules created by translation from particular rule templates as described above. Moreover, R contains rule of the form

$$(q, \sigma) \to (q, \texttt{child}[*])$$

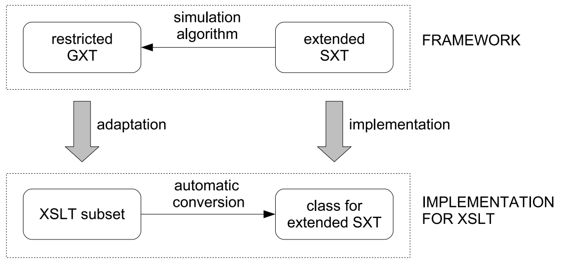


Figure 6: An implementation of the framework for XSLT language.

for each mode q and name $\sigma \in \Sigma$ such that xsl does not contain a template matching σ in the mode q. Such additional rules correspond to the XSLT implicit built-in rule templates[8].

In a similar way, XSLT subsets corresponding to restricted GXTs can be identified. According to the principle of the formal framework, a restricted GXT (GXT_r) can be simulated by some extended SXT (SXT_e) such that the simulation algorithm is known. Then XSLT stylesheets from the XSLT subset associated with GXT_r can be converted to SXT_e using the simulation algorithm. The conversion can be performed automatically since the simulation algorithm exactly determines how to convert constructs of the given XSLT subset into the rules of SXT_e. The resulting SXT_e is constructed explicitly as an object and its method $transform()$ performs streaming processing of the transformation specified by the stylesheet. The relation between the formal framework and the implementation for XSLT is shown in Fig. 6.

5.3 Modules of Streaming Processor

To sum up, the streaming processor works in three steps (see also Fig. 7):

1. *Analysis.* The analyzer examines the constructs in the input XSLT stylesheet (both XPath constructs and XSLT constructs themselves). It checks whether there is specified an XSLT subset that allows all the constructs encountered. If there are more such subsets, the smallest one is chosen.

2. *Translation.* The translator creates an object for the extended SXT associated with the XSLT subset chosen. The creation is automatic, following the simulation algorithm provided for the XSLT subset.

3. *Processing.* The method $transform()$ of the new SXT object is run on the input XML document. The streaming transformation performed is equivalent to the one specified by the input XSLT stylesheet.

[8]The built-in XSLT rules actually ensure that the resulting GXT is complete. Note that it is also deterministic since xsl cannot contain two templates matching the same name in the same mode by definition of the valid XSLT stylesheet.

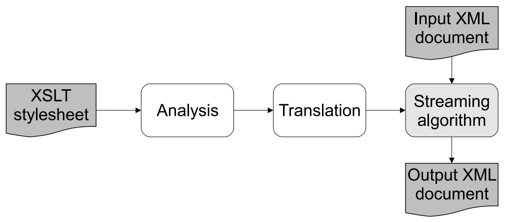


Figure 7: Modules os the automatic streaming processor.

6 Related work

Most of the earlier work was devoted to analyzing the streaming processing of the querying language XPath (1; 2; 6; 7; 14; 16; 22; 24). Recently, several streaming processors for the transformation languages XQuery and XSLT have appeared.

XML Streaming Machine (XSM) (19) processes a subset of XQuery on XML streams without attributes and recursive structures. It is based on a model called XML streaming transducer. The processor have been tested on XML documents of various sizes against a simple query. Using XSM the processing time grows linearly with the document size, while in the case of standard XQuery processors the time grows superlinearly. However, more complex queries have not been tested.

BEA/XQRL (12) is a streaming processor that implements full XQuery. The processor was compared with Xalan-J XSLT processor on the set of 25 transformations and another test was carried on XMark Benchmarks. BEA processor was fast on small input documents, however, the processing of large documents was slower since the optimizations specially designed for XML streams are limited in this engine.

FluXQuery (17) is a streaming XQuery processor based on a new internal query language *FluX* which extends XQuery with constructs for streaming processing. XQuery query is converted into FluX and the memory size is optimized by examining the query as well as the input DTD. FluXQuery supports a subset of XQuery. The engine was benchmarked against XQuery processors Galax and AnonX on selected queries of the XMark benchmark. The results show that FluXQuery consumes less memory and runtime.

SPM (Streaming Processing Model) (15) is a simple one-pass streaming XSLT processor without an additional memory. Authors present a procedure that tries to converts a given XSLT stylesheet into SPM. However, no algorithm for testing the streamability of XSLT is introduced, and thus the class of XSLT transformations captured by SPM is not clearly characterized.

The effectiveness of the processors mentioned was examined only through empirical tests. The test results show that streaming processors tend indeed to be less time and space consuming than tree-based processors. However, since no formal characterizations of the transformation class captured were given, the results hold only for a few (typically one or two) XML transformations chosen for experiments.

In other approaches (3; 13; 21), a new specification language is developed which supports streaming processing, and the streaming processor for this new language is designed. In all cases the connection to the commonly used transformation languages is not clearly stated and the computational complexity of the streaming processing is not addressed.

7 Implementation

The formal framework introduced has been implemented on .Net platform. The pilot implementation includes the stack-based algorithm described in Section 4. The evaluation of the algorithm implementation shows that it is highly efficient in practice - it requires memory proportional to the depth of the input XML document. Since this depth is generally not depending on the document size and common XML documents are relatively shallow (99% of XML documents have fewer than 8 levels whereas the average depth is 4 according to (20)), the memory requirements for most of the XML documents are constant, independent to the document size. On the contrary, standard XSLT processors are tree-based and thus require memory proportional to the document size. We refer the reader to (11) for a more detailed description of the stack-based algorithm implementation and evaluation.

8 Conclusion

A design of an automatic streaming processor for XSLT transformations have been presented. Comparing to other similar processors, the contribution of our approach is that the resource usage for streaming processing of particular types of XSLT transformations is known. Our processor includes several streaming algorithms, and it automatically chooses the most efficient one for a given XSLT stylesheet. The process of choice has a solid formal base – a framework consisting of tree transducers that serve as models both for the streaming algorithms and for the transformation types.

In the future work, we plan to include algorithms for the local and non-order-preserving transformations to obtain a processor for a a large subset of practically needed XML transformations. We intend to demonstrate the usage of such processor by integration into the Trisolda semantic repository and carry out performance tests and comparison to other implementations subsequently.

Acknowledgement

This work was supported in part by the grant VEGA 1/3106/06.

References

[1] Z. Bar-Yossef, M. Fontoura, and V. Josifovski. On the Memory Requirements of XPath Evaluation over XML Streams. In *PODS '04: Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 177–188, New York, NY, USA, 2004. ACM.

[2] C. Barton, P. Charles, D. Goyal, M. Raghavchari, M. Fontoura, and V. Josifovski. An Algorithm for Streaming XPath Processing With Forward and Backward Axis. In *Proceedings of ICDE 2003*, 2003.

[3] O. Becker. Transforming XML on the Fly. In *Proceedings of XML Europe 2003*, 2003.

[4] G. J. Bex, S. Maneth, and F. Neven. A Formal Model for an Expressive Fragment of XSLT. *Inf. Syst.*, 27(1):21–39, 2002.

[5] N. Bruno, L. Gravano, N. Koudas, and D. Srivastava. XML & Data Streams. In *Stream Data Management*, pages 59–82. Springer Verlag, 2005.

[6] F. Bry, F. Coskun, S. Durmaz, T. Furche, D. Olteanu, and M. Spannagel. The XML Stream Query Processor SPEX. In *Proceedings of ICDE 2005*, pages 1120–1121, 2005.

[7] Y. Chen, S. B. Davidson, and Y. Zheng. An Efficient XPath Query Processor for XML Streams. In *Proceedings of ICDE 2006*, pages 79–79, 2006.

[8] Y. Diao, M. Altinel, M. J. Franklin, H. Zhang, and P. Fischer. Path Sharing and Predicate Evaluation for High-performance XML Filtering. *ACM Trans. Database Syst.*, 28(4):467–516, 2003.

[9] J. Dokulil, J. Tykal, J. Yaghob, and F. Zavoral. Semantic Web Repository And Interfaces. In *International Conference on Advances in Semantic Processing SEMAPRO 2007*. IEEE Computer Society, 2007.

[10] J. Dvořáková and B. Rovan. A Transducer-Based Framework for Streaming XML Transformations. In *Proceedings of SOFSEM (2) 2007*, pages 50–60, 2007.

[11] J. Dvořáková and F. Zavoral. An Implementation Framework for Efficient XSLT Processing. In *Proceedings of IDC 2008, Studies in Computational Intelligence*, Springer-Verlag, 2008.

[12] D. Florescu, C. Hillery, D. Kossmann, P. Lucas, F. Riccardi, T. Westmann, M. J. Carey, A. Sundararajan, and G. Agrawal. The BEA/XQRL Streaming XQuery Processor. In *Proceedings of VLDB 2003*, pages 997–1008, 2003.

[13] A. Frisch and K. Nakano. Streaming XML Transformations Using Term Rewriting. In *Proceedings of PLAN-X 2007*, 2007.

[14] P. Genevès and K. Rose. Compiling XPath for Streaming Access Policy. In *Proceedings of ACM DOCENG 2005*, pages 52–54, 2005.

[15] Z. Guo, M. Li, X. Wang, and A. Zhou. Scalable XSLT Evaluation. In *Advanced Web Technologies and Applications, LNCS 3007/2004*. Springer Berlin / Heidelberg, 2004.

[16] K. Jittrawong and R. K. Wong. Optimizing XPath Queries on Streaming XML Data. In *ADC '07: Proceedings of the eighteenth conference on Australasian database*, pages 73–82, Darlinghurst, Australia, Australia, 2007. Australian Computer Society, Inc.

[17] C. Koch, S. Scherzinger, N. Schweikardt, and B. Stegmaier. FluXQuery: An Optimizing XQuery Processor for Streaming XML Data. In *VLDB'2004: Proceedings of the Thirtieth International Conference on Very Large Databases*, pages 1309–1312, 2004.

[18] X. Li and G. Agrawal. Efficient Evaluation of XQuery over Streaming Data. In *VLDB '05: Proceedings of the 31st international conference on Very large data bases*, pages 265–276. VLDB Endowment, 2005.

[19] B. Ludäscher, P. Mukhopadhyay, and Y. Papakonstantinou. A Transducer-Based XML Query Processor. In *Proceedings of VLDB 2002*, pages 227–238, 2002.

[20] I. Mlýnková, K. Toman and J. Pokorný. Statistical Analysis of Real XML Data Collections. In *COMAD'06: Proc. of the 13th Int. Conf. on Management of Data*, pages 20–31, 2006.

[21] K. Nakano. An Implementation Scheme for XML Transformation Languages through Derivation of Stream Processors. In *Proceedings of the Second ASIAN Symposium on Programming Languages and Systems (APLAS'04)*, 2004.

[22] D. Oltenau, H. Meuss, T. Furche, and F. Bry. XPath: Looking Forward. In *Proceedings of XMLDM Workshop*, pages 109–127, 2002.

[23] T. Pankowski. Transformation of XML Data Using an Unranked Tree Transducer. In *EC-Web*, pages 259–269, 2003.

[24] F. Peng and S. S. Chawathe. XPath Queries on Streaming Data. In *Proceedings of ACM SIGMOD 2003*, 2003.

[25] J. Thatcher. Tree automata: An informal survey. In A. V. Aho, editor, *Currents in the Theory of Computing*, chapter 4, pages 143–172. Prentice-Hall, 1973.

[26] W3C. *XML Path Language (XPath), version 1.0, W3C Recommendation*, 1999. `http://www.w3.org/TR/xpath`.

[27] W3C. *XSL Transformations (XSLT) Version 1.0, W3C Recommendation*, 1999. `http://www.w3.org/TR/xslt`.

[28] W3C. *Extensible Markup Language (XML) 1.0 (Fourth Edition), W3C Recommendation*, 2006. `http://www.w3.org/TR/REC-xml`.

[29] W3C. *XQuery 1.0: An XML Query Language, W3C Recommendation*, 2007. `http://www.w3.org/TR/xquery`.

On Interchange between Drools and Jess

Oana Nicolae, Adrian Giurca and Gerd Wagner
Brandenburg University of Technology, Germany
E-mail: {nicolae, giurca, G.Wagner}@tu-cottbus.de

Keywords: Drools (aka JBossRules), Jess, RuleML, R2ML, RIF, Rete, ReteOO, business rules, interchange, standardisation

Received: March 15, 2008

There is a growing demand for research in order to provide insights into challenges and solutions based on business rules, related to target PSMs (Platform Specific Model in OMG's MDA terms - Implementation Model). As an answer to these needs, the paper argues on the relevance of business rules target platforms for the actual IT and business context, by emphasising the important role of business rules interchange initiatives. Therefore, the rule-system developers can do their work without any concern about a vendor-specific format, and in particular without any concern about the compatibility between the technologies. The paper provides a description of the business rules translation from a particular object oriented rule-system such as Drools, to another rule-system as Jess coming from the AI area, using R2ML as interchange language. The transformation preserves the semantic equivalence for a given rule set, taking also into account the rules vocabulary.

Povzetek: Prispevk opisuje prenos pravil iz objektnega sistema Drools v AI sistem Jess.

1 Introduction

There is a growing request for business rules technology standardisation from both UML and ontology architects communities. Due to these reasons, business rules aim to express rules in a platform independent syntax.

A number of initiatives on rules interchange have been started. They include the RuleML (2), OMG Production Rules Representation (PRR) (8), RIF (1), and the REWERSE I1 Rule Markup Language (R2ML[1]) (10). We mention here the efforts to establish some standards for expressing business rules and their vocabularies in natural language such as OMG's SBVR (9) and Attempto Controlled English (ACE) (4). SBVR, this human readable format of business rules comes under OMG's Model Driven Architecture (MDA[2]) standards and is defined as Computation-Independent Model (CIM[3]). CIM is most frequently used in the context of the Model Driven Architecture (MDA) approach which corresponds the Object Management Group (OMG) vision of Model Driven Engineering (MDE). The Meta-Object Facility (MOF), is the OMG standard for Model Driven Engineering.

The second layer in OMG's MDA is Platform-Independent Model (PIM)[4] where rule interchange formats (i.e. RuleML, RIF, R2ML) try to accomplish their general purpose: a PSM to PSM business rules migration through the PIM level. The third MDA level is Platform-Specific Model (PSM[5]) containing rule specific languages together with their specific engines/platforms like: F-Logic (5), JRules(ILOG[6]), Jess[7] or Drools[8].

The main purpose of an interchanging approach is to provide means for reusing, publication and interchange of rules between different systems and tools. Actually, it also plays an important role in facilitating business-to-customer (B2C) and business-to-business (B2B) interactions over the Internet. Moreover, an interchange approach always supposes less transformations than PSM-to-PSM translations.

Our rule interchange work addresses Drools as source platform and Jess as a target platform, using the approach suggested by OMG's MDA, because these languages are actually in business market interest as popular business logic frameworks, used by Java developers to create complex rule-based applications by combining Java platform and business rule technology. Another reason for choosing these two rule systems is their efficiency in "pattern" matching, especially to handle updates to its working set of facts, as both Drools and Jess use an algorithm known as the Rete (i.e. Latin for "net") algorithm. Computational complexity per iteration of this algorithm is linear in the size of the fact base.

The main standardisation communities, OMG[9] and W3C[10] focus their work on providing business rules specification languages for all MDA layers of models in order

[1]R2ML - `http://oxygen.informatik.tu-cottbus.de/rewerse-i1/?q=node/6`

[2]MDA - Model Driver Architecture is a framework for distinguishing different abstraction levels defined by the Object Management Group.

[3]CIM - Computational Independent Model

[4]PIM - Platform Independent Model

[5]PSM - Platform Specific Model

[6]*ILog*, `http://www.ilog.com`

[7]*Jess*, `http://herzberg.ca.sandia.gov/jess/`

[8]*JBossRules*, `http://labs.jboss.com/jbossrules/`

[9]OMG - http://www.omg.org/

[10]W3C - http://www.w3.org/

to obtain rules interchange. Their standards are not sustained by most of business rules management system tools, as they implement proprietary rule languages. The reasons for this situation imply the existence of only a few interchange works in the academia i.e. RIF (1) language still has no well defined guidelines of how to implement the transformations and it also does not specify how to test the correction of the translation.

In this context, EU network of Excelence REWERSE[11] developed R2ML as an interchange language for deploying and sharing rules between different rule systems and tools (e.g. Object Oriented rule languages, Semantic Web rule languages, Artificial Intelligence rule languages). Actually, R2ML (now at version 0.5) is a mature and experienced enough rule interchange language to provide a concrete interchange format for different rule systems and languages (i.e.`http://oxygen.informatik.tu-cottbus.de/rewerse-i1/?q=node/15`).

R2ML has a rich syntax, so it can represent business rules from both Drools and Jess languages, providing this way the interchange possibility. As an interchange language, R2ML addresses the PIM level. The main idea is to use a model transformation language (MTL), or an application transformation language (ATL) to transform a PIM model into a PSM as in the Figure 1.

Business rules are built following a business model representation. In many cases, a business model is first represented in a natural language description based on core ontologic concepts like classes and variables (OMG's MDA - CIM level).

At this stage, we can identify all objects referenced in the rules, and for each object we identify all referenced properties. For each property, we identify all its constraints.

2 Drools to R2ML mapping

In this section we describe the general JBoss business rules transformation into R2ML interchange language. Drools engine project, (now at version $4.0.x$) is an open source and standards-based business rule engine and it uses an enhanced implementation named ReteOO[12].

Drools is classified as an Object-Oriented Production Rules engine written entirely in Java language, and more specifically it is a Forward-Chaining rule engine.

A Production Rules System (i.e. PRS) relies on an Inference Engine that is able to scale to a large number of rules and facts. The Inference Engine matches facts and data, against PRs, also called Productions or just Rules, to infer conclusions which result in actions. The Rules are stored in the Production Memory. The facts that the Inference Engine matches against the rules are stored in the Working Memory.

R2ML is a visual rule markup, XML-based language, whose purpose is to capture rules formalised in different languages and to interchange them between rule systems and tools. It provides support for all kind of rules:

- Integrity Rules
- Derivation Rules
- Production Rules
- Reaction Rules

A R2ML production rule has *conditions* and *post-conditions*. The conditions and post-conditions of a R2ML production rule are usually interpreted as logical formulae which correspond to a general first order formula: quantified formula, existentially quantified or universally quantified (i.e. R2ML uses the concept of `r2ml:QuantifiedFormula` and by default, all R2ML formulae are universally quantified). Usually, PRS does not explicitly refer to events, but events can be simulated in a production rule system by externally asserting corresponding facts into the Working Memory. The R2ML production rules metamodel is depicted in the Figure 2:

The mapping from Drools to R2ML is possible as R2ML supports the representation of the PRs by relying on the OMG's PRR (8) Specification. Following sections describe general principles of mapping from JBoss rules into R2ML PRs.

2.1 Mapping rules vocabularies

Object oriented rules systems as Drools and ILOG JRules are build on top of Java vocabularies. Drools is designed to use Java beans as *facts*. These facts represent the domain of the rules, meaning the rules vocabulary. Java beans objects are defined by users in their applications.

These objects inserted into Working Memory represent the valid facts that rules can access. Facts are the application data, meanwhile the rules represent the logic layer of the application. This vocabulary is used by rules through the *import* declarations, which are specified inside of the rules file (*drl* files or *xml* files). For example, a rule from Drools may use one or many Java beans classes in order to describe its own vocabulary. The Java bean classes represent a description of the facts used by the Drools rule engine.

A R2ML rule always refers to a vocabulary which can be R2ML own vocabulary or an imported one (i.e. UML[13], RDF(S)[14] and OWL[15] - see lines 3.-4. from Section 2.2 i.e. an example of the importing an OWL external vocabulary for an entire R2ML production rule set). R2ML vocabulary is a serialisation of an UML fragment of class diagrams. Below, we describe the corresponding translation class from a usual Java bean into R2ML elements of the vocabulary namespace with the help of the optionally element `r2mlv:Vocabulary` i.e. Since almost all names from

[11]REWERSE - http://rewerse.net/

[12]RETE adaptation for an object-oriented language, a descendant of the well-known RETE algorithm

[13]UML - http://www.uml.org

[14]RDF(S) - http://www.w3.org/TR/rdf-schema

[15]OWL - http://www.w3.org/2004/OWL

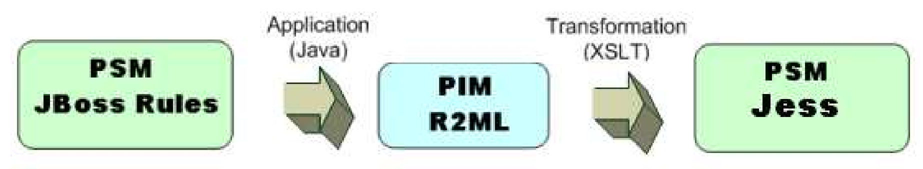


Figure 1: Interchanging between Drools and Jess.

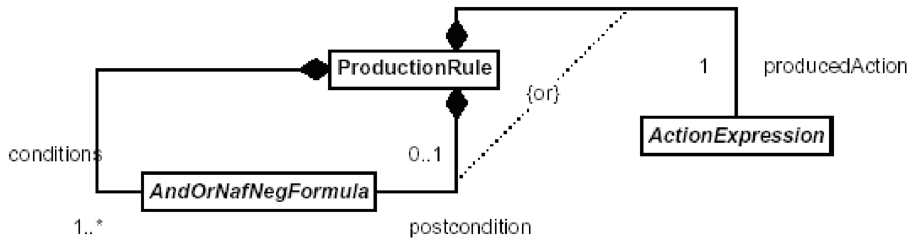


Figure 2: R2ML Production Rules Representation Metamodel.

```
 1.<r2ml:RuleBase
 2.    xmlns:r2ml=
       "http://www.rewerse.net/I1/2006/R2ML"
 3.    xmlns:dc=
       "http://purl.org/dc/elements/1.1/"
 4.    xmlns:ex=
       "http://www.businessrulesforum.com/2007/"
 5.    xmlns:xsi=
       "http://www.w3.org/2001/XMLSchema-instance"
 6.    xsi:schemaLocation=
       "http://www.rewerse.net/I1/2006/R2ML
 7.    http://oxygen.informatik.tu-cottbus.de/
                                 R2ML/0.5/R2ML.xsd"
 8.<r2mlv:Vocabulary>
 9. <r2mlv:Class r2mlv:ID="Cheese">
10.  <r2mlv:Attribute r2mlv:ID="type">
11.   <r2mlv:range><r2mlv:Datatype
       r2mlv:ID="xs:string"/>
12.  </r2mlv:range>
13.  </r2mlv:Attribute>
14.  <r2mlv:Attribute r2mlv:ID="price">
15.   <r2mlv:range><r2mlv:Datatype
       r2mlv:ID="xs:integer"/>
16.  </r2mlv:range>
17.  </r2mlv:Attribute>
18.  <r2mlv:Attribute r2mlv:ID="bestBefore">
19.   <r2mlv:range><r2mlv:Datatype
       r2mlv:ID="xs:dateTime"/>
20.   </r2mlv:range>
21.  </r2mlv:Attribute>
22. </r2mlv:Class>
23.</r2mlv:Vocabulary>
24.</r2mlv:RuleBase>
```

R2ML rule bases are qualified names (`xs:QName`), they must have declared the corresponding namespaces (i.e. see above lines 2.-5.). In the same manner, any Java qualified class name will be translated into a qualified name (`xs:QName`) together with the corresponding names declarations.

For example, if we assume the Drools import declaration (i.e. a Java qualified name): `org.drools.usecase.Cheese`, this will translate into the following namespace declaration `xmlns:ex="http://www.drools.org/usecase"` used in the qualified name (i.e. `ex:Cheese`), in order to reference the class name.

2.2 Rule Sets Mapping

All imported Java beans in Drools rule packages form the rules vocabulary. The set of Drools rules is individualized by its *package namespace*, declared at the beginning of the rules file, namespace that can be equal or can differ from Drools *import* declarations i.e. The Drools package of rules

```
package org.drools.rules;

import org.drools.usecase.Cheese;
/* set of Drools rules */
```

finds its correspondent into the `r2ml:ProductionRuleSet` element. It contains three optional attributes:

- `r2ml:ruleSetID` - is the name of the rule set. The name of the Java package of classes identifies in an unique way the name of a R2ML ProductionRuleSet (i.e. see line 2. from below R2ML code example).
- `r2ml:externalVocabulary` - represents an URI of an external vocabulary. We used OWL to represent the vocabulary of the rule.
- `r2ml:externalVocabularyLanguage` - refers the language of the external vocabulary.

```
1. <r2ml:ProductionRuleSet
2.  r2ml:ruleSetID="org.drools.rules"
3.  r2ml:externalVocabulary="http://..."
4.  r2ml:externalVocabularyLanguage="OWL">
```

Excepting the *rules* and their *import* declarations, a Drools package may contain other specific constructs like: *globals*, user-defined *functions* and *queries*, but they do not represent the subject of our translation.

```
rule "<name>"
 when
  <LHS>
 then
  <RHS>
end
```

2.3 Rule Mapping

In Drools, a rule consists of the rule *identifier*, the *conditions* part called LHS (i.e. Left Hand Side) and the *actions* part called RHS (i.e. Right Hand Side). The general principles of mapping a Drools rule into a R2ML production rule is:

- Every JBoss production rule is translated into a `r2ml:ProductionRule` element. An optional element `r2ml:Documentation` can contain elements which comprise the rule text and also the representation of the rule in a specific rules language.
- A R2ML `r2ml:ruleID` production rule attribute is generated using the JBoss `<name>` value. The `r2ml:ruleID` unique identifies a rule inside a rule set.
- A JBoss rule has a conditions part (i.e. *when* part) and an action part (i.e. *then* part). The condition part of a JBoss rule is mapped into the content of `r2ml:conditions` role element. The RHS part of a JBoss rule which contains multiple actions maps into the content of `r2ml:producedActionExpr` role element.
- The Drools language syntax also contains the comments expressed in Java-like syntax, such as:

  ```
  // java-like, single line comment
  #  single line comment
  /* ...
     java-like, multi lines comment
     ... */
  ```

  When translated into R2ML syntax, they map into the XML `<[!CDATA[...]]>` construct. For example:

  ```
  <r2ml:Documentation>
   <r2ml:RuleText r2ml:textFormat="plain">
     <[!CDATA[
      JBoss rule expressed in natural language...
       ]]>
   </r2ml:RuleText>
  </r2ml:Documentation>
  ```

In the following lines we describe the mapping of Drools conditions into R2ML appropriate ones.

The LHS (i.e. *when* part) of a JBoss rule consists of *patterns* (i.e. columns) and `eval` as *Conditional Elements* (i.e. CE) in order to facilitate the encoding of propositional logic and First Order Logic i.e. FOL. The entire LHS of a Drools rule is in fact a tuple of facts (i.e. a tuple of patterns). Each pattern may have zero or more field constraints i.e. the pattern terms (see Figure 4). The `and` (i.e. `&&`) CE is implicit when the JBoss rule condition contains multiple patterns. Field constraints compare and assess the field values from the fact object instances. Drools facts from Working Memory are Java beans objects instances, therefore these field constraints can be accessed from the "no arguments" methods, also called the accessors (i.e. getters).

2.3.1 Mapping Drools patterns without Field Constraints

A Drools pattern without field constraints, will map into the `r2ml:ObjectClassificationAtom`. For example, the following Drools pattern, which corresponds to universally quantified formula from classical logic: $\forall ?c\, Cheese(?c)$ is expressed in Drools as following:

```
$c: Cheese()
```

This Drools pattern finds its R2ML translation into the below code. As an explanation, we mention that all the R2ML formulae are implicitly universal quantified:

```
1.<r2ml:ObjectClassificationAtom
2. r2ml:class="Cheese">
3. <r2ml:ObjectVariable r2ml:name="c"
4.  r2ml:class="Cheese"/>
5.</r2ml:ObjectClassificationAtom>
```

Following RuleML, R2ML framework defines the generic concepts of variable. However, R2ML makes a clear distinction between *object terms* and *data terms*.

Typed terms are either *object terms* standing for *objects*, or *data terms* standing for *data values*. The concrete syntax of first-order non-Boolean OCL (7) expressions can be directly mapped to R2ML abstract concepts of *ObjectTerm* and *DataTerm*, which can be viewed as a predicate-logic-based reconstruction of the standard OCL abstract syntax.

The bounded variable `c` represents the value of the `r2ml:name` attribute of the corresponding term (`r2ml:ObjectName` and/or `r2ml:ObjectVariable`) and the name of the Java bean class (i.e. `Cheese`) is the value of `r2ml:class` attribute. The above Drools pattern can be declared inside rules conditions also without the `c` variable, such as: `Cheese()`, but to be able to refer to the matched facts, usually, the rules conditions use a pattern binding variable such as `c` (i.e. in Drools terminology we refer to it as a *fact variable* or *declaration*).

Any JBoss variables translate into R2ML variables. Notice that the translation of the Drools variables into R2ML eliminates the `$` symbol (used in Drools only as a notation convention) from the names of the variables. The JBoss *fact variable* used in the previous pattern example (i.e. `c:Cheese()`) is mapped into `r2ml:ObjectVariable` using the value of `r2ml:name="c"` property to describe the variable name, which represents an instance of the `Cheese` class (see lines 3.-4.). The usage of this instance gives us the possibility to call properties and functions of `Cheese` class in the actions part of a JBoss rule. The optional `r2ml:class` property (see line 4. from the above example) specifies the type of the object variable (i.e. `Cheese`). An `r2ml:ObjectVariable` is a variable that can be only instantiated by objects.

2.4 Mapping Drools patterns with Field Constraints

In many cases, a JBoss *pattern* (see Figure 3) may contain many field constraints, all of them referring to the same context variable. The Drools field constraints may be of the following possible types (i.e. string, numeric, boolean and date). When separated by the following operators (i.e. enumerated here in their priority order see also Figure 5): `&&`, `||` and `,` (i.e. comma), they form a Drools pattern formula.

A Drools pattern formula translates into R2ML formula, using the R2ML simple/imbricated concepts of `r2ml:qf.Disjunction` and `r2ml:qf.Conjunction` (`qf` stands for "quantifier free") applied on R2ML atoms, in order to serialize the Drools CE `||` and `&&`, respectively.

In the example below, we have two Drools patterns that in classical logic have the following representation, taking into account the operators order from Drools i.e.

$$\forall ?c\ \forall ?p\ \exists ?youngCheese\ (Person(?p) \land like(?p, ?youngCheese)\ \land\ (Cheese(?c) \land (type(?c, ?youngCheese)\ \land\ price(?c) < 10) \lor bestBefore(?c)\ <\ "27-Oct-2010"))$$

```
1.$p:Person($youngCheese:like)
2.$c:Cheese(type == $youngCheese &&
3.          price < 10 ||
4.          bestBefore < "27-Oct-2010")
```

The `Cheese` pattern (see lines 2.-4.) has three field constraints combined with a conjunctive connector (i.e. `&&`) and a disjunctive connector (i.e. `||`). We mention that the comma represents by default the conjunctive logic operator. The pattern refers literal constraints used to match the facts (i.e. instances of `Cheese` class): `type` (i.e. string constraint), `price` (i.e. numeric constraint) and `bestBefore` (i.e. date type constraint). The valid operators that apply for the numeric and date operands are: ==, != , <,>, <=, >=.

The above Drools pattern translates into the following R2ML formula (i.e. the example below describes only the imbrication of the operators (`&&`, `||`) inside the Drools pattern):

```
 1.<r2ml:qf.Disjunction>
 2. <r2ml:qf.Conjunction>
 3.  <r2ml:DatatypePredicateAtom>
 6.    ...
 7.  </r2ml:DatatypePredicateAtom>
 8.   <r2ml:DatatypePredicateAtom>
 9.    ...
10.  </r2ml:DatatypePredicateAtom>
11. </r2ml:qf.Conjunction>
12. <r2ml:DatatypePredicateAtom>
14.    ...
15. </r2ml:DatatypePredicateAtom>
16.</r2ml:qf.Disjunction>
```

In the absence of the `qf.Disjunction` or `qf.Conjunction`, all atoms from the R2ML rule body are implicitly connected by conjunction.

First pattern from the Drools example above (see line 1.) contains as field constraint a bound variable, called declaration. The JBoss variable `$youngCheese` is bound to the `like` property, so it can later constrain the `type` property of the `Cheese` class. Since the `like` property is a data type property (i.e. String), the R2ML mapping is an `r2ml:AttributionAtom`, while for an object type property would find its mapping into the `r2ml:ReferencePropertyAtom`.

```
<!--$p:Person($youngCheese:like)-->
 1.<r2ml:AttributionAtom
 2. r2ml:attribute="ex:Person.like">
 3. <r2ml:subject>
 4.  <r2ml:ObjectVariable
 5.   r2ml:name="p"
 6.   r2ml:class="Person"/>
 7. </r2ml:subject>
 8. <r2ml:dataValue>
 9.  <r2ml:DataVariable
10.   r2ml:name="youngCheese"
11.   r2ml:datatype="xs:string"/>
12. </r2ml:dataValue>
13.</r2ml:AttributionAtom>
```

The `r2ml:AttributionAtom` contains the `r2ml:subject` element which encloses the object term we refer. This can be expressed, for example, by an `r2ml:ObjectVariable` (i.e. lines 4.-6.). The value of the JBoss property is referred by the `r2ml:dataValue`. In this example, the value is encoded by the `r2ml:DataVariable` element (i.e. lines 9.-12.). The `youngCheese` variable borrows the type of the `youngCheese` property. R2ML uses XML Schema Datatypes[16] as its default namespace for encoding basic datatypes. The usage of this set of pre-declared datatypes is not mandatory, the user can specify any other appropriate URI and namespace for referring later in the rules its datatypes declaration. The Java/XML type correspondence it is done according to the JAXB[17] binding style correspondence principle, therefore a Java String value will be translated into `xs:string` qualified name (see line 11.).

The relational operations from Drools are serialized into R2ML language using the `r2ml:DatatypePredicateAtom` construct. Until this version, R2ML had not declared its own built-in constructs, but it allows the use of external ones, such as SWRL[18] built-ins for representing the predicate type of the relational operations (i.e. `swrlb:lessThen`). We also use the construct of `r2ml:DatatypePredicateAtom` to serialize the Drools literal field constraints that test equality / inequality of data types properties. When serializing object types literal field constraints we use the `r2ml:EqualityAtom` to express the concept of equality and the `r2ml:InequalityAtom` to express the concept of inequality (`!=`).

The following Drools pattern describes a Drools literal String constraint. Drools pattern translates into `r2ml:DatatypePredicateAtom` using the `r2ml:datatypePredicate="swrlb:equal"` SWRL build-in to represent the equality operator and serialises the `type` property into the `r2ml:AttributeFunctionTerm` (see

[16]XML Schema Part 2: Datatypes Second Edition - http://www.w3.org/TR/xmlschema-2/

[17]Java Architecture for XML Binding - java.sun.com/developer/technicalArticles/WebServices/jaxb/

[18]SWRL - http://www.w3.org/Submission/SWRL

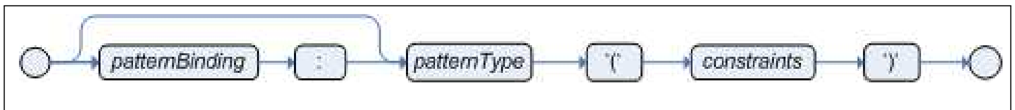


Figure 3: Drools Pattern concept.

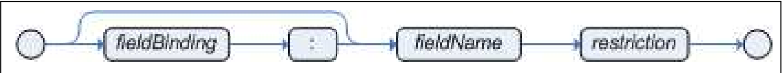


Figure 4: Drools Field Constraint concept.

lines 4.-10.). The `r2ml:dataArguments` attribute comprises

```
$c:Cheese(type == $youngCheese)
```

the Drools operands translation into R2ML terms objects or data types, depending on the types of the involved properties. The Drools `$youngCheese` variable is expressed using the concept of `r2ml:DataVariable` i.e.

```
<!--$c:Cheese(type == $youngCheese)-->
1.<r2ml:DatatypePredicateAtom
2. r2ml:datatypePredicate="swrlb:equal">
3. <r2ml:dataArguments>
4.  <r2ml:AttributeFunctionTerm
5.   r2ml:attribute="ex:Cheese.type">
6.   <r2ml:contextArgument>
7.    <r2ml:ObjectVariable r2ml:name="c"
8.     r2ml:class="ex:Cheese"/>
9.    </r2ml:contextArgument>
10.  </r2ml:AttributeFunctionTerm>
11.  <r2ml:DataVariable r2ml:name="youngCheese"
12.   r2ml:datatype="xs:string"/>
13. </r2ml:dataArguments>
14.</r2ml:DatatypePredicateAtom>
```

The relational operation `$c:Cheese(price < 10)` is expressed by the `r2ml:DatatypePredicateAtom` and the build-in `swrlb:lessThan`. The case also involves the `r2ml:AttributeFunctionTerm` for representing the property `price` of the Cheese class (i.e. `ex:Cheese.price` (see lines 4.-10.) and the `r2ml:TypedLiteral` term (see lines 11.-12.) for encoding the Java integer value into XML `xs:integer`.

```
<!--$c:Cheese(price < 10)-->
1.<r2ml:DatatypePredicateAtom
2. r2ml:datatypePredicate="swrlb:lessThan">
3. <r2ml:dataArguments>
4.  <r2ml:AttributeFunctionTerm
5.   r2ml:attribute="ex:Cheese.price">
6.   <r2ml:contextArgument>
7.    <r2ml:ObjectVariable r2ml:name="c"
8.     r2ml:class="ex:Cheese"/>
9.    </r2ml:contextArgument>
10.  </r2ml:AttributeFunctionTerm>
11.  <r2ml:TypedLiteral r2ml:lexicalValue="10"
12.   r2ml:datatype="xs:integer"/>
13. </r2ml:dataArguments>
14.</r2ml:DatatypePredicateAtom>
```

The Drools literal date type field constraints are represented into R2ML using XML qualified name `xs:dateTime`.

The Drools numeric operators work analogous for this type of field constraint, so the serialization into R2ML code is also the `r2ml:DatatypePredicateAtom` i.e.

```
<!--$c:Cheese(bestBefore<"27-Oct-2007")-->
 1.<r2ml:DatatypePredicateAtom
 2. r2ml:datatypePredicate="swrlb:lessThan">
 3.  <r2ml:dataArguments>
 4.  <r2ml:AttributeFunctionTerm
 5.   r2ml:attribute="bestBefore">
 6.   <r2ml:contextArgument>
 7.    <r2ml:ObjectVariable r2ml:name="c"
 8.     r2ml:class="Cheese"/>
 9.   </r2ml:contextArgument>
10.  </r2ml:AttributeFunctionTerm>
11.  <r2ml:TypedLiteral
12.   r2ml:datatype="xs:dateTime"
13.   r2ml:lexicalValue="2007-10-27Z"/>
14. </r2ml:dataArguments>
15.</r2ml:DatatypePredicateAtom>
```

Another meaningful example of Drools field constraints is testing the equality or inequality of a property against the Java `null` value i.e.

```
$c:Cheese(buyer == null)
$c:Cheese(buyer != null)
```

The corresponding formula from the classical logic would be:

$$\forall ?c\, \forall ?t\; Cheese(?c) \wedge \neg buyer(?c, ?t)$$

$$\forall ?c\, \exists ?t\; Cheese(?c) \;\wedge\; buyer(?c, ?t)$$

Taking into account the above classical logical formula, the R2ML serialization results in the `r2ml:EqualityAtom`, having its meaning negated using `r2ml:isNegated=true` attribute. The child elements for the `r2ml:EqualityAtom` are object terms. Our example involves an `r2ml:ReferencePropertyTerm` with the attribute `r2ml:referenceProperty="ex:Cheese.buyer"` and a generated object term expressed using `r2ml:ObjectVariable` with the generated attribute value `r2ml:name="t_24535899"` and the `r2ml:class="ex:Person"` as the type of the `buyer` property.

The second logic formula involves the existence of a `Cheese` fact into Working Memory, whose `buyer` property is not null. The R2ML first step in the R2ML serialization is the generation of an object term (i.e. the `r2ml:ObjectVariable` `t_57685642`) of `ex:Person` type which is bounded to `buyer` property. We use the `r2ml:ReferencePropertyAtom` element i.e.

We have mentioned before that implicitly, the `and` operator binds the Drools patterns inside the rule condition.

```
 <!--$c:Cheese(buyer==null)-->
 1.<r2ml:EqualityAtom
 2. r2ml:isNegated="true">
 3. <r2ml:ReferencePropertyFunctionTerm
 4.  r2ml:referenceProperty="ex:Cheese.buyer">
 5.  <r2ml:contextArgument>
 6.   <r2ml:ObjectVariable
 7.    r2ml:name="c"
 8.    r2ml:class="ex:Cheese"/>
 9.  </r2ml:contextArgument>
10. </r2ml:ReferencePropertyFunctionTerm>
11. <r2ml:ObjectVariable
12.  r2ml:name="t_24535899"
13.  r2ml:class="ex:Person"/>
14.</r2ml:EqualityAtom>
```

```
 <!--$c:Cheese(buyer!=null)-->
 1.<r2ml:ReferencePropertyAtom
 2. r2ml:referenceProperty="ex:Cheese.buyer">
 3. <r2ml:subject>
 4.  <r2ml:ObjectVariable
 5.  r2ml:name="c" r2ml:class="ex:Cheese"/>
 6. </r2ml:subject>
 7. <r2ml:object>
 8.  <r2ml:ObjectVariable r2ml:name="t_57685642"
 9.   r2ml:class="ex:Person"/>
10. </r2ml:object>
11.</r2ml:ReferencePropertyAtom>
```

```
//$p:Person()
1.$c:Cheese(buyer == $p, inStock == true)
```

A Drools pattern containing a formula that implies only field constraints conjunctions, can be split into as many Drools patterns as field constraints it contains i.e.

```
//$p:Person()
1.$c:Cheese(buyer == $p)
2.$c:Cheese(inStock == true)
```

There are two possibilities to markup the above `Cheese` patterns (see lines 1.-2.): to use a conjunction of R2ML appropriate atoms or to use the `r2ml:ObjectDescriptionAtom` construct.

First option implies the use of the `r2ml:ReferencePropertyAtom` in order to markup the first pattern (see line 1. from Drools example) and a `r2ml:AttributionAtom` for the data type boolean field constraint (see line 2. from Drools example) i.e.

```
<!--$c:Cheese(buyer == $p)-->
 1.<r2ml:ReferencePropertyAtom
 2. r2ml:referenceProperty="ex:Cheese.buyer">
 3. <r2ml:subject>
 4.  <r2ml:ObjectVariable
 5.   r2ml:name="c"
 6.   r2ml:class="ex:Cheese"/>
 7. </r2ml:subject>
 8. <r2ml:object>
 9.  <r2ml:ObjectVariable
10.   r2ml:name="p"
11.   r2ml:class="ex:Person"/>
12. </r2ml:object>
13.</r2ml:ReferencePropertyAtom>
```

A `r2ml:ReferencePropertyAtom` associates two object terms, having different meanings: `subject` (see lines 3.-7.) and `object` (see lines 8.-12.). For example, to express the concept of: "buyer of cheese" we use the code above, where the subject is a `Cheese` instance and the object is a `Person` instance.

The second solution translates directly the two `Cheese` patterns of the rule using the `r2ml:ObjectDescriptionAtom`. The referred R2ML serialization is a conjunction of equality constraints i.e. an enumeration of `r2ml:DataSlot`(s) or `r2ml:ObjectSlot`, depending on the type of the involved properties objects or data, respectively. The attribute `r2ml:class="Cheese"` corresponds to the patterns name from Drools implementation i.e.

```
 1.<r2ml:ObjectDescriptionAtom
 2. r2ml:class="Cheese">
 3. <r2ml:subject>
 4.  <r2ml:ObjectVariable r2ml:name="c"/>
 5. </r2ml:subject>
 6. <r2ml:ObjectSlot
 7.  r2ml:referenceProperty="Cheese.buyer">
 8.  <r2ml:object>
 9.   <r2ml:ObjectVariable r2ml:name="p"
10.    r2ml:class="Person"/>
11.   </r2ml:object>
12.  </r2ml:ObjectSlot>
13.  <r2ml:DataSlot
14.   r2ml:attribute="ex:Cheese.inStoc">
15.   <r2ml:value>
16.    <r2ml:DataVariable
17.     r2ml:name="true"
18.     r2ml:datatype="xs:boolean"/>
19.   </r2ml:value>
20.  </r2ml:DataSlot>
21.</r2ml:ObjectDescriptionAtom>
```

Another CE from the LHS of a Drools rule is the pattern disjunction (`||` / `or`). The Drools disjunction of multiple patterns results in multiple rule generation, called subrules, for each possible outcome i.e.
$\forall ?c\, Cheese(?c) \wedge (type(?c, stilton) \vee type(?c, cheddar))$

```
1.Cheese(type=="stilton") or Cheese(type=="cheddar")
2.Cheese(type=="stilton") || Cheese(type=="cheddar")
```

In the above examples the Drools `or` CE is a shortcut for generating two additional rules. There can be multiple activations for a rule, if both sides of the CE are true. The R2ML serialization uses the `r2ml:qf.Disjunction` that contains each of the `Cheese` properties mapped with the `r2ml:AttributionAtom`.

The Drools negation `not` represents the existential quantifier that checks for the non existence of some facts in Working Memory. Currently, this existential quantifier it is applied only for patterns i.e. $\forall ?c\, \neg Cheese(?c)$.

```
not Cheese()
```

The above pattern tests if there are not `Cheese` facts in the Working Memory. The "not" pattern (see Figure 7) can

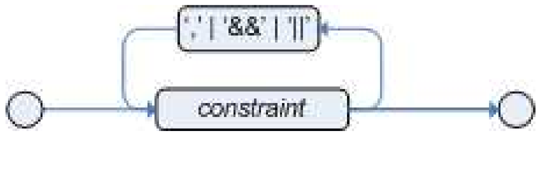


Figure 5: Drools constraints inside a pattern.

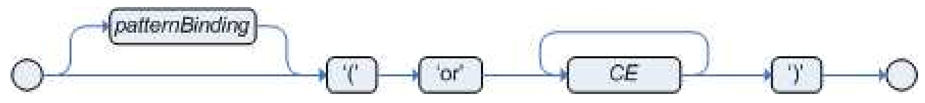


Figure 6: Drools "or" CE.

not have a pattern binding. We still can not serialize the existentially quantifier concept into R2ML language.

But, by applying the negation to the entire formula, we obtain the following expression from the classical logic: $\forall ?c \neg Cheese(?c)$. The R2ml serialization of the above formula uses the concept of `r2ml:qf.Negation` which embeds a `r2ml:ObjectClassificationAtom`.

```
<r2ml:ObjectClassificationAtom r2ml:isNegated="true"
 r2ml:classID="Car">
 <r2ml:ObjectVariable r2ml:name="c_974376"/>
</r2ml:ObjectClassificationAtom>
```

```
not Cheese(type == "stilton")
```

The above pattern tests if there are not `Cheese` facts of type `stilton` in the Working Memory. In the classical logic would be i.e.

$$\forall ?c\, Cheese(?c) \;\wedge\; \neg\, type(?c, stilton)$$

. We serialize it using the `r2ml:AttributionAtom` i.e.

```
<r2ml:AttributionAtom
r2ml:attribute="ex:Cheese.type"
r2ml:isNegated="true">
<r2ml:subject>
 <r2ml:ObjectVariable r2ml:name="c_6587483"
  r2ml:class="ex:Cheese"/>
</r2ml:subject>
<r2ml:dataValue>
 <r2ml:TypedLiteral
  r2ml:lexicalValue="stilton"
  r2ml:datatype="xs:string"/>
</r2ml:dataValue>
</r2ml:AttributionAtom>
```

2.4.1 Mapping Drools Actions

Java beans objects/instances are defined by users in their applications. These objects inserted into Working Memory (i.e. WM) represent the valid facts which the rules can access. Facts are the application data, meanwhile the rules represent the logic layer of the application. The term Working Memory Actions is used to describe assertions, retractions and modifications of facts within Working Memory. When discussing about the Drools - R2ML mapping of actions, we are only referring to the JBoss rule actions that find their mapping into R2ML.

R2ML actions are built according with the OMG PRR Specification (8), which stipulates that an action is either an `r2ml:InvokeActionExpression` or an `r2ml:UpdateStateActionExpr`. The R2ML actions are encoded by the content of `r2ml:producedActionExpr` role element i.e.

`r2ml:InvokeActionExpression` - invokes an operation (by means of the `r2ml:operation` attribute) with an ordered, possible empty list of parameter arguments represented as R2ML terms. In the following example, we map a Java output operation, which has as `r2ml:arguments` the Cheese instance (i.e. `c`), previously, supposed to be declared in the JBoss rule condition (see lines 4.-5.) and a String argument, that is translated into `r2ml:TypedLiteral` (see lines 6.-8.).

```
then
 System.out.println($c+" out of stock.");
end
```

The R2ML translation:

```
 1.<r2ml:InvokeActionExpression
 2. r2ml:operation="System.out.println">
 3. <r2ml:arguments>
 4.  <r2ml:ObjectVariable r2ml:name="c"
 5.   r2ml:Class="ex:Cheese"/>
 6.  <r2ml:TypedLiteral
 7.   r2ml:lexicalValue=" out of stock"
 8.   r2ml:datatype="xs:string"/>
 9. </r2ml:arguments>
10.</r2ml:InvokeActionExpression>
```

`AssertActionExpr` - contains a collection of slots i.e. property-value pairs (e.g. `r2ml:DataSlot` / `r2ml:ObjectSlot`) in order to represent the data/object type properties of a fact. We use this R2ML action call in order to map the JBoss `insert(object)` / `insertLogical(object)` Working Memory Actions, which has the purpose to insert new memory data.

```
then
  // Offer(cheese, price)
  Offer offer = new Offer($cheese,100)
  insert(offer);
end
```

The R2ML translation needs an instance of the `Offer` class for its `r2ml:contextArgument`, which encode the context of the action call, so we generate an `r2ml:ObjectVariable` with the `r2ml:name="offer"`. We translate the instance of the Cheese class to a `r2ml:ObjectSlot` and the direct value `100` as a `r2ml:TypedLiteral` having the type of the `price` property of the `Offer` class (i.e. `xs:integer`).

```
<r2ml:AssertActionExpr
r2ml:class="ex:Offer">
<r2ml:contextArgument>
 <r2ml:ObjectVariable
  r2ml:name="offer"
  r2ml:class="ex:Offer"/>
</r2ml:contextArgument>
<r2ml:ObjectSlot
 r2ml:referenceProperty="ex:Offer.cheese">
 <r2ml:object>
  <r2ml:ObjectVariable
   r2ml:name="cheese"
```

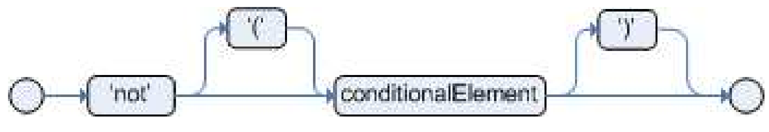


Figure 7: Drools "not" CE.

```
     r2ml:class="ex:Cheese"/>
   </r2ml:object>
  </r2ml:ObjectSlot>
  <r2ml:DataSlot
   r2ml:attribute="ex:Offer.price">
   <r2ml:value>
    <r2ml:TypedLiteral
     r2ml:datatype="xs:integer"
     r2ml:lexicalValue="100"/>
   </r2ml:value>
  </r2ml:DataSlot>
</r2ml:AssertActionExpr>
```

`RetractActionExpr` - deletes an object. Its `r2ml:contextArgument` always evaluates to an object term. We use this R2ML construct to map the JBoss WM Action for removing a previously declared fact (i.e. `c:Cheese()`) from the memory i.e.

```
then
  retract($c);
end
```

The R2ML translation is:

```
<r2ml:RetractActionExpr r2ml:class="ex:Cheese">
 <r2ml:contextArgument>
  <r2ml:ObjectVariable
   r2ml:name="c"
   r2ml:class="ex:Cheese"/>
 </r2ml:contextArgument>
</r2ml:RetractActionExpr>
```

`UpdateActionExpr` - updates a property of a specific object term specified by the `r2ml:contextArgument`. The below Drools example modifies the `type` property of a particular `Cheese` instance (i.e. `c`). The `update(c)` is necessary in order to notify the Drools engine about the changes from WM.

```
then
 $c.setType("cheddar");
 update($c);
end
```

The R2ML translation is:

```
<r2ml:UpdateActionExpr
 r2ml:property="ex:Cheese.type">
 <r2ml:contextArgument>
  <r2ml:ObjectVariable r2ml:name="c"
   r2ml:class="ex:Cheese"/>
 </r2ml:contextArgument>
 <r2ml:TypedLiteral
  r2ml:lexicalValue="cheddar"
  r2ml:datatype="xs:string"/>
</r2ml:UpdateActionExpr>
```

3 R2ML to Jess mapping

In this section we describe how R2ML rules are translated into Jess rule language. In Jess, rules are defined using the `defrule` construct. They are, in fact written as lists where the head is the special symbol `defrule`.

Jess provides two main categories of rules: forward-chaining rules and backward-chaining rules. Forward-chaining rules are the most common and used rules in Jess, and our translation will obtain Jess forward-rules. Jess is a forward-chaining reasoning engine, backward-chaining rules being simulated in terms of forward chaining (3).

To translate a R2ML rule into a Jess rule we will use Jess *unordered facts*. Unordered facts from Jess are alternatives for Java bean instances: objects that have named fields (i.e. properties) in which data appears (although the properties are traditionally called slots (see code example from 3.1)).

Any R2ML rule will translate into a Jess rule which uses unordered facts, because they are nice structured and are better emulating the internal structure of a R2ML rule (which includes the rule vocabulary).

We also use the *new-style*, simplified syntax of the Jess language, introduced in version 7.0[19]. We mention that the Drools language syntax had received and still receives a strong influence from Clips/Jess made and ongoing researches.

3.1 Mapping rules vocabulary

Jess business rules vocabulary consists of `deftemplate`(s) structures.

A `deftemplate` describes a fact, in the same way as a Java class describes an object. In particular, a `deftemplate` is a Jess concept which includes a name, an optional documentation string, an optional "extends" clause, an optional list of declarations, and a list of zero or more member variables (called slot descriptions) with a type qualifier. Each slot description can optionally include a type qualifier or a default value qualifier.

Using vocabulary classes from R2ML, corresponding Jess `deftemplates` are generated. The deftemplate[20] structure corresponds to the class description from the R2ML vocabulary 2.1 i.e. They can be placed in the same

```
(deftemplate Cheese
  (slot type (type STRING))
  (slot price (type INTEGER))
  (slot bestBefore (type OBJECT)) )
```

[19]Jess 7.0 - http://herzberg.ca.sandia.gov/docs/70/release_notes.html

[20]http://herzberg.ca.sandia.gov/docs/71/api/jess/Deftemplate.html

file as the rules, or in a separate file, which need to be imported into the rules file, using the `import` keyword.

3.2 Rule Sets Mapping

The output of the translation from R2ML to Jess is a `.jess` batch file (i.e. a Jess knowledge base). This batch file contains the facts and the rules which represent the input data and the logic for the Jess Rete engine, of which the current version is 7.0. A Jess rule set is a Jess batch file. The name of this file is obtained from the `r2ml:ruleSetID` attribute (see Section 2.2).

3.3 Rule Mapping

– The `r2ml:ruleID` attribute value is used to obtain a Jess rule ID, which is not allowed to contain spaces.

– In R2ML framework the content of `r2ml:conditions` role element which corresponds to an universally quantified formula is translated into the conditions part of a Jess rule i.e. LHS pattern. The LHS of a Jess rule consists of patterns that match facts.

– The content of a `r2ml:producedActionExpr` is markup as the actions part of a Jess rule i.e. RHS pattern introduced by the symbol => , which roughly denotes implication. The actions of a Jess rule are composed only of function calls.

– Jess language supports two kinds of comments: Lisp-style line comments (;) and C-style block comments (/*...*/), in order to translate the R2ML comments.

```
(defrule ruleName
  (pattern1)
  (pattern2)
    ;; ...
  =>
  (function calls))
```

3.3.1 Mapping r2ml:ObjectClassificationAtom

An R2ML atom corresponds to a Jess pattern. The `r2ml:ObjectClassificatioAtom` is used to capture the *instanceOf* relationship between objects and classes. Any R2ML object classification atom consists from a mandatory attribute `r2ml:class` with a (`xs:QName`) value and an object term as an argument.

A positive `r2ml:ObjectClassificationAtom` (see Section 2.3.1) is mapped into a Jess pattern without field constraints i.e. `?fact_variable <- (PatternName{})` where `?fact_variable` is the corresponding term (i.e. `r2ml:ObjectVariable`) and `PatternName` is the value of `r2ml:class` attribute i.e.

```
?c <- (Cheese{})
```

3.3.2 Mapping R2ML Formulae

Any `r2ml:qf.Conjunction` (*qf* stands from *quantifier free*) of atoms corresponds to a conjunction inside of Jess patterns. It represents an enumeration of Jess field constraints. If conjunctions contain conditions referring to the same context, then we translate all the R2ML atoms (i.e. the R2ML formula) into a Jess single pattern with a number of field constraints. Every R2ML atom finds its mapping into the Jess field constraint concept.

Any R2ML variable name is mapped into Jess variable identifier by adding the ? symbol as first character: `?fact_variables` and `?field_variables`. R2ML variables are provided in the form of `r2ml:ObjectVariable` and `r2ml:DataVariable`. `r2ml:ObjectVariable` are variables that can be only instantiated by objects, meanwhile `r2ml:DataVariable` are variables that can be only instantiated by data literals.

The `r2ml:ObjectVariable` is mapped into the Jess concept of `?fact_variable` using the value of the `r2ml:name` attribute as the variable name with type `xs:NCName`. The optional `r2ml:class` attribute specifies the membership of the object variable.

The `r2ml:DataVariable` with attribute `r2ml:typeCategory='individual'` is mapped into Jess as `?field_variable`, being instantiated only by data literal types properties.

The `r2ml:AttributionAtom` captures data valued properties of objects. Any `r2ml:AttributionAtom` maps into a Jess pattern i.e. `?fact_variable<-(PatternName {property ?value})` where `?fact_variable` is the content of the child role element `r2ml:subject` of the involved atom (see Section 2.4 lines 3.-7.), `PatternName` represents the content of attribute `r2ml:class` (see Section 2.4 line 6.), `property` is the value of the attribute `r2ml:attribute` and `?value` is the content of the child role element `r2ml:dataValue` of the atom (see Section 2.4 lines 8.-12.). The appropriate translation into Jess language is:

```
?p <- (Person{} (like ?youngCheese))
```

The `r2ml:DatatypePredicateAtom` is designed to capture built-in predicates. It refers to a user-defined datatype predicate by its mandatory `r2ml:datatypePredicate` attribute and consists of a number of data terms as data arguments (the children of `r2ml:dataArguments` attribute).

Current translation in Jess supports only operations which have two arguments. The `r2ml:datatypePredicate`, represents qualified names which express the appropriate build-ins: (e.g. `swrlb:equal`, `swrlb:lessThan`, `swrlb:lessThanOrEqual`, `swrlb:greaterThan`, `swrlb:greaterThanOrEqual`), and corresponds to appropriate, Jess operators used inside the patterns (==, <, <=, >, >=, ! =, <>).

The `r2ml:DatatypePredicateAtom` code examples from Section 2.4 find their mapping into the following Jess pattern with field constraints i.e.

```
?c <- (Cheese{type == ?youngCheese &&
            price < 10 ||
            bestBefore <= "27/10/2010"})
```

The `r2ml:AttributeFunctionTerm` is used by R2ML in order to express data valued properties of objects. The name of the attribute is revealed through the mandatory `r2ml:attribute` value. Usually, a R2ML `r2ml:AttributeFunctionTerm` is contained, as a child of `r2ml:dataArguments` construct in R2ML atoms (i.e. `r2ml:DatatypePredicateAtom`), in order to assign values to object properties (i.e. `type`, `price`, `bestBefore`).

The Jess translation will always take into consideration, the relation between `r2ml:AttributeFunctionTerm` and other R2ML atoms (see Section 2.4 lines 3.-10.).

R2ML data literals i.e. `r2ml:TypedLiteral` are mapped into Jess using the values of the `r2ml:lexicalValue` and `r2ml:datatype` attributes, into corresponding, Jess valid types. Internally, all Jess values (symbols, numbers, strings, lists etc) are represented by instances of the class `jess.Value`. Its possible values are enumerated by a set of constants defined into the `jess.RU` (i.e. Rete Utilities) class (i.e. ANY, INTEGER, FLOAT, NUMBER, SYMBOL, STRING, LEXEME, and OBJECT).

As Jess language does not provide a particular datatype in order to express the XML `xs:dateTime` value, a possible translation could be the use of the Jess class: jess.RU (i.e. OBJECT constant). As a consequence, the `bestBefore` slot from the `cheese` deftemplate construct has the type `OBJECT` and encapsulates the `java.util.Date` class.

An R2ML `r2ml:ReferencePropertyAtom` associates an object term as `r2ml:subject` with other object term as `r2ml:object`.

Therefore, the R2ML example of `r2ml:ReferencePropertyAtom` (see Section 2.4) will translate into the following Jess patterns:

```
?p <- (Person{})
?c <- (Cheese{buyer == ?p &&
            inStock == true})
```

The `r2ml:EqualityAtom` is intended to express equality of two object terms (i.e. `r2ml:ObjectVariable`). The corresponding translation into Jess implementation is the symbol of == operator. The `r2ml:isNegated` attribute set to `true` value, involves the use of != operator.

The `r2ml:InequalityAtom` is just a convenience construct to express the negation of the object terms equality. The corresponding translation into Jess implementation is the symbol of != operator. The `r2ml:isNegated` attribute set to `true` value, involves the use of == operator.

The Jess `nil` value has an equivalent meaning with the `null` value from Java language. Therefore, the corresponding mappings from Section 2.4 are the following i.e.

```
?c <- (Cheese{type == nil})
?c <- (Cheese{type != nil})
```

The `r2ml:ObjectDescriptionAtom` is a convenience construct to describe a set of property-object value pairs and/or a set of attribute-value pairs which refer to the same object as `r2ml:subject`.

We translate the `r2ml:ObjectDescriptionAtom` into a Jess pattern with field constraints. These constraints refer to the same child role element `r2ml:subject` and can be object terms (i.e. `r2ml:ObjectVariable`) captured in `r2ml:ObjectSlots`, or data terms (i.e. `r2ml:DataVariable`) captured in `r2ml:DataSlots`.

The R2ML code example from Section 2.4 will translate into:

```
?p <- (Person{})
?c <- (Cheese{buyer == ?p &&
            inStock == true})
```

Any `r2ml:qf.Disjunction` (*qf* stands from *quantifier free*) is translated into a disjunctive list of Jess patterns, using the Jess conditional element `or`, all bound to the same fact-variable. Any number of patterns can be enclosed in a list with `or` conditional element as the head.

This structure is a shortcut for generating two or more additional rules. For a rule which contains such disjunction of patterns, there could be multiple activations if multiple sides of the `or` are true i.e.

```
(or (Cheese{type=="stilton"})
    (Cheese{type=="cheddar"}) )
```

All R2ML negations i.e. `r2ml:qf.Negation`, and here we refer to both `r2ml:qf.StrongNegation`, and `r2ml:qf.NegationAsFailure` collapse in the Jess negation denoted by the keyword `not`, which currently applies only for patterns (see R2ML example from Section 2.4).

```
not(Cheese{})
```

Any R2ML atom has an optional, boolean property `r2ml:isNegated` which tells if the atom is, or is not negated. The corresponding Jess translation describes the negated R2ML atom. If the attribute `r2ml:isNegated` is missing, this is interpreted in R2ML by the default `r2ml:isNegated="false"`. The example from Section 2.4 describes an `r2ml:AttributionAtom` which supports a negation through its property `r2ml:isNegated="true"`. The Jess corresponding translation pattern is:

```
not (Cheese{type == "stilton"})
```

We mention that a `not` pattern cannot define any variables that are used in subsequent patterns (since a `not` pattern does not match any facts, it can not be used to define the values of any variables). Also, in our both previous examples a `not` pattern can not have a pattern binding.

3.4 Mapping R2ML actions into Jess Function Calls

The possibles actions of a R2ML production rule are defined using OMG's PRR Proposal (the content of `r2ml:producedActionExpr` role element). They are mapped into the corresponding Jess function calls i.e.

3.4.1 Mapping r2ml:InvokeActionExpression

An `r2ml:InvokeActionExpression` will map into Jess into a function call with/without arguments. Notice that `?c` is a bound variable to the `Cheese` pattern in the LHS of the rule. The R2ML code example from Section 2.4.1 finds its translation into the following Jess code:

```
(printout t ?c " out of stock!")
```

3.4.2 Mapping r2ml:AssertActionExpr

The `r2ml:AssertActionExpr` assert a new data into Working Memory. The analogous construct in Jess is the `assert` function containing the name of the `deftemplate` name and (slot value) pairs. The R2ML code example from Section 2.4.1 translates into:

```
(assert(offer (cheese ?c) (price 100)) )
```

3.4.3 Mapping r2ml:RetractActionExpr

The `r2ml:RetractActionExpr` deletes an object term from WM. According with the R2ML code from Section 2.4.1 the corresponding Jess translation is:

```
(retract (?c))
```

3.4.4 Mapping r2ml:UpdateActionExpr

The `r2ml:UpdateActionExpr` from our R2ML rule example (see Section 2.4.1) corresponds to the `modify` function invocation from Jess language, which updates an unordered fact from WM. Notice that `?c` is a bound variable to the `score` field value i.e.

```
(modify ?c (type cheddar))
```

4 Limitations of the proposed interchange

The translation process from PSM to PSM using intermediate R2ML (as PIM level) demands also the mapping of rules vocabulary. The Drools to R2ML Translator is a Java application that requests access to the Drools vocabulary (Java compiled classes) in order to establish the types of objects and primitives. As R2ML supports Production Rules format(8), the Drools to R2ML translation of rule conditions part relies naturally.

Interchange limits appear in the translation of the actions part of a JBoss rule, which may contain any Java valid code: variable declaration, non-declarative structures like `if...then` structures or cycling structures(i.e. `while`, `for`).

R2ML intends to solve this problem by providing in its future version an `<r2ml:OpaqueExpression>` with the role to encapsulate the code which do not find its semantic equivalent into R2ML `<r2ml:producedActionExpr>` role element. Also Drools tries to emerge its syntax from the OMG proposal for PRR, therefore the future design will take much more into consideration the standard actions.

The purpose of R2ML, as PIM level markup language, is to provide PSM to PSM rules translation by sharing a vocabulary model (actually R2ML vocabulary), which can easily be mapped into Jess vocabulary (deftemplate(s) structures) i.e.

$$\mathcal{V}_{Drools} \rightarrow \mathcal{V}_{R2ML} \rightarrow \mathcal{V}_{Jess}$$

Therefore, it makes possible the translation from R2ML to Jess language, as the business rules expressed in R2ML format do not require any conceptual changes in order to be implemented in PSM target platforms (e.g. Jess, F-Logic, RuleML, OCL, SWRL, Drools, Jena[21]). However, R2ML is not concerned with the vocabulary interchange issues, therefore this interchange of rules depends of the capabilities of a vocabulary interchange format. In this case, we use the R2ML vocabulary but rules may come with different vocabularies (for example UML, RDF(S) or OWL).

5 Interchange soundness

The soundness of the discussed interchange brought under attention the lack of a well established semantics for both languages (i.e. Drools and Jess). Our solution to establish the interchange soundness is by *testing rules*.

Let $\mathcal{W}_0 = \{f_1, \ldots, f_m\}$ the initial facts from Drools Working Memory and $\mathcal{R} = \{R_1, \ldots, R_n\}$ the current rules set of Drools encapsulating the logic of a specific application. The inference engine (Drools) will execute the rule set $\mathcal{R}$ against $\mathcal{W}_0$ obtaining $\mathcal{W} = \{g_1, \ldots, g_p\}$.

We encode the logic of the same application using the following set of Jess rules: $\mathcal{R}' = \{R'_1, \ldots, R'_n\}$ and as data the initial set of facts $\mathcal{W}'_0 = \{f'_1, \ldots, f'_m\}$. Running the Jess Inference Engine, we obtain a set of final Jess facts i.e. $\mathcal{W}' = \{g'_1, \ldots, g'_p\}$.

We translate the JBoss production rules into Jess implementation via R2ML and we execute those rules based on analogous facts from the Working Memory. The correctness of the translation implies the same obtained results regarding the facts from Working Memory.

A translation from Drools to Jess involves:

– a transformation function

$$Tr_{(Drools,R2ML)} : Drools \rightarrow R2ML$$

describing the serialisation from Drools to R2ML, where: Drools = ($\mathcal{V}_{Drools}$ x $\mathcal{W}$ x $\mathcal{R}$)

– a transformation function

$$Tr_{(R2ML,Jess)} : R2ML \rightarrow Jess$$

describing the mapping from R2ML to Jess, where: Jess = ($\mathcal{V}_{Jess}$ x $\mathcal{W}'$ x $\mathcal{R}'$)

[21] Jena Framework - `http://jena.sourceforge.net/`

Table 1: Drools to Jess Mapping Rules

Excerpt from Mapping from Drools to Jess		
Drools	R2ML	Jess
Drools rule	R2ML PR	Jess rule
import JavaBeans files	declare R2ML vocabulary	import deftemplates
fact variable	ObjectVariable	fact variable
field variable	ObjectVariable	field variable
Drools pattern	ObjectClassificationAtom	LHS's pattern
conjunction of field constraints	ObjectDescriptionAtom	conjunction of field constraints
field variable binding	AttributionAtom	field variable binding
relational operators on data	DatatypePredicateAtom	relational operators
==/ != operators on objects	ReferencePropertyAtom	==/ != on object terms
not	isNegated="true"	Jess negation
Drools pattern conjunction	qf.Conjunction	Jess conjunction
Drools pattern disjunction	qf.Disjunction	Jess disjunction
function call	InvokeActionExpression	Jess function call
create/assert an object	AssertActionExpr	assert function call
delete an object	RetractActionExpr	retract function call
setter call	UpdateActionExpr	modify function call

– a translation function $T_{(Drools,Jess)} = Tr_{(Drools,R2ML)}(Tr_{(R2ML,Jess)})$

and results in:

– $T_{(Drools,Jess)}(\mathcal{V}_{Drools}$ x $\mathcal{W}$ x $\mathcal{R}$) = ($\mathcal{V}_{Jess}$ x $\mathcal{W}'$ x $\mathcal{R}'$) (i.e. by applying the translation function $T_{(Drools,Jess)}$ on the $\mathcal{W} = \{g_1, \ldots, g_p\}$ set of facts, we obtain the following set of facts $\mathcal{W}_h = \{h_1, \ldots, h_p\}$ which is semantically and syntactically equivalent with the set $\mathcal{W}' = \{g'_1, \ldots, g'_p\}$, obtained from Jess inference process.

An informal rules translation from Drools to Jess, using R2ML as interchange language is presented in Table 1. The reverse translation, from Jess to Drools is also possible, as Jess supports a new richer syntax[22] which offers the capability to represent object types using *deftemplate* structures. One limitation of this syntax is that it can only be used with unordered facts.

6 Conclusion and future works

The paper provides a description of rule translation from Drools, an Object Oriented rule language, into Jess, an Artificial Intelligence rule language using R2ML as interchange format. The results presented in this paper are based on our previous work (6).

Our future works intend to establish a mathematical model of semantic soundness of rule interchange. Also compatibility of the interchange with the work in progress of W3C (i.e. RIF) will be analysed. Further results will be reported in a future work.

[22]http://herzberg.ca.sandia.gov/docs/71/memory.html

Acknowledgements

We would like to thanks to Edson Tirelli and Kris Verlaenen from the Drools Team for their helpful comments on our questions as well to Ernest Friedman-Hill, a precious mentor in all issues concerning the Jess language.

References

[1] H. Boley, M. Kifer (2007). *RIF Core Design*, W3C Working Draft, March 30, 2007 `http://www.w3.org/TR/rif-core/`

[2] H. Boley, S. Tabet and G.Wagner (2001) Design Rationale of RuleML: A Markup Language for Semantic Web Rules,*In Proc. of Int. Semantic Web Working Symposium (SWWS)*, Stanford University, California, USA.

[3] E.Friedman-Hill (2003), *Jess in Action - Rule-Based Systems in Java*, Manning Publications Co.

[4] N. E. Fuchs, U. Schwertel, R. Schwitter (1997), Attempto Ů Englisch als (formale) Spezifikationssprache, In: F. Bry, B. Freitag, D. Seipel (eds.), *Proceedings of the Twelfth Workshop on Logic Programming WLP*, Munich.

[5] M. Kifer, G. Lausen and J. Wu, Logical Foundations of Object-Oriented and Frame-Based Languages, *Journal of ACM*, May 1995.

[6] Oana Nicolae, Adrian Giurca and Gerd Wagner.*On Interchange between JBossRules and Jess* Proceedings of 1st International Symposium on Intelligent

and Distributed Computing (IDC 2007), October, 2007.

[7] OMG (2005). *OCL 2.0 Specification*, June 06, 2005, `www.omg.org/docs/ptc/05-06-06`

[8] OMG (2007). *Production Rule Representation Ver. 1.0* , March 5, 2007, `http://www.omg.org/docs/bmi/07-03-05.pdf`

[9] OMG (2006). *Semantics of Business Vocabulary and Business Rules (SBVR)*, `http://www.omg.org/docs/dtc/06-03-02.pdf`

[10] G. Wagner, A. Giurca and S. Lukichev (2005), R2ML: A General Approach for Marking up Rules, *Dagstuhl Seminar Proceedings 05371, In F. Bry, F. Fages, M.Marchiori, H. Ohlbach (Eds.) Principles and Practises of Semantic Web Reasoning*, ISSN:1862-4405.

An Approach to Extracting Sub-schema Similarities from Semantically Heterogeneous XML Schemas

Pasquale De Meo and Giovanni Quattrone
Università Mediterranea di Reggio Calabria
Via Graziella, Località Feo di Vito
89122 Reggio Calabria, Italy
E-mail: demeo@unirc.it, quattrone@unirc.it

Giorgio Terracina
Dipartimento di Matematica
Università della Calabria
Via Pietro Bucci
87036 Rende (CS), Italy
E-mail: terracina@mat.unical.it

Domenico Ursino
Università Mediterranea di Reggio Calabria
Via Graziella, Località Feo di Vito
89122 Reggio Calabria, Italy
E-mail: ursino@unirc.it

Keywords: XML Schema, sub-schema similarities, interschema properties, semantically heterogeneous information sources

Received: November 7, 2007

This paper presents a semi-automatic approach to deriving sub-schema similarities from semantically heterogeneous XML Schemas. The proposed approach is specific for XML, almost automatic and light. It consists of two phases: the first phase selects the most promising pairs of sub-schemas, the second one examines them and returns only those which are similar. This paper describes the approach in all details and illustrates a large variety of experiments to test its performance. Furthermore, it presents a comparison between this approach and others which have already been proposed in the literature.

Povzetek: Opisano je iskanje podobnosti podshem v XML.

1 Introduction

The derivation of semantic matchings among concepts of different sources (known also as "schema matching" activity in the literature) has become a challenging issue in the field of Information Systems; as a matter of fact, their knowledge allows the improvement of source interoperability and plays a key role in various applications, such as data source integration, ontology matching, e-commerce, semantic query processing, data warehousing, source clustering and cataloguing, and so on.

In the past, most of the proposed approaches to deriving matchings were manual (1); today, due to the enormous number of available sources, it is widely recognized the need of semi-automatic techniques (4; 5; 11; 13; 20). Moreover, most of the matching derivation theory has been *developed to operate on classical, structured databases*, and the main focus has been on deriving similarities and dissimilarities between *single classes of objects* (e.g., two entities, two relationships, an entity and a relationship, and so on).

However, in the last few years, the Web is becoming the reference infrastructure for many applications conceived to handle the interoperability among different partners. Web sources are quite different from classical databases, since they are semi-structured. In order to make Web activities easier, World Wide Web Consortium (W3C) proposed XML (eXtensible Markup Language) as a new standard information exchange language, that aims at unifying representation capabilities, typical of HTML, and data management features, typical of classical DBMSs. In order to improve the capability of representing and handling the intensional component of XML sources, W3C proposed to associate XML Schemas with XML documents. An XML Schema can be considered as a catalogue of the information that can be found in the corresponding XML documents.

The exploitation of the semi-structured paradigm in general, and of XML in particular, makes it evident the necessity to develop new approaches to deriving semantic matchings; these approaches are quite different from the traditional ones. As a matter of fact, in semi-structured information sources, a concept is not generally expressed by a single class of objects but it is represented by a group of them; as an example, in XML, concepts are expressed by elements which can be, in their turn, described by sub-elements.

In such a situation, the emphasis shifts away from the extraction of *semantic correspondences between object classes* to the derivation of *semantic correspondences between portions of information sources* (i.e., sub-sources). We call *sub-schema* a self contained sub-source such that a concept represented therein is connected to other concepts of the sub-source by means of at least one relationship. We call *sub-schema similarity* a similarity between two sub-schemas belonging to different sources.

Due to its intrinsic complexity, the sub-schema similarity extraction problem goes beyond the classic problem of deriving semantic correspondences among single concepts belonging to different schemas and allows more complex relationships to be handled.

This paper aims at providing a contribution in this setting; in fact, it presents an approach to extracting similarities between XML sub-schemas. Our approach is characterized by the following features:

- *It is almost automatic*, in that it requires the user intervention only for validating obtained results; the present overwhelming amount of available information sources on the Web makes such a feature particularly relevant.

- *It has been specifically conceived for operating on XML Schemas*; in fact, the framework underlying our approach has been defined for directly covering the XML specificities (see, below, Section 2.1). With regard to this choice, we point out that XML source interoperability will play a more and more relevant role in the future; as a consequence, it will be more and more common the necessity to handle the interoperability of a group of information sources that are all XML-based. In this scenario, the possibility to exploit a technique specifically tailored for XML sources appears extremely useful; our approach has been conceived exactly for providing such a chance.

- *It is light*, since it does not exploit any threshold or weight; as a consequence, it does not need any tuning activity; in spite of this, obtained results are satisfactory, as pointed out in Section 3.

Our approach assumes the existence of an *Interschema Property Dictionary* (IPD), i.e., a catalogue storing relationships between *single concepts* represented in the involved XML Schemas. Specifically, it assumes that IPD stores the following properties: *(i) Synonymies*: a synonymy indicates that two concepts have the same meaning; *(ii) Hyponymies/Hypernymies*: given two concepts c_1 and c_2, c_1 is a hyponym of c_2 (which is, in its turn, a hypernym of c_1) if c_1 has a more specific meaning than c_2; *(iii) Overlappings*: an overlapping exists between two concepts if they are neither synonyms nor one a hyponym of the other but represent, to some extent, the same reality. In the literature, many approaches to deriving synonymies, hyponymies and overlappings have been proposed (see, for example, (4; 5; 13; 19)); any of them could be exploited for constructing IPD. However, in the prototype implementing our approach, we have adopted the technique described in (6) for deriving properties to be stored in IPD.

It is worth pointing out that the exploitation of IPD does not introduce scalability problems; in fact, even if IPD must be computed for each pair of XML Schemas into consideration, the worst case time complexity of its derivation is smaller than that associated with the extraction of sub-schema similarities (see (6), Theorems 2.2, 2.4 and 2.5 and Section 3.9).

Given an XML Schema, the number of possible sub-schemas that could be derived from it is extremely high; in certain circumstances it could be even exponential against the number of elements and attributes of the Schema. In order to avoid huge numbers of pairs of sub-schemas to be handled, we propose a heuristic technique for singling out only the most promising ones. A pair of sub-schemas is considered "promising" if the sub-schemas at hand include a large number of pairs of concepts whose similarity has been already stated (i.e., a large number of pairs of concepts for which a synonymy, a hyponymy or an overlapping has been already derived). In this way it is probable that the overall similarity of the promising pair of sub-schemas will be high.

After the most promising pairs of sub-schemas have been selected, they must be examined for detecting those ones that are really similar. The similarity degree associated with each pair of sub-schemas is determined by applying some matching functions defined on suitable bipartite graphs, constructed from the components of the sub-schemas into consideration (see below). The idea underlying the adoption of graph matching algorithms as the core step for "measuring" the similarity of two sub-schemas is motivated by the following reasoning: two sub-schemas can be detected to be similar only if it is possible to state that there exists a form of similarity (e.g., a synonymy, a hyponymy or an overlapping) for many of their elements. The graph matching algorithm is, thus, used to carry out such a verification.

It is worth pointing out that, in the past, we have presented in this journal an approach to extracting interschema properties from XML Schemas (see (6)); this approach was explicitly conceived to extract synonymies, homonymies, hyponymies and overlappings. The approach proposed in this paper derives another kind of interschema properties (i.e., sub-schema similarities), particularly important in the current Internet era, by following the same guidelines followed in (6). As a consequence, the two papers, in the whole, define a new complete approach for uniformly extracting a large variety of interschema properties. In our opinion, this is a particularly important issue; in fact, as we pointed out also in (6), the capability of uniformly deriving distinct properties appears a crucial feature for a new interschema property derivation approach. As a matter of fact, different strategies for extracting distinct interschema properties could lead to different interpretations of the same reality, and this is a situation that should be avoided.

The outline of the paper is as follows: Section 2 provides a detailed illustration of our approach. Section 3 is devoted to present the results of several tests we have carried out for verifying its performance. In Section 4 we compare it with several approaches previously proposed in the literature. Finally, in Section 5, we draw our conclusions.

2 Approach description

2.1 Preliminary concepts

In this section we introduce some preliminary concepts that will be largely exploited in this paper. Preliminarily, we point out that XML Schemas are usually designed by adopting one of the following three classical techniques: *(i)* the "Russian doll" design, in which the schema structure mirrors the document structure; in particular, it defines one single global element whereas all other elements are local; *(ii)* the "Salami slice" approach, in which, contrarily to the Russian doll, all elements declarations are global; *(iii)* the "Venetian blind" technique, which defines one single global element, as the Russian doll design, but exploits named complex types and element groups instead of element declarations.

Since simple rules have been defined to switch among these three representations, in the following, for the sake of simplicity, we assume that XML Schemas have been designed with the "Salami slice" approach.

First of all we introduce the concept of *x-component*; it denotes an element or an attribute of a Schema S. Given two x-components x_S and x_T of an XML Schema S:

- x_S is defined $veryclose$ to x_T if and only if: *(i)* $x_T = x_S$, or *(ii)* x_T is an attribute of x_S, or *(iii)* x_T is a simple sub-element of x_S.
- x_S is defined $close$ to x_T if and only if x_T is a complex sub-element of x_S.
- x_S is defined $near$ to x_T if and only if x_S is either $veryclose$ or $close$ to x_T.
- x_T is defined $reachable$ from x_S if and only if there exists a sequence of k *distinct* x-components $x_1, x_2, \ldots, x_k$ such that $x_S = x_1$, x_1 is $near$ to x_2, x_2 is $near$ to x_3, ..., x_{k-1} is $near$ to x_k, $x_k = x_T$.

We now introduce the concept of *Connection Cost* $CC(x_S, x_T)$ from an x-component x_S to an x-component x_T. Specifically, *(i)* $CC(x_S, x_T) = 0$ if x_S is $veryclose$ to x_T; *(ii)* $CC(x_S, x_T) = 1$ if x_S is $close$ to x_T; *(iii)* $CC(x_S, x_T) = \mathcal{C}_{ST}$ if x_T is $reachable$ from x_S and x_S is not $near$ to x_T; $CC(x_S, x_T) = +\infty$ if x_T is not $reachable$ from x_S. Here $\mathcal{C}_{ST} = min_{x_A}\ (CC(x_S, x_A) + CC(x_A, x_T))$ for each x_A such that $reachable(x_S, x_A) = reachable(x_A, x_T) = true$.

We are now able to introduce the concept of neighborhood of an x-component, that plays a key role in our approach.

Definition 2.1. *Let S be an XML Schema and let x_S be an x-component of S. The d^{th}* neighborhood *of x_S is defined as:*

$$nbh(x_S, d) = \{x_T |\ x_T \textit{ is an x-component of } S,\ CC(x_S, x_T) \leq d\} \qquad \square$$

We call *significant neighborhoods* of x_S all neighborhoods $nbh(x_S, d)$ such that $nbh(x_S, d) \neq nbh(x_S, d-1)$.

As far as the previous concepts are concerned, the following propositions and the following theorem can be introduced; the interested reader can find the corresponding proofs in the Appendix available at the address `http://www.ing.unirc.it/ursino/informatica/Appendix.pdf`.

Proposition 2.1. Let S be an XML Schema; let x_S and x_T be two x-components of S; let m be the number of complex elements of S. If $CC(x_S, x_T) \neq +\infty$, then $CC(x_S, x_T) < m$. □

Proposition 2.2. Let S be an XML Schema; let x_S be an x-component of S; let m be the number of complex elements of S; then $nbh(x_S, d) = nbh(x_S, m-1)$ for each d such that $d \geq m$. □

Theorem 2.1. Let S be an XML Schema; let n be the number of x-components of S. The worst case time complexity for constructing all neighborhoods of all x-components of S is $O(n^3)$. □

Theorem 2.1 is particularly important since it guarantees that our approach is polynomial (see, below, Theorems 2.4 and 2.5). It could appear that a polynomial complexity to the degree of three for neighborhood derivation causes scalability problems for the whole approach. Actually, this is not the case. In fact, in an XML source exploited as a database, the intensional component (i.e., the schema-level information, corresponding, in our application context, to XML Schemas) is generally much smaller than the extensional one (i.e., the instance-level information, corresponding, in our application context, to XML documents); as a consequence, the number of involved x-components (i.e., n) is generally very small. Moreover, the derivation of the neighborhoods of a Schema S must be carried out once and for all when S is examined for the first time; derived neighborhoods can be, then, exploited each time a sub-schema similarity extraction task involving S is performed. Only a change in the intensional component of S requires to update the corresponding neighborhoods; such a task, however, is uncommon and, in any case, it does not imply to re-compute, but simply to incrementally update, them.

A more detailed analysis concerning the scalability of our approach can be found in Section 3.9.

2.1.1 A case example

Consider the XML Schema S_1, shown in Figure 1, representing a University. Here, *professor* is $veryclose$ to *identifier* because *identifier* is an attribute of *professor*; analogously, *university* is $close$ to *professor* because *professor* is a complex sub-element of *university*; as a consequence, *university* is $near$ to *professor* and *professor* is $near$ to *identifier*; finally, *identifier* is *reachable* from *university* because *university* is $near$ to *professor* and *professor* is $near$ to *identifier*. As for neighborhoods, we have that:

> $nbh(university, 1)$ = *{university, professor, phdstudent, paper, course, student, identifier, name, cultural_area, papers, advisor, thesis, research_interests, authors, type, volumes, pages, argument, duration, attended_by, taught_by, program, students, enrollment_year, attends}*

For instance, *professor* belongs to *nbh(university, 1)* because *CC(university, professor) = 1*. All the other neighborhoods can be determined analogously.

2.2 Selection of the most promising pairs of sub-schemas

2.2.1 Overview

The first problem our approach must face is the extremely high number of possible sub-schemas that could be derived from an XML Schema S; in fact, this number might be exponential against the number of x-components of S.

In order to avoid huge numbers of pairs of sub-schemas to be examined, we have designed a heuristic technique for singling out only the most promising ones. This technique receives two XML Schemas S_1 and S_2 and an Interschema Property Dictionary IPD, storing synonymies, hyponymies and overlappings holding between complex elements of S_1 and S_2. The most promising pairs of sub-schemas are derived as follows: for each pair $\langle x_{1_j}, x_{2_k}\rangle$ belonging to IPD, such that $x_{1_j} \in S_1$ and $x_{2_k} \in S_2$, x_{1_j} and x_{2_k} are taken as the "seeds" for the construction of promising pairs of sub-schemas.

Specifically, our technique:

- considers the pairs $\langle nbh(x_{1_j}, \delta), nbh(x_{2_k}, \gamma)\rangle$, such that $nbh(x_{1_j}, \delta)$, (resp., $nbh(x_{2_k}, \gamma)$) is a *significant neighborhood* (see Section 2.1) of x_{1_j} (resp., x_{2_k});
- derives a pair of sub-schemas $\langle prosub_{1j_\delta}, prosub_{2k_\gamma}\rangle$, from each pair $\langle nbh(x_{1_j}, \delta), nbh(x_{2_k}, \gamma)\rangle$, such that $prosub_{1j_\delta}$ (resp., $prosub_{2k_\gamma}$) is obtained from $nbh(x_{1_j}, \delta)$ (resp., $nbh(x_{2_k}, \gamma)$) by removing from it those portions that are dissimilar with $nbh(x_{2_k}, \gamma)$ (resp., $nbh(x_{1_j}, \delta)$), i.e., those x-components not involved in semantic relationships with x-components of $nbh(x_{2_k}, \gamma)$ (resp., $nbh(x_{1_j}, \delta)$) - see below for more details.

2.2.2 Technical Details

In this section we formalize our technique for selecting the most promising pairs of sub-schemas. Specifically, given two XML Schemas S_1 and S_2, the set SPS of the most promising pairs of sub-schemas associated with them is obtained by calling a suitable function Φ as follows:

$$SPS = \Phi\left(S_1, S_2, IPD\right)$$

For each tuple $\langle x_{1_j}, x_{2_k}\rangle \in IPD$, Φ invokes a function Ψ for deriving the set of the most promising pairs of sub-schemas having x_{1_j} and x_{2_k} as their seeds. The formal definition of Φ is:

$$\Phi\left(S_1, S_2, IPD\right) = \bigcup_{\langle x_{1_j}, x_{2_k}\rangle \in IPD} \Psi\left(S_1, S_2, x_{1_j}, x_{2_k}, IPD\right)$$

The function Ψ receives two XML Schemas S_1 and S_2, two complex elements $x_{1_j} \in S_1$ and $x_{2_k} \in S_2$ and an Interschema Property Dictionary IPD; for each pair of significant neighborhoods $nbh(x_{1_j}, \delta)$ and $nbh(x_{2_k}, \gamma)$, Ψ calls a function ξ, which extracts the most promising pair of sub-schemas $\langle prosub_{1j_\delta}, prosub_{2k_\gamma}\rangle$ associated with it. Ψ can be defined as follows:

$$\begin{gathered}\Psi\left(S_1, S_2, x_{1_j}, x_{2_k}, IPD\right) = \\ \bigcup_{\substack{0 \le \delta < \mu(S_1) \\ 0 \le \gamma < \mu(S_2)}} \xi\left(S_1, S_2, nbh(x_{1_j}, \delta), nbh(x_{2_k}, \gamma),\right. \\ \left.\nu\left(IPD, nbh(x_{1_j}, \delta), nbh(x_{2_k}, \gamma)\right)\right)\end{gathered}$$

Here, the function μ receives an XML Schema and returns the number of its complex elements. The function ν receives an Interschema Property Dictionary IPD and two neighborhoods $nbh(x_{1_j}, \delta)$ and $nbh(x_{2_k}, \gamma)$; it returns the set $IPD_{\delta\gamma} \subseteq IPD$ of interschema properties involving only pairs of x-components belonging to both $nbh(x_{1_j}, \delta)$ and $nbh(x_{2_k}, \gamma)$.

The function ξ receives two XML Schemas S_1 and S_2, two neighborhoods $nbh(x_{1_j}, \delta)$ and $nbh(x_{2_k}, \gamma)$ and the set $IPD_{\delta\gamma}$, as constructed by ν; in order to extract the most promising pair of sub-schemas $\langle prosub_{1j_\delta}, prosub_{2k_\gamma}\rangle$, associated with $nbh(x_{1_j}, \delta)$ and $nbh(x_{2_k}, \gamma)$, ξ activates functions ζ, θ and π for pruning $nbh(x_{1_j}, \delta)$ and $nbh(x_{2_k}, \gamma)$ in such a way as to eliminate the most dissimilar portions. ξ can be formalized as follows:

$$\begin{gathered}\xi\left(S_1, S_2, nbh(x_{1_j}, \delta), nbh(x_{2_k}, \gamma), IPD_{\delta\gamma}\right) = \\ \langle \zeta(\theta(nbh(x_{1_j}, \delta), \pi(S_1, IPD_{\delta\gamma})), S_1), \\ \zeta(\theta(nbh(x_{2_k}, \gamma), \pi(S_2, IPD_{\delta\gamma})), S_2)\rangle\end{gathered}$$

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
            <xs:attribute name="identifier" type="xs:ID"/>
            <xs:attribute name="name" type="xs:string"/>
            <xs:attribute name="cultural_area" type="xs:string"/>
            <xs:attribute name="papers" type="xs:IDREFS"/>
            <xs:attribute name="advisor" type="xs:IDREF"/>
            <xs:attribute name="thesis" type="xs:string"/>
            <xs:attribute name="research_interests" type="xs:string"/>
            <xs:attribute name="authors" type="xs:IDREFS"/>
            <xs:attribute name="type" type="xs:string"/>
            <xs:attribute name="volumes" type="xs:integer"/>
            <xs:attribute name="pages" type="xs:integer"/>
            <xs:attribute name="argument" type="xs:string"/>
            <xs:attribute name="duration" type="xs:duration"/>
            <xs:attribute name="attended_by" type="xs:IDREFS"/>
            <xs:attribute name="taught_by" type="xs:IDREFS"/>
            <xs:attribute name="program" type="xs:string"/>
            <xs:attribute name="students" type="xs:IDREFS"/>
            <xs:attribute name="enrollment_year" type="xs:date"/>
            <xs:attribute name="attends" type="xs:IDREFS"/>
    <xs:element name="professor">
        <xs:complexType>
            <xs:attribute ref="identifier"/>
            <xs:attribute ref="name"/>
            <xs:attribute ref="cultural_area"/>
            <xs:attribute ref="papers"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="phd-student">
        <xs:complexType>
            <xs:attribute ref="identifier"/>
            <xs:attribute ref="advisor"/>
            <xs:attribute ref="thesis"/>
            <xs:attribute ref="research_interests"/>
            <xs:attribute ref="papers"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="paper">
        <xs:complexType>
            <xs:attribute ref="identifier"/>
            <xs:attribute ref="authors"/>
            <xs:attribute ref="type"/>
            <xs:attribute ref="volumes"/>
            <xs:attribute ref="pages"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="course">
        <xs:complexType>
            <xs:attribute ref="identifier"/>
            <xs:attribute ref="name"/>
            <xs:attribute ref="argument"/>
            <xs:attribute ref="duration"/>
            <xs:attribute ref="attended_by"/>
            <xs:attribute ref="taught_by"/>
            <xs:attribute ref="program"/>
            <xs:attribute ref="students"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="student">
        <xs:complexType>
            <xs:attribute ref="identifier"/>
            <xs:attribute ref="name"/>
            <xs:attribute ref="enrollment_year"/>
            <xs:attribute ref="attends"/>
        </xs:complexType>
    </xs:element>
    <!-- root -->
    <xs:element name="university">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="professor" maxOccurs="unbounded"/>
                <xs:element ref="phd-student" maxOccurs="unbounded"/>
                <xs:element ref="paper" maxOccurs="unbounded"/>
                <xs:element ref="course" maxOccurs="unbounded"/>
                <xs:element ref="student" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

Figure 1: The XML Schema S_1

Here, the function π receives an XML Schema S_h, $h \in \{1, 2\}$, and the set $IPD_{\delta\gamma}$, computed by ν; it returns the set $AtLeastOne$ of the complex elements belonging to S_h and involved in at least one property of $IPD_{\delta\gamma}$.

The function θ receives a neighborhood $nbh(x_S, d)$, associated with an XML Schema S, and the set $AtLeastOne$ as computed by the function π. It constructs a set of x-components $XSet_{S_d} \subseteq nbh(x_S, d)$ by removing from $nbh(x_S, d)$ each complex element x_R (along with all its sub-elements and attributes) that satisfies both the following conditions: *(i)* $x_R \notin AtLeastOne$; *(ii)* for each complex element x_{R_i} such that $reachable(x_R, x_{R_i}) = true$ and $x_{R_i} \in nbh(x_S, d)$, $x_{R_i} \notin AtLeastOne$.

In other words a complex element x_R, belonging to $nbh(x_S, d)$, is not inserted in $XSet_{S_d}$ if both it and all complex elements in $nbh(x_S, d)$ reachable from it are not involved in any interschema property stored in $IPD_{\delta\gamma}$. Note that the two conditions above guarantee that if x_R is not inserted in $XSet_{S_d}$, then no x-components reachable from it are inserted therein. In fact, if the two conditions above are valid for x_R, then they must be also valid for all x-components reachable from it.

The function ζ receives the set of x-components $XSet_{S_d}$ returned by θ and constructs a sub-schema $prosub_{S_d}$ taking into account the initial structure of S. Specifically, $prosub_{S_d}$ is constructed from $XSet_{S_d}$ in such a way that the following two conditions hold: *(i)* it must contain all, and only, the x-components of $XSet_{S_d}$; *(ii)* all x-components of $XSet_{S_d}$ must preserve, in $prosub_{S_d}$, the same hierarchical organization they have in S[1].

The next theorems state the worst case time complexity for computing all promising pairs of sub-schemas, as well as an upper bound to the number of promising pairs of sub-schemas returned by the function Φ. Their proofs can be found in the Appendix available at the address `http://www.ing.unirc.it/ursino/informatica/Appendix.pdf`.

Theorem 2.2. Let S_1 and S_2 be two XML Schemas. Let IPD be the Interschema Property Dictionary associated with S_1 and S_2; let m be the maximum between the number of complex elements of S_1 and S_2; let n be the maximum between the number of x-components of S_1 and S_2. The worst case time complexity for computing, by means of the function Φ, the set SPS of the most promising pairs of sub-schemas associated with S_1 and S_2 is $max\{O(m^7), O(m^4 \times n^2)\}$. $\square$

Theorem 2.3. Let S_1 and S_2 be two XML Schemas; let IPD be the corresponding Interschema Property Dictionary; let m be the maximum between the number of complex elements of S_1 and S_2. The maximum cardinality of SPS is $O(m^4)$. $\square$

As for these two theorems, all considerations about the value of n, that we have drawn after Theorem 2.1, are still valid. Moreover, since in an XML document the number of attributes and simple elements is generally much greater than the number of complex elements, the value of m is even much smaller than that of n.

[1]Note that the sub-schema $prosub_{S_d}$ obtained by the function ζ is a well-formed and self contained XML Schema because of the function θ. In fact, this function constructs $XSet_{S_d}$ in such a way that, if an x-component is not inserted in $XSet_{S_d}$, then no x-components reachable from it are inserted therein.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:attribute name="ID" type="xs:ID"/>
    <xs:attribute name="first_name" type="xs:string"/>
    <xs:attribute name="last_name" type="xs:string"/>
    <xs:attribute name="type" type="xs:string"/>
    <xs:attribute name="roles" type="xs:string"/>
    <xs:attribute name="research" type="xs:string"/>
    <xs:attribute name="argument" type="xs:string"/>
    <xs:attribute name="budget" type="xs:string"/>
    <xs:attribute name="funds" type="xs:string"/>
    <xs:attribute name="responsibles" type="xs:IDREFS"/>
    <xs:attribute name="termination" type="xs:date"/>
    <xs:attribute name="authors" type="xs:IDREFS"/>
    <xs:attribute name="title" type="xs:string"/>
    <xs:attribute name="volume" type="xs:integer"/>
    <xs:attribute name="pages" type="xs:integer"/>
    <xs:attribute name="year" type="xs:date"/>
    <xs:attribute name="booktitle" type="xs:string"/>
    <xs:attribute name="address" type="xs:string"/>
    <xs:attribute name="publisher" type="xs:string"/>
    <xs:attribute name="chief" type="xs:IDREF"/>
    <xs:attribute name="people" type="xs:IDREF"/>
    <xs:attribute name="projects" type="xs:IDREFS"/>
    <xs:attribute name="locations" type="xs:string"/>
    <xs:attribute name="labs" type="xs:string"/>
    <xs:element name="article">
        <xs:complexType>
            <xs:choice>
                <xs:element ref="journal"/>
                <xs:element ref="conference"/>
            </xs:choice>
        </xs:complexType>
    </xs:element>
    <xs:element name="researcher">
        <xs:complexType>
            <xs:attribute ref="ID"/>
            <xs:attribute ref="first_name"/>
            <xs:attribute ref="last_name"/>
            <xs:attribute ref="type"/>
            <xs:attribute ref="roles"/>
            <xs:attribute ref="research"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="project">
        <xs:complexType>
            <xs:attribute ref="ID"/>
            <xs:attribute ref="argument"/>
            <xs:attribute ref="budget"/>
            <xs:attribute ref="funds"/>
            <xs:attribute ref="responsibles"/>
            <xs:attribute ref="termination"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="journal">
        <xs:complexType>
            <xs:attribute ref="ID"/>
            <xs:attribute ref="authors"/>
            <xs:attribute ref="title"/>
            <xs:attribute ref="volume"/>
            <xs:attribute ref="pages"/>
            <xs:attribute ref="year"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="conference">
        <xs:complexType>
            <xs:attribute ref="ID"/>
            <xs:attribute ref="authors"/>
            <xs:attribute ref="title"/>
            <xs:attribute ref="booktitle"/>
            <xs:attribute ref="address"/>
            <xs:attribute ref="year"/>
            <xs:attribute ref="pages"/>
            <xs:attribute ref="publisher"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="department">
        <xs:complexType>
            <xs:attribute ref="ID"/>
            <xs:attribute ref="chief"/>
            <xs:attribute ref="people"/>
            <xs:attribute ref="projects"/>
            <xs:attribute ref="locations"/>
            <xs:attribute ref="labs"/>
        </xs:complexType>
    </xs:element>
    <!-- root -->
    <xs:element name="university">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="article" maxOccurs="unbounded"/>
                <xs:element ref="project" maxOccurs="unbounded"/>
                <xs:element ref="researcher" maxOccurs="unbounded"/>
                <xs:element ref="department" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

Figure 2: The XML Schema S_2

x-component of S_1	*x-component of S_2*	*interschema property typology*
university	university	synonymy
professor	researcher	overlapping
phd-student	researcher	overlapping
paper	article	synonymy
paper	journal	hyponymy
paper	conference	hyponymy

Table 1: The Interschema Property Dictionary IPD associated with S_1 and S_2

2.2.3 A case example (cnt'd)

Consider the XML Schemas S_1 and S_2, associated with a University and illustrated in Figures 1 and 2. Consider the corresponding Interschema Property Dictionary IPD shown in Table 1 [2].

In order to construct SPS, first the function Φ is activated. For each tuple of IPD, Φ calls the function Ψ. In order to show the behaviour of Ψ, we consider its application to the pair of complex elements $\langle university_{[S_1]}$, $university_{[S_2]}\rangle$ [3].

For each pair $\langle nbh(university_{[S_1]}, \delta)$, $nbh(university_{[S_2]}, \gamma)\rangle$ of significant neighborhoods, Ψ activates the function ξ. In order to illustrate the behaviour of ξ, we consider its application to $nbh(university_{[S_1]}, 1)$ and $nbh(university_{[S_2]}, 2)$; $nbh(university_{[S_1]}, 1)$ has been shown in the previous section; $nbh(university_{[S_2]}, 2)$ is as follows:

$nbh(university_{[S_2]}, 2)$ = *{university, article, project, researcher, department, journal, conference, ID, first_name, last_name, type, roles, research, argument, budget, funds, responsibles, termination, authors, title, volume, pages, year, booktitle, address, publisher, chief, people, projects, locations, labs}*

For this pair of neighborhoods the set $IPD_{\delta\gamma}$, returned by the function ν, is equal to IPD. ξ activates θ for pruning $nbh(university_{[S_1]}, 1)$ and $nbh(university_{[S_2]}, 2)$ in such a way as to remove the most dissimilar portions. As an example, the complex element $student_{[S_1]}$ and all its attributes are pruned from $nbh(university_{[S_1]}, 1)$ because: *(i)* $student_{[S_1]}$ is not involved in any interschema property of $IPD_{\delta\gamma}$; *(ii)* there does not exist any complex element x_{R_i} such that $reachable(student_{[S_1]}, x_{R_i}) = true$, $x_{R_i} \in nbh(university_{[S_1]}, 1)$, and x_{R_i} is involved in some interschema property of $IPD_{\delta\gamma}$.

The final sets of x-components returned by the function θ, when applied on $nbh(university_{[S_1]}, 1)$ and $nbh(university_{[S_2]}, 2)$, are:

{university, professor, phd-student, paper, identifier, name, cultural_area, papers, advisor, thesis, research_interests, authors, type, volumes, pages}

[2]As previously pointed out, we have chosen to construct IPD by applying the approaches described in (6); however, any other approach proposed in the literature for deriving synonymies, hyponymies and overlappings among elements of different XML Schemas could be exploited.

[3]Here and in the following, whenever necessary, we use the notation $x_{[S]}$ for indicating the x-component x of an XML Schema S.

{university, article, researcher, journal, conference, ID, first_name, last_name, type, roles, research, authors, title, organization, year, booktitle, address, pages, publisher}

After this, ξ activates ζ that constructs the promising sub-schemas corresponding to $nbh(university_{[S_1]}, 1)$ and $nbh(university_{[S_2]}, 2)$.

The final promising pair of sub-schemas corresponding to $nbh(university_{[S_1]}, 1)$ and $nbh(university_{[S_2]}, 2)$ returned by ξ is illustrated in Figure 3. All the other promising pairs of sub-schemas can be determined analogously.

2.3 Derivation of sub-schema similarities

In the previous section we have seen how the most promising pairs of sub-schemas can be determined. In this section we illustrate how these pairs can be analyzed in order to derive sub-schema similarities. Before describing this task in detail some preliminary considerations are needed.

Applications possibly exploiting sub-schema similarities (and, more in general, interschema properties) are extremely heterogeneous. Some of them (i.e., the most critical ones) require, for each pair of involved sub-schema similarities, a high level of trustworthiness; in other words they require the correspondences between the elements belonging to the involved sub-schemas to be precisely and unambiguously determined. In order to achieve this guarantee, it is necessary to pay the price of filtering out the weakest sub-schema similarities, i.e., those involving sub-schemas whose elements might have a form of similarity different from synonymy. In fact, synonymy represents the strongest form of similarity; it is the only one capable of guaranteeing that the similar elements belonging to the involved sub-schemas can be precisely and unambiguously mapped each other.

By contrast, other (possibly non critical) applications could prefer to have a more complete picture of existing sub-schema similarities and, therefore, could choose to consider also the weakest ones. As previously pointed out, while a strong similarity requires most of the elements of the corresponding schemas to be related by a synonymy, a weak similarity can accept other kinds of similarity property, e.g., hyponymies and overlappings; it can also exist between two schemas characterized by quite a different structure. As a consequence of these reasonings, even if weak similarities cannot be considered in critical applications, in non-critical scenarios they can provide a richer vision of the reality.

Clearly, the two exigencies outlined above (i.e., strength and breadth of discovered similarities) are divergent and, consequently, it appears extremely difficult to satisfy both of them simultaneously.

In order to address this issue, our technique allows the derivation of two levels of sub-schema similarities, namely strong similarities, that guarantee a strong correspondence between the x-components of similar sub-schemas, and weak similarities, that allow the existence of less characterizing semantic relationships between the corresponding x-components. Specifically, *strong sub-schema similarities* are derived by taking only synonymies into account; *weak sub-schema similarities* cannot be derived with the only support of synonymies but need also the contribution of hyponymies and overlappings.

Our technique for deriving sub-schema similarities between two XML Schemas S_1 and S_2 receives the set SPS of the most promising pairs of sub-schemas and the Interschema Property Dictionary IPD associated with S_1 and S_2 and selects two sets of pairs of similar sub-schemas, namely:

$$SSS_{strong} = \rho_{strong}(SPS, IPD)$$
$$SSS_{weak} = \rho_{weak}(SPS, IPD)$$

Here, the function ρ_{strong} derives the strong sub-schema similarities, whereas the function ρ_{weak} extracts the weak ones.

2.3.1 Derivation of strong similarities

ρ_{strong} operates by computing the objective function associated with a maximum weight matching defined on a suitable bipartite graph. Specifically, let $\langle prosub_{1j_\delta}, prosub_{2k_\gamma} \rangle \in SPS$ be a promising pair of sub-schemas; let $BG_{\delta\gamma} = \langle NSet, ESet \rangle$ be the bipartite graph associated with $prosub_{1j_\delta}$ and $prosub_{2k_\gamma}$. $NSet = PSet \cup QSet$ is the set of nodes of $BG_{\delta\gamma}$; there is a node in $PSet$ (resp., $QSet$) for each complex element of $prosub_{1j_\delta}$ (resp., $prosub_{2k_\gamma}$). $ESet$ is the set of edges of $BG_{\delta\gamma}$; in $ESet$ there exists an edge $\langle p, q \rangle$ between two nodes $p \in PSet$ and $q \in QSet$ if and only if, in IPD, there exists a synonymy between the element corresponding to p and that corresponding to q.

The maximum weight matching on $BG_{\delta\gamma}$ is the set $ESet^* \subseteq ESet$ such that, for each node $x \in NSet$, there exists at most one edge of $ESet^*$ incident onto x and $|ESet^*|$ is maximum (the interested reader is referred to (15) for details about the maximum weight matching problem). The objective function we associate with the maximum weight matching is $\chi_{BG} = \frac{2|ESet^*|}{|PSet|+|QSet|}$. Here $|ESet^*|$ represents the number of matches associated with $BG_{\delta\gamma}$, as well as the number of synonymies involving $prosub_{1j_\delta}$ and $prosub_{2k_\gamma}$. $2|ESet^*|$ indicates the number of matching nodes in $BG_{\delta\gamma}$, as well as the number of similar complex elements present in $prosub_{1j_\delta}$ and $prosub_{2k_\gamma}$. $|PSet| + |QSet|$ denotes the total number of nodes in $BG_{\delta\gamma}$ as well as the total number of complex elements associated with $prosub_{1j_\delta}$ and $prosub_{2k_\gamma}$. Finally, χ_{BG} represents the share of matching nodes in $BG_{\delta\gamma}$, as well as the share of similar complex elements present in $prosub_{1j_\delta}$ and $prosub_{2k_\gamma}$.

We assume that $prosub_{1j_\delta}$ and $prosub_{2k_\gamma}$ are similar if $\chi_{BG} > \frac{1}{2}$. Such an assumption derives from the consideration that two sets of objects can be considered similar if the number of similar elements is greater than the number

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:attribute name="identifier" type="xs:ID"/>
    <xs:attribute name="name" type="xs:string"/>
    <xs:attribute name="cultural_area" type="xs:string"/>
    <xs:attribute name="papers" type="xs:IDREFS"/>
    <xs:attribute name="advisor" type="xs:IDREF"/>
    <xs:attribute name="thesis" type="xs:string"/>
    <xs:attribute name="research_interests" type="xs:string"/>
    <xs:attribute name="authors" type="xs:IDREFS"/>
    <xs:attribute name="type" type="xs:string"/>
    <xs:attribute name="volumes" type="xs:integer"/>
    <xs:attribute name="pages" type="xs:integer"/>
    <xs:element name="professor">
        <xs:complexType>
            <xs:attribute ref="identifier"/>
            <xs:attribute ref="name"/>
            <xs:attribute ref="cultural_area"/>
            <xs:attribute ref="papers"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="phd-student">
        <xs:complexType>
            <xs:attribute ref="identifier"/>
            <xs:attribute ref="advisor"/>
            <xs:attribute ref="thesis"/>
            <xs:attribute ref="research_interests"/>
            <xs:attribute ref="papers"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="paper">
        <xs:complexType>
            <xs:attribute ref="identifier"/>
            <xs:attribute ref="authors"/>
            <xs:attribute ref="type"/>
            <xs:attribute ref="volumes"/>
            <xs:attribute ref="pages"/>
        </xs:complexType>
    </xs:element>
    <!-- root -->
    <xs:element name="university">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="professor" maxOccurs="unbounded"/>
                <xs:element ref="phd-student" maxOccurs="unbounded"/>
                <xs:element ref="paper" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:attribute name="ID" type="xs:ID"/>
    <xs:attribute name="first_name" type="xs:string"/>
    <xs:attribute name="last_name" type="xs:string"/>
    <xs:attribute name="type" type="xs:string"/>
    <xs:attribute name="roles" type="xs:string"/>
    <xs:attribute name="research" type="xs:string"/>
    <xs:attribute name="authors" type="xs:IDREFS"/>
    <xs:attribute name="title" type="xs:string"/>
    <xs:attribute name="organization" type="xs:string"/>
    <xs:attribute name="year" type="xs:date"/>
    <xs:attribute name="booktitle" type="xs:string"/>
    <xs:attribute name="address" type="xs:string"/>
    <xs:attribute name="pages" type="xs:integer"/>
    <xs:attribute name="publisher" type="xs:string"/>
    <xs:element name="article">
        <xs:complexType>
            <xs:choice>
                <xs:element ref="journal"/>
                <xs:element ref="conference"/>
            </xs:choice>
        </xs:complexType>
    </xs:element>
    <xs:element name="researcher">
        <xs:complexType>
            <xs:attribute ref="ID"/>
            <xs:attribute ref="first_name"/>
            <xs:attribute ref="last_name"/>
            <xs:attribute ref="type"/>
            <xs:attribute ref="roles"/>
            <xs:attribute ref="research"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="journal">
        <xs:complexType>
            <xs:attribute ref="ID"/>
            <xs:attribute ref="authors"/>
            <xs:attribute ref="title"/>
            <xs:attribute ref="organization"/>
            <xs:attribute ref="year"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="conference">
        <xs:complexType>
            <xs:attribute ref="ID"/>
            <xs:attribute ref="authors"/>
            <xs:attribute ref="title"/>
            <xs:attribute ref="booktitle"/>
            <xs:attribute ref="address"/>
            <xs:attribute ref="year"/>
            <xs:attribute ref="pages"/>
            <xs:attribute ref="publisher"/>
        </xs:complexType>
    </xs:element>
    <!-- root -->
    <xs:element name="university">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="article" maxOccurs="unbounded"/>
                <xs:element ref="researcher" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

Figure 3: The promising pair of sub-schemas associated with $nbh(university_{[S_1]}, 1)$ and $nbh(university_{[S_2]}, 2)$

of the dissimilar ones or, in other words, if the number of similar elements is greater than half of the total number of elements.

The following theorem states the worst case time complexity for computing all strong similarities. Its proof can be found in the Appendix at the address `http://www.ing.unirc.it/ursino/informatica/Appendix.pdf`.

Theorem 2.4. Let S_1 and S_2 be two XML Schemas; let IPD be the corresponding Interschema Property Dictionary; let m be the maximum between the number of complex elements of S_1 and S_2. The worst case time complexity for computing SSS_{strong} is $O(m^7)$. $\square$

With regard to this result, the same reasoning about the extremely small number of complex elements in an XML Schema, that we have presented after Theorems 2.2 and 2.3, is still valid.

2.3.2 Derivation of weak similarities

ρ_{weak} receives SPS and IPD and returns weak sub-schema similarities. We call them "weak" because, differently from ρ_{strong}, which takes only synonymies into account, ρ_{weak} considers also overlappings and hyponymies, that are weaker properties than synonymies in the representation of concept similarities.

When we introduce hyponymies and overlappings in the computation of sub-schema similarities we must consider that, often, more than one element of a schema could be hyponymous or overlapping with an element of the other schema.

A consequence of this reasoning is that, in order to derive weak sub-schema similarities, it is not suitable to apply maximum weight matching techniques; in fact, they would associate an element of a schema with at most one element of the other schema.

Taking into account the reasoning above, ρ_{weak} has been defined as follows. Let $\langle prosub_{1j_\delta}, prosub_{2k_\gamma}\rangle \in SPS$ be a promising pair of sub-schemas; let $BG'_{\delta\gamma} = \langle NSet', ESet'\rangle$ be a bipartite graph associated with $prosub_{1j_\delta}$ and $prosub_{2k_\gamma}$. Here, $NSet' = PSet' \cup QSet'$ is the set of nodes of $BG'_{\delta\gamma}$; there is a node in $PSet'$ (resp., $QSet'$) for each complex element of $prosub_{1j_\delta}$ (resp., $prosub_{2k_\gamma}$). $ESet'$ is the set of edges of $BG'_{\delta\gamma}$;

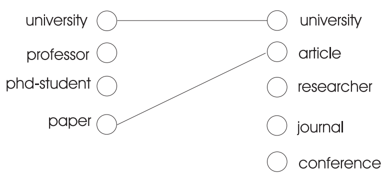


Figure 4: The bipartite graph $BG_{\delta\gamma}$ associated with the promising pair of sub-schemas illustrated in Figure 3

in $ESet'$ there exists an edge $\langle p, q\rangle$ between two nodes $p \in PSet'$ and $q \in QSet'$ if and only if, in IPD, a synonymy, a hyponymy or an overlapping holds between the element corresponding to p and that corresponding to q.

Let η_p and η_q be the sets of nodes of $PSet'$ and $QSet'$ involved in at least one interschema property; specifically, $\eta_p = \{p \in PSet'$ such that at least one edge of $BG'_{\delta\gamma}$ is incident onto it$\}$ and $\eta_q = \{q \in QSet'$ such that at least one edge of $BG'_{\delta\gamma}$ is incident onto it$\}$; we assume that $prosub_{1j_\delta}$ and $prosub_{2k_\gamma}$ are weakly similar if $\chi'_{BG'} = \frac{|\eta_p|+|\eta_q|}{|PSet'|+|QSet'|} > \frac{1}{2}$. Such an assumption indicates that two sub-schemas are weakly similar if at least half of their elements are someway related by an interschema property. The justification underlying such an assumption is analogous to that we have seen for strong similarities.

The following theorem states the worst case time complexity for computing all strong similarities. Its proof can be found in the Appendix at the address `http://www.ing.unirc.it/ursino/informatica/Appendix.pdf`.

Theorem 2.5. Let S_1 and S_2 be two XML Schemas; let IPD be the corresponding Interschema Property Dictionary; let m be the maximum between the number of complex elements of S_1 and S_2. The worst case time complexity for computing SSS_{weak} is $O(m^6)$. $\Box$

2.3.3 A case example (cnt'd)

Consider the XML Schemas illustrated in Figures 1 and 2 and the promising pair of sub-schemas derived in Section 2.2.3 and illustrated in Figure 3.

For this pair, $BG_{\delta\gamma}$ is shown in Figure 4; the objective function χ_{BG} computed on it is equal to $\frac{4}{9} < \frac{1}{2}$; as a consequence, we can conclude that the sub-schemas of the pair are not strongly similar.

For the same pair, $BG'_{\delta\gamma}$ is shown in Figure 5; the value of $\chi'_{BG'}$, computed by ρ_{weak}, is equal to $\frac{9}{9} > \frac{1}{2}$, which allows us to conclude that a weak similarity holds between the two sub-schemas into consideration.

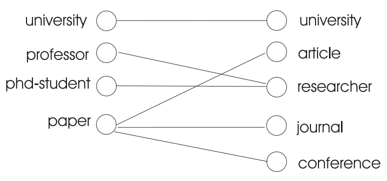


Figure 5: The bipartite graph $BG'_{\delta\gamma}$ associated with the promising pair of sub-schemas illustrated in Figure 3

3 Experimental results

3.1 Introduction

In this section we provide a detailed description of the experiments we have carried out in order to test the performance of our approach. Specifically, in Section 3.2 we describe the characteristics of the sources exploited in our experimental tests. The adopted accuracy measures are illustrated in Section 3.3; the results we have obtained by applying these measures are presented in Section 3.4. In Section 3.5 we compare the accuracy of our approach with that achieved by some other approaches previously proposed in the literature. Section 3.6 illustrates our study about the role of our heuristics for the extraction of promising pairs of sub-schemas in the improvement of the efficiency of our approach. An analysis about the improvements of our approach against manual "naive" approaches, based on identifying synonyms and expanding around them for constructing sub-schemas, is illustrated in Section 3.7. The robustness of our approach is estimated in Section 3.8. Finally, Section 3.9 is devoted to discuss our experimental results about its scalability.

3.2 Characteristics of the exploited sources

In our tests we have exploited a large variety of XML Schemas associated with disparate application contexts, such as Biomedical Data, Project Management, Property Register, Industrial Companies, Universities, Airlines, Scientific Publications and Biological Data.

Specifically, we have compared all pairs of schemas within a particular domain. Biomedical Schemas have been derived from various sites; one of these sites has been `http://www.biomediator.org`. Schemas concerning Project Management, Property Register and Industrial Companies have been derived from Italian Central Government Office sources and are shown at the address `http://www.mat.unical.it/terracina/tests.html`. Schemas concerning Universities have been downloaded from the Web address `http://anhai.cs.uiuc.edu/archive/domains/courses.html`. Schemas concerning Airlines have been found in (26). Schemas concerning Scientific Publications have been supplied by the authors of (17). Finally, Biological Schemas have been downloaded from the addresses `http://smi-web.stanford./`

`edu.projects/helix/pubs/ismb02/schemas/`, `http://www.cs.toronto.edu/db/clio/testSchemas.html` and `http://www.genome.ad.jp/kegg/genes.html`.

Examined sources were characterized by the following properties: *(i) Number of schemas*: we have considered 30 XML Schemas whose characteristics are reported in Table 2. *(ii) Maximum depth of schemas*: for each domain we have computed the maximum depth of the involved schemas; it is shown in the third column of Table 2. In our opinion, this parameter is particularly interesting for an approach specifically conceived for XML Schemas; in fact, the maximum depth is an indicator of the complexity of the sub-schemas that can be generated. *(iii) Size of schemas*: the size of the evaluated XML Schemas, i.e., the number of their elements and attributes, are shown in the fourth column of Table 2. The size of a test schema is relevant because it influences the quality of obtained results; in fact, as mentioned in (8), the bigger the input schemas are, the greater the search space for candidate pairs is, and the lower the quality of obtained results will be.

The *total number of pairs of schemas we have compared* for each domain is shown in the last column of Table 2.

3.3 Accuracy Measures exploited in our experimental tests

All accuracy measures adopted in our experimental tests have been computed according to the following general framework: *(i)* a set of experts has been asked to identify the sub-schema similarities existing among involved schemas; *(ii)* sub-schema similarities among the same schemas have been determined by running our algorithm; *(iii)* the sub-schema similarities provided by the experts and those returned by our algorithm have been compared and accuracy measures have been computed.

The number of experts that have been involved in manually solving the match tasks is as follows: 6 for Biomedical Data, 3 for Project Management, 3 for Property Register, 4 for Industrial Companies, 4 for Universities, 2 for Airlines, 2 for Scientific Publications and 7 for Biological Data.

Let A be the set of sub-schema similarities provided by the experts and let C be the set of sub-schema similarities returned by our approach; two basic accuracy measures are:

- *Precision* (hereafter *Pre*), that specifies the share of correct sub-schema similarities detected by the system among those it derived. It is defined as: $Pre = \frac{|A \cap C|}{|C|}$.
- *Recall* (hereafter *Rec*), that indicates the share of correct sub-schema similarities detected by the system among those the experts provided. It is defined as: $Rec = \frac{|A \cap C|}{|A|}$.

Precision and Recall are typical measures of Information Retrieval (see (29)). Both of them fall within the real interval $[0, 1]$; in the ideal case (i.e., when $A \equiv C$) they are both equal to 1.

Property Typology	*Average Precision*	*Average Recall*	*Average F-Measure*	*Average Overall*
weak similarities	0.89	0.77	0.83	0.67
strong similarities	0.92	0.72	0.81	0.66

Table 3: Accuracy measures of our approach for weak and strong similarities

However, neither Precision nor Recall alone can accurately measure the quality of an interschema property extraction algorithm; in order to improve the quality of results, it appears necessary the computation of a joint measure of them. Two very popular measures satisfying these requirements are:

- *F-Measure* (3; 29), that represents the harmonic mean between Precision and Recall. It is defined as: $F\text{-}Measure = 2 \cdot \frac{Pre \cdot Rec}{Pre + Rec}$.
- *Overall* (9; 22), that measures the post-match effort needed for adding false negatives and removing false positives from the set of similarities returned by the system to evaluate. It is defined as: $Overall = Rec \cdot (2 - \frac{1}{Pre})$.

F-Measure falls within the interval $[0, 1]$ whereas Overall ranges between $-\infty$ and 1; the higher they are, the better the accuracy of the tested approach will be.

3.4 Discussion of obtained results

As for the evaluation of Precision and Recall associated with our approach we considered particularly interesting to compute them by distinguishing weak and strong similarities. Before the experiments we expected that passing from weak to strong similarities would have caused an increase of the Precision and a decrease of the Recall of our approach. This intuition was motivated by considering that strong similarities are a subset of the weak ones, obtained from them by eliminating the most uncertain ones; this should cause the set of strong similarities to be more precise than the set of the weak ones. However, this filtering task could erroneously discard some valid similarities; for this reason the set of strong similarities could have a lower Recall w.r.t. the set of the weak ones.

In order to verify this intuition and, possibly, to quantify it, we have applied our approach on all pairs of XML Schemas belonging to the same application domain and we have computed Precision, Recall, F-Measure and Overall for each pair into consideration; after this, we have computed the average values of all obtained measures for weak and strong similarities; they are reported in Table 3. From the analysis of this table we can draw the following conclusions:

- As for weak similarities, *(i)* Precision is quite high, even if our approach returns some false positives; *(ii)* Recall is quite high, given the specificity of the interschema property typology we are studying in this

Application context	*Number of Schemas*	*Maximum depth of Schemas*	*Minimum, Average and Maximum Number of x-components*	*Minimum, Average and Maximum Number of complex elements*	*Total Number of Comparisons*
Biomedical Data	6	8	15 - 26 - 38	4 - 8 - 16	15
Project Management	3	4	37 - 40 - 42	6 - 7 - 8	3
Property Register	2	4	64 - 70 - 75	14 - 14 - 14	1
Industrial Companies	5	4	23 - 28 - 46	6 - 8 - 9	10
Universities	5	5	15 - 17 - 19	3 - 4 - 5	10
Airlines	2	4	12 - 13 - 13	4 - 4 - 4	1
Scientific Publications	2	6	17 - 18 - 18	8 - 9 -9	1
Biological Data	5	8	70 - 136 - 262	21 - 41 - 103	10

Table 2: Characteristics of the XML Schemas exploited for testing the performance of our approach

paper (see below); as a consequence, our approach returns most of the valid properties or, in other words, it returns a very small number of false negatives.

– If we consider the strong similarities, *(i)* the set of similarities returned by our approach contains a smaller number of false positives w.r.t. the previous case; specifically, Precision increases about 4%; *(ii)* Recall decreases of about 6% w.r.t. the previous case; in other words, a certain increase of false negatives can be observed.

All these experiments confirm our original intuition about the trend of Precision and Recall when passing from weak to strong similarities.

Note that in the weak similarity context a user is willing to accept false positives if this allows him to obtain a wide set of similarities. On the contrary, in the strong similarity context, a user is willing to receive an incomplete set of similarities by the system but he desires that proposed properties are (almost surely) correct. These observations fully agree with the trend of Precision and Recall registered in our tests.

As a final remark about this experiment, we observe that the not particularly high values of Recall are explained by considering that: *(i)* the possible number of sub-schema similarities might be exponential against the number of x-components of the corresponding XML Schemas; *(ii)* we have used a heuristics for selecting the most promising pairs of sub-schemas.

After this, we have computed the variation of our accuracy measures in presence of a variation of the dimension of the input schemas. Specifically, given two XML Schemas S_1 and S_2 such that $n_1 = |XCompSet(S_1)|$ and $n_2 = |XCompSet(S_2)|$, we have computed the average values of Precision, Recall, F-Measure and Overall for the extraction of weak and strong similarities for different values of the number of involved x-components $n_t = n_1 + n_2$.

The obtained results are shown in Figures 6, 7, 8 and 9. From their analysis it is possible to conclude that all our accuracy measures slightly decrease in presence of an increase of n_t. Such quite an intuitive result confirms observations and results presented in (8).

Finally, we have verified if the accuracy of our approach depends on the application domain which the test XML Schemas belong to. The value of each accuracy measure for a domain has been determined by computing the accuracy measure for all possible pairs of XML Schemas belonging to the domain and, then, by averaging these values. The results we have obtained are shown in Figures 10, 11, 12 and 13. From the analysis of these figures it is possible to conclude that the accuracy of our approach is quite independent of the application domain (the only, quite significant, differences can be found in the biological domain). As far as our experiments are concerned, we have obtained the best accuracy for the Airlines domain; here, Precision reaches its best value, i.e., 0.94, obtained for the derivation of strong similarities; Recall, F-Measure and Overall are maximum for the extraction of weak similarities and are 0.79, 0.84 and 0.71, respectively. The worst accuracy results have been obtained in the Biological domain; here, Precision is maximum for the strong similarity derivation and is 0.86; Recall, F-Measure and Overall reach their best values for the weak similarity extraction and are 0.70, 0.76 and 0.55, respectively.

3.5 Comparison of the accuracy of our approach with that achieved by some related approaches

In this section we report the results of some experimental tests aiming to compare the accuracy of our approach with that achieved by other approaches already presented in the literature.

Our experimental comparison has been inspired by the ideas and methodologies proposed in (8). The authors of (8) considered the following systems: *Autoplex* (2), *Automatch* (3), *COMA* (9), *Cupid* (20), LSD (10), GLUE (12), *SemInt* (18) and *SF* (*Similarity Flooding*) (22); they ran each of these prototypes on the same data sources. For each prototype, Precision, Recall, F-Measure and Overall were computed; these measures were averaged across all input data sources.

We believe that the authors of (8) have provided a meaningful survey which can help us to objectively assess the accuracy of our system with that achieved by the systems mentioned above. To this purpose, we ran our system on the same data sources exploited in (8) and computed the Precision, the Recall, the F-Measure and the Overall achieved by it. Obtained results are reported in Table 4 [4]. Before discussing them, we point out that the accuracy

[4] It is worth pointing out that the values of the accuracy measures of the other systems reported in this table are exactly those specified in (8).

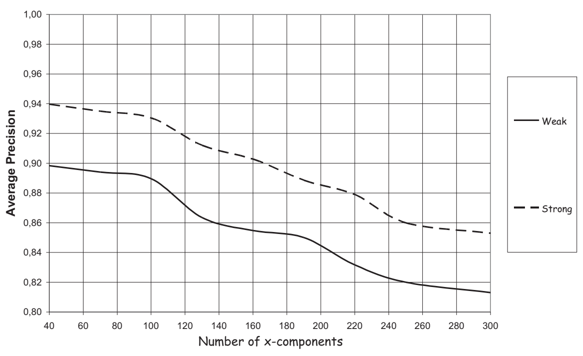


Figure 6: Variation of the Average Precision when the dimension n_t of involved XML Schemas grows

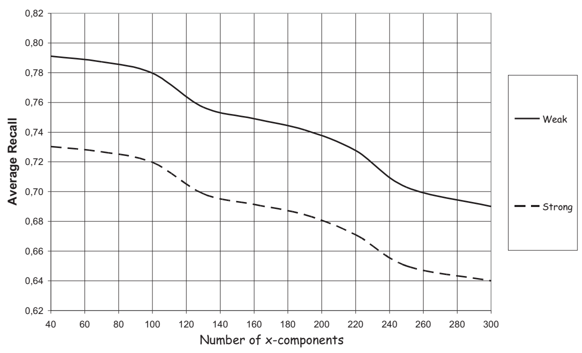


Figure 7: Variation of the Average Recall when the dimension n_t of involved XML Schemas grows

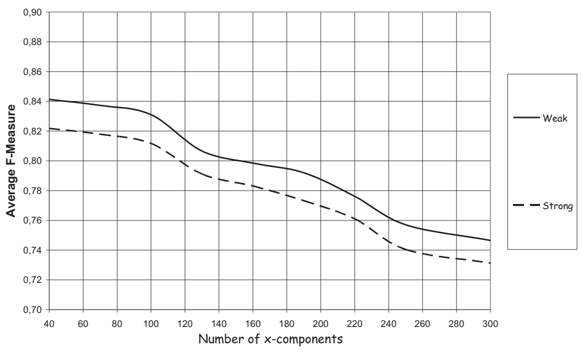


Figure 8: Variation of the Average F-Measure when the dimension n_t of involved XML Schemas grows

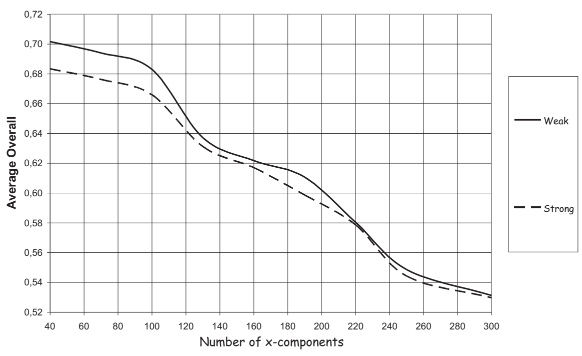


Figure 9: Variation of the Average Overall when the dimension n_t of involved XML Schemas grows

System	*Precision*	*Recall*	*F-Measure*	*Overall*
Our system (*weak*)	0.88	0.78	0.84	0.67
Our system (*strong*)	0.94	0.71	0.81	0.66
Autoplex & Automatch	0.84	0.82	0.82 & 0.72	0.66
COMA	0.93	0.89	0.90	0.82
Cupid	–	–	–	–
LSD	~ 0.80	0.80	~ 0.80	~ 0.60
GLUE	~ 0.80	0.80	~ 0.80	~ 0.60
SemInt	0.78	0.86	0.81	0.48
SF	–	–	–	~ 0.60

Table 4: Comparison of the accuracy of our approach with that of the other approaches evaluated in (8)

measures of the other approaches described in (8) concern both the derivation of sub-schema similarities and the extraction of similarities between single concepts; this last problem is simpler and, generally, the corresponding task shows better accuracy measures, especially for Recall.

From the analysis of Table 4 we can conclude that:

– If strong similarities are computed, our approach achieves the highest Precision and the lowest Recall. This result confirms the main findings emerging from Section 3.4 stating that if strong sub-schema similarities are computed then only few similarities are found but, generally, they are characterized by a high level of reliability. As for weak similarities, our approach achieves the third highest value of Precision and its Recall is high if we consider that we are extracting sub-schema similarities that are complex properties. This result points out the great flexibility of our approach which can be adapted to prioritize Precision over Recall (or vice versa) depending on the exigencies of the application context which it operates in.

– Approaches like *Autoplex*, *Automatch*, LSD, GLUE (12) and *SemInt* (18) use machine learning techniques and, as will be clear in Section 4, analyze data instances along with a wealth of auxiliary information (i.e., data type information or key constraints) to derive semantic matchings. This explains the high values of Precision and Recall achieved by them. The flip side of the coin is that they require a meaningful human effort to provide an initial set of training examples; moreover, if these examples are incomplete and/or incorrect, the accuracy achieved by these approaches may drastically decrease (7; 8; 10; 11).

– Approaches like *SF* and *COMA* may yield highly accurate results; however, their accuracy strongly depends on the feedbacks provided by human operators.

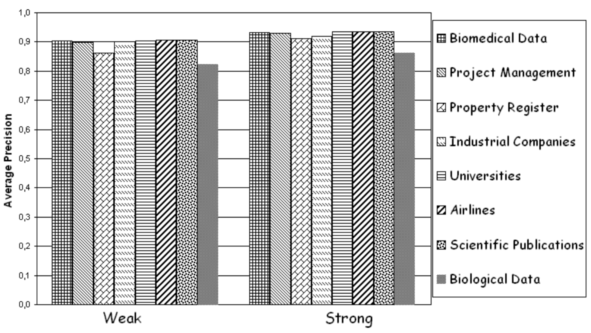


Figure 10: Average Precision of our approach in different application domains

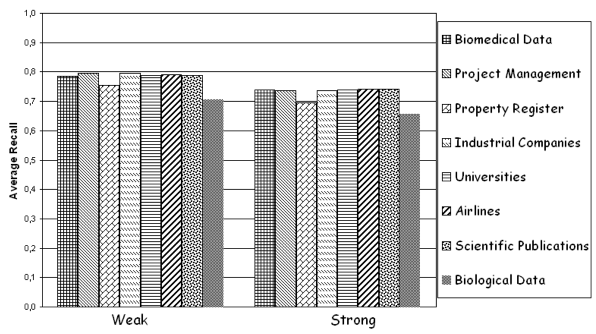


Figure 11: Average Recall of our approach in different application domains

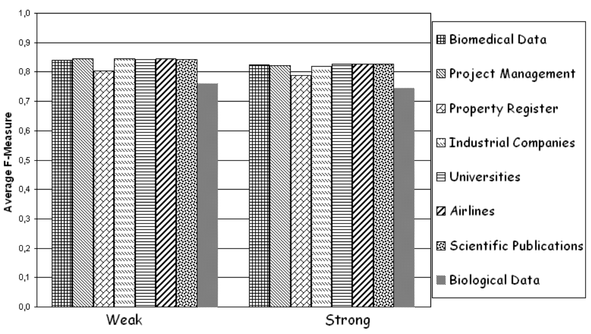


Figure 12: Average F-Measure of our approach in different application domains

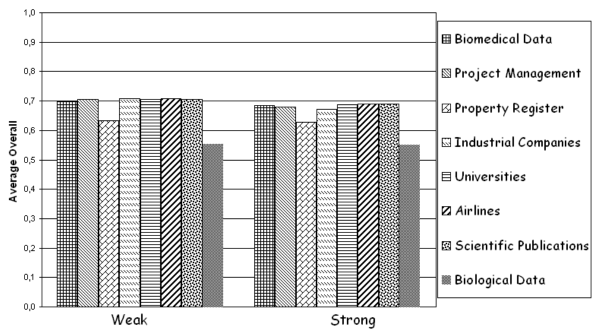


Figure 13: Average Overall of our approach in different application domains

In fact, *SF* iteratively computes semantic matchings; a human user is asked to check the matchings generated at each iteration and to fix a threshold beyond which no further iteration must be performed. Clearly, this threshold influences the number and the quality of discovered matchings.

COMA is a flexible library of matchers; a user exploiting *COMA* can select the matchers best fitting both his needs and the scenario he is operating in and can suitable combine them to improve the quality of obtained results. Clearly, the combination strategy adopted by him influences the overall quality of the results returned by *COMA*.

3.6 Role of our heuristics for the extraction of the most promising pairs of sub-schemas

In order to evaluate the role of our heuristics for the extraction of the most promising pairs of sub-schemas in the improvement of the efficiency of our approach, we have implemented a simple prototype that receives an XML Schema and evaluates the number of possible pairs of (well-formed and self contained) sub-schemas that can be derived from it. The prototype has been exploited for computing the following parameter:

Application context	*Average* $E_{Promising}$
Biomedical Data	2.47×10^{-8}
Project Management	1.61×10^{-10}
Property Register	9.18×10^{-13}
Industrial Companies	2.36×10^{-8}
University	3.88×10^{-5}
Airlines	5.07×10^{-4}
Scientific Publications	3.66×10^{-5}
Biological Data	1.97×10^{-20}

Table 5: Values of $E_{Promising}$ for the various application domains

$$E_{Promising} = \frac{\textit{Number of promising pairs of sub-schemas}}{\textit{Number of possible pairs of sub-schemas}}$$

We have carried out some tests for evaluating this parameter; Table 5 shows the average values we have obtained in the various application domains; these values have been computed by following a procedure analogous to that previously illustrated for accuracy measures.

From the analysis of this table it is possible to conclude that the value of $E_{Promising}$ is extremely low in all application domains; this confirms the importance, for our approach, of the task that singles out the most promising pairs of sub-schemas. The results shown in this table, coupled with the results about the accuracy measures reported previously, allow us to conclude that the extraction of the most promising pairs of sub-schemas plays a fundamental role for obtaining a scalable approach, applicable on real cases and producing good results. Such an idea is further enforced if we consider that the number of possible sub-schemas in an XML Schema might be exponential against the number of its x-components and, consequently, neither a manual approach nor an automatic one, exhaustively examining all pairs of sub-schemas, might be applied.

3.7 Improvement w.r.t. "naive" approaches

This class of experiments has been performed for verifying the improvements of our approach against manual, "naive" ones; here we use the term "naive" for indicating an approach that is capable of constructing only immediate and quite simple pairs of similar sub-schemas; it is generally based on identifying synonyms and expanding around them for constructing sub-schemas. Usually, a "naive" approach is little time expensive, but it tends to detect only immediate sub-schema similarities, whereas it tends to exclude many complex and potentially significant similarities. In our opinion, a comparison between our approach and the "naive" one is useful to identify the capabilities of our approach of finding complex sub-schema similarities that could be discovered by a human expert only spending a great amount of time in the analysis of the involved Schemas.

In order to carry out such a comparison, we applied our approach to our test schemas by following the guidelines illustrated in the previous experiments. For each considered pair of sub-schemas we asked human experts to determine the number N_{Naive} of sub-schema similarities, identified by our approach, that, in their opinion, had a "naive" structure. After this, we have computed the following parameter:

$$R_{Naive} = \frac{N_{Naive}}{N_{Total}}$$

where N_{Total} indicates the total number of sub-schema similarities derived by our approach. Clearly, the lower R_{Naive} is, the higher the improvement caused by our approach will be.

Table 6 shows the average value of R_{Naive} for weak and strong sub-schema similarities in the various application domains. The average values have been computed by following the same guidelines illustrated in the previous experiments.

From the analysis of this table we can observe that the best values of R_{Naive} can be obtained for domains characterized by large information sources, such as Biological Data and Property Register. This fact is explained by considering that, if involved information sources have a great number of x-components, the structures of the sub-schemas can be more complex and, consequently, N_{Total} can be extremely greater than N_{Naive}.

This result is confirmed by Figure 14 that illustrates the values of the average R_{Naive} for the extraction of weak and strong similarities for different values of the dimension of the input XML Schemas, i.e., of the parameter n_t, measuring the dimension of involved XML Schemas (see Section 3.4).

Application context *Application context*	R_{Naive} *for* *weak similarities*	R_{Naive} *for* *strong similarities*
Biomedical Data	0.53	0.62
Project Management	0.36	0.43
Property Register	0.21	0.28
Industrial Companies	0.51	0.60
University	0.61	0.70
Airlines	0.64	0.72
Scientific Publications	0.51	0.63
Biological Data	0.08	0.09

Table 6: Values of R_{Naive} for the extraction of weak and strong sub-schema similarities in the various application domains

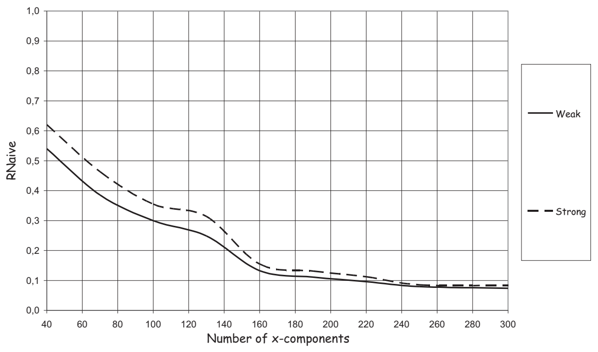


Figure 14: Variation of R_{Naive} for the extraction of weak and strong similarities when the parameter n_t, measuring the dimension of involved XML Schemas, grows

From the analysis of this figure we can observe that R_{Naive} significantly decreases, for both weak and strong sub-schema similarities, when n_t grows. This implies that, for large XML Schemas, the fraction of "naive" sub-schema similarities identified by our approach is actually negligible if compared with the total number of similarities it derives. This further confirms that a "naive" approach, even if it is less time expensive, might exclude many potentially significant similarities.

3.8 Robustness analysis

3.8.1 Robustness against structural dissimilarities

XML is inherently hierarchical; it allows nested, possibly complex, structures to be exploited for representing a domain. As a consequence, two human experts might model the same reality by means of two XML Schemas characterized by deep structural dissimilarities. We have performed a robustness analysis of our approach, devoted to verify if it is resilient to structural dissimilarities.

Before describing our experimental tests about this issue, we point out that the particular features of our approach make it intrinsically robust for a specific case, that is very common in practice. Specifically, if the typology of an x-component x_{1_j} of an XML Schema S_1 is changed from "simple element" to "attribute", or vice versa, no modifications of the sub-schema similarities involving x-components of S_1 occur. This result directly derives from the definitions of $veryclose$ and *neighborhood* (see Section 2.1).

There are further structural modifications that could influence the results of our approach and for which it is not intrinsically robust; for these cases an experimental measure of its robustness appears necessary. As an example, consider Figure 15 illustrating two portions of XML Schemas representing persons; in the first XML Schema, the concept "Person" is represented by means of a nested hierarchical structure; on the contrary, in the second XML Schema, the same concept is represented by means of a flat structure.

In order to determine the robustness of our approach in the management of these cases, for each pair of XML Schemas into consideration, we have progressively altered the structure of one of them by transforming a certain percentage of its x-components from a nested structure to a flat one. For each of these transformations, we have derived the sub-schema similarities associated with the "modified" versions of the XML Schemas and we have computed the corresponding average values of the accuracy measures. Specifically, for each pair of XML Schemas within each domain, we have considered five cases, corresponding to a percentage of flattened x-components (hereafter FXP - *Flattened X-component Percentage*) equal to: *(a)* 0%; *(b)* 7%; *(c)* 14%; *(d)* 21%; *(e)* 28%. The results we have obtained are shown in Figures 16, 17, 18 and 19.

From the analysis of these figures it is possible to observe that our approach shows a good robustness against increases of FXP. In fact, even if structural dissimilarities occur, the changes in the accuracy measures are generally quite small. However, we stress that if the increases of FXP would be significantly greater than those considered above, the changes in the accuracy measures could be significant; this behaviour is correct since it guarantees that our approach shows the right degree of sensitivity to changes to the structure of involved XML Schemas.

3.8.2 Robustness against errors in IPD

In this experiment we have tested the effects of errors and inaccuracies in the IPD received in input by our approach. We have performed the experiment as follows. First, we have exploited the approaches described in (6) for constructing IPD[5]; then, we have asked experts to validate IPD properties in such a way as to remove any possible error.

After this, we have performed some variations on the correct IPD and, for each of them, we have computed the Average Precision, the Average Recall, the Average F-Measure and the Average Overall; this activity has been performed by following the same guidelines illustrated for the previous experiments. Specifically, we have performed two different typologies of variations on IPD; in a first series of experiments we have discarded a certain percentage of correct properties from the correct IPD, without adding any new wrong property; in a second series of experiments, we have added a certain percentage of wrong properties to the correct IPD, without removing any existing correct property. Variations we have carried out on IPD are: *(a)* 10% of correct properties have been filtered out; *(b)* 20% of correct properties have been filtered out; *(c)* 30% of correct properties have been filtered out; *(d)* 50% of correct properties have been filtered out; *(e)* 10% of wrong properties have been added; *(f)* 20% of wrong properties have been added; *(g)* 30% of wrong properties have been added; *(h)* 50% of wrong properties have been added.

[5]We point out again that any other approach conceived for deriving synonymies, hyponymies and overlappings could be exploited for constructing IPD.

Case	*Average Precision*	*Average Recall*	*Average F-Measure*	*Average Overall*
No errors	0.89	0.77	0.83	0.67
(a)	0.88	0.71	0.79	0.62
(b)	0.88	0.65	0.75	0.56
(c)	0.87	0.60	0.71	0.51
(d)	0.87	0.51	0.64	0.43
(e)	0.82	0.77	0.79	0.59
(f)	0.75	0.76	0.76	0.51
(g)	0.69	0.76	0.72	0.42
(h)	0.58	0.76	0.66	0.22

Table 7: Variation of the Average Precision, the Average Recall, the Average F-Measure and the Average Overall w.r.t. possible errors in IPD for the extraction of weak similarities

Case	*Average Precision*	*Average Recall*	*Average F-Measure*	*Average Overall*
No errors	0.92	0.72	0.81	0.65
(a)	0.91	0.67	0.77	0.61
(b)	0.91	0.62	0.74	0.56
(c)	0.90	0.58	0.71	0.52
(d)	0.90	0.50	0.64	0.44
(e)	0.86	0.72	0.78	0.60
(f)	0.80	0.71	0.75	0.54
(g)	0.75	0.71	0.73	0.47
(h)	0.65	0.71	0.68	0.33

Table 8: Variation of the Average Precision, the Average Recall, the Average F-Measure and the Average Overall w.r.t. possible errors in IPD for the extraction of strong similarities

Tables 7 and 8 present the values of Precision, Recall, F-Measure and Overall we have obtained for the extraction of weak and strong similarities in all these tests. These results show that our system is quite robust w.r.t. errors and inaccuracies in IPD. In fact, its accuracy significantly decreases only for cases *(d)* and *(h)*; i.e., when the correct properties of IPD that are filtered out or the wrong properties of IPD that are added are greater than 30%. This shows, also, that our system presents a good sensitivity in addition to a satisfying robustness.

```
<xs:element name="person">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="address"/>
        </xs:sequence>
        <xs:attribute name="first_name" type="xs:string"/>
        <xs:attribute name="last_name" type="xs:string"/>
        <xs:attribute name="gender" type="xs:string"/>
        <xs:attribute name="birthdate" type="xs:date"/>
    </xs:complexType>
</xs:element>
<xs:element name="address">
    <xs:complexType>
        <xs:attribute name="city" type="xs:string"/>
        <xs:attribute name="state" type="xs:string"/>
        <xs:attribute name="country" type="xs:string"/>
        <xs:attribute name="zip" type="xs:string"/>
    </xs:complexType>
</xs:element>
```

```
<xs:element name="person">
    <xs:complexType>
        <xs:attribute name="first_name" type="xs:string"/>
        <xs:attribute name="last_name" type="xs:string"/>
        <xs:attribute name="gender" type="xs:string"/>
        <xs:attribute name="birthdate" type="xs:date"/>
        <xs:attribute name="city" type="xs:string"/>
        <xs:attribute name="state" type="xs:string"/>
        <xs:attribute name="country" type="xs:string"/>
        <xs:attribute name="zip" type="xs:string"/>
    </xs:complexType>
</xs:element>
```

Figure 15: Example of "nested" and "flat" structures

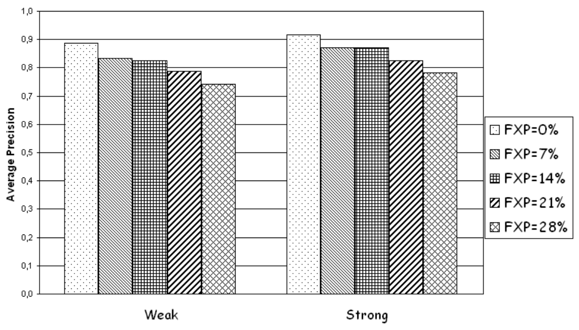


Figure 16: Average Precision of our approach for various values of FXP

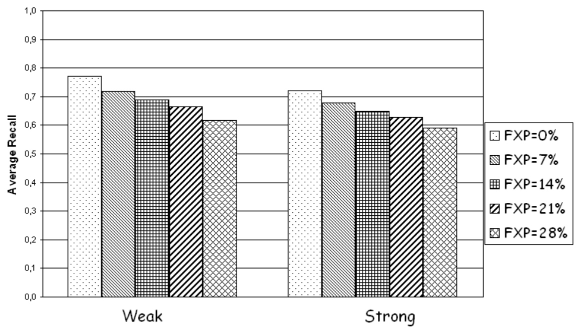


Figure 17: Average Recall of our approach for various values of FXP

3.9 Scalability Issues

3.9.1 Analysis of the cardinality of SPS

One of the most important factors that may influence the scalability of our system is the number of the most promising pairs of sub-schemas, i.e., the cardinality of SPS. In the previous sections we have shown that our heuristics for the construction of SPS allow both a very high accuracy of results to be maintained and the number of pairs of sub-schemas into examination to be significantly reduced. In this section we analyze how this number grows when the number of complex elements of the input Schemas grows. Specifically, Figure 20 plots the increase of the cardinality of SPS against m, i.e., the maximum between the number of complex elements of S_1 and that of S_2.

From the analysis of this figure we can observe that this increase is much lower than that we could expect from the theoretical worst case analysis (see Theorem 2.3). This result is quite interesting because it further confirms that the number of promising sub-schemas generated by our approach is large enough to yield accurate results (see Section 3.4) but small enough to prevent a untenable computational effort.

This result depends on the following factors:

– In order to construct SPS, we need to apply the function Ψ, introduced in Section 2.2, on all pairs of complex elements belonging to the Interschema Property Dictionary (IPD) associated with the input sources. In real cases, a complex element is involved in a very

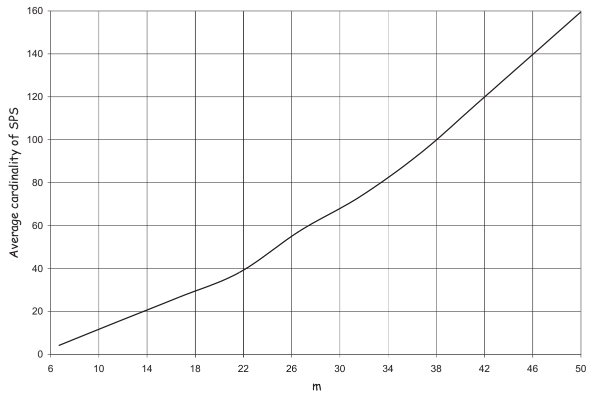


Figure 20: Cardinality of SPS against the number m of complex elements

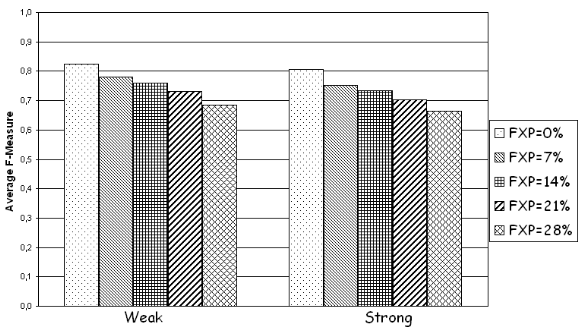


Figure 18: Average F-Measure of our approach for various values of FXP

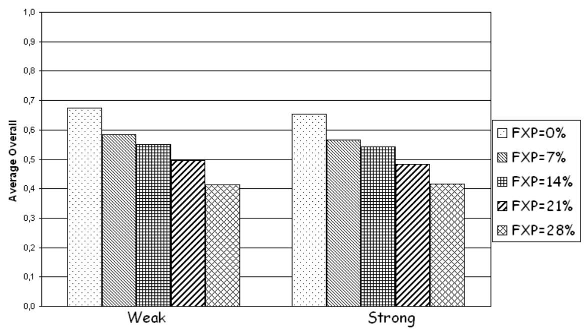


Figure 19: Average Overall of our approach for various values of FXP

low number of interschema properties and, consequently, the cardinality of IPD is much less than the overall number of possible pairs of complex elements existing between S_1 and S_2. As a consequence, the number of times our approach needs to call the function Ψ is significantly lower than that we could expect from the theoretical analysis, and this produces a significant time saving.

– In order to construct a promising sub-schema we need to apply also functions ξ and θ (see Section 2.2). Recall that θ receives a pair $\langle nbh(x_{1_j}, \delta), nbh(x_{2_k}, \gamma)\rangle$, where x_{1_j} (resp., x_{2_k}) is a complex element of S_1 (resp., S_2) and δ (resp., γ) is an integer ranging from 0 to m. It may be that θ receives a pair of neighborhoods $\langle nbh(x_{1_j}, \delta), nbh(x_{2_k}, \gamma)\rangle$ such that no elements of $nbh(x_{1_j}, \delta)$ share any interschema property with any element of $nbh(x_{2_k}, \gamma)$. In this case $nbh(x_{1_j}, \delta)$ and $nbh(x_{2_k}, \gamma)$ would be completely "pruned" by θ and, then, ξ would not return any promising sub-schema. This contributes to reduce the overall number of promising sub-schemas w.r.t. the theoretical upper bound specified by Theorem 2.3.

3.9.2 Analysis of the average size of a promising sub-schema

A second, important, factor that can influence the scalability of our approach concerns the average cardinality of promising sub-schemas; in fact, in order to derive sub-schema similarities, our approach computes some matchings on suitable bipartite graphs constructed starting from the complex elements of the involved sub-schemas (see Section 2.3). According to the reasoning illustrated in Section 2.2, measuring the average cardinality of a promising sub-schema is equivalent to measure the average cardinality of the set of complex elements generated by applying the function θ.

In Figure 21 we plot the Average Cardinality AC of a promising sub-schema against the number n of x-components of the corresponding XML Schema. From the analysis of this figure we can observe that AC depends on n in a sub-linear fashion. This result is encouraging because it shows that the size of the promising sub-schemas generated by our approach does not "explode" when the number of x-components of the corresponding input Schemas grows; this influences the scalability of our approach positively.

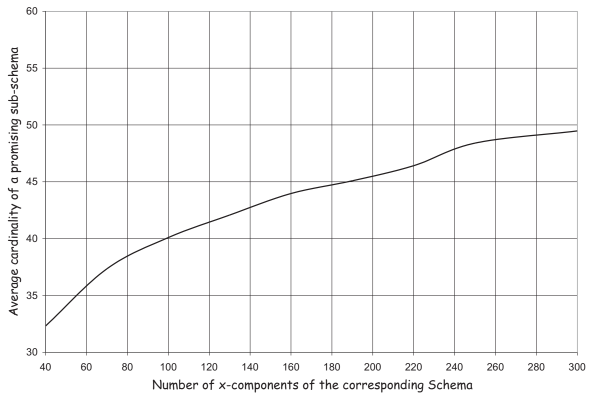


Figure 21: Average Cardinality against the number n of involved x-components

This behaviour can be justified by the following reasoning: if the overall number of x-components of an XML Schema grows then both the number of its complex elements and that of its simple elements and attributes grows. However, in real cases, this growth is not "balanced", in the sense that the number of complex elements does not grow as quick as the number of simple elements and attributes; as a consequence, if n becomes large, we expect that the number of complex elements of the Schema into consideration grows slowly, whereas the number of its simple elements/attributes increases significatively. This implies that the Interschema Property Dictionary IPD associated with input Schemas grows slowly in presence of an increase of n because IPD contains only pairs of complex elements and they must be also semantically related. Now, the number of interschema properties stored in IPD has a substantial impact on the pruning activity performed by θ (see Section 2.2) and, ultimately, on the average cardinality of a promising sub-schema; this impacts on the scalability of our approach positively.

3.9.3 Analysis of the Response Time

A third important parameter that we have considered in order to evaluate the scalability of our approach is its Response Time. To this purpose we have conducted an experimental study on our test XML Schemas to compute the increase of the Response Time caused by an increase of the sizes of schemas. All these tests have been performed on a machine with a Pentium IV 3 GHz CPU and 1 Gb of RAM.

This experiment was carried out as follows: given two XML Schemas S_1 and S_2 such that $n_1 = |XCompSet(S_1)|$ (resp., $n_2 = |XCompSet(S_2)|$) and m_1 (resp., m_2) is the number of complex elements of S_1 (resp., S_2), we have computed the average values of the Response Time of our approach against the values of $n_t = n_1 + n_2$ and $m_t = m_1 + m_2$. The obtained results are shown in Figures 22 and 23.

From the analysis of Figure 22 we can observe that the increase of the Response Time against m_t is much "softer" than that we could expect from the theoretical, worst case, analysis (see Theorems 2.4 and 2.5). In our opinion this result is even more important if we consider that:

– in real XML Schemas the number of complex elements is generally very low;

– even when the number of complex elements in one or both of the involved XML Schemas is quite high (e.g., $m_t \simeq 50$) the time necessary to our system to determine sub-schema similarities is quite low (e.g., at most some minutes for $m_t \simeq 50$);

– the extraction of sub-schema similarities is generally an activity to be performed offline.

All these considerations are further strengthened by Figure 23 where we analyze the increase of the Response Time of our system against the increase of n_t i.e., against the increase of the total number of x-components belonging to the involved XML Schemas (which is a reliable and precise indicator of the complexity of the involved Schemas).

All the reasonings above allow us to conclude that our approach is scalable and really adequate in those contexts characterized by numerous and large information sources.

Finally, Figures 22 and 23 show also that the Response Time for deriving the strong properties is comparable with that required for extracting the weak ones. This confirms the theoretical results illustrated in Theorems 2.4 and 2.5.

3.9.4 Analysis of the percentage of Response Time spared by a human expert

Finally, an interesting study, someway related to scalability, regards the computation of the average percentage of time spared by the experts by applying our approach and validating its results w.r.t. doing the same task manually.

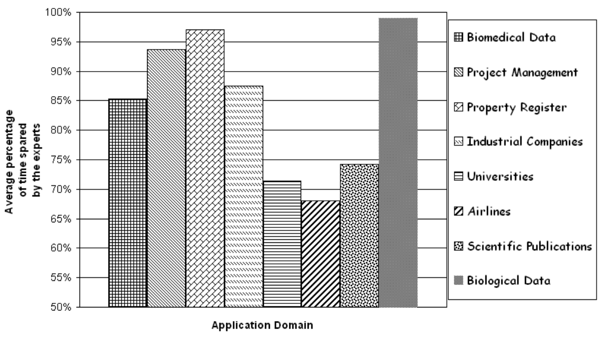


Figure 24: Average percentage of time spared by the experts by applying our approach and validating its results than doing the same task manually

Actually, this percentage is not a direct measure of the scalability of our approach; however, in our opinion, it provides a precise idea of the positive impact of our approach on the daily life of a human expert working in this application context.

The obtained results are shown in Figure 24. From the analysis of this figure we can see that the exploitation of our system really allows experts to save a great amount of time, especially in those domains involving large source schemas, such as the Property Register domain (97% of spared time) and the Biological Data domain (99% of spared time).

4 Related works

4.1 Introduction

The problem of deriving interschema properties is also called *schema matching* or *ontology alignment* in the Information Systems and Artificial Intelligence research communities; the corresponding algorithms are known as matchers (27). The problem of extracting semantic similarities between two single elements of different schemas or ontologies is often referred as *1:1* matching, whereas the problem of deriving similarities between two groups of elements or attributes is also known as *1:n*, *n:1* or, more in general, *m:n* matching.

In the literature various classification criteria have been proposed for comparing schema matching approaches (see, for example, (27)). They allow approaches to be examined from various points of view. In the following we report those criteria appearing particularly interesting in our context and exploit them to compare our approach with the other ones already presented in the past. The most common of these criteria are the following:

– *Schema Types*: Some matching algorithms can operate only on a specific kind of data source (e.g., XML, relational, and so on); these approaches are called *specific* in the following. On the contrary, other approaches are able to manage various kinds of data source; we

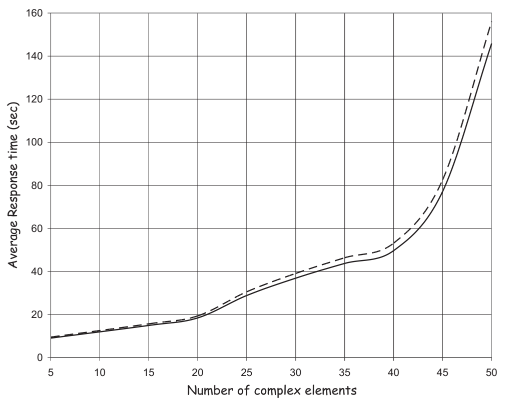


Figure 22: Response Time of our approach against the number m_t of involved complex elements

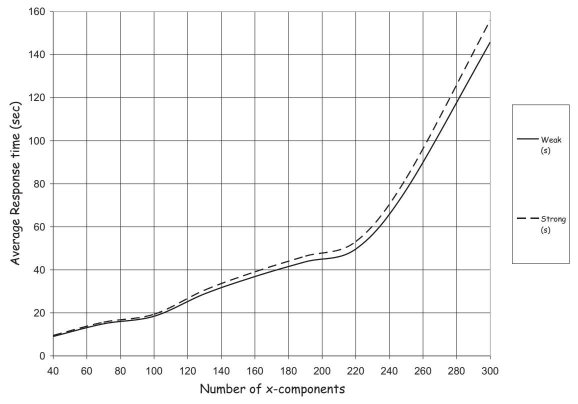


Figure 23: Response Time of our approach against the number n_t of involved x-components

call them *generic* in the following. A generic approach is usually more versatile than a specific one because it can be applied on data sources characterized by heterogeneous representation formats. On the contrary, a specific approach can take advantage of the peculiarities of the corresponding data model.

– *Instance-Based versus Schema-Based*: In order to detect interschema properties, matching approaches can consider data instances (i.e., the so-called *extensional information*) or schema-level information (i.e., the so-called *intensional information*). The former class of approaches is called instance-based; the latter one is known as schema-based. An intermediate category is represented by *mixed approaches*, i.e. those ones exploiting both intensional and extensional information.

  Instance-based approaches are generally very precise because they look at the actual content of the involved sources; however, they are quite expensive since they must examine the extensional component of the involved sources; moreover, the results of an instance-based approach are valid only for the sources it has been applied to. On the contrary, schema-based approaches look at the intensional information only and, consequently, they are less expensive; however, they could be also less precise; the results of a schema-based approach are valid for all sources conforming to the considered schemas.

– *Exploitation of Auxiliary Information*: Some approaches could exploit auxiliary information (e.g., dictionaries, thesauruses, and so on) for their activity; on the contrary, this information is not needed in other approaches. Auxiliary information represents an effective way to enrich the knowledge that an approach can exploit. However, in order to maintain its effectiveness, the time required to compile and/or retrieve it must be negligible w.r.t. the time required by the whole approach to perform its matches. For this reason, pre-built or automatically computed auxiliary information would be preferred to the manually provided one.

– *Individual versus Combinatorial*: An individual matcher exploits just one matching criterion; on the contrary, combinatorial approaches integrate different individual matchers to perform schema matching activities. Combinatorial matchers can be further classified as: *(i) hybrid matchers*, if they directly combine several schema matching approaches into a unique matcher; *(ii) composite matchers*, if they combine the results of several independently executed matchers; they are sometimes called *multi-strategy approaches*. The individual matchers are simpler, and consequently less time-consuming, than the combinatorial ones; however, the results they obtain are often not very accurate.

4.2 Some related approaches

In (23) the authors propose a logic-based matcher called *SKAT* (Semantic Knowledge Articulation Tool). In *SKAT* the user has to initially specify matching and mismatching relationships existing between two ontologies/schemas. After this, the system exploits a set of *first-order logic rules* to refine available relationships and derive new semantic matchings. These matchings can be approved or rejected by the user. Obtained results can be reused in the subsequent schema matching activities.

In (22) the *Similarity Flooding* (SF) algorithm, capable of operating on a wide variety of data sources, is proposed. SF is a graph-based matcher; first it converts input schemas into labeled graphs; then it uses a *fixpoint computation* to determine semantic matchings between the nodes of the graphs; these matchings are refined by means of specific software modules called *filters*. Generated matchings are checked by the human experts at each iteration of the fixpoint computation.

In (20) *Cupid*, a system for deriving interschema properties among heterogeneous information sources, is presented. *Cupid* takes an external thesaurus as input; its approach consists of two phases, named *linguistic* and *structural*. *Cupid* exploits sophisticated techniques, taking into account various characteristics of involved schemas; as a consequence, it is particularly suited when the precision of results is compulsory and the involved schemas are not numerous.

In (7) the authors propose the `iMAP` prototype. `iMAP` operates in two phases: the first one exploits Artificial Intelligence techniques (like *Bayesian Network* or *beam search*) to generate a set of rough matchings; the second one uses auxiliary information (like domain integrity constraints, past matchings, etc.) for refining these matchings. Interesting properties of `iMAP` are its *modularity* and its *extensibility*, since new matching algorithms might be easily embodied in it. It is worth observing that in `iMAP` the required user effort is (quite) limited; in this aspect it follows the same philosophy of our approach.

In (18) the system *SemInt* is presented. *SemInt* operates on relational schemas. First it associates each attribute with a coefficient (*signature*), computed by taking into account both intensional and extensional information. Then, it exploits the signatures of the attributes of the first schema to train a neural network that is used to cluster similar attributes of the first schema. Finally, it feeds the neural network with the signatures of the attributes of the second schema to find the attributes (resp., the groups of attributes) of the first schema best mapping the attributes (resp., the groups of attributes) of the second schema.

In (9) the authors describe a graph-based system called COMA (COmbining MAtch). It first transforms input schemas into rooted, directed, acyclic graphs; then, it activates different schema matching algorithms on these graphs; finally, it suitably combines the results produced by each algorithm to generate accurate matchings. Interestingly enough, COMA allows a user to specify the matching strategy, i.e., to choose the algorithms for performing the schema matching task.

(32) proposes a schema matching approach operating as follows: first it represents a schema by means of a rooted graph; in this way it can uniformly manage different data source typologies; after this, it combines four different techniques for computing semantic similarities between the elements of the two schemas; this last information is further exploited to derive *m:n* matchings.

In (31) an approach to deriving *1:1* and *1:n* semantic matchings holding among Web query interfaces (i.e., among data sources containing the results of the execution of queries posed through Web interfaces) is proposed. First, it derives *1:1* matchings by means of a hierarchical agglomerative clustering algorithm. After this, it extracts *1:n* matchings by applying a suitable clustering algorithm on derived *1:1* matchings.

In (16) an algorithm exploiting data mining techniques for deriving interschema properties holding among Web query interfaces is presented. This approach first translates involved sources in a suitable format; after this, it derives matchings by analyzing the co-occurrence patterns of attributes belonging to involved sources. Differently from most of the schema matching approaches proposed in the literature, the approach of (16) simultaneously examines all involved schemas.

In (11) the system CGLUE is proposed. It exploits machine learning techniques for deriving both *1:1* and *1:n* semantic matchings between two given ontologies O_1 and O_2. CGLUE receives an initial set of matchings (*training matches*) from the user; then it exploits suitable artificial intelligence techniques (e.g. Bayesian learner) to derive new interschema properties. These techniques are implemented on specific software modules called *learners*. Each learner independently operates on input schemas and generates its set of matchings; the results obtained by each learner are, then, combined to produce the final set of interschema properties.

In (14) the authors propose a schema matching approach particularly suited for Web sources. It first derives *1:1* matchings by solving a matching problem on a suitable weighted bipartite graph; in this task, several parameters (e.g., constraints associated with data types and ranges, linguistic similarities, etc.) are taken into account. After this, it derives *1:n* matchings by applying a polynomial-time heuristic algorithm on previously derived *1:1* matchings.

In (21) the MAFRA (ontology MApping FRAmework) prototype, capable of extracting mappings among distributed ontologies in the Semantic Web, is presented. MAFRA derives both *1:1* and *1:n* matchings as follows. First it represents available ontologies as RDF schemas; then, it adopts a composite approach, taking into account both structural and linguistic matchings, for deriving interschema properties. In order to carry out its activities, MAFRA requires quite a limited human intervention.

In (30) an approach to deriving *m:n* matchings is proposed. It first represents each input schema by means of a graph; after this, it asks the user to provide some basic similarities and dissimilarities. Finally, it derives similarities by taking into account information provided by users, as well as structural and linguistic information; this last is constructed with the support of a suitable thesaurus.

In (24; 25) the system *DIKE* is presented; it is devoted to extract interschema properties from E/R schemas. DIKE has been conceived to operate with quite a small number of information sources; as a consequence, it privileges accuracy to computation time. The extraction task is graph-based and takes into account the “context” of the concepts into examination; it exploits a large variety of thresholds and weights in order to better adapt its behaviour to the sources which it must operate on; these thresholds and weights must be tuned during a training phase.

4.3 Contribution of our approach

We are now able to illustrate the main novelties introduced by our approach w.r.t. the previous ones illustrated above. These novelties can be summarized as follows:

– For each pair of sub-schemas into examination, our approach analyzes both interschema properties and the structural relationships holding among the complex elements stored therein; structural relationships are modelled and handled by means of the *reachable* function.

  In the literature some *schema-based approaches* consider both the similarity of the elements belonging to promising sub-schemas and their structural relationships; however, the notion of similarity considered in these approaches is less rich and expressive than that emerging from the usage of interschema properties. For instance, *Cupid* (20) considers only lexical matchings stored in a thesaurus and the "adjacency" of schema elements (e.g., if an element X is a sub-element of an element Y); *Similarity Flooding* (22) defines an ad-hoc graph matching algorithm which uses a string-matching technique to determine the similarity of two groups of schema elements; MAFRA (21) and the approach of (14) represent input Schemas as graphs and use linguistic and structural constraints to derive *1:1* and *1:m* matchings; finally, the approach of (30) considers structural properties of input schemas (represented as graphs) and uses information extracted from a thesaurus to find sub-schema similarities.

  By contrast, *instance-based approaches*, like iMAP (7), *SemInt* (18), the approach of (31), the approach of (16) and CGLUE (11), perform a detailed analysis of the extensional component of each data source. This analysis is quite complex and refined because it considers not only the similarities existing among single elements but also complex co-occurrence patterns involving concepts belonging to different schemas. This analysis yields accurate and important results because it is often able to derive interesting semantic correspondences which would be usually neglected by a traditional schema-based approach. However, these approaches require a significant *data preparation* phase (as in (16; 31)) and a, often long, *training* phase (as in iMAP (7), *SemInt* (18), and CGLUE (11)).

  Our approach tries to overcome the shortcomings characterizing schema-based and instance-based approaches, while preserving their merits. Specifically, unlike most of schema-based approaches, it considers interschema properties, instead of lexical similarities or string matchings, as the basic properties for the computation of sub-schema similarities. The exploitation of interschema properties allow our approach to achieve a great accuracy since these properties are able to capture the semantic correspondences that would be usually neglected by lexical-based matchings (because two terms might be conceptually equivalent even though they have different names), or to discard semantic matchings that are erroneously recognized by lexical approaches (because two terms might represent different real-world concepts even though they are associated with the same, or at least quite similar, names). In addition, analogously to schema-based approaches, our own is scalable (see Section 3.9); this important feature derives from the fact that it mainly manages schema-based information.

  Unlike instance-based approaches, our approach does not inspect the extensional component of involved data sources and does not need a training phase. Owing to these reasons, it requires a less computational effort and a much more reduced human intervention. In spite of this fact, experimental tests performed in Section 3 show that the accuracy achieved by it is fully satisfactory and comparable with that obtained by instance-based approaches.

– Our approach conceptually separates the derivation of *1:1* matchings and the extraction of *1:m* and *m:n* matchings. In fact, it proposes a technique for the derivation of sub-schema similarities (i.e., *1:n* and *m:n* matchings) which is separate and independent of (although conceptually uniform with) the approach for the extraction of *1:1* matchings described in (6). Our sub-schema similarity derivation approach simply requires an Interschema Property Dictionary as input and does not oblige the user to apply the approach of (6) for constructing it.

  As a consequence, our approach can take advantage of the fact that some *1:1* matching derivation techniques are competitive in some scenarios, whereas other, conceptually different, techniques operate well in other different scenarios. A human expert could select the schema matching technique producing the best results in his scenario and could apply it to derive *1:1* matchings; then, he could use these matchings to derive new sub-schema similarities.

  In the literature, some approaches (e.g., (7; 11; 16; 18; 22; 23; 30; 32)) explicitly designed to derive *m:n* and *1:n* matchings regard *1:1* matchings as special cases of *m:n* matchings. Other approaches (e.g., (9; 14; 20; 21; 24; 25)) propose a two-phase technique: first they derive *1:1* matchings and, then, exploit these matchings, along with other support information derived during the first phase, for extracting *m:n* matchings. As previously pointed out, our approach follows a third philosophy that does not consider *1:1* matchings as special cases of *1:n* and *m:n* matchings and, at the same time, in order to derive *m:n* matchings, it does not need any further information derived during the computation of *1:1* matchings.

– Our approach considers two kinds of sub-schema similaritie, namely, *strong similarities*, computed starting from synonymies, and *weak similarities*, computed by taking also hyponymies and overlappings into account. Strong similarities detected by our system are usually few and characterized by a high level of trustworthiness; on the contrary, weak similarities are able to provide a wide picture of the semantic relationships between two schemas, even if this picture might contain some sub-schema similarities that could be not completely reliable in some application contexts.

  A clear distinction between strong and weak similarities is not present in any of the approaches described in Section 4.2.

  As a consequence of this distinction, our system is characterized by a *great flexibility*; in fact, according to the operating scenario, a human expert could prefer to manage a small set of highly reliable sub-schema similarities or, alternatively, he could want to consider a wide set of sub-schema similarities, some of which could be not precise.

5 Conclusions

In this paper we have presented a semi-automatic approach to deriving sub-schema similarities between XML Schemas; we have shown that our approach is specialized for XML sources, is almost automatic and "light". It consists of two steps: the first one selects a set of promising pairs of sub-schemas, whereas the second one computes sub-schema similarities.

We have pointed out that our approach is part of a more general framework that allows a uniform derivation of similarities and dissimilarities among concepts and groups of concepts represented in semantically heterogeneous XML Schemas. We have also presented the experimental results we have obtained by applying our approach on some, quite variegate, XML Schemas. Finally, we have examined various other related approaches previously proposed in the literature and we have compared them with ours by pointing out their similarities and differences.

At present we are working on the development of an XML Schema integration approach taking sub-schema similarities into account. In the future, we plan to study the possibility to make our sub-schema similarity derivation technique more refined by taking into account the "context" which the sub-schemas into consideration are involved in, in such a way as to define their semantics in a more precise fashion.

In addition, we plan to develop techniques exploiting sub-schema similarities in other application contexts such as those we have mentioned in the Introduction.

Finally, we argue that several other semantic relationships, already studied for single concepts could be extended to sub-schemas. In the future, we plan to verify if this intuition is really feasible and, in the affirmative case, to define suitable approaches.

References

[1] C. Batini and M. Lenzerini. A methodology for data schema integration in the entity relationship model. *IEEE Transactions on Software Engineering*, 10(6):650–664, 1984.

[2] J. Berlin and A. Motro. Autoplex: Automated discovery of content for virtual databases. In *Proc. of the International Conference on Cooperative Information Systems (CoopIS 2001)*, pages 108–122, Trento, Italy, 2001. Lecture Notes in Computer Science, Springer.

[3] J. Berlin and A. Motro. Database schema matching using machine learning with feature selection. In *Proc. of the International Conference on Advanced Information Systems Engineering (CAiSE 2002)*, pages 452–466, Toronto, Canada, 2002. Lecture Notes in Computer Science, Springer.

[4] S. Castano, V. De Antonellis, and S. De Capitani di Vimercati. Global viewing of heterogeneous data sources. *IEEE Transactions on Data and Knowledge Engineering*, 13(2):277–297, 2001.

[5] C.E.H. Chua, R.H.L. Chiang, and E.P. Lim. Instance-based attribute identification in database integration. *The International Journal on Very Large Databases*, 12(3):228–243, 2003.

[6] P. De Meo, G. Quattrone, G. Terracina, and D. Ursino. An approach for extracting interschema properties from XML schemas at various severity levels. *Informatica*, 31:217–232, 2007.

[7] R. Dhamankar, Y. Lee, A. Doan, A. Halevy, and P. Domingos. `iMAP`: Discovering complex semantic matches between database schemas. In *Proc. of the ACM International Conference on Management of Data (SIGMOD 2004)*, pages 383–394, Paris, France, 2004. ACM Press.

[8] H. Do, S. Melnik, and E. Rahm. Comparison of schema matching evaluations. In *Proc. of the International Workshop on Web, Web-Services, and Database Systems*, pages 221–237, Erfurt, Germany, 2002. Lecture Notes in Computer Science, Springer.

[9] H. Do and E. Rahm. COMA- a system for flexible combination of schema matching approaches. In *Proc. of the International Conference on Very Large Databases (VLDB 2002)*, pages 610–621, Hong Kong, China, 2002. VLDB Endowment.

[10] A. Doan, P. Domingos, and A. Halevy. Reconciling schemas of disparate data sources: a machine-learning approach. In *Proc. of the ACM International Conference on Management of Data (SIGMOD*

2001), pages 509–520, Santa Barbara, California, USA, 2001. ACM Press.

[11] A. Doan, J. Madhavan, R. Dhamankar, P. Domingos, and A. Halevy. Learning to match ontologies on the Semantic Web. *The International Journal on Very Large Databases*, 12(4):303–319, 2003.

[12] A. Doan, J. Madhavan, P. Domingos, and A. Halevy. Learning to map between ontologies on the Semantic Web. In *Proc. of the ACM International Conference on World Wide Web (WWW 2002)*, pages 662–673, Honolulu, Hawaii, USA, 2002. ACM Press.

[13] A. Gal, A. Anaby-Tavor, A. Trombetta, and D. Montesi. A framework for modeling and evaluating automatic semantic reconciliation. *The International Journal on Very Large Databases*, 14(1):50–67, 2005.

[14] A. Gal, G. Modica, and H.M. Jamil. Improving Web Search with Automatic Ontology Matching. Technical Report TR-IDB-2002-09, Department of Computer Science, Mississippi State University, 2002.

[15] Z. Galil. Efficient algorithms for finding maximum matching in graphs. *ACM Computing Surveys*, 18:23–38, 1986.

[16] B. He, K. Chen-Chuan Chang, and J. Han. Discovering complex matchings across web query interfaces: a correlation mining approach. In *Proc. of the ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD 2004)*, pages 148–157, Seattle, Washington, United States, 2004. ACM Press.

[17] M.L. Lee, L.H. Yang, W. Hsu, and X. Yang. XClust: clustering XML schemas for effective integration. In *Proc. of the ACM International Conference on Information and Knowledge Management (CIKM 2002)*, pages 292–299, McLean, Virginia, USA, 2002. ACM Press.

[18] W. Li and C. Clifton. SEMINT: A tool for identifying attribute correspondences in heterogeneous databases using neural networks. *Data and Knowledge Engineering*, 33(1):49–84, 2000.

[19] J. Madhavan, P.A. Bernstein, A. Doan, and A.Y. Halevy. Corpus-based schema matching. In *Proc. of the IEEE International Conference on Data Enginnering (ICDE 2005)*, pages 57–68, Tokyo, Japan, 2005. IEEE Computer Society Press.

[20] J. Madhavan, P.A. Bernstein, and E. Rahm. Generic schema matching with Cupid. In *Proc. of the International Conference on Very Large Data Bases (VLDB 2001)*, pages 49–58, Roma, Italy, 2001. Morgan Kaufmann.

[21] A. Maedche, B. Motik, N. Silva, and R. Volz. MAFRA - a MApping FRAmework for distributed ontologies. In *Proc. of the International Conference on Knowledge Engineering and Knowledge Management (EKAW 2002)*, pages 235–250, Siguenza, Spain, 2002. Lecture Notes in Computer Science, Springer.

[22] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity Flooding: A versatile graph matching algorithm and its application to schema matching. In *Proc. of the IEEE International Conference on Data Engineering (ICDE 2002)*, pages 117–128, San Jose, California, USA, 2002. IEEE Computer Society Press.

[23] P. Mitra, G. Wiederhold, and J. Jannink. Semi-automatic integration of knowledge sources. In *Proc. of Fusion'99*, Sunnyvale, California, USA, 1999.

[24] L. Palopoli, D. Saccà, G. Terracina, and D. Ursino. Uniform techniques for deriving similarities of objects and subschemes in heterogeneous databases. *IEEE Transactions on Knowledge and Data Engineering*, 15(2):271–294, 2003.

[25] L. Palopoli, G. Terracina, and D. Ursino. Experiences using DIKE, a system for supporting cooperative information system and data warehouse design. *Information Systems*, 28(7):835–865, 2003.

[26] K. Passi, L. Lane, S.K. Madria, B.C. Sakamuri, M.K. Mohania, and S.S. Bhowmick. A model for XML Schema integration. In *Proc. of the International Conference on E-Commerce and Web Technologies (EC-Web 2002)*, pages 193–202, Aix-en-Provence, France, 2002. Lecture Notes in Computer Science, Springer.

[27] E. Rahm and P.A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, 10(4):334–350, 2001.

[28] E. van der Vlist. Using W3C XML Schema. `http://www.xml.com/pub/a/2000/11/29/schemas/part1.html`, 2001.

[29] C.J. Van Rijsbergen. *Information Retrieval*. Butterworth, 1979.

[30] G. Wang, J.A. Goguen, Y.K. Nam, and K. Lin. Critical points for interactive schema matching. In *Proc. of the Asia-Pacific Web Conference on Advanced Web Technologies and Applications (APWeb 2004)*, pages 654–664, Hangzhou, China, 2004. Lecture Notes in Computer Science, Springer.

[31] W. Wu, C.T. Yu, A. Doan, and W. Meng. An interactive clustering-based approach to integrating source query interfaces on the Deep Web. In *Proc. of the ACM International Conference on Management of Data (SIGMOD 2004)*, pages 95–106, Paris, France, 2004. ACM Press.

[32] L. Xu and D.W. Embley. Discovering direct and indirect matches for schema elements. In *Proc. of IEEE International Conference on Database Systems for Advanced Applications (DASFAA '03)*, pages 39–46, Kyoto, Japan, 2003. IEEE Computer Society.

DNA Algorithms for Petri Net Modeling

Alfons Schuster
School of Computing and Mathematics, Faculty of Engineering, University of Ulster, Shore Road, Newtownabbey, Co. Antrim BT37 0QB, Northern Ireland
E-mail: a.schuster@ulster.ac.uk
http://www.infc.ulst.ac.uk/cgi-bin/infdb/buscard?email=a.schuster

Keywords: DNA computing, algorithms, Petri nets

Received: March 3, 2008

The paper applies, in a theoretical investigation, the DNA computing paradigm to the modeling of Petri nets. A run-through example demonstrates the feasibility of the approach as well as its potential practical value.

Povzetek: Podani so DNA algoritmi za Petri mreže.

1 Introduction

A defining moment for DNA computing was Adleman's (1) fundamental contribution in which he demonstrated the potential of this novel computing paradigm by solving an instance of the Hamiltonian Path Problem in theoretical as well as practical terms. Since then, DNA computing has been proposed and tested in numerous areas including, finite automata (14), machine learning (12), relational database modeling (13), and, of course, solving computationally expensive problems (e.g., (6), (2), and (3)).

This paper investigates Petri nets as a novel DNA computing application area. The paper provides brief introductions to Petri nets and DNA computing and demonstrates via an example algorithm how the DNA computing paradigm can be successfully applied for Petri net modeling. It is necessary to mention that the the paper does not include simulations of the work on a silicon computer or practical, experimental work involving real-life DNA material. Rather, the paper is of theoretical value only and largely neglects aspects of practical realizations of the proposed work (e.g., error rates). In a sense, the algorithm presented in this work is a high-level description for a program. The program/algorithm describes a sequence of biochemical events and these events are meant to execute/run in a biochemical environment—a DNA computer. Once this sequence of events is executed correctly, which is not a trivial bioengineering task, the result is available as/in the form of DNA strings. In order to extract the outcome of the algorithm, it is necessary to readout these strings and decode their information, but this is similar to reading out a sequence in a human genome (e.g., identifying a protein-encoding gene). Perhaps, one could think of the following analogy. It is possible to add and subtract two numbers with an electronic calculator, but the same thing can be done with an abacus (the abacus made of wood). Both procedures produce the same result but use entirely different machines and very different algorithms. Similarly, a DNA computer operates in a biochemical environment, executes real biochemical events, and uses real (usually synthetically modified) DNA.

In the remainder, Section 2 provides a brief introduction to Petri nets, their design, and working. Section 3 starts with a summary on DNA computing and then describes a DNA algorithm for a Petri net example the paper uses as a run-through vehicle to explain the presented work. Section 4 provides a discussion and Section 5 ends the paper with a summary.

2 Petri nets

Petri nets are the brainchild of Carl Adam Petri (9). Since their conception, Petri nets are a very lively field where findings in theoretical and applied work are continually added to the field. A summary may describe Petri nets as a formal, graphical, executable technique for the specification and analysis of concurrent, discrete-event dynamic systems (e.g., see http://www.petrinets.info/). Over time, the field attracted a lot of interest not only in the computing community but in a diverse spectrum of application areas including software & hardware (the complete software lifecycle from analysis, specification, design, modeling, simulation, and testing; e.g., (11) and (5)), complex systems (16), particle interaction in atomic physics (10), as well as model validation of biological pathways (4), for example. This section introduces Petri nets via a simple example network. The paper uses this example network as a run-through vehicle in forthcoming sections. For a start, Figure 1 illustrates the main Petri net components (place, active place, transition, directed arc, and token).

In order to build a network, the components in Figure 1 are combined in a systematic way by a set of relatively straightforward rules. Note that some of these definitions are adopted from (8), which is one of several excellent books available on Petri nets. The main design rules are:

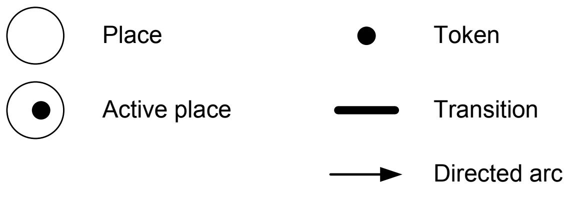


Figure 1: Main Petri net components.

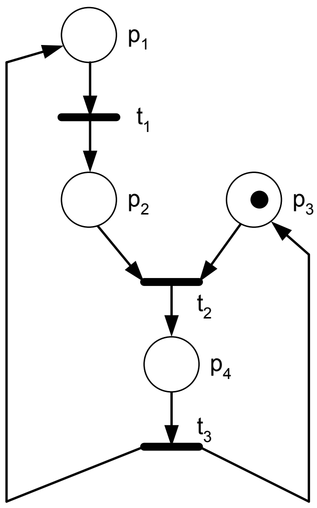


Figure 2: A Petri net with four places and three transitions.

- An arc always connects a place to a transition (in either direction).
- An arc never connects a place directly to another place nor a transition directly to another transition.
- Each place and each transition should have at least one incoming and at least one outgoing arc.
- There is no upper limit to the number of arcs that can connect to a place or a transition.

The Petri net in Figure 2, for example, is constructed by these rules. The network has four places (p_1, p_2, p_3, and p_4) and three transitions (t_1, t_2, and t_3). Directed arcs connect places and transitions, and place p_3 is an active place with one token in it. The main rules for Petri net tokens, and Petri net operations in general, are equally simple:

- Tokens are used to indicate which places are "active" (see Figure 1 and Figure 2). An active place may contain more than one token.
- If all its incoming places are active, a transition will "fire".
- When a transition fires then (a) all its incoming places lose a token, and (b) all its outgoing places gain a token.

$$P = \{p_1, p_2, p_3, p_4\}$$
$$T = \{t_1, t_2, t_3\}$$

$$I(t_1) = \{p_1\}$$
$$I(t_2) = \{p_2, p_3\}$$
$$I(t_3) = \{p_4\}$$

$$O(t_1) = \{p_2\}$$
$$O(t_2) = \{p_4\}$$
$$O(t_3) = \{p_1, p_3\}$$

Figure 3: Petri net structure $C = \{P, T, I, O\}$ for the Petri net in Figure 2.

Although Petri nets and their operations are relatively easy to understand via graphical illustrations, it is necessary mentioning that the field rests on a rigorous mathematical underpinning (e.g., see (8)). In formal terms, a Petri net is composed of four parts:

- A set P of places.
- A set T of transitions.
- An "input" function I. The input function I is a mapping from a transition t_j to a collection of places (input places) $I(t_j)$.
- An "output" function O. The output function O is a mapping from a transition t_j to a collection of places (output places) $O(t_j)$.
- The "structure" C of a Petri net is defined by its places, transitions, input function, and output function; $C = (P, T, I, O)$.

Figure 3 uses the latter definitions for the description of the Petri net in Figure 2.

It is possible to describe a Petri net and its working entirely in a rigorous mathematical way. The paper uses a simpler approach. It keeps the mathematical notation to a minimum, and instead uses graphical illustrations for the sake of ease of understanding. The paper uses Figure 4 to demonstrate the operating behavior of the Petri net in Figure 2.

Figure 4 (a) illustrates an "unmarked" Petri net. An unmarked Petri net has no tokens assigned to any of its places. In Figure 4 (b) the Petri net is "marked" with two tokens, one token is in place p_1, and the other token in place p_3. The previous text mentioned that the dynamic behavior of a Petri net is determined by the number and distribution of tokens in the Petri net. According to these rules, transition t_1 fires, because all its incoming places (p_1) have a token. Consequently, place p_1 loses its token and place p_2 gains a token (see Figure 4 (c)). Now, transition t_2 fires, because all its incoming places (p_2, and p_3) have a token. As a result, place p_2 and place p_3 lose their token, and place p_4 receives a token (see Figure 4 (d)). Next, transition t_3 fires,

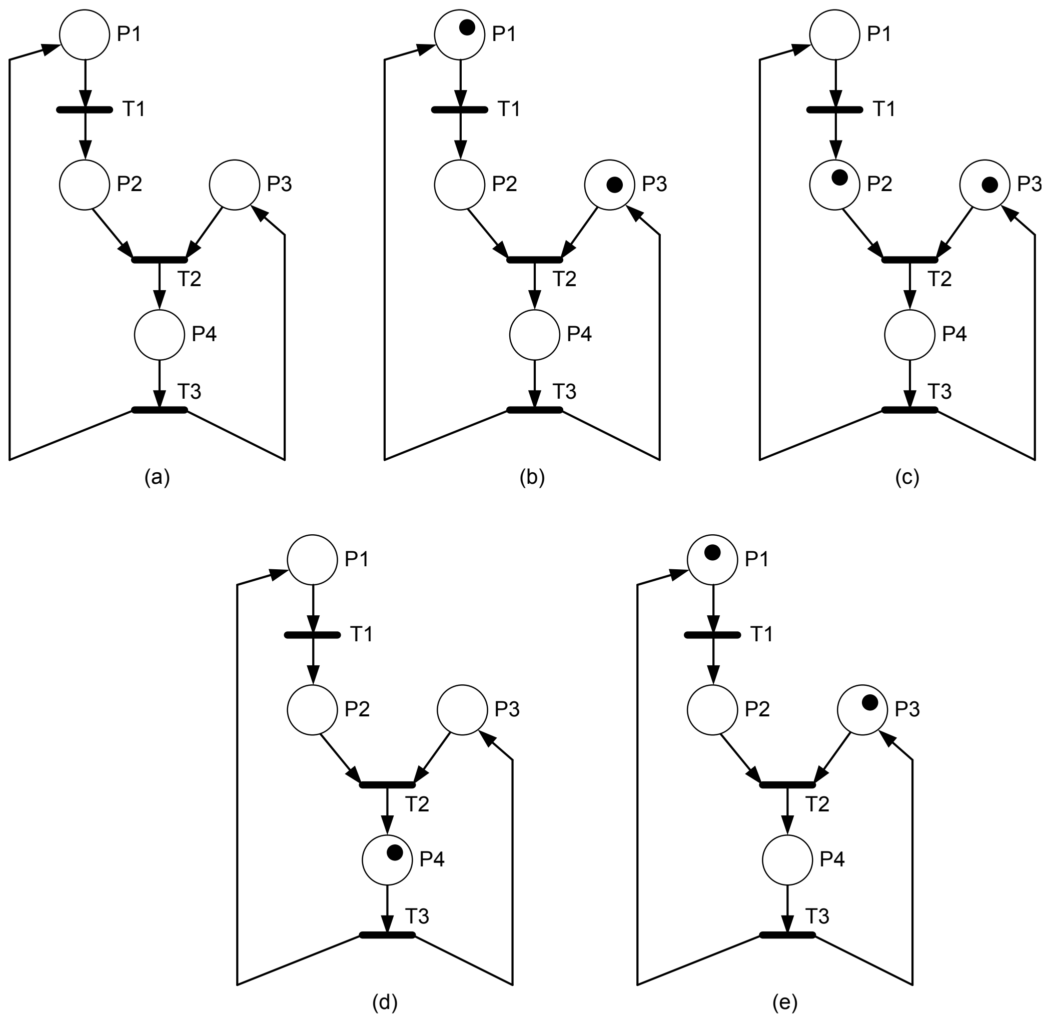


Figure 4: Execution sequence for the Petri net in Figure 2.

causing p_4 losing its token, and placing tokens into place p_1 and place p_3 (see Figure 4 (e)). A closer look reveals that Figure 4 (e) is equivalent to Figure 4 (b), and it should be clear that the example illustrates a loop.

3 DNA computing

DNA computing is a relatively young computing paradigm. Among other things, the potential of DNA computing lies in its inherent capacity for vast parallelism, the scope for high-density storage, and its intrinsic ability for potentially solving many combinatorial problems. In simple terms, DNA computing is based on the design, manipulation, and processing of nucleotides. These nucleotides are chemical compounds including a chemical base, a sugar, and a phosphate group. Four main nucleotides are distinguished, *adenine* (*A*), *guanine* (*G*), *cytosine* (*C*) and *thymine* (*T*). Nucleotides can combine or bond as "single stranded" DNA, or "double stranded" DNA. Single stranded DNA is generated through the subsequent bonding of any of the four types of nucleotides, and is often illustrated as a string of letters (e.g., TATCGGATCGGTATATCCGA). Double stranded DNA is generated from single stranded DNA and its complementary strand. This type of bonding follows "Watson-Crick Complementary", which says that base *A* only bonds with base *T*, base *G* only with base *C*, and vice versa. For example, the strand ATAGCCTAGCCATATAGGCA is the Watson-Crick Complement of the DNA strand TATCGGATCGGTATATCCGA just mentioned. The literature often illustrates a resulting double strand as two parallel strands (e.g., $\frac{TATCGGATCGGTATATCCGA}{ATAGCCTAGCCATATAGGCA}$, where the fraction line symbolizes bonding).

From a computing perspective, the field aims for the construction of DNA computers and programs that run on such a computer. Typically, the four nucleotides mentioned before provide the basis for an alphabet (e.g., Σ = {A, G, C, T}). From this alphabet a particular language (L) may be constructed. This language is used to define algorithms and computer programs. In practical terms, a DNA computer bears similarity to a biochemical machine in which biochemical events perform algorithms and execute programs by manipulating DNA strands in a series of carefully orchestrated biochemical processes. They are usually mediated by molecular entities called enzymes and include the lengthening, shortening, cutting, linking, and multiplying of DNA, for example. It is necessary to point out that these events and processes are quite challenging from a biochemical engineering perspective, but it is beyond the scope of this paper to indulge into the many challenges the field holds in this regard. There is a large body of literature available on the subject, and the interested reader is referred to one of the excellent books (7) available for the field. It may be useful however to direct a reader to some of the major contributions in the field (e.g., (15), (1), and (6)). The more imminent goal is to demonstrate the potential application of the DNA computing paradigm to the field of Petri nets.

3.1 DNA-based model for Petri nets

The goal is the design and behavioral modeling of Petri nets via DNA computing principles. The paper mentioned that the behavior of a Petri net is linked to the firing of transitions, which essentially boils down to the monitoring of activated places (i.e., places with tokens in them). For example, transition t_2 in Figure 4 (a) fires only if place p_2 and place p_3 at least hold one token each. This could be presented by a simple it-then rule: if p_2 and p_3 then t_2. The presented approach therefor has two main features:

1. It models activated places via DNA strands. For example, it is possible to represent the active place p_1 in Figure 4 (b) via the DNA strand s_1 = TATCGGATCGGTATATCCGA.

2. An algorithm describes the behavior or logic of a Petri net. This algorithm is similar to a sequence of biochemical reactions on DNA strands.

For the forthcoming sections it is necessary to introduce some of the most common operations (biochemical reactions) on DNA strands. Note that some of the following definitions are adopted from (7).

– **amplify:** Given a tube N, *amplify*(N) produces two copies of it.

– **detect:** Given a tube N, *detect*(N) returns *true* if N contains at least one DNA strand, otherwise return is *false*.

– **merge:** Given tubes N_1 and N_2, *merge*(N_1, N_2) produces a new tube N_3 that forms the union $N_1 \cup N_2$ of the two tubes.

– **separate:** Given a tube N and a DNA strand w composed of nucleotides $m \in$ {A, T, C, G}, *separate*($+N, w$) produces a new tube N_1 that consists of all strands in N which contain w as a consecutive sub-strand. Similarly, *separate*($-N, w$) produces a new tube N_1 that consists of all strands in N which do not contain w as a consecutive sub-strand.

– **lengthSeparate:** Given a tube N and an integer n, *lengthSeparate*($N, \leq n$) produces a new tube N_1 consisting of all strands in N with length less then or equal to n.

Without further ado, the paper uses Figure 5 to illustrate a DNA algorithm for the Petri net scenario in Figure 4.

The algorithm starts in line one with tube N_0. This tube is completely empty (indicated by the symbol). It is helpful to imagine that this state is equivalent to the unmarked Petri net in Figure 4 (a). The algorithm uses four boolean variables ($b1$ to $b4$ in line two) to represent the presence of any of the strands s_1 to s_4 in tube N_0. Remember that the presence of a strand is similar to an active place in the Petri net. For example, the presence of strand s_1 in tube N_0 indicates that there is a token in place p_1. The boolean variables indicate the presence of a strand by the value *true*, and its absence by the value *false*. Initially tube N_0 is empty, and so $b1 = b2 = b3 = b4 = false$ in line two. Please note that the discussion in Section 4 comments on these variables in more detail.

Line three to ten mark the Petri net. There are several possibilities for marking this particular net, and the current example uses one of them. It is possible therefore, to compare line three to ten to a function call, for instance function Mark_Petri_net(), in a computer programme where parameters are passed to the function. Line four is a simple comment. A double slash (//) always indicates a comment in the algorithm. Anyhow, line five adds strand s_1 into tube N_0, which is equivalent to adding a token into place p_1, and line six adds strand s_3 into tube N_0, which is equivalent to adding a token into place p_3. The Petri net is now in the state illustrated by Figure 4 (b). The settings of variable $b1 = true$, and $b3 = true$ in line seven and line eight reflect the presence of these strands in tube N_0. Petri nets are often models of real world systems. The appearance of a token in a place usually triggers some event in this system. This is the reason for line nine, which is a reference to some external task that my be executed by the algorithm.

Line 11 introduces the variable n. The algorithm uses this variable in the *repeat-until* loop where it defines an application specific exit criterion (line 36). The repeat-until loop extends from line 12 to line 36, and contains three *if-then* statements. Essentially, each if-then statement does two things, first, it checks the firing status of a particular transition, and second, it contains instructions for what happens when a transition fires. For example, line 14 checks for strand s_1 in tube N_0. In case the result is negative, nothing happens, and the algorithm advances to the next if-then statement. If the result is positive then place p_1 loses a token (line 15, $b1 = false$) and place p_2 gains a token (line 16, $b2 = true$). In a similar fashion, the second if-then statement monitors transition t_2, and the third if-then statement transition t_3. The comment in line 16 indicates that setting any of the boolean variables to $true$ is equal to adding a corresponding strand to tube N_1. Note that the algorithm deals with this at a later stage (lines 30 to 33).

```
(1)  input($N_0$) =
(2)  $b1 = b2 = b3 = b4 = false$,
(3)  Mark Petri net begin
(4)    //For example
(5)    add($s_1, N_0$)
(6)    add($s_3, N_0$)
(7)     $b1 = true$
(8)     $b3 = true$
(9)     Do some task.
(10) end
(11) $n = 0$
(12) repeat
(13)    input($N_1$) =
(14)    if detect($N_0(s_1)$) then begin
(15)       $b1 = false$
(16)       $b2 = true$, //add($s_2, N_1$) later
(17)       Do some task.
(18)    end
(19)    if detect(($N_0(s_2)$ and $N_0(s_3)$) then begin
(20)       $b2 = b3 = false$
(21)       $b4 = true$, //add($s_4, N_1$) later
(22)       Do some task.
(23)    end
(24)    if detect($N_0(s_4)$) then begin
(25)       $b4 = false$
(26)       $b1 = true$, //add($s_1, N_1$) later
(27)       $b3 = true$, //add($s_3, N_1$) later
(28)       Do some task.
(29)    end
(30)    if $b1 = true$ then add($s_1, N_1$)
(31)    if $b2 = true$ then add($s_2, N_1$)
(32)    if $b3 = true$ then add($s_3, N_1$)
(33)    if $b4 = true$ then add($s_4, N_1$)
(34)    $N_0 = N_1$
(35)    $n = n + 1$
(36) until (some condition regarding $n$ is met)
```

Figure 5: DNA-based algorithm for the Petri net in Figure 4 (a).

The final lines requiring explanation are line 13 and lines 30 to 35. Line 13 introduces a new tube N_1. This tube is always empty ($N_1 =$) when the repeat-until loop enters a new iteration. Between line 30 to line 33, it depends on the values for $b1$ to $b4$, which strands (s_1, s_2, s_3, or s_4) are added to tube N_1. It is important to understand that at line 35, the Petri net went through one complete state transition (e.g., from Figure 4 (b) to Figure 4 (c)). Inside the repeat-until loop, the current "state" is always represented by the content of tube N_0, and the so-called "next-state" by that of tube N_1 in line 34. Line 13 empties tube N_1 in order to prepare it for the new next state. Line 36 decides whether the loop enters a new iteration. This depends on the new value for n, which was incremented in line 35.

The paper now goes through a couple of iterations to demonstrate the algorithm in more detail. A decision table (Table 1) keeps track of the boolean variables (which represent the behavior of the Petri net in terms of active places and content of tube N_0).

Column "Start" in Table 1 captures line one and two of the algorithm and is equivalent to the unmarked Petri net in Figure 4 (a). Column "Marking" represents line three to line ten in the algorithm and is equivalent to the marked Petri net in Figure 4 (b). Note two things here, first that strands are added to tube N_0 (line five and six), and second that this is reflected by corresponding settings by the boolean variables (line seven and eight). Note also that Table 1 indicates changes in boolean variable values (as the Petri net moves from one state to the next) by underlining these values. For instance, from the "Start" state to the "Marking" state in Table 1 the values for $b1$ and $b3$ change and therefore are underlined. Anyhow, at line ten the settings are $b1 = true$, $b2 = false$, $b3 = true$, and $b4 = false$.

			Iteration, repeat loop					
	Start	Marking	1	2	3	4	5	6
$b1$, *detect*$(N_0(s_1))$	0	$\underline{1}$	$\underline{0}$	0	$\underline{1}$	$\underline{0}$	0	$\underline{1}$
$b2$, *detect*$(N_0(s_2))$	0	0	$\underline{1}$	$\underline{0}$	0	$\underline{1}$	0	0
$b3$, *detect*$(N_0(s_3))$	0	$\underline{1}$	1	$\underline{0}$	$\underline{1}$	1	0	$\underline{1}$
$b4$, *detect*$(N_0(s_4))$	0	0	0	$\underline{1}$	$\underline{0}$	0	$\underline{1}$	$\underline{0}$
State similar to Figure 4	(a)	(b)	(c)	(d)	(b)	(c)	(d)	(b)

Table 1: Decision table, illustrating the behavior of the Petri net in Figure 4.

Next, variable n is set to nil in line 11. According to the values for $b1$ to $b4$, the repeat loop enters the first if-then statement (line 14) only. Consequently, when the index n is incremented in line 35 then $b1 = false$ (line 15) and $b2 = true$ (line 16), whereas $b3$ and $b4$ remain unaltered ($b3 = true$ from line eight, and $b4 = false$ from line two). So, the settings after the first iteration are $b1 = false$, $b2 = true$, $b3 = true$, and $b4 = false$. This state is equivalent to Figure 4 (c).

In the second iteration these settings activate the second if-then statement only ($b2 = true$, and $b3 = true$). Consequently, $b2 = b3 = false$ (line 20), $b4 = true$ (line 21), and variable $b1$ remains unaltered (*false*). Now, the settings are $b1 = false$, $b2 = false$, $b3 = false$, and $b4 = true$. This state is equivalent to Figure 4 (d).

In iteration three, these settings activate the third if-then statement (line 24) only ($b4 = true$). Therefore, $b4 = false$ (line 25), $b1 = true$ (line 26), $b3 = true$ (line 27), and $b2$ remains unaltered $false$. So, the settings are $b1 = true$, $b2 = false$, $b3 = true$, and $b4 = false$. This state is equivalent to Figure 4 (b) again, and the Petri net process illustrated in Figure 4 starts again. Table 1 captures a few more iterations. Playing these iterations through demonstrates that the algorithm indeed models the behavior of the Petri net in Figure 4, however, this also indicates that the paper achieved its main goal.

4 Discussion

The previous sections successfully led to our goal, the modeling of Petri nets based on DNA computing principles. It is necessary, however, mentioning that the algorithm presented here is an ad hoc solution to the problem. It is possible, for instance, to use only one tube N_0, and to write the algorithm without any of the four boolean variables by replacing them with corresponding biochemical operations. For example, imagine the state in Figure 4 (c) equivalent to strand s_2 and strand s_3 in tube N_0. Now, imagine a progression to Figure 4 (d) where strands s_2 and s_3 need to be separated from tube N_0 and strand s_4 added to the same tube. This could be achieved by the following biochemical operations, *separate*$(-N_0, s_2)$, *separate*$(-N_0, s_3)$, and *add*(s_4, N_0). We found, however, statements such as $b2 = false$, $b3 = false$, etc. much simpler to handle and follow, helping a reader to better understand the logic of the algorithm, and also providing an easier mapping between the logic of the algorithm and the behavior of the Petri net in Figure 4.

Another possible modification relates to the modeling of active places. Currently, an active place is modeled by a single DNA strand, and the firing conditions for a transition are determined by checking for the activity of its input places (i.e., corresponding DNA strands). Line 19 in the algorithm, for example, checks for the two individual stands s_2 and s_3 in tube N_0. Another way would be to model a transition by connecting all its active places into a single strand. The Watson-Crick complement can be used for connecting two stands in a pre-defined fashion. One possibility would be to connect s_2 and s_3 via the strand $(e_{2\to3})$, where $(e_{2\to3})$ is the Watson-Crick complement of the second half of s_2 concatenated with the Watson-Crick complement of the first half of s_3. If the complements for s_2 and s_3 are $s\prime_2$ and $s\prime_3$, respectively, then we may have the example illustrated in Figure 6 (a), and in case strands s_2, s_3, and $e_{2\to3}$ where mixed together in a tube (e.g., N_0) then the double strands illustrated in Figure 6 (b) might be generated by bonding.

It is now possible to write an algorithm including some of the biochemical procedures mentioned in Section 3.1, as well as others, to check for the existence of strand s_2s_3 in tube N_0. If the strand exists then the code executed after may be similar to that following line 19 in Figure 5. The paper does not go into further details here about biochemical procedures or the algorithm reflecting these procedures, but the reader is directed to Adleman's (1) work, which entails details that are very similar to the facts just mentioned.

In terms of other issues, there is also the fact that the presented work deals with a single example only. Although the example may not really bring to the fore the great advantage DNA computing provides, namely parallelism, it is not difficult to envisage two or more Petri nets running in parallel and interacting amongst each other (e.g., interactions may be messages exchanged in the form of tokens). This should not devalue the paper, because the major contribution of in this work is the synergy of two fields—Petri nets and DNA computing. A final though considers the purely theoretical treatment of the subject. Such a treatment should not suggest any ignorance of the many challenges DNA computing still poses for engineers working in a broad variety of disciplines involved with DNA computing.

(a)

s_2 = TATCGGATCGGTATATCCGA
s_3 = GCTATTCGAGCTTAAAGCTA
s_2s_3 = TATCGGATCGGTATATCCGAGCTATTCGAGCTTAAAGCTA
s'_2 = CATATAGGCT
s'_3 = CGATAAGCTC
$e_{2\to3} = s'_2s'_3$ = CATATAGGCTCGATAAGCTC

(b)

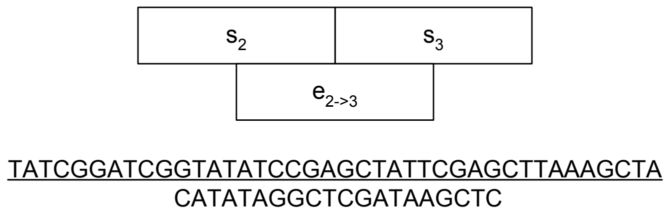


Figure 6: Alternative modeling of transitions and places.

5 Summary

The paper suggests Petri nets as a novel DNA computing application area. The paper demonstrated the feasibility of the approach in theory. The paper indicates that real-life applications of the presented work may be problematic to achieve because of various engineering challenges in the field of DNA computing. This does not mean, however, that the approach could not be verified in vitro in a DNA computing project.

References

[1] Adleman, L. (1994). Molecular computation of solutions to combinatorial problems. *Science*, 266:1021–1024.

[2] Chang, W. and Guo, M. (2003). Solving the set-cover problem and the problem of exact cover by 3-sets in the Adleman-Lipton's model. *BioSystems*, 72(3):263–275. Elsevier Science.

[3] Chang, W. and Guo, M. (2004). Molecular solutions for the subset-sum problem on DNA-based supercomputing. *BioSystems*, 73(2):117–130. Elsevier Science.

[4] Heiner, M., K. I. and Willa, J. (2004). Model validation of biological pathways using Petri nets–demonstrated for apoptosis. *Biosystems*, 75:15–28. Computational Systems Biology.

[5] Kounev, S. and Buchmann, A. (2006). SimQPN–a tool and methodology for analyzing queueing Petri net models by means of simulation. *Performance Evaluation*, 63(4–5):364–394.

[6] Lipton, J. (1995). DNA solution to hard computational problems. *Science*, 268:542–545.

[7] Paun, G., R. G. and Salomaa, A. (1998). *DNA Computing–New Computing Paradigms*. Springer-Verlag, N.Y.

[8] Peterson, J. (1981). *Petri Net Theory And The Modelling Of Systems*. Prentice Hall, Inc., Englewood Cliffs, N.J.

[9] Petri, C. (1961). *Kommunikation mit Automaten*. PhD thesis, University Bonn, Germany. PhD thesis.

[10] Petri, C. (1982). State-transition structures in physics and in computation. *International Journal of Theoretical Physics*, 21(12):979–992.

[11] Reza, H. (2006). A methodology for architectural design of concurrent and distributed software systems. *The Journal of Supercomputing*, 37(3):227–248.

[12] Schuster, A. (2003). DNA algorithms for rough set analysis. In Liu, J., Cheung, Y.M, and Yin, H., editors, *Intelligent Data Engineering and Automated Learning*, volume 2690 of *Lecture Notes in Computer Science*, pages 498–513. Springer-Verlag, Berlin.

[13] Schuster, A. (2005). DNA databases. *BioSystems*, 81(3):234–246. Elsevier Science.

[14] Soreni, M., Yogev, S., Kossoy, E., Shoham, Y., and Keinan, E. (2005). Parallel biomolecular computation on surfaces with advanced finite automata. *J. Am. Chem. Soc. (Article)*, 127(11):3935–3943.

[15] Watson, J. and Crick, F. (1953). Molecular structure of nucleic acids. *Nature*, 171:734–737.

[16] Zhu, P. and Schnieder, E. (2000). Holistic modeling of complex systems with Petri nets. In *Proceedings IEEE International Conference on Systems, Man, and Cybernetics (SMC'2000), 8-11 October 2000, Nashville, TN*, volume 4, pages 3075–3080.

A Readability Checker with Supervised Learning Using Deep Indicators

Tim vor der Brück, Sven Hartrumpf, Hermann Helbig
Intelligent Information and Communication Systems (IICS)
FernUniversität in Hagen, 58084 Hagen, Germany
E-mail: firstname.lastname@fernuni-hagen.de

Keywords: readability, syntactic and semantic analysis, nearest neighbor

Received: May 11, 2008

Checking for readability or simplicity of texts is important for many institutional and individual users. Formulas for approximately measuring text readability have a long tradition. Usually, they exploit surface-oriented indicators like sentence length, word length, word frequency, etc. However, in many cases, this information is not adequate to realistically approximate the cognitive difficulties a person can have to understand a text. Therefore we use deep syntactic and semantic indicators in addition. The syntactic information is represented by a dependency tree, the semantic information by a semantic network. Both representations are automatically generated by a deep syntactico-semantic analysis. A global readability score is determined by applying a nearest neighbor algorithm on 3,000 ratings of 300 test persons. The evaluation showed that the deep syntactic and semantic indicators lead to promising results comparable to the best surface-based indicators. The combination of deep and shallow indicators leads to an improvement over shallow indicators alone. Finally, a graphical user interface was developed which highlights difficult passages, depending on the individual indicator values, and displays a global readability score.

Povzetek: Strojno učenje z odvisnostnimi drevesi je uporabljeno za ugotavljanje berljivosti besedil.

1 Introduction

Readability checkers are used to highlight text passages that are difficult to read. They can help authors to write texts in an easy-to-read style. Furthermore they often display a global readability score which is derived by a readability formula. Such a formula describes the readability of a text numerically. There exists a large amount of readability formulas (13). Most of them use only surface-oriented indicators like word frequency, word length, or sentence length. Such indicators have only indirect and limited access to judging real understandability. Therefore, we use deep syntactic and semantic indicators[1] in addition to surface-oriented indicators. The semantic indicators operate mostly on a semantic network (SN); in contrast, the syntactic indicators mainly work on a dependency tree containing linguistic categories and surface text parts. The SNs and the dependency trees are derived by a deep syntactico-semantic analysis based on word-class functions.

Furthermore, we collected a whole range of readability criteria from almost all linguistic levels: morphology, lexicon, syntax, semantics, and discourse[2] (7). To make these criteria operable, each criterion is underpinned by one or more readability indicators that have been investigated in the (psycho-)linguistic literature and can be automatically determined by NLP tools (see (11) for details). Two typical readability indicators for the syntactic readability criterion of *syntactic ambiguity* are the *center embedding depth of subclauses* and the *number of argument ambiguities* (concerning their syntactic role[3]).

[1]An indicator is called *deep* if it requires a deep syntactico-semantic analysis.

[2]In this article, discourse criteria are subsumed under the heading semantic because they form only a small group and rely directly on semantic information.

2 Related work

There are various methods to derive a numerical representation of text readability. One of the most popular readability formulas is the so-called Flesch Reading Ease (4). The formula employs the average sentence length and the average number of syllables for estimating readability. The sentence length is intended to roughly approximate sentence complexity, while the number of syllables approximates word frequency since usually long words are less used. Later on, this formula was adjusted to German (1). Despite of its age, the Flesch formula is still widely used.

Also, the revised Dale-Chall readability index (2) mainly depends on surface-oriented indicators. Actually, it is based on sentence length and the occurrences of words in a given list of words which are assumed to be difficult to read.

Recently, several more sophisticated approaches which use advanced NLP technology were developed. They determine for instance the embedding depth of clauses, the usage of active/passive voice or text cohesion (17; 9; 21).

[3]Such ambiguities can occur in German because of its relatively free constituent order.

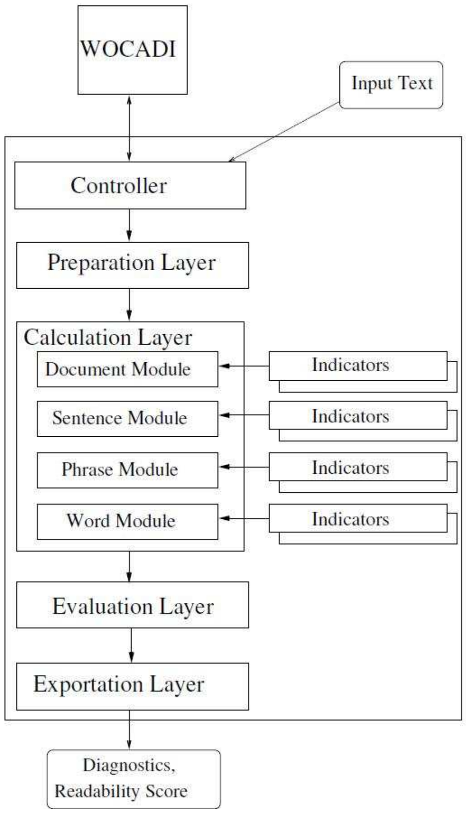


Figure 1: System architecture of the readability checker DeLite.

The methods of (3; 22) go a step beyond pure analysis and also create suggestions for possible improvements. Some approaches, e.g., (20), integrate their readability checkers into a graphical user interface, which is vital for practical usage.

As far as we know, all approaches for determining text readability are based on surface or syntactic structures but not on a deep semantic representation which represents the cognitive difficulties for text understanding more adequately. Readability formulas usually combine several so-called readability indicators like sentence or word length by a linear combination. Examples for non-linear approaches are the nearest neighbor approach of Heilman et al. (9) and the employment of support vector machines by Larsson (15) to separate the vectors of indicator values for given texts into the three different readability classes *easy*, *medium*, and *difficult*. A drawback of the latter method is that this classification is rather rough.

3 System architecture

A text is processed in several steps (see Figure 1) by our readability checker DeLite (an association of *Lite* as in light/easy reading and *De* as in Deutsch/German; there is also a prototype EnLite for English). First, the Controller passes the text to a deep syntactico-semantic analysis (WOCADI[4] parser, (6)), which is based on a word-class functional analysis and is supported by a large semantically oriented lexicon (8). The parser output for each sentence is a morpho-lexical analysis, one or more (in case of ambiguities) syntactic dependency trees, one or more SNs, and intrasentential and intersentential coreferences determined by a hybrid rule-statistical coreference resolution module. The resulting SNs follow the MultiNet formalism (multilayered extended semantic network, (10), example in Figure 2). On the basis of this analysis, the text is divided into sentences, phrases, and words in the Preparation Layer.

The individual indicator values are determined by the Calculation Layer. DeLite currently uses 48 morphological, lexical, syntactic, and semantic indicators; below we concentrate on some deep syntactic and semantic ones. Each indicator is attached to a certain processing module depending on the type of required information: words, phrases, sentences, or the entire document. Each module iterates over all objects of its associated type that exist in the text and triggers the calculation of the associated indicators. Examples for indicators operating on the word level are the indicators *number of word characters* or *number of word readings*. Semantic and syntactic indicators usually operate on the sentence level. As the result of this calculation step an association from text segments to indicator values is established.

In the Evaluation Layer, the values of each indicator are averaged to the so-called *aggregated* indicator value. Note that there exists for each indicator only one aggregated indicator value per text. The readability score is then calculated (see Sect. 4) by the k-nearest neighbor algorithm of the machine learning toolkit RapidMiner (18). In contrast to surface-oriented indicators, a deep indicator can usually only be determined for a given sentence (most deep indicators operate on sentences) if certain prerequisites are met (e.g., full parse or chunk parse is available). If this is not the case, the associated sentence is omitted for determining the aggregated indicator value. If an indicator could not be calculated for any sentence of the text at all, its value is set to some fixed constant.

Finally, all this information is marked up in XML and in a user-friendly HTML format and is returned to the calling process by the Exportation Layer.

4 Deriving a readability score using the k-nearest neighbor algorithm

A nearest neighbor algorithm is a supervised learning method. Thus, before this method can be applied to new data, a training phase is required. In this phase, a vector of aggregated indicator values is determined by RapidMiner (see Sect. 4) for each text of our readability study comprising 3,000 ratings from 300 users. The vector components are normalized and multiplied by weights representing the

[4]WOCADI is the abbreviation of **Wo**rd-**Cla**ss based **Di**sambiguating.

importance of the individual indicators where the weights are automatically learned by an evolutionary algorithm. All vectors are stored together with the average user ratings for the associated texts. To derive a readability score for a previously unseen text, the vector of weighted and normalized aggregated indicator values is determined for this text first (see above). Afterwards, the k vectors of the training data with the smallest distance to the former vector are extracted. The readability score is then given as a weighted sum of the user ratings associated with those k vectors.

5 Syntactic indicators

5.1 Clause Center Embedding Depth

A sentence is difficult to read if the syntactic structure is very complex (5). One reason for a high complexity can be that the sentence contains deeply embedded subordinate clauses. The difficulty can be increased if the subordinate clause is embedded into the middle of a sentence since the reader has to memorize the superior clause until its continuation after the termination of the subordinate clause (12), for example: *Er verließ das Haus, in dem die Frau, die er liebte, wohnte, sofort.* (literal translation from German: *He left the house where the woman he loved lived immediately.*) In contrast to (21), two separate indicators are employed for the embedding depth: one measuring embedding depth in general and one focusing only on center embedding depth which allows it to compare both effects. In our experiments only center embedding depth was considerably correlated to the readability ratings from the participants. Center embedding depth is calculated for each main verb in the following way. First, we determine the path from the root of the dependency tree to each main verb. Then, we count the occurrences of the dependency relations for relative or other subordinated clauses on this path. However, they are only taken into account if the embedded clause is not located on the border of the superior clause.

5.2 Distance between Verb and Separable Prefix

In German, so-called separable prefix verbs are split into two words in clauses with main clause word order, for example *einladen* (*invite*) $\Rightarrow$ *Er lädt ... ein.* (*He invites ....*). If the verb is far away from the verb prefix, it can be difficult for readers to associate both parts.

5.3 Number of Words per Nominal Phrase

According to (19), long NPs degrade readability. Hence, some information from the long NP should better be placed in a subordinate clause or a new sentence. Therefore we count the average number of words contained in an NP where a larger number results in a worse readability score. Note that we only consider maximal NPs (i.e., NPs not contained in a larger NP); otherwise a large indicator value for the long NP could be compensated by small indicator values for the contained NPs which should be avoided.

6 Semantic indicators

6.1 SN Quality

An incomplete parse from WOCADI is mainly caused by syntactic or semantic defects of the sentence since the parser builds the syntactic structure as a dependency tree and the semantic representation as an SN in parallel. Therefore, the indicator *SN quality* is a mixed one: semantic and syntactic. Consider for instance the two sentences *Das Werk kam vor allem bei jungen Theatergängern an. Schulbusse reisten an, um es sich anzusehen.*[5] (*The work was very well accepted by young visitors of the theater. School buses arrived to watch it.*) The second sentence, which is syntactically correct, is semantically incorrect and therefore difficult to read. The semantic lexicon employed by the parser requires that the first argument (which plays the semantic role of the agent) of *ansehen.1.1*[6] (*to watch*) is of type *human*. Thus, this sentence is rejected by the parser as incorrect. In other cases the sentence might be accepted but considered as semantically improbable. This information, which is provided by the parser, is used by DeLite and turned out to be very valuable for estimating text readability.

Three parse result types are differentiated: complete parse (around 60% of the sentences; note that this means complete syntactic structure *and* semantic representation at the same time), chunk parse (25%), failure (15%).[7] These three cases are mapped to different numerical values of the indicator *SN quality*. Additionally, if a full parse or a chunk parse is available, the parser provides a numerical value specifying the likelihood that the sentence is semantically correct which is determined by several heuristics. This information is incorporated into the quality score of this indicator too.[8]

6.2 Number of Propositions per Sentence

DeLite also looks at the number of propositions in a sentence. More specifically, all SN nodes are counted which have the ontological sort *si*(tuation) (10, p. 412) or one of its subsorts. In a lot of cases, readability can be judged more accurately by the number of propositions than by sentence length or similar surface-oriented indicators. Consider for instance a sentence containing a long list of NPs: *Mr. Miller, Dr. Peters, Mr. Schmitt, Prof. Kurt, ... were*

[5]from the newspaper *Schleswig-Holstein am Sonntag, 2007*

[6]Note that the readings of a lexeme are distinguished by numerical suffixes.

[7]The absence of a complete parse is problematic only for a part of the indicators, mainly deep syntactic and semantic ones. And even for some of these indicators, one can define fallback strategies to approximate indicator values by using partial results (chunks).

[8]Naturally, this indicator depends strongly on the applied parser. A different parser might lead to quite different results.

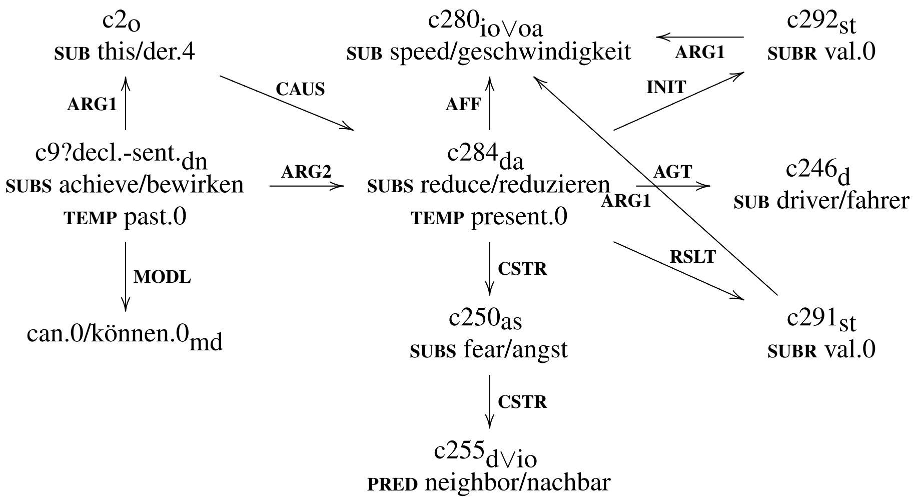


Figure 2: Simplified SN for the corpus sentence *Das könnte bewirken, dass der Fahrer aus Angst vor den Nachbarn die Geschwindigkeit reduziert.* (*This could achieve that the driver reduces the speed for fear of the neighbors.*)

present. Although this sentence is quite long it is not difficult to understand (14). In contrast, short sentences can be dense and contain many propositions, e.g., concisely expressed by adjective or participle clauses.

6.3 Number of Connections between SN Nodes/Discourse Entities

The average number of nodes which are connected to an SN node is determined. A large value often indicates a lot of semantic dependencies. For this indicator, the arcs leading to and leaving from an SN node are counted. Note that the evaluation showed that better results (stronger correlation and higher weight) are achieved if only SN nodes are regarded which are assigned the ontological sort *object* (10, p. 409–411). These SN nodes roughly represent the discourse entities of a sentence.

6.4 Length of Causal and Concessive Chains

Argumentation is needed to make many texts readable. But if an author puts too many ideas in too few words, the passage becomes hard to read. For example, the following sentence from a newspaper corpus has been automatically identified as pathologic because it contains three causal relations (CAUS and CSTR in Figure 2) chained together: *Das könnte bewirken, dass der Fahrer aus Angst vor den Nachbarn die Geschwindigkeit reduziert.* (*This could achieve that the driver reduces the speed for fear of the neighbors.*). Again, length measurements on the surface will not help to detect the readability problem, which exists for at least some user groups. Splitting such a sentence into several ones is a way out of too dense argumentation.

7 Evaluation

To judge the viability of our approach, we conducted an online readability study with 500 texts, more than 300 participants, and around 3,000 human ratings for individual texts. The participants rated the text readability on a 7 point Likert scale (16).

Almost 70 % of the participants were between 20 and 40 years old; the number of participants over 60 was very small (3 %). The participants were mainly well-educated. 58 % of them owned a university or college degree. There is none who had no school graduation at all.

Our text corpus originated from the municipal domain and differs significantly from newspaper corpora, which are widely used in computational linguistics. It contains a lot of ordinances with legal terms and abbreviations, e.g., *ğ 65 Abs. 1 Satz 1 Nr. 2 i.V.m. ğ 64 Abs. 1 Satz 2 LWG NRW* (*section 65.1.1 (2) in connection with section 64.1.2 LWG NRW*). This corpus has been chosen because local administrations in Germany have committed themselves to make their web sites accessible; one central aspect of accessibility is simple language.

Figure 4 shows the mean absolute error (MAE) and the root mean square error (RMSE) of DeLite's global readability score in contrast to the average user rating determined by a 10 fold cross-validation over all 500 test documents. The ordinate contains MAE and RMSE, the abscissa, on a logarithmic scale, the number of neighbors used. The lowest errors (MAE: 0.122, RMSE: 0.148) and highest correlation (0.528) were obtained with 30 nearest neighbors. The nearest neighbor algorithm determined the weights of each indicator using an evolutionary algorithm. The resulting indicator weights, in the case all indicators are used simultaneously, are given in Table 1.

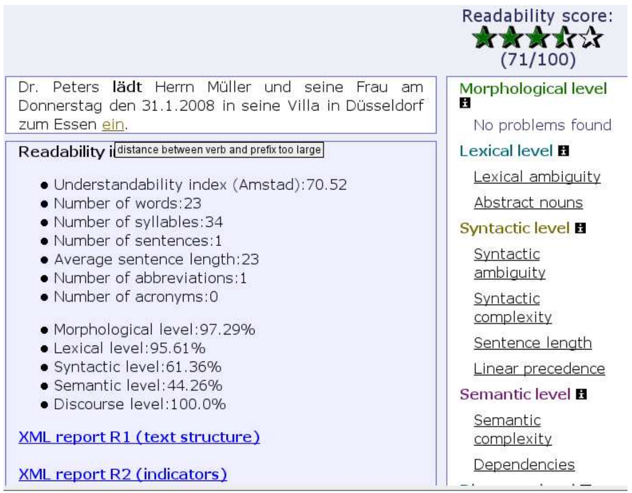
The correlations of the indicators in comparison with the user ratings are displayed in Table 2. Correlation and weights of deep syntactic and semantic indicators turned

Indicator	Weight	Type
Number of words per sentence	0.679	Sur
Passive without semantic agent	0.601	Syn/Sem
Number of word readings	0.520	Sem
Distance between verb and complement	0.518	Syn
SN quality	0.470	Syn/Sem
Number of connections between discourse entities	0.467	Sem
Inverse concept frequency	0.453	Sem
Clause center embedding depth	0.422	Syn
Number of sentence constituents	0.406	Syn
Maximum path length in the SN	0.395	Sem
Number of causal relations in a chain	0.390	Sem
Number of compound simplicia	0.378	Sur
...	...	...
Word form frequency	0.363	Sur
...	...	...
Number of connections between SN nodes	0.326	Sem

Table 1: Indicators with largest weights in our readability function (Syn=syntactic, Sem=semantic, and Sur=surface indicator type).

Indicator	Corr.	Type
Number of words per sentence	0.430	Sur
SN quality	0.399	Syn/Sem
Inverse concept frequency	0.330	Sem
Word form frequency	0.262	Sur
Number of reference candidates for a pronoun	0.209	Sem
Number of propositions per sentence	0.180	Sem
Clause center embedding depth	0.157	Syn
Passive without semantic agent	0.155	Syn/Sem
Number of SN nodes	0.148	Sem
Pronoun without antecedent	0.140	Sem
Number of causal relations in a chain	0.139	Sem
Distance between pronoun and antecedent	0.138	Sem
Maximum path length in the SN	0.132	Sem
Number of connections between discourse entities	0.132	Sem

Table 2: Indicators most strongly correlated with user ratings (Syn=syntactic, Sem=semantic, and Sur=surface indicator type).



Readability score:
★★★⯪☆
(71/100)

Dr. Peters **lädt** Herrn Müller und seine Frau am Donnerstag den 31.1.2008 in seine Villa in Düsseldorf zum Essen ein.

Readability i distance between verb and prefix too large

- Understandability index (Amstad):70.52
- Number of words:23
- Number of syllables:34
- Number of sentences:1
- Average sentence length:23
- Number of abbreviations:1
- Number of acronyms:0

- Morphological level:97.29%
- Lexical level:95.61%
- Syntactic level:61.36%
- Semantic level:44.26%
- Discourse level:100.0%

XML report R1 (text structure)

XML report R2 (indicators)

Morphological level

No problems found

Lexical level

Lexical ambiguity

Abstract nouns

Syntactic level

Syntactic ambiguity

Syntactic complexity

Sentence length

Linear precedence

Semantic level

Semantic complexity

Dependencies

Figure 3: DeLite screenshot showing a sentence which contains a large distance between verb (*lädt*) and separable verb prefix (*ein*). English translation for the example sentence: *Dr. Peters invites Mr. Müller and his wife for dinner on Thursday, Jan. 31, 2006 to his villa in Düsseldorf.*

out to be quite comparable to surface-oriented indicators.

Finally as a baseline, DeLite was compared to the readability index resulting from employing the nearest neighbor approach only on the indicators of the Flesch readability index, i.e. average sentence length and number of syllables per word. The correlation of DeLite with the user ratings is 0.528, which clearly outperforms the Flesch indicators (0.432).

8 User interface

Besides a low-level server interface, DeLite provides a graphical user interface for comfortable usage. In Figure 3, a screenshot of this interface is shown.[9] The types of readability problems found in the text are displayed on the right side. If the user clicks on such a type, the associated difficult-to-read text segments are highlighted. Additional support for the user is provided if he/she wants to have more information about the readability problem. By moving the mouse pointer over the highlighted text segment, a fly-over help text with a more detailed description is displayed. Moreover, if the user clicks on the highlighted segment, additional text segments are marked in bold face. These additional segments are needed to fully describe and explain specific readability problems. Figure 3 shows the readability analysis of a verb which is too far away from its separable prefix (see Sect. 5.2). The prefix *ein-* is highlighted as problematic and additionally the main verb *lädt* is marked in bold face for better understanding.

[9]Note that the classification of indicators is slightly different in the screenshot than in this article. This is caused by the fact that we want to evaluate surface-oriented indicators in comparison to linguistically informed indicators.

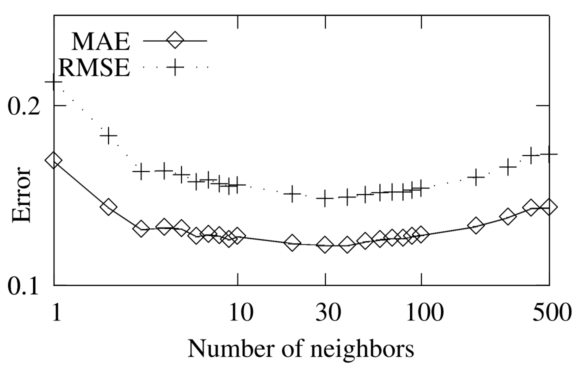


Figure 4: Mean absolute error (MAE) and root mean square error (RMSE) between the DeLite readability score and the average user rating of a text depending on the number of neighbors.

9 Conclusion

An overview of some typical examples of deep syntactic and semantic readability indicators has been given. In our evaluation, it turned out that these indicators have weights and correlations comparable to the best surface-based indicators in accurately judging readability.

In the future, the parser employed in DeLite will be continually improved. Currently, DeLite is only a diagnosis tool; we will investigate how DeLite can propose reformulations for improving readability. Finally, the automatic distinction between real ambiguities that exist for humans and spurious ambiguities that exist only for machines (e.g., NLP methods like PP attachment and interpretation) must be sharpened.

Deep syntactic and semantic indicators turned out to be quite valuable for assessing readability and are expected to be a vital part of future readability checkers.

Acknowledgments

We wish to thank our colleagues Christian Eichhorn, Ingo Glöckner, Johannes Leveling, and Rainer Osswald for their support for this work. The research reported here was in part funded by the EU project Benchmarking Tools and Methods for the Web (BenToWeb, FP6-004275).

References

[1] Amstad, T. (1978). *Wie verständlich sind unsere Zeitungen?* Ph.D. thesis, Universität Zürich, Zurich, Switzerland.

[2] Chall, J. and Dale, E. (1995). *Readability Revisited: The New Dale-Chall Readability Formula*. Brookline, Massachusetts: Brookline Books.

[3] Chandrasekar, R. and Srinivas, B. (1996). Automatic induction of rules for text simplification. Technical Report IRCS Report 96-30, University of Pennsylvania, Philadelphia, Pennsylvania.

[4] Flesch, R. (1948). A new readability yardstick. *Journal of Applied Psychology*, 32:221–233.

[5] Groeben, N. (1982). *Leserpsychologie: Textverständnis – Textverständlichkeit*. Münster, Germany: Aschendorff.

[6] Hartrumpf, S. (2003). *Hybrid Disambiguation in Natural Language Analysis*. Osnabrück, Germany: Der Andere Verlag.

[7] Hartrumpf, S.; Helbig, H.; Leveling, J.; and Osswald, R. (2006). An architecture for controlling simple language in web pages. *eMinds: International Journal on Human-Computer Interaction*, 1(2):93–112.

[8] Hartrumpf, S.; Helbig, H.; and Osswald, R. (2003). The semantically based computer lexicon HaGenLex – Structure and technological environment. *Traitement automatique des langues*, 44(2):81–105.

[9] Heilman, M. J.; Collins-Thompson, K.; Callan, J.; and Eskenazi, M. (2007). Combining lexical and grammatical features to improve readability measures for first and second language texts. In *Proceedings of the Human Language Technology Conference*. Rochester, New York.

[10] Helbig, H. (2006). *Knowledge Representation and the Semantics of Natural Language*. Berlin, Germany: Springer.

[11] Jenge, C.; Hartrumpf, S.; Helbig, H.; Nordbrock, G.; and Gappa, H. (2005). Description of syntactic-semantic phenomena which can be automatically controlled by NLP techniques if set as criteria by certain guidelines. EU-Deliverable 6.1, FernUniversität in Hagen.

[12] Kimball, J. (1973). Seven principles of surface structure parsing in natural language. *Cognition*, 2:15–47.

[13] Klare, G. (1963). *The Measurement of Readability*. Ames, Iowa: Iowa State University Press.

[14] Langer, I.; von Thun, F. S.; and Tausch, R. (1981). *Sich verständlich ausdrücken*. München, Germany: Reinhardt.

[15] Larsson, P. (2006). *Classification into Readability Levels*. Master's thesis, Department of Linguistics and Philology, University Uppsala, Uppsala, Sweden.

[16] Likert, R. (1932). A technique for the measurement of attitudes. *Archives of Psychology*, 140:1–55.

[17] McCarthy, P.; Lightman, E.; Dufty, D.; and McNamara, D. (2006). Using Coh-Metrix to assess distributions of cohesion and difficulty: An investigation of the structure of high-school textbooks. In *Proc. of the Annual Meeting of the Cognitive Science Society*. Vancouver, Canada.

[18] Mierswa, I.; Wurst, M.; Klinkenberg, R.; Scholz, M.; and Euler, T. (2006). Yale: Rapid prototyping for complex data mining tasks. In *Proc. of KDD*. Philadelphia, Pennsylvania.

[19] Miller, G. (1962). Some psychological studies of grammar. *American Psychologist*, 17:748–762.

[20] Rascu, E. (2006). A controlled language approach to text optimization in technical documentation. In *Proc. of KONVENS 2006*, pp. 107–114. Konstanz, Germany.

[21] Segler, T. M. (2007). *Investigating the Selection of Example Sentences for Unknown Target Words in ICALL Reading Texts for L2 German*. Ph.D. thesis, School of Informatics, University of Edinburgh, Edinburgh, UK.

[22] Siddharthan, A. (2003). *Syntactic simplification and text cohesion*. Ph.D. thesis, Computer Laboratory, University of Cambridge, Cambridge, UK.

Improving Morphosyntactic Tagging of Slovene Language through Meta-tagging

Jan Rupnik, Miha Grčar and Tomaž Erjavec
Jožef Stefan Institute, Jamova cesta 39, Ljubljana
E-mail : {jan.rupnik, miha.grcar, tomaz.erjavec}@ijs.si
http://kt.ijs.si

Keywords: PoS tagging, meta-tagger, Slavic languages, FidaPLUS, JOS corpus, machine learning, Orange, decision trees, CN2 rules, Naive Bayes

Received: June 17, 2008

Part-of-speech (PoS) or, better, morphosyntactic tagging is the process of assigning morphosyntactic categories to words in a text, an important pre-processing step for most human language technology applications. PoS-tagging of Slovene texts is a challenging task since the size of the tagset is over one thousand tags (as opposed to English, where the size is typically around sixty) and the state-of-the-art tagging accuracy is still below levels desired. The paper describes an experiment aimed at improving tagging accuracy for Slovene, by combining the outputs of two taggers – a proprietary rule-based tagger developed by the Amebis HLT company, and TnT, a tri-gram HMM tagger, trained on a hand-annotated corpus of Slovene. The two taggers have comparable accuracy, but there are many cases where, if the predictions of the two taggers differ, one of the two does assign the correct tag. We investigate training a classifier on top of the outputs of both taggers that predicts which of the two taggers is correct. We experiment with selecting different classification algorithms and constructing different feature sets for training and show that some cases yield a meta-tagger with a significant increase in accuracy compared to that of either tagger in isolation.

Povzetek: V članku je opisan poskus izboljšanja točnosti označevanja slovenskih besedil z združevanjem dveh neodvisnih orodij za označevanje.

1 Introduction

Morphosyntactic tagging, also known as part-of-speech tagging or word-class syntactic tagging is a process in which each word appearing in a text is assigned an unambiguous tag, describing the morphosyntactic properties of the word token. Such tagging is the basic pre-processing step for a number of applications or more advanced analysis steps, such as syntactic parsing. Morphosyntactic tagging is, in general, composed of two parts: the program first assigns, on the basis of a morphological lexicon all the possible tags that a word form can be associated with (morphological look-up), and then chooses the most likely tag on the basis of the context in which the word form appears in the text (disambiguation). For words not appearing in the lexicon, various taggers either ignore them or employ heuristics to guess at their tag.

Unlike English, morphologically richer Slavic languages such as Czech (Hajič and Hladka, 1998) or Slovene typically distinguish more than a thousand morphosyntactic tags. In the multilingual MULTEXT-East specification (Erjavec, 2004) almost 2,000 tags (morphosyntactic descriptions, MSDs) are defined for Slovene. MSDs are represented as compact strings, with positionally coded attribute values, so they effectively serve as shorthand notations for feature-structures. For example, the MSD `Agufpa` expands to `Category = Adjective, Type = general, Degree = undefined, Gender = feminine, Number = plural, Case = accusative`.

Having such a large number of tags makes assigning the correct one to each word token a much more challenging task than it is e.g. for English. The problem for Slovene has been exacerbated by the lack of large and available validated tagged corpora, which could serve as training sets for statistical taggers.

Recently, new annotated language resources have become available for Slovene. FidaPLUS[1] (Arhar & Gorjanc, 2007) is a 600 million word monolingual reference corpus automatically annotated with MULTEXT-East MSDs by the Slovene HLT company Amebis[2]. But while FidaPLUS is freely available for research via a Web concordancer, it is not generally available as a dataset. In order to remedy the lack of publicly available annotated corpora for HLT research on Slovene, the JOS project (Erjavec and Krek, 2008) is making available two corpora under the Creative Commons license. Both contain texts sampled from FidaPLUS, with the smaller jos100k containing 100,000 words with fully validated morphosyntactic annotations, and the larger, jos1M having 1 million words, and partially hand validated annotations – project resources preclude fully validating the latter.

Previous experiments (Erjavec et al., 2000) showed that from various publicly accessible taggers the best

[1] http://www.fidaplus.net/
[2] http://www.amebis.si/

results were achieved by TnT (Brants, 2000). TnT is a Hidden Markov Model tri-gram tagger, which also implements an unknown-word guessing module. It is fast in training and tagging, and is able to accommodate the large tagset used by Slovene.

Having the validated jos100k at our disposal, we experimented with training TnT and seeing how its errors compare to the ones assigned by the Amebis tagger. It turned out that the two taggers are comparable in accuracy, but make different mistakes. This gave us a method of selecting the words that should be manually corrected in jos1M – only those tokens where the annotations between the taggers differ were selected for manual inspection. This approach concentrated on validating the words where state-of-the-art taggers are still able to make correct decisions, at the price of ignoring cases where both taggers predict the same but incorrect tag, i.e. the truly difficult cases.

Having several automatically tags for each word also offers the possibility of combining their outputs in order to increase accuracy, say, over the whole FidaPLUS corpus. Experiments in combining PoS taggers have been attempted before, using various learning strategies, and for various languages, e.g. voting, stacking, etc. for Swedish (Sjöbergh, 2003) or multi-agent systems for Arabic (Othmane Zribi et al., 2006). An experiment, more similar to ours, is reported in Spoustová et al. (2007) for Czech, also using a rich positional tagset, where several stochastic taggers are combined with a rule based one; the rule based tagger is used predominantly as a pre-disambiguation step, to filter out unacceptable tags from the ambiguity classes of the tokens.

This paper presents a similar experiment, which, however, uses only two independent taggers therefore precluding combination methods such as voting or pipelining. But as in the Czech case, we also need to deal with a very large and positionally encoded tagset.

The rest of this paper is structured as follows: Section 2 presents the dataset used in the experiments, Section 3 explains the methods used to combine the output of the taggers, Sections 4 and 5 give the results of experiments on the jos100k and jos1M corpora with different methods and features, and Section 6 gives the conclusions and directions for further work.

2 Dataset

The dataset used in the first set of experiments is based on the jos100k corpus; the corpus contains samples from almost 250 texts from FidaPLUS, cca. 1,600 paragraphs or 6,000 sentences. The corpus has just over 100,000 word tokens, and, including punctuation, 120,000 tokens. jos100k contains only manually validated MSDs, of which 1,064 different ones appear in the corpus.

For the dataset we added MSDs assigned by Amebis and TnT to the manually assigned ones. Two sentences from the dataset are given in Figure 1. Annotations marking texts and paragraphs have been discarded and end of sentence is marked by an empty line. Punctuation is tagged with itself.

Prišlo	Vmep-sn	Vmep-sn	Vmep-sn
je	Va-r3s-n	Va-r3s-n	Va-r3s-n
do	Sg	Sg	Sg
prerivanja	Ncnsg	Ncnsg	Ncnsg
in	Cc	Cc	Cc
umrla	Vmep-sf	Vmep-sf	Vmep-sf
je	Va-r3s-n	Va-r3s-n	Va-r3s-n
.	.	.	.
Tega	Pd-nsg	Pd-msa	Pd-msg
se	Px------c	Px------c	Px------c
sploh	Q	Q	Q
nisem	Va-r1s-y	Va-r1s-y	Va-r1s-y
zavedel	Vmep-sm	Vmep-sm	Vmep-sm
.	.	.	.

Figure 1: Example stretch of the corpus dataset ("*Prišlo je do prerivanja in umrla je. Tega se sploh nisem zavedel.*"). First column is the word-form, second the gold standard manually assigned tag, third the one assigned by TnT, and the fourth by Amebis. Note the first word of the second sentence, where both taggers make a mistake.

The source FidaPLUS corpus also contains, for each word token, all possible MSDs that could be assigned to it, i.e. its ambiguity class. Based on this information, we computed the average per-word MSD ambiguity which turns out to be 3.13 for the jos100k corpus. So, on the average, a tagger needs to choose the correct MSD tag between three possibilities. Note that disambiguation is only possible for known words.

2.1 Amebis MSDs

The Amebis MSDs were taken from the source FidaPLUS corpus; as mentioned, the Amebis tagger is largely a rule-based one, although with heuristics and quantitative biases. The tagger uses a large lexicon, leaving only 2% of the word tokens in jos100k unknown. Amebis doesn't tag these words, and they have all been given a distinguished PoS/MSD "unknown". Furthermore, FidaPLUS is annotated according to the MULTEXT-East specification, while the JOS corpus uses a modification, based on, but different from the MULTEXT-East/FidaPLUS one. Differences concern reordering of attribute positions, changes in allowed values, etc., as well as lexical assignment. For the most part an information-preserving conversion is possible, but for MSDs (attributes) of some lexical items only heuristics can be used for the conversion. Taking into account that all Amebis "unknowns" are by definition wrong, as all words are manually annotated with specific MSDs, and that a certain number of errors is introduced by the tagset mapping, Amebis obtains 87.9% accuracy on all tokens (incl. punctuation) in the dataset.

2.2 TnT MSDs

The TnT tagger was trained on the dataset itself, using 10-fold cross-tagging. The dataset was split into 10 parts, with 9 folds used for training, and the remaining fold tagged with the resulting model, and this process repeated for all 10 folds. As the lexical stock of jos100k is small, the tagging model used a backup lexicon which was extracted from the FidaPLUS corpus and its annotations. In other words, tri-gram statistics and lexicon containing uni-gram statistics of word-forms (their ambiguity classes) of frequent words were learned from jos100k, while less frequent words obtained their ambiguity classes from MSDs assigned by the Amebis tagger. Given such a tagging set-up, the obtained accuracy over the all dataset tokens (incl. punctuation) for TnT is 88.7%, slightly better than Amebis; but TnT has the advantage of learning how to correctly tag at least some unknown words (such as those marked as "foreign", i.e. tokens in spans of non-Slovene text), as well as having less problems with tagset conversion. Nevertheless, on the dataset it performs better than Amebis, so the TnT accuracy can be taken to constitute the baseline for the experiment.

2.3 Error comparison

Table 1 compares the errors made by the taggers against the gold standard. The first line gives the complete size of the corpus in words. The second gives the number of correct MSD assignment to word tokens for TnT (86.6% per-word accuracy), and the third for Amebis (85.7%). The fourth line covers cases where both taggers predict the correct MSD, for 78% of the words.

Lines 5 and 6 cover cases where one tagger correctly predicts the tag, while the other makes a mistake. These two lines cover a significant portion (2/3) of all the errors, so if such mistakes can be eliminated by deciding which tagger made the correct choice, the gains in accuracy are considerable.

The last two lines indicate upper bounds on the gains achieved by concentrating on choosing the correct tag. Line 7 gives cases where both taggers agree, but on an incorrect tag (3.2%), and line 8 the number of cases where both are wrong, but in different ways (2.4%); the upper bound on combination accuracy is thus 94.3%.

Let us look at two typical examples of cases 7 and 8. An example of both taggers being wrong, but agreeing on the assigned tag is exemplified in the fragment *"ni mogoče povedati" (it is not possible to tell)* where *"mogoče"* should be an adverb but both taggers assign it an adjectival tag. An example of both taggers being wrong in different ways is the fragment *"ni priporočene/Adj zgornje/Adj mejne/Adj vrednosti/Adj" (there is no recommended upper bound value).* The correct tag for the noun is *Ncfsg*, i.e. feminine singular genitive, the genitive being determined by the (long distance) dependency on "*ni*". The Amebis tagger correctly predicts this tag, while TnT makes a mistake, and assigns to the noun the plural accusative. As adjectives must agree with the noun in gender, number and case, the three adjectives preceding the noun must also be tagged as feminine singular genitive. Here both taggers are wrong: while TnT correctly posits the agreement between the noun and adjectives, all the adjective tags are wrong, due to the noun being incorrectly tagged. Amebis, on the other hand, does not pick up the agreement, and tags all three adjectives as masculine ones.

	Words	Gold	Amebis	TnT	Gloss
1	100,003	MSD1			Words in dataset
2	86,623	MSD1		MSD1	TnT tagger correct
3	85,718	MSD1	MSD1		Amebis tagger correct
4	78,018	MSD1	MSD1	MSD1	Both taggers correct
5	7,700	MSD1	MSD1	MSD2	Amebis correct, TnT error
6	8,605	MSD1	MSD2	MSD1	Amebis error, TnT correct
7	3,232	MSD1	MSD2	MSD2	Both wrong, and identical
8	2,448	MSD1	MSD2	MSD3	Both wrong, and different

Table 1: Comparison of tagging accuracy of Amebis and TnT over the 100k dataset.

3 Combining the taggers

As mentioned, our meta-tagger is built on top of two taggers, the Amebis rule-based tagger and TnT. The sole task of the meta-tagger is to decide which tag to consider correct. The meta-tagger is implemented as a classifier which, if the two underlying taggers disagree, classifies the case into one of the two classes indicating which of the two taggers is more likely to be correct. To train the classifier, we needed two things: a way to describe a case with a set of features, and a classification algorithm. The following section describes the feature construction process and the subsequent section the classification algorithms we tried out for this task.

3.1 Feature construction

To be able to train the classifier we needed to describe each case with a set of features. We decided to keep our meta-tagger relatively simple and to construct features solely out of tags predicted by the underlying taggers. Alternatively, we could compute content features as well (such as *n*-grams, prefixes, and suffixes) as it is the case with the SVM-based taggers such as SVMTool (Giménez & Márquez, 2004).

For training and testing we used the dataset discussed in Section 2, with each word assigned three tags: the correct tag (assigned manually), the tag assigned by TnT, and the tag assigned by the Amebis tagger. Each of these three tags can be decomposed into 15 attributes such as the part-of-speech category, type, gender, number, and so on. For a given tag, not all attribute values are set, therefore the data is sparse in this sense (e.g. the value of gender and number for prepositions is "undefined").

The attributes of the tags assigned by the two taggers (but not those of the manually assigned tags) were directly used as features for training. In addition, we constructed features that indicate whether the two taggers agree on a particular attribute value or not (the so called agreement features). The example was labeled according to the tagger which correctly tagged the word (the label was thus either TnT or Amebis). Note that we built a training feature vector only when the two taggers disagreed and one of them was correct (if none of the taggers was correct, we were unable to label the feature vector). The entire feature construction process is illustrated in Figure 2.

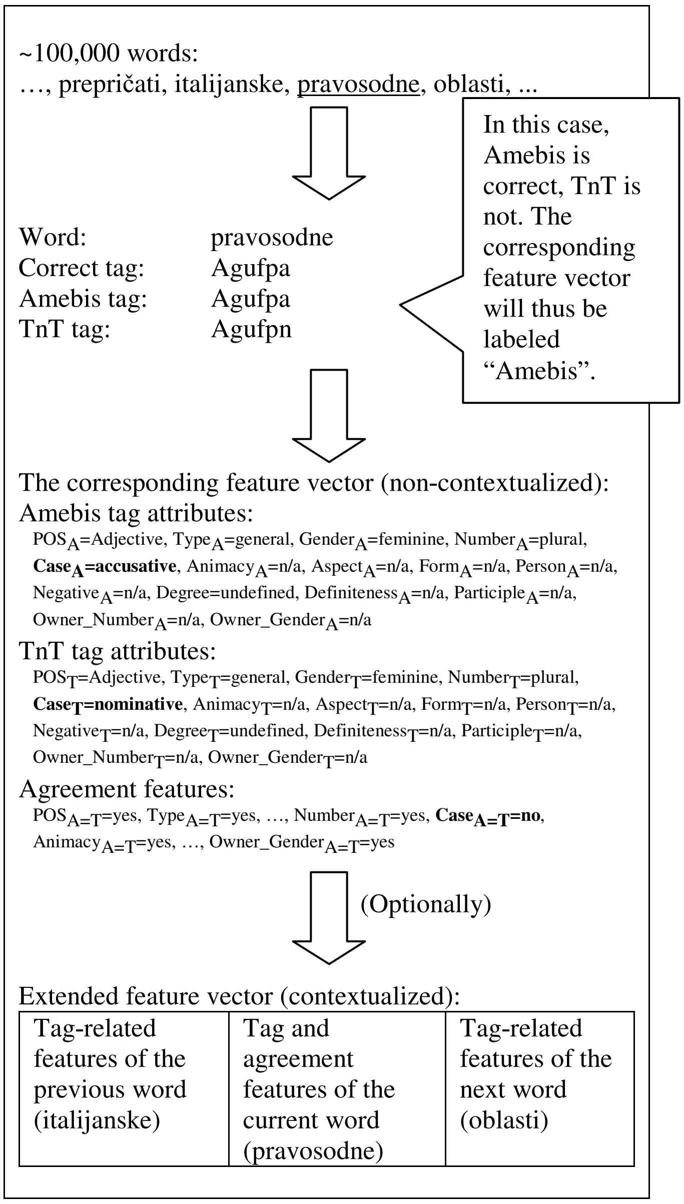
For the first set of experiments we used the tag attributes and agreement features of the current word to construct a feature vector (termed non-contextualized features in Figure 2). In the second set of experiments, on the other hand, we also added tag features (from both, TnT and Amebis) from the previous and the next word (termed contextualized features in Figure 2). It is also important to mention that we ran a set of experiments where we excluded punctuation from the text and a set of experiments where each different type of punctuation was treated as a separate part-of-speech category (e.g. POS_T=,) with all the other attributes set to "not applicable". Each of these settings gave slightly different results. The results are discussed in Section 4 in more detail.

3.2 Learning algorithms

We experimented with three different classification algorithms: the Naive Bayes classifier, CN2 rule-induction algorithm, and C4.5 decision tree building algorithm. In this section, we briefly describe each of them.

The **Naive Bayes (NB)** classifier is a probabilistic classifier based on Bayes' theorem.[3] It naively assumes a strong independence of features. Furthermore, it is a black box classifier in the sense that its decisions are not easily explainable.

CN2 is an if-then rule-induction algorithm (Clark & Niblett, 1989). It is a covering algorithm meaning that each new rule covers a set of examples which are thus removed from the dataset. Unlike the Naive Bayes classifier, the trained model (i.e. a set of induced rules) provides an explanation for a decision (i.e. an if-then rule that was taken into account when classifying the example). Looking at the induced rules, it is also possible to read, understand, and also verify the knowledge that was discovered in the training set.


~100,000 words:
…, prepričati, italijanske, pravosodne, oblasti, ...

Word:	pravosodne
Correct tag:	Agufpa
Amebis tag:	Agufpa
TnT tag:	Agufpn

In this case, Amebis is correct, TnT is not. The corresponding feature vector will thus be labeled "Amebis".

The corresponding feature vector (non-contextualized):
Amebis tag attributes:
POS_A=Adjective, $Type_A$=general, $Gender_A$=feminine, $Number_A$=plural, **$Case_A$=accusative**, $Animacy_A$=n/a, $Aspect_A$=n/a, $Form_A$=n/a, $Person_A$=n/a, $Negative_A$=n/a, Degree=undefined, $Definiteness_A$=n/a, $Participle_A$=n/a, $Owner\_Number_A$=n/a, $Owner\_Gender_A$=n/a

TnT tag attributes:
POS_T=Adjective, $Type_T$=general, $Gender_T$=feminine, $Number_T$=plural, **$Case_T$=nominative**, $Animacy_T$=n/a, $Aspect_T$=n/a, $Form_T$=n/a, $Person_T$=n/a, $Negative_T$=n/a, $Degree_T$=undefined, $Definiteness_T$=n/a, $Participle_T$=n/a, $Owner\_Number_T$=n/a, $Owner\_Gender_T$=n/a

Agreement features:
$POS_{A=T}$=yes, $Type_{A=T}$=yes, …, $Number_{A=T}$=yes, **$Case_{A=T}$=no**, $Animacy_{A=T}$=yes, …, $Owner\_Gender_{A=T}$=yes

(Optionally)

Extended feature vector (contextualized):

Tag-related features of the previous word (italijanske)	Tag and agreement features of the current word (pravosodne)	Tag-related features of the next word (oblasti)

Figure 2: The feature construction process.

C4.5 is an algorithm for building decision trees; it is based on information entropy[4] (Quinlan, 1993). C4.5 uses the fact that each attribute of the data can be used to make a decision that splits the data into smaller subsets. It examines the normalized information gain (difference in entropy) that results from choosing an attribute for splitting the data. The attribute with the highest normalized information gain is the one used to make the decision. This process is repeated several times on smaller and smaller subsets of data. Similarly to CN2 (the rule-induction algorithm), C4.5 builds glass box models. Unlike its predecessor, the ID3 algorithm, C4.5 knows how to handle data with missing values (i.e. sparse data) and prunes the tree by cutting off branches that do not contribute to the classification accuracy.

[3] c.f. http://en.wikipedia.org/wiki/Naive_Bayes_classifier

[4] c.f. http://en.wikipedia.org/wiki/C4.5_algorithm

4 Experiments

In this section, we present tagging accuracies of the meta-tagger for different combinations of feature sets and underlying classification models. The size of the set of examples for training and testing is 16,305 and consists of 8,605 cases where TnT tagger predicted the correct tag and Amebis tagger did not and 7,700 cases where Amebis was correct and TnT was not. All experiments were conducted with the Orange data mining tool (Demšar et al., 2004). 5-fold cross validation method was used to evaluate the tagging accuracy of the meta-tagger in all experimental scenarios. We first discuss two baseline models for the meta-tagger, after that we define several different feature sets, then continue with the description of non-contextualized models and end the section with models that incorporate context features.

4.1 Baselines

The first baseline is the majority classifier which always predicts that TnT tagger is correct. This classifier achieves the accuracy of 52.8%.

The second baseline model is a Naive Bayes model trained on only one feature: Amebis MSD. This is a very simple model, since to classify a new example (with only one feature f, that is the Amebis MSD), all one needs to do is count the number of cases with MSD equal to f where Amebis was correct and the number of cases with MSD equal to f where Amebis was incorrect (*P(x = f, y = amebis-correct)* and *P(x = f, y = amebis-incorrect)*) and predict the class (amebis-correct or amebis-incorrect) with the higher count. This model achieves the accuracy of 70.95% (approx. 18% higher than the first baseline).

Let us consider two examples. Assume that there were 200 cases where Amebis predicted the tag Pd-nsg, and it was correct in 150 of these cases (this means the TnT was correct in the remaining 50 cases). This means that P(Amebis-predicts: Pd-nsg, Amebis-correct) = 0.75. In this case the meta-tagger would always predict the tag Pd-nsg if Amebis predicted it as well.

Now, if we assumed that Amebis was correct in 80 of 200 cases, P(Amebis-predicts: Pd-nsg, Amebis-correct) = 0.4), then the meta-tagger would always predict the tag predicted by TnT, given that Amebis predicted Pd-nsg (the evidence in the training data tells us not to trust the Amebis tagger, since the probability of it being correct is less than 0.5).

4.2 Feature sets

We will now describe the features for the non-contextualized models. The first set of features for the non-contextualized models are the so called FULL features; they only include full Amebis MSD and full TnT MSD (two features). The second set of features called DEC is a decomposition of the FULL features as described in Section 3.1 (45 features: 15 Amebis features, 15 TnT features, 15 Agreement features). The third set of features, BASIC, is a subset of DEC features, where we only take the features corresponding to Category, Type, Gender, Number and Case into account (10 features: 5 for Amebis and 5 for TnT). The final set of features, ALL, is a union of FULL and DEC (47 features).

Feature sets for contextualized models (with and without punctuation) are extensions of non-contextualized feature sets, where the features of examples surrounding our training example are added (see Section 3.1). The context features (i.e. the features of the previous and next word) are the same ones as that of the current word except for the Agreement features which are only computed for the current word (in the DEC feature set we thus keep only 15 Agreement features: the ones of the current word).

Features ALL, when contextualized, include six features for MDS tags (Amebis-Prev, Amebis, Amebis-Next, TnT-Prev, TnT, TnT-Next), 45 for Amebis tag features (3 × 15 features), 45 for TnT tag features and 15 Agreement features, which sums up to 111 features.

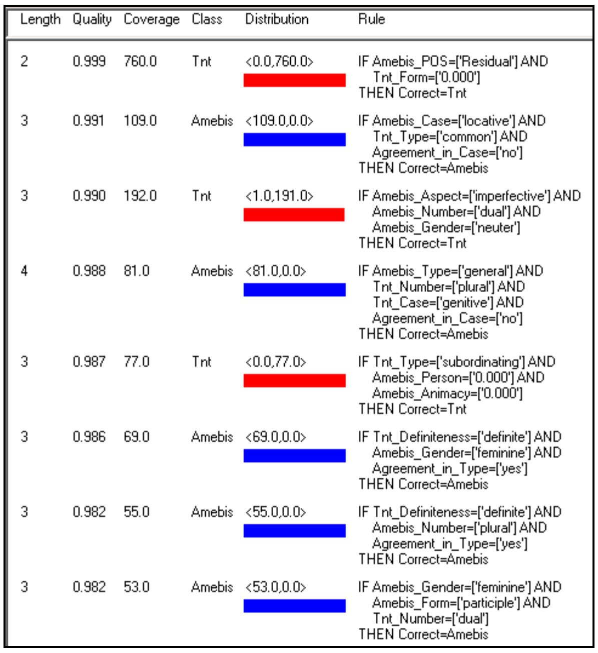
4.3 Non-contextualized models

Experiments with features that do not take context into account (Table 2) show that C4.5 is the most robust classifier with respect to different feature sets and that it can achieve the highest accuracy. We can also observe that tag features are not very suitable for the Naive Bayes classifier because the conditional independence assumptions are too strongly violated.

Feature set / Classifier	FULL	DEC	BASIC	ALL
NB	73.90	67.55	67.50	69.65
C4.5	73.51	**74.70**	74.23	73.59
CN2	60.61	72.57	71.68	70.90

Table 2: Non-contextualized models (accuracy in %). Feature sets FULL, DEC, BASIC and ALL are explained in Section 4.2.

Even though the CN2 algorithm results in slightly lower accuracy it can prove useful since the rules that it produces are easy to interpret and thus discover the strengths and weaknesses of the TnT and Amebis classifiers (see Figure 3).



Length	Quality	Coverage	Class	Distribution	Rule
2	0.999	760.0	Tnt	<0.0,760.0> 🟥	IF Amebis_POS=['Residual'] AND Tnt_Form=['0.000'] THEN Correct=Tnt
3	0.991	109.0	Amebis	<109.0,0.0> 🟦	IF Amebis_Case=['locative'] AND Tnt_Type=['common'] AND Agreement_in_Case=['no'] THEN Correct=Amebis
3	0.990	192.0	Tnt	<1.0,191.0> 🟥	IF Amebis_Aspect=['imperfective'] AND Amebis_Number=['dual'] AND Amebis_Gender=['neuter'] THEN Correct=Tnt
4	0.988	81.0	Amebis	<81.0,0.0> 🟦	IF Amebis_Type=['general'] AND Tnt_Number=['plural'] AND Tnt_Case=['genitive'] AND Agreement_in_Case=['no'] THEN Correct=Amebis
3	0.987	77.0	Tnt	<0.0,77.0> 🟥	IF Tnt_Type=['subordinating'] AND Amebis_Person=['0.000'] AND Amebis_Animacy=['0.000'] THEN Correct=Tnt
3	0.986	69.0	Amebis	<69.0,0.0> 🟦	IF Tnt_Definiteness=['definite'] AND Amebis_Gender=['feminine'] AND Agreement_in_Type=['yes'] THEN Correct=Amebis
3	0.982	55.0	Amebis	<55.0,0.0> 🟦	IF Tnt_Definiteness=['definite'] AND Amebis_Number=['plural'] AND Agreement_in_Type=['yes'] THEN Correct=Amebis
3	0.982	53.0	Amebis	<53.0,0.0> 🟦	IF Amebis_Gender=['feminine'] AND Amebis_Form=['participle'] AND Tnt_Number=['dual'] THEN Correct=Amebis

Figure 3: List of rules discovered by CN2 in Orange. Rules are ordered by their quality which is a function of rule coverage and rule accuracy. The second rule, for example, tells us that if Amebis predicted locative case and TnT predicted some other case and TnT predicted common type, then the meta-tagger should predict the same tag as Amebis. The first rule, IF Amebis_POS=['Residual'] AND TnT_Form=['0.000'] THEN Correct = TnT, covers the examples mentioned in Section 2.1, where Amebis predicts POS tag "unknown" (by definition incorrect). The rule says that in such case, TnT is always correct, which is what is expected.

4.4 Context and punctuation

When comparing the results of experiments with context, we notice that taking punctuation into account (see Section 3.1) is beneficial in almost all cases (see Tables 3 and 4). This can be explained by the fact that ignoring punctuation can yield unintuitive context tags, for instance the sequence of tags T1, T2, T3, where T1 is the last word of a sentence, T2 the first word and T3 the second word of the next sentence.

We notice that C4.5 can best benefit from extra contextual features, whereas the performance of the other algorithms does not change notably.

Feature set / Classifier	FULL	DEC	BASIC	ALL
NB	73.10	68.29	67.96	70.55
C4.5	73.10	78.51	**79.23**	76.72
CN2	62.16	73.26	72.75	72.29

Table 3: Context without punctuation (accuracy in %).

Feature set / Classifier	FULL	DEC	BASIC	ALL
NB	73.44	68.32	68.14	70.53
C4.5	74.18	78.91	**79.73**	77.68
CN2	62.23	74.27	72.82	73.01

Table 4: Context with punctuation (accuracy in %).

5 Large-scale experiment

In addition to the experiments on the jos100k corpus, we also performed a large-scale experiment on a larger subset of FidaPLUS, the jos1M corpus, consisting of 1,000,017 word tokens (without punctuation). The corpus was first tagged by both taggers (i.e. Amebis and TnT). Amebis is a rule-based tagger and does not require training, TnT, on the other hand, was trained on the complete jos100k corpus. Then, if (and only if) the two taggers disagreed on a particular word token, the token was manually validated. Consequently, we are unable to determine cases when both taggers are correct or agree on an incorrect tag. Dataset statistics (analogous to the ones in Table 1) are given in Table 5.

	Words	Gold	Amebis	TnT	Gloss
1	1,000,017				Words in dataset
2	809,897		MSD1	MSD1	Both taggers agree
3	75,378	MSD1	MSD1	MSD2	Amebis correct, TnT error
4	88,657	MSD1	MSD2	MSD1	Amebis error, TnT correct
5	26,085	MSD1	MSD2	MSD3	Both wrong, and different

Table 5: The jos1M corpus statistics.

5.1 Experimental setting

We confronted Naive Bayes with C4.5 (building CN2 rules was computationally too expensive). We experimented with all defined feature sets: FULL, DEC, BASIC, and ALL, with and without context. Punctuation was included in the contextualized cases. For some reason, the C4.5 algorithm was unable to handle feature sets FULL and ALL when contextualized. We speculate that the implementation in Orange does not manage memory efficiently when it comes to attributes with 1000+ different values. The results of the experiments are presented in the following section.

5.2 Results

In this section, we present tables analogous to the ones in Section 4. We show how the algorithms perform under different feature sets. As already said, we do not show results for the CN2 algorithm and for C4.5 under certain conditions (denoted with "N/A"). The results fully support our observations on the smaller jos100k corpus and are presented in Tables 6 and 7. Note also that the second baseline yields 72.39% accuracy on the jos1M corpus.

Feature set / Classifier	FULL	DEC	BASIC	ALL
NB	73.93	66.85	66.67	69.81
C4.5	76.45	**76.56**	76.29	76.49

Table 6: The jos1M corpus – non-contextualized models (accuracy in %).

Feature set / Classifier	FULL	DEC	BASIC	ALL
NB	73.74	67.59	67.86	70.28
C4.5	N/A	**84.18**	84.01	N/A

Table 7: The jos1M corpus – context and punctuation (accuracy in %).

6 Conclusions

The paper presents a meta-tagger built on top of two taggers, namely the TnT HMM-based tagger and the Amebis rule-based tagger. The purpose of the meta-tagger is to decide which tag to take into account if the two taggers disagree in a particular case.

The experimental results show that the two taggers are quite orthogonal since very little information is needed to get a significant increase in performance from the first baseline.

Furthermore, using context can improve the performance of some models and taking punctuation into account when constructing context features is better than ignoring it. C4.5 with context and punctuation features achieves the highest accuracy, 79.73% on jos100k and 84.18% on jos1M, which results in a meta-tagger with significantly higher accuracy than Amebis tagger or TnT tagger. The overall accuracies are given in Figure 4. Note that the first baseline is equal to the TnT overall accuracy.

There are roughly 5% cases in which both taggers assign an incorrect tag. By using the technique discussed in this paper (i.e. rule inference), it would be possible to learn under which conditions the two taggers are both mistaken and thus alert the user about such tags.

Furthermore, it would be possible to apply our technique on a per-attribute basis. We would be able to predict incomplete tags, i.e. tags with some attributes missing, where the missing attributes would be those most likely predicted falsely by both taggers. This would be very useful as guidance for human taggers preparing the JOS corpus. The missing attributes would have to be entered manually; the rest would only need to be validated.

Tagging on a per-attribute basis and looking at cases in which both taggers predict an incorrect tag will be the focus of our future research. In addition, we will consider including more taggers into the system. The main idea is to develop taggers, specialized to handle cases in which the two currently used taggers are not successful.

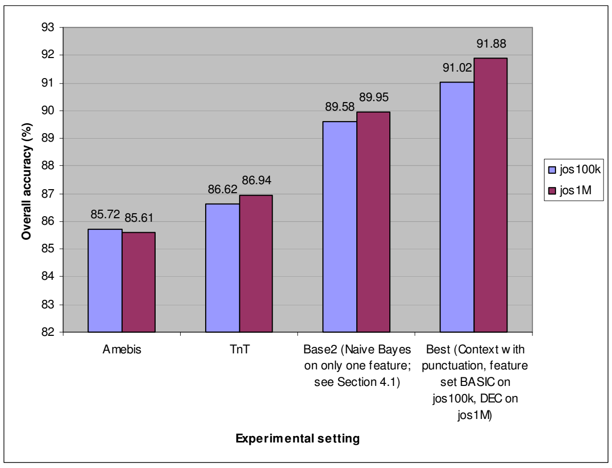


Figure 4: The overall accuracies (%). We can see that our meta-tagger exhibits around 4%–5% overall improvement over the two underlying taggers (i.e. TnT and Amebis). For computing the accuracies on the jos1M corpus, we needed to estimate the number of cases where the two taggers agreed on a correct tag. Looking at the statistics of the jos100k corpus (Table 1), we can see that the taggers are correct in 96.4% of the cases where they agree on the tag. Therefore, we computed the required number as 96.4% of 809,897 which is 780,740.71.

Acknowledgements

The work described in this paper was supported in part by grant ARRS J2-9180 "Jezikoslovno označevanje slovenskega jezika: metode in viri" and EU 6FP-033917 SMART "Statistical Multilingual Analysis for Retrieval and Translation".

References

[1] Arhar, Š. and Gorjanc, V. (2007). Korpus FidaPLUS: nova generacija slovenskega referenčnega korpusa. Jezik in slovstvo, 52(2): 95–110.

[2] Brants T. (2000). TnT – A Statistical Part-of-Speech Tagger. In Proceedings of the Sixth Applied Natural Language Processing Conference ANLP-2000, 224–231.

[3] Clark, P. and Niblett, T. (1989). The CN2 Induction Algorithm. Machine Learning, 3(4): 261–283.

[4] Demšar J., Zupan B. and Leban G. (2004). Orange: From Experimental Machine Learning to Interactive Data Mining. White Paper (www.ailab.si/orange), Faculty of Computer and Information Science, University of Ljubljana.

[5] Erjavec, T., Džeroski, S. and Zavrel, J. (2000). Morphosyntactic Tagging of Slovene: Evaluating PoS Taggers and Tagsets. In Proceedings of the Second International Conference on Language Resources and Evaluation (LREC'2000). ELRA, Paris.

[6] Erjavec, T. (2004). MULTEXT-East Version 3: Multilingual Morphosyntactic Specifications, Lexicons and Corpora. In Proceedings of the Fourth

International Conference on Language Resources and Evaluation, LREC 2004, 1535–1538.

[7] Erjavec, T. and Krek, S. (2008). The JOS morphosyntactically tagged corpus of Slovene. In Proceedings of the Sixth International Conference on Language Resources and Evaluation, LREC 2008.

[8] Giménez, J. and Márquez, L. (2004). SVMTool: A General POS Tagger Generator Based on Support Vector Machines. In Proceedings of the 4th International Conference on Language Resources and Evaluation (LREC'04).

[9] Hajič, J. and Hladka, B. (1998). Tagging Inflective Languages: Prediction of Morphological Categories for a Rich, Structured Tagset. COLING-ACL'98. ACL.

[10] Quinlan, J.R. (1993). C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, Inc.

[11] Sjöbergh, J. (2003). Combining POS-taggers for improved accuracy on Swedish text. In NoDaLiDa 2003, 14th Nordic Conference on Computational Linguistics. Reykjavik.

[12] Spoustová, D., Hajič, J., Votrubec, J., Krbec, P. and Květoň, P. (2007). The Best of Two Worlds: Cooperation of Statistical and Rule-Based Taggers for Czech. Proceedings of the Workshop on Balto-Slavonic Natural Language Processing. June 2007. Prague, Czech Republic. Association for Computational Linguistics.

[13] Zribi, C.B.O., Torjmen, A. and Ahmed, M.B. (2006). An Efficient Multi-agent System Combining POS-Taggers for Arabic Texts. In Computational Linguistics and Intelligent Text Processing. LNCS Volume 3878/2006, Springer.

Improving Part-of-Speech Tagging Accuracy for Croatian by Morphological Analysis

Željko Agić and Zdravko Dovedan
Department of Information Sciences, Faculty of Humanities and Social Sciences, University of Zagreb
Ivana Lučića 3, HR-10000 Zagreb, Croatia
E-mail: {zeljko.agic, zdravko.dovedan}@ffzg.hr

Marko Tadić
Department of Linguistics, Faculty of Humanities and Social Sciences, University of Zagreb
Ivana Lučića 3, HR-10000 Zagreb, Croatia
E-mail: marko.tadic@ffzg.hr

Keywords: part-of-speech tagging, morphological analysis, inflectional lexicon, Croatian language

Received: May 22, 2008

This paper investigates several methods of combining a second order hidden Markov model part-of-speech (morphosyntactic) tagger and a high-coverage inflectional lexicon for Croatian. Our primary motivation was to improve tagging accuracy of Croatian texts by using our newly-developed tagger CroTag, currently in beta-version. We also wanted to compare its tagging results – both standalone and utilizing the morphological lexicon – to the ones previously described in (Agić and Tadić 2006), provided by the TnT statistical tagger which we used as a reference point having in mind that both implement the same tagging procedure. At the beginning we explain the basic idea behind the experiment, its motivation and importance from the perspective of processing the Croatian language. We also describe tools – namely tagger and lexicon – and language resources used in the experiment, including their implementation method and input/output format details that were of importance. With the basics presented, we describe in theory four possible methods of combining these resources and tools with respect to their operating paradigm, input and production capabilities and then put these ideas to test using the F-measure evaluation framework. Results are then discussed in detail and conclusions and future work plans are presented.

Povzetek: Opisana je nova metoda označevanja hrvaških besedil.

1 Introduction

After obtaining satisfactory results of the preliminary experiment with applying a second order hidden Markov model part-of-speech/morphosyntactic tagging paradigm by using TnT tagger on Croatian texts, we decided to attempt reaching a higher level of accuracy based on these results. Detailed description of the previous experiment is given in (Agić and Tadić 2006) and TnT tagger is described in (Brants 2000). Please note that abbreviation HMM is used instead hidden Markov model and PoS (MSD) tagging instead part-of-speech (morphosyntactic) tagging further in the text.

In the section about our future work plans in (Agić and Tadić 2006), we provided two main directions for further enhancements:

a. Producing new, larger and more comprehensive language resources, i.e. larger, more precisely annotated and systematically compiled corpora of Croatian texts, maybe with special emphasis on genre diversity and
b. Developing our own stochastic tagger based on HMMs (being that TnT is available to public only as a black-box module) and then altering it by adding morphological cues about Croatian language or other rule-based modules.

We considered both courses of action as being equally important. HMM PoS/MSD trigram taggers make very few mistakes when trained on large and diverse corpora encompassing most of morphosyntactic descriptions for a language and, on the other hand, they rarely seem to surpass 98% accuracy on PoS/MSD, excluding the tiered tagging approach by (Tufis 1999.) and (Tufis and Dragomirescu 2004), not without help of rule-based modules, cues from morphological lexica or other enhancements which in fact turn stochastic tagging systems into hybrid ones. We have therefore chosen to undertake both courses of action in order to create a robust version of Croatian PoS/MSD tagger that would be able to provide us with high-quality MSD-annotated Croatian language resources automatically.

However, knowing that manual production of MSD-tagged corpora takes substantial amounts of time and human resources, we put an emphasis on developing and fine-tuning the trigram tagger in this experiment. Here we describe what is probably the most straightforward of currently available fine-tuning options for Croatian –

combining CroTag tagger and Croatian morphological lexicon. The lexicon itself is described in (Tadić and Fulgosi 2003) and implemented in form of Croatian lemmatization server, described in (Tadić 2006) and available online at http://hml.ffzg.hr. Our notion of tagger-lexicon combination in this paper refers to several possibilities of utilizing high coverage of the lexicon on Croatian texts in order to assist CroTag where it makes most errors, namely while tagging tokens that were not encountered by its training procedure.

Section 2 of the paper describes all the tools, language resources, annotation standards, input and output formats used in the experiment, while section 3 deals in theory with four conceptually different but functionally similar methods of pairing CroTag tagger and Croatian morphological lexicon. Section 4 defines the evaluation framework that would finally provide us with results. Discussion and conclusions along with future plans are given in sections 5 and 6.

2 Resources and tools

In this section, we give detailed insight on tools and resources used in the experiment, along with other facts of interest – basic characteristics of available annotated corpora and input-output file format standard used.

2.1 Inflectional lexicon

At the first stage of the experiment, we had available the Croatian morphological lexicon in two forms – one was the generator of Croatian inflectional word forms, described in (Tadić 1994) and another was the Croatian lemmatization server, detailed in (Tadić 2006). As it can be verified at http://hml.ffzg.hr, the server takes as input a UTF-8 encoded verticalized file. File verticalization is required because the server reads each file line as a single token which is used as a query in lemma and MSD lookup. Output is provided in form of a text file and an equivalent HTML browser output. Figure 2.1 represents a simplified illustration of this output: first token is the word form given at input and it is followed by pairs of lemmas and corresponding morphosyntactic descriptors compliant to MULTEXT-East v3 specification, given by e.g. (Erjavec 2004).

```
da [da2 Qr] [dati Vmia2s] [dati Vmia3s]
        [dati Vmip3s] [da1 Css]
```

Figure 2.1 Output of inflectional lexicon (illustration)

Therefore, a text document was extracted from the server containing all (lemma, token, MSD) triples and any computer program or a programming library implementing fast search capability over this document could be utilized in our experiment as a black-box module. For this purpose, we used the Text Mining Tools library (TMT), described in (Šilić et al. 2007), that had implemented a very fast and efficient dictionary module based on finite state automata, storing triples of word forms, lemmas and tags into an incrementally constructed deterministic automaton data structure. This TMT dictionary module has thus provided us with the needed object-oriented interface (conveniently developed in C++, same as CroTag) that we could use to get e.g. all lemmas and MSDs for a token, all MSDs for a (token, lemma) pair etc. By utilizing this library, a working inflectional lexicon interface was at our disposal to be used both as an input-output black-box and rule-based module for integration with CroTag at runtime.

2.2 Stochastic tagger

Stochastic PoS/MSD trigram tagger for Croatian (or just CroTag from this point on) was developed and made available in form of an early beta-version for purposes of validation in this experiment, enabling us to envision future improvement directions and implementation efforts. Although many stochastic taggers have been made available to the community for scientific purposes during the years – for example, the TnT tagger (Brants 2000) and its open source reimplementation made in OCaml programming language, named HunPos (Halacsy et al. 2007) – and could be utilized in research scheme of our experiment, we still chose to develop our own trigram HMM tagger. This enabled us to alter its operation methods whenever required and also allowed us to integrate it with larger natural language processing systems that are currently under development for Croatian, such as the named entity recognition and document classification libraries. CroTag is developed using standard C++ with some helpful advice from the HunPos development team and additional interpretation of the OCaml source of HunPos tagger itself.

At this moment, the tagger implements only a second order hidden Markov model tagging paradigm (trigram tagging), utilizing a modified version of the Viterbi algorithm (Thede and Harper 1999), linear interpolation, successive abstraction and deleted interpolation as smoothing and default unknown word handling paradigms. These are de facto standard methods, also found in both TnT and HunPos. CroTag presumes token emission upon reached state and is trained as a visible Markov model, i.e. on pre-tagged corpora, from which it acquires transition and emission probability matrices, as described in e.g. (Manning and Schütze 1999).

Input and output formats of CroTag are once again virtually identical to ones of TnT and HunPos The training procedure takes a verticalized, sentence delimited corpus and creates the language model – i.e. tag transition and token emission probability matrices – while the tagging procedure takes as input a verticalized, sentence delimited, non-tagged text and utilizes the language model matrices to provide an output formatted identical to that required for training input: verticalized text containing a token and MSD per line.

Since CroTag is still under heavy development taking several different implementation directions, tagging procedures do not offer any possibility of setting the parameters to the user at the moment, although implementation of these options is placed on our to-do list. Once we develop a final version of CroTag, it will be made available to the community as a web service and possibly as an open source project as well. Additional work planned for CroTag beta is discussed in section 6 together with other possible research directions.

2.3 Annotated corpus

The Croatia Weekly 100 kw newspaper corpus (CW100 corpus further in the text) consists of articles extracted from seven issues of the Croatia Weekly newspaper, which has been published from 1998 to 2000 by the Croatian Institute for Information and Culture. This 100 kw corpus is a part of Croatian side of the Croatian-English parallel corpus, as described by (Tadić 2000).

PoS	Corpus %	Different MSD
Noun	30.45	119
Verb	14.53	62
Adjective	12.06	284
Adposition	09.55	9
Conjunction	06.98	3
Pronoun	06.16	312
Other	20.27	107

Table 2.1 PoS distribution on the CW100 corpus

The CW100 corpus was manually tagged using the MULTEXT-East version 3 morphosyntactic descriptors specification, detailed in (Erjavec 2004) and encoded using XCES encoding standard (Ide et al. 2000). The corpus consists of 118529 tokens, 103161 of them being actual wordforms in 4626 different sentences, tagged by 896 different MSD tags. Nouns make for a majority of corpus wordforms (30.45%), followed by verbs (14.53%) and adjectives (12.06%), which is in fact a predictable distribution for a newspaper corpus.

Some details are provided in Table 2.1. Please note that PoS category Other includes acronyms, punctuation, numerals, etc. A more detailed insight on the CW100 corpus stats and pre-processing methods can be found in (Agić and Tadić 2006).

3 Combining lexicon and tagger

Four different methods were considered while planning this experiment. They all shared the same preconditions for input and output file processing, as described in the previous section. We now describe in theory these methods of pairing our trigram tagger and morphological lexicon.

3.1 Tagger resolving lexicon output

The first idea is based on very high text coverage displayed by the inflectional lexicon (more than 96.5% for contemporary newspaper texts documented). The text, consisting of one token per line to be tagged, could serve as input to the lexicon, providing all known MSDs given a wordform in each output line. The tagger would then be used only in context of tag sequence probabilities obtained by the training procedure and stored in the transition probability data structure. Namely, a program module could be derived from basic tagger function set, using tagger's tag transition probabilities matrix to find the optimal tag sequence in the search space, narrowed by using output of the inflectional lexicon instead of a generally poor lexical database stored in the emission probability matrix acquired at training.

3.2 Lexicon handling unknown words

A second-order HMM tagger such as CroTag is largely (almost exclusively) dependent of matrices of transition and emission probabilities, both of which are usually obtained from previously annotated corpora by a training procedure. As mentioned before, both CroTag and TnT (and HunPos, for that matter) use visible Markov model training procedures. It is well-known that it this case a large gap occurs when comparing PoS/MSD tagging accuracies on tokens known and unknown to the tagger in terms of the training procedure. If the training procedure encounters wordforms and discovers their respective tag distributions at training, error rates for tagging these words decrease substantially compared to tagging words that were not encountered at training. Improving trigram tagger accuracy therefore often means implementing an advanced method of guessing distributions of tags for unknown wordforms based on transition probabilities and other statistical methods, e.g. deleted interpolation, suffix tries and successive abstraction. Namely, TnT tagger implements all the methods listed above. However, most of these heuristic procedures frequently assign MSD tag distributions containing morphosyntactic descriptions having no linguistic sense for given unknown wordforms. We based our second method of pairing CroTag and inflectional lexicon on that fact alone; it would be worth investigating whether lexicon – as a large, high-coverage database of wordforms and associated lemmas and MSDs – could serve as unknown word handling module for the tagger at runtime. As it is expected that in most cases lexicon would recognize more word forms than tagger, implementation of this setting seemed to us as a logical and feasible course of action.

Suffix trie	Lexicon	Distribution
$p(tag_{i1} \mid suff_i)$	$(w_i\ l_1\ tag_1)$	
...	...	
$p(\boxed{tag_{ii}} \mid suff_i)$	$(w_i\ l_1\ \boxed{tag_{ii}})$	$p'(tag_{ii} \mid w_i)$
$p(\boxed{tag_{ij}} \mid suff_i)$	$(w_i\ l_1\ \boxed{tag_{ij}})$	$p'(tag_{ij} \mid w_i)$
$p(\boxed{tag_{ik}} \mid suff_i)$	$(w_i\ l_2\ \boxed{tag_{ik}})$	$p'(tag_{ik} \mid w_i)$
...	...	$\sum p' = \sum p$
$p(tag_{in} \mid suff_i)$	$(w_i\ l_m\ tag_1)$	

Table 3.1 Lexicon improving the suffix trie

In more detail, the idea builds on (Halacsy et al. 2006) and (Halacsy et al. 2007) and is basically a simple extension of the unknown word handling paradigm using suffix tries and successive abstraction (Samuelsson 1993). Trigram tagger such as TnT uses algorithms to disambiguate between tags in tag lists provided by emission probability matrix for a known wordform. Upon encountering an unseen wordform, such a list cannot be found in the matrix and must be constructed from another distribution, e.g. based on wordform suffixes acquired from specific types of encountered wordforms and implemented in the suffix trie data structure. Successive abstraction module contributes by iteratively choosing a more general distribution, i.e. distribution for shorter suffixes, shortening until a distribution of tags for a matching is finally assigned to the unknown token. This results in large and consequently low-quality distributions of MSD tag probabilities for unknown word forms, resulting in lower tagging accuracy. Taking high coverage of the inflectional lexicon into consideration, our idea was to choose from the suffix trie distribution only those MSDs on which both lexicon and suffix trie intersect, falling back to suffix tries and successive abstraction alone when both lexicon and tagger fail to recognize the wordform. By this proposition, we utilize wordform and tag probabilities as given by the suffix trie and yet choose only meaningful wordform and tag pairs, i.e. pairs confirmed by reading the lexicon. Probabilities of tags that remain in distributions after the selection are recalculated, increasing and thus becoming more reliable for calculating the optimal tag sequence. Table 3.1 illustrates this principle: if suffix trie tag and lexicon tag for an unknown token match, this tag is chosen for the new emission distribution of the previously unknown wordform and emission probability is recalculated.

3.3 Lexicon as pre-processing module

In this method, we train CroTag and obtain matrices containing transition and emission probabilities. The latter one, emission probability matrix, links each of the tokens found in the training corpus to its associated tags and counts, i.e. probabilities as is shown in Figure 3.1. The figure provides an insight on similarities and differences of storing language specific knowledge of tagger and inflectional lexicon.

```
%% ...
ime 26 Ncnsa 24 Ncnsn 2
imena 8 Ncnpa 1 Ncnpg 1 Ncnpn 3 Ncnsg 3
imenima 2 Ncnpd 1 Ncnpi 1
imenom 3 Ncnsi 3
imenovan 2 Vmps-smp 2
imenovana 1 Vmps-sfp 1
imenovanja 3 Ncnpg 2 Ncnsg 1
imenovanje 1 Ncnsv 1
imenovanjem 1 Ncnsi 1
imenovanju 4 Ncnsl 4
   %% ...
```

```
%% ...
ime ime Ncnsa ime Ncnsn ime Ncnsv
imenima ime Ncnpd ime Ncnpi ime Ncnpl
imenom ime Ncnsi
%% ...
```

Figure 3.1 Emission probability matrix file and lexicon output file comparison

It was obvious that inflectional lexicon and tagger lexicon acquired by training have common properties, making it possible to create a lexicon-derived module for error detection and correction on the acquired lexicon used internally by the tagger. From another perspective, inflectional lexicon and tagger lexicon could also be merged into a single resource by some well-defined merging procedure.

3.4 Lexicon as post-processing module

Similar to using language knowledge of the inflectional lexicon before tagging, it could also be used afterwards. Output of the tagger could then be examined in the following manner:

1. Input is provided both to tagger and inflectional lexicon, each of them giving an output.
2. The two outputs are then compared, leading to several possibilities and corresponding actions:
   a. Both tagger and lexicon give an answer. Lexicon gives an unambiguous answer identical to the one provided by the tagger. No action is required.
   b. Both tagger and lexicon give an answer. Lexicon gives an unambiguous answer and it is different from the one provided by the tagger. Action is required and we choose to believe the lexicon as a manually assembled and thus preferred source of language specifics.
   c. Both tagger and lexicon give an answer. Lexicon gives an ambiguous answer, i.e. a sequence of tags. One of the tags in the sequence is identical to taggers answer. We keep the tagger's answer, being now confirmed by the lexicon.
   d. Both tagger and lexicon give an answer. Lexicon gives an ambiguous answer and none of the tags in the sequence matches the one provided by the tagger. A module should be written that takes into account the sequence provided by the lexicon and does re-tagging in a limited window of tokens in order to provide the correct answer. Basically, we define a window sized 3 tokens/tags and centred on the ambiguous token, lookup the most frequent of various trigram combinations available for the window (these are given by the lexicon!) in transition probability matrix of the tagger and assign this trigram to the window, disambiguating the output. By this we bypass tagger knowledge and once again choose to prefer lexicon output, unfortunately disregarding the fact that Viterbi algorithm outperforms this simple heuristic disambiguation.
   e. Tagger provides an answer, but token is unknown to the lexicon. We keep the tagger's answer, this being the only possible course of action.
   f. Tagger does not provide an answer and lexicon does. If its answer is unambiguous, we assign it

to the token. If it is ambiguous, we apply the procedure described in option 2d.

3. Final output produced by the merge is then investigated by the evaluation framework.

It should by all means be noted that each of the presented paradigms had to undergo a theoretical debate and possibly – if considered to be a reasonable course of action – a full sequence of tests described in section 4 in order to be accepted or rejected for introducing overall improvement of tagging accuracy or creating additional noise, respectively. Details are given in the following sections.

4 Evaluation method

As a testing paradigm, we chose the F-measure framework for evaluation on specific PoS and general accuracy for overall tagging performance. Firstly, we provide a comparison of CroTag beta and TnT: overall PoS vs. MSD accuracy and also F-measures on nouns, pronouns and adjectives, proven to be the most difficult categories in (Agić and Tadić 2006). We then discuss the proposed tagger-lexicon combinations and provide the measures – overall accuracy and F1-scores for those methods judged as suitable and meaningful at the time of conducting the investigation.

Each test consists of two parts: the worst-case scenario and the default scenario. Worst-case is a standard tagging accuracy measure scenario created by taking 90% of the CW100 corpus sentences for training and leaving the other 10% for testing. Therefore, in a way, this scenario guarantees the highest number of unknown words to be found at runtime given the corpus. The default scenario chooses 90% of sentences from the CW100 pool for training and then 10% for testing from the same pool, making it possible for sentences to overlap in these sets. The default scenario is by definition not a standard measure scenario and was introduced in order to respect the nature of random occurrences in languages, leaving a possibility (highly improbable) of tagger encountering identical sentences at training and at runtime. Also, we argue that investigating properties of errors occurring on highest accuracy scores, derived by the default testing scenario, provides additional insight on properties of trigram tagging in general.

Note that we do not include testing scenarios debating on training set size as a variable: in this test, we consider improving overall tagging accuracy and not investigating HMM tagging paradigm specifics as in (Agić and Tadić 2006), being that conclusions on this specific topic were already provided there.

5 Results

The first set of results we present is from the set of tests evaluating overall tagging accuracy of CroTag on full MULTEXT East v3 MSD and on PoS information only (by PoS we imply the first letter of the MSD tag – not comparable to English PoS of e.g. English Penn Treebank). Acquired results are displayed in Table 5.1.

It could be stated from this table that results on TnT and CroTag are virtually identical and the differences exist merely because testing environment – mainly the number of unknown words – was variable. It is however quite apparent that CroTag outperformed TnT on part-of-speech, especially regarding unknown tokens, but this should be taken with caution as well, being that CroTag dealt with fewer unknowns in that specific test.

Second testing case considers combining CroTag and the inflectional lexicon. Before presenting the results and in order to interpret them correctly, it should be stated that only two of the four initially proposed merging methods were chosen to proceed to the practical testing session: method (3.2) using the inflectional lexicon as an unknown world handler (3.4) using the inflectional lexicon as a postprocessing module to resolve potential errors produced by the tagger. We rejected applying (3.1) tagger as a disambiguation module for inflectional lexicon output because it would be costly to develop yet another tagger-derived procedure to handle transition probabilities only. This procedure would, in fact, do nothing different than a common HMM-based tagger does with its own acquired lexicon: disambiguates its ambiguous entries upon encountering them in the text and applying the transition probability matrix and handling procedures on unknown words.

		TnT		**CroTag**	
		MSD	**PoS**	**MSD**	**PoS**
Worst case	Overall	86.05	96.53	86.05	96.84
	Known	89.05	98.29	89.26	98.42
	Unknown	66.04	86.02	65.95	87.29
	Corp. unk.	13.07	14.40	13.77	14.11
Default case	Overall	97.54	98.51	97.51	99.31
	Known	98.04	98.74	98.05	99.43
	Unknown	62.21	83.11	63.75	88.39
	Corp. unk.	01.42	01.51	01.59	01.13

Table 5.1 Overall tagging accuracy on MSD and PoS

The idea of inflectional lexicon as preprocessing module (3.3) was also rejected, mainly because we were unable to define precisely how to merge its database to the one acquired by tagger at training procedure. Being that tagger training procedure assigns each entry with a number of its occurrences overall and number of occurrences under various MSDs, in order to apply the inflectional lexicon as proposed by (3.3), we would have to assign these numbers so the tagger could understand the new entries. If we assign all to 1, it does not contribute and is redundant and if we assign any other number, we are in fact altering the tagging procedure outcome in such a manner that is not in any way bound by the language model, i.e. the training corpus. Therefore, we proceed with considering proposed cases (3.2) and (3.4) only.

We have also omitted PoS results from this testing case because TnT and CroTag are both able to achieve an accuracy over 95% without additional modules so we were focused in investigating MSD accuracy, keeping in mind that most errors do not occur on PoS but on sub-PoS levels resolvable by the lexicon. Details are provided by Table 5.2.

The first apparent conclusion is that method (3.4) that cleans up the errors on tagger output has failed and that it has failed on unknown words – where we may have expected it (or hoped for it) to perform better. The reason is, on the other hand and second thought, quite obvious: the tagger applies a tag to an unknown word using transition probabilities and smoothing procedures that are proven to operate quite satisfactory in TnT, HunPos and CroTag. When the postprocessing lexicon-based module encounters a word tagged as unknown, this word is rarely unambiguous in the inflectional lexicon. Therefore, a resolution module using transition probabilities has to be applied quite frequently and this module clearly and expectedly does not outperform default unknown word handling procedures.

		TnT	CroTag +3.2	CroTag +3.4
Worst case	Overall	86.05	85.58	83.94
	Known	89.05	88.84	88.18
	Unknown	66.04	65.13	57.38
	Corp. unk.	13.07	13.77	13.77
Default case	Overall	97.54	97.97	97.88
	Known	98.04	98.53	98.51
	Unknown	62.21	63.49	59.40
	Corp. unk.	01.42	01.59	01.59

Table 5.2 Tagging accuracy with (3.2) unknown word handler and (3.4) postprocessing

Based on other stats in Table 5.2, we could end the section by stating that CroTag, when combined with the inflectional lexicon in such a manner that the lexicon provides morphological cues to the tagger upon encountering unknown words, outperforms TnT by a narrow margin on the default MSD test case. However, a more sincere and exact statement – taking in regard all section 5 tables – would be that both TnT and CroTag share the same functional dependency regarding the number of unknown words they encounter in the tagging procedure. That is, CroTag outperforms TnT when less unknown tokens occur for him at runtime and vice versa, the inflectional lexicon contributing for around 1.3% improvement on unknown words. We can thus argue that our beta-version of CroTag tagger performs as well as TnT tagger and that we succeeded in implementing a state-of-the-art solution for tagging large-scale corpora of Croatian, given the test environment we had at hands, its drawbacks noted and hereby included.

In Table 5.3 we present results of evaluation broken down by three most difficult PoS categories: adjectives, nouns and pronouns. Data and analysis is given for PoS information only, as mentioned before.

		Adjective	Noun	Pronoun
TnT	Worst case	64.56	81.63	75.42
	Default case	94.79	96.75	96.94
CroTag	Worst case	65.31	80.85	74.62
	Default case	95.86	97.40	95.88
CroTag +3.2	Worst case	66.72	82.61	77.32
	Default case	95.06	96.79	95.82

Table 5.3 Tagging accuracy with adjectives, nouns and pronouns

It can be clearly noticed that suggested combination mode (3.2) outperforms both TnT and CroTag in the worst case scenario on all parts of speech since it has the support of HML when handling unknown words, that obviously do occur somewhat more frequently in this scenario. In the default case scenario, results are expectedly more even and inconclusive – default CroTag actually outperforms lexicon combination (3.2) because unknown tokens were found in small numbers in the test sets, much too small for the inflectional lexicon to contribute significantly to overall tagging accuracy.

6 Conclusion

In this contribution we have presented CroTag – an early beta-version of statistical PoS/MSD tagger for Croatian and proposed combining it with a large scale inflectional lexicon of Croatian, creating a hybrid system for high-precision tagging of Croatian corpora. We have presented several possible types of combinations, tested and evaluated two of them using the F-measure evaluation framework. CroTag provided results virtually identical to TnT, differing only in fractions of percentage in both directions in different evaluating conditions. This way we have shown that CroTag functions at the level of state-of-the-art regarding HMM-based trigram tagging and PoS/MSD-tagging in general.

Our future directions for improvement of this system could and probably can and probably will fall into several different research pathways.

The first of them should be analyzing tagging accuracy on morphological (sub-part-of-speech) features in more detail and fine-tuning the tagger accordingly.

Various parameterization options could also be provided at tagger runtime. Such options could include parameters for unigram, bigram and trigram preference or implementing token emissions depending on previously encountered sequences (multiword unit dependencies). As was previously mentioned, once we remove the beta-version appendix from CroTag by implementing these features and optimizing and tidying its source code, it will firstly be made available as a web service and then most probably as a freely-downloadable open source project on the web.

Fine-tuned rule-based modules for Croatian language specifics could also be considered and applied before or after the statistical procedure. Another option would be integration of inflectional lexicon into tagger as they have been programmed as separate modules, inducing some overhead to execution speed.

The next direction would be to build a full lemmatizer which, unlike inflectional lexicon presented in this paper, gives fully disambiguated lemmas as output relying on the results of the tagger. Selection of proper lemmas from sets of possible ones would be done on the basis of tagger output, once again fine-tuning levels of confidence between tagger and lemmatizer similar to section 3 of the paper.

It should also be noted that (Agić and Tadić 2008) takes into account an entirely different approach, putting an emphasis on corpora development. Namely, all the methods presented in previous sections are made exclusively for handling unknown word occurrences and all of them required lots of time and human effort to be implemented. On the other hand, manual corpora development – although obviously also requiring time and effort – is by definition a less demanding and at the same time reasonable course of action: larger, better and more diverse corpora are always a necessity for any language, necessity that implicitly resolves many unknown wordform issues as well. Courses of action could therefore be argued; we decided to take most of them throughout our future work in order to additionally improve tagging accuracy on Croatian texts.

Acknowledgement

This work has been supported by the Ministry of Science, Education and Sports, Republic of Croatia, under the grants No. 130-1300646-1776, 130-1300646-0645 and 036-1300646-1986.

References

[1] Agić, Ž., Tadić, M. (2006). Evaluating Morphosyntactic Tagging of Croatian Texts. In Proceedings of the Fifth International Conference on Language Resources and Evaluation. ELRA, Genoa – Paris 2006.

[2] Agić, Ž., Tadić, M. (2008). Investigating Language Independence in HMM PoS/MSD-Tagging. In Proceedings of the 30th International Conference on Information Technology Interfaces. Cavtat, Croatia, 2008, pp. 657-662.

[3] Brants, T. (2000). TnT – A Statistical Part-of-Speech Tagger. In Proceedings of the Sixth Conference on Applied Natural Language Processing. Seattle, Washington 2000.

[4] Erjavec, T. (2004). Multext-East Version 3: Multilingual Morphosyntactic Specifications, Lexicons and Corpora. In Proceedings of the Fourth International Conference on Language Resources and Evaluation. ELRA, Lisbon-Paris 2004, pp. 1535-1538.

[5] Halácsy, P., Kornai, A., Oravecz, C. (2007). HunPos - an open source trigram tagger. In Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions. Association for Computational Linguistics, Prague, Czech Republic, pp. 209-212.

[6] Halácsy, P., Kornai, A., Oravecz, C., Trón, V., Varga, D. (2006). Using a morphological analyzer in high precision POS tagging of Hungarian. In Proceedings of 5th Conference on Language Resources and Evaluation (LREC). ELRA, pp. 2245-2248.

[7] Ide, N., Bonhomme, P., Romary, L., (2000). An XML-based Encoding Standard for Linguistic Corpora. In Proceedings of the Second International Conference on Language Resources and Evaluation, pp. 825-830. (see also at http://www.xces.org).

[8] Manning, C., Schütze, H. (1999). Foundations of Statistical Natural Language Processing, The MIT Press, 1999.

[9] Samuelsson, C. (1993). Morphological tagging based entirely on Bayesian inference. 9th Nordic Conference on Computational Linguistics NODALIDA-93. Stockholm University, Stockholm, Sweden.

[10] Šilić, A., Šarić, F., Dalbelo Bašić, B., Šnajder, J. (2007). TMT: Object-Oriented Text Classification Library. Proceedings of the 29th International Conference on Information Technology Interfaces. SRCE, Zagreb, 2007. pp. 559-566.

[11] Tadić, M. (1994). Računalna obrada morfologije hrvatskoga književnog jezika. Doctoral thesis. Faculty of Humanities and Social Sciences, University of Zagreb, 1994.

[12] Tadić, M. (2000). Building the Croatian-English Parallel Corpus. In Proceedings of the Second International Conference on Language Resources and Evaluation. ELRA, Paris-Athens 2000, pp. 523-530.

[13] Tadić, M., Fulgosi, S. (2003). Building the Croatian Morphological Lexicon. In Proceedings of the EACL2003 Workshop on Morphological Processing of Slavic Languages. Budapest 2003, ACL, pp. 41-46.

[14] Tadić, M. (2006). Croatian Lemmatization Server. Formal Approaches to south Slavic and Balkan Languages. Bulgarian Academy of Sciences, Sofia, 2006. pp. 140-146.

[15] Thede, S., Harper, M. (1999). A second-order Hidden Markov Model for part-of-speech tagging. In Proceedings of the 37th annual meeting of the Association for Computational Linguistics, pp. 175-182.

[16] Tufiş, D. (1999). Tiered Tagging and Combined Classifiers. In F. Jelinek, E. Nöth (Eds.) Text, Speech and Dialogue, Lecture Notes in Artificial Intelligence 1692, Springer, 1999, pp. 28-33.

[17] Tufiş, D., Dragomirescu, L. (2004). Tiered Tagging Revisited. In Proceedings of the 4th LREC Conference. Lisbon, Portugal, pp. 39-42.

A Heuristic Search Algorithm for Flow-Shop Scheduling

Joshua Poh-Onn Fan
Graduate School of Business
University of Wollongong, NSW, Australia
E-mail: joshua@uow.edu.au

Graham K. Winley
Faculty of Science and Technology
Assumption University, Bangkok, Thailand
E-mail: gkwinley@scitech.au.edu

Keywords: Admissible heuristic function, dominance, flow-shop scheduling, optimal heuristic search algorithm

Received: October 25, 2007

This article describes the development of a new intelligent heuristic search algorithm (IHSA*) which guarantees an optimal solution for flow-shop problems with an arbitrary number of jobs and machines provided the job sequence is constrained to be the same on each machine. The development is described in terms of 3 modifications made to the initial version of IHSA*. The first modification concerns the choice of an admissible heuristic function. The second concerns the calculation of heuristic estimates as the search for an optimal solution progresses, and the third determines multiple optimal solutions when they exist. The first 2 modifications improve performance characteristics of the algorithm and experimental evidence of these improvements is presented as well as instructive examples which illustrate the use of initial and final versions of IHSA*.

Povzetek: Opisan je nov hevristični iskalni algoritem IHSA.*

1 Introduction

The optimal solution to the flow-shop scheduling problem involving n jobs and m machines determines the sequence of jobs on each machine in order to complete all the jobs on all the machines in the minimum total time (i.e. with minimum makespan) where each job is processed on machines 1, 2, 3, …, m, in that order. The number of possible schedules is $(n!)^m$ and the general problem is NP-hard. For this general problem it is known that there is an optimal solution where the sequence of jobs is the same on the first two machines and the sequence of jobs is the same on the last two machines [4]. Consequently, for the general problem with 2 or 3 machines there is an optimal solution where the jobs are processed in the same sequence on each machine and the optimal sequence is among only n! job sequences. However, in the optimal solution for the general problem with more than 3 machines the jobs are not necessarily processed in the same sequence on each machine. This article is concerned with the development of an algorithm (IHSA*) which is guaranteed to find an optimal solution for flow-shop scheduling problems involving an arbitrary number of jobs and machines where the problem is constrained so that the same job sequence is used on each machine.

Early research on flow-shop problems is based mainly on Johnson's theorem, which gives a procedure for finding an optimal solution with 2 machines, or 3 machines with certain characteristics [20], [21]. Other approaches for the general problem include integer linear programming and combinatorial programming, which use intensive computation to obtain optimal solutions and are generally not feasible from a computational standpoint because the number of variables increases exponentially as the number of machines increases [35]. Branch-and-bound methods use upper or lower bounds to guide the direction of the search. Depending on the effectiveness of the heuristic and the search strategy this method may return only near optimal solutions but with long computation time [19], [29], [30], [36]. Heuristic methods have received significant attention [9], [18], [26], [27], [37], [40], [41], [42]. However, even the most powerful heuristic method to-date, the NEH heuristic developed by Nawaz et al. [31] fails to reach solutions within a reasonable bound of the optimal solution in some difficult problem cases [47]. A review of approaches by Zobolas et al. [47] indicates that there has been strong interest in artificial intelligence optimization methods referred to as metaheuristics including: Simulated Annealing [32], [34]; Tabu Search [11]; Genetic Algorithms, which may give an optimal solution but due to the evolutionary nature of this approach the computation time is unpredictable [2], [3], [5], [38]; Fuzzy Logic [13], [14], [15], [16], [17]; Ant Colony and Particle Swarm Optimization [28], [43]; Iterated Local Search [39]; and Differential Evolution [33]. The strong interest in metaheuristics generated the development of hybrid approaches which combine different components of more than one metaheuristic [1].

An initial version of a new intelligent heuristic search algorithm (IHSA*) for flow-shop problems with an arbitrary number of jobs and machines and subject to the constraint that the same job sequence is used on each machine has been proposed in [6], [7], [8]. It is based on the Search and Learning A* algorithm presented in [44], [45], [46] which is a modified version of the Learning Real Time A* algorithm in [24], [25] which is, in turn, a modified version of the original A* algorithm [10], [12].

At the start of the search using IHSA* different methods are considered for computing estimates for the total time needed to complete all of the jobs on all of the machines assuming in turn that each of the jobs is placed first in the job sequence. It is shown that if there are m machines then there are m different methods that should be considered. Among the estimates associated with each method the smallest estimate is referred to as the value of the heuristic function that is associated with that method and it identifies the job that would be placed first in the job sequence at the start of the search if that method is used. If the value of the heuristic function does not exceed the minimum makespan for the problem then the heuristic function is said to be admissible and in such cases IHSA* is guaranteed to find an optimal solution provided the job sequence is the same on each machine. The proof of this result for IHSA* is given in [8] and is similar to that given in [22] and [23] in relation to the A* algorithm from which IHSA* is derived. The term "heuristic" is used in the title of IHSA* because the optimality of the algorithm and its performance depends on the selection of an appropriate admissible heuristic function at the start of the search and this function continues to guide the search to an optimal solution.

The purpose of this article is to describe the development of IHSA* which has occurred since the initial version was first presented in [6]. For simplicity of presentation the development is described in terms of problems involving an arbitrary number of jobs with 3 machines. However, the notations, definitions, proofs, and concepts presented may be extended to problems involving more than 3 machines if the job sequence is the same on each machine and these extensions are noted at the appropriate places throughout the presentation. Three significant modifications have been made to the initial version of IHSA*. The first concerns the choice of an admissible heuristic function at the start of the search. The second concerns the calculation of heuristic estimates as the search progresses, and the third determines multiple optimal solutions when they exist.

Following an introduction to the initial version of IHSA* each of the 3 modifications is presented. Experimental evidence of improvements in performance characteristics of the algorithm which result from the first 2 modifications is provided in the Appendix and discussed in section 5. Instructive examples are given to illustrate the initial and final versions of IHSA* and these have been limited to 3 machines in order to allow interested readers to familiarize themselves with the algorithm by reworking the examples by hand. The proofs of results related to the modifications are presented in the Appendix.

2 The initial version of IHSA*

Before presenting the initial version of the algorithm notations and definitions are introduced and the state transition process associated with IHSA* is described together with the features of search path diagrams which are used to illustrate the development of an optimal job sequence.

2.1 Notations and definitions

The following notations and definitions are introduced for a flow-shop problem involving n jobs $J_1, J_2, \ldots, J_n$ and 3 machines M_1, M_2, M_3.

O_{ij} is the operation performed on job J_i by machine M_j and there are 3n operations. For job J_i the processing times a_i, b_i, and c_i denote the times required to perform the operations O_{i1}, O_{i2}, and O_{i3}, respectively and these processing times are assumed to be non negative integers. If O_{ij} has commenced but has not been completed then p_{ij} represents the additional time required to complete O_{ij} and at the time when O_{ij} starts p_{ij} is one of the values among a_i, b_i, or c_i. The sequence $\phi_{st} = \{J_s, \ldots, J_t\}$ represents a sequence of the n jobs with J_s scheduled first and J_t scheduled last. $T(\phi_{st})$ is the makespan for the job sequence ϕ_{st} and $S(\phi_{st})$ is the time at which all of the jobs in ϕ_{st} are completed on machine M_2.

Using these notations and definitions Figure 1 illustrates the manner in which the operations associated with the job sequence ϕ_{st} are performed on the 3 machines.

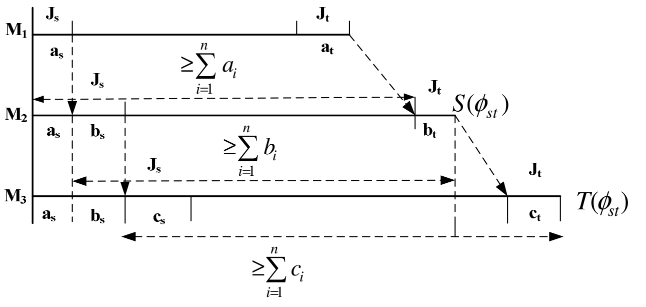


Figure 1: Processing the Job Sequence ϕ_{st}

A method for calculating an estimate of the total time to complete all of the n jobs on all of the 3 machines when job J_i is the first job in the sequence is given by $a_i + b_i + \sum_{i=1}^{n} c_i$. Then the heuristic function associated with this method is H_3 where,

$$H_3 = \min\,[a_1 + b_1, a_2 + b_2, \ldots, a_s + b_s, \ldots, a_n + b_n] + \sum_{i=1}^{n} c_i . \qquad (1)$$

If $\min\,[a_1 + b_1, a_2 + b_2, \ldots, a_s + b_s, \ldots, a_n + b_n] = a_s + b_s$ then $H_3 = a_s + b_s + \sum_{i=1}^{n} c_i$ and job J_s would be scheduled first in the job sequence. It is seen from Figure 1 and proved in the Appendix that H_3 is an admissible heuristic function and H_3 is the heuristic function that was used in the initial version of the IHSA*.

2.2 The state transition process

The procedure for developing the optimal job sequence using IHSA* proceeds by selecting an operation which may be performed next on an available machine. At the time that selection is made each of the 3n operations is in only one of 3 states: the *not scheduled* state; the *in-progress* state; or the *finished* state and the operation which is selected is among those in the *not scheduled* state. Operations not in the *finished* state are referred to as incomplete. A state transition occurs when one or more of the operations move from the *in-progress* state to the *finished* state and at any time the state level is the number of operations in the *finished* state.

IHSA* describes the procedure which takes the state transition process from one state level to the next and the development of the optimal job sequence is illustrated graphically using search path diagrams.

2.3 Search path diagrams

A search path diagram consists of nodes drawn at each state level with one of the nodes connected to nodes at the next state level. Each node contains 3 cells which are used to display information about operations on the machines M_1, M_2, and M_3, respectively. When a state transition occurs one of the nodes at the current state level is expanded and it is connected to the nodes at the next state level where each node represents one of the different ways of starting operations that are in the *not scheduled* state. At each of these nodes a cell is labeled with J_i to indicate that the operation O_{ij} is either in progress or is one of the operations that may start on M_j, and p_{ij} which is the time needed to complete J_i on M_j. A blank cell indicates that no operation can be performed on that machine at this time.

Near each node the heuristic estimate (h) associated with the node is recorded. The heuristic estimate is calculated using the heuristic function chosen in Step 1 of the algorithm in conjunction with the procedure used in Step 2. It is an estimate of the time required to complete the operation at the node identified by the procedure in Step 2 as well as all of the other operations that are not in the *finished* state. At each state level the node selected for expansion is the one which has associated with it the minimum heuristic estimate among all of the estimates for the nodes at that state level. Near this selected node the value of f = h + k is recorded where the edge cost (k) is the time that has elapsed since the preceding state transition occurred.

A comparison is made between f and h' where h' is the minimum heuristic estimate at the preceding state level. Based on that comparison the search path either backtracks to the node expanded at the preceding state level or moves forward to the next state level. If backtracking occurs then the value of h' is changed to the current value of f and the search moves back to that node. If the path moves forward then the value of the edge cost (k) is recorded below the expanded node. For convenience of presentation a new search path diagram is drawn when backtracking in the previous diagram is completed.

At state level 0 there are n root nodes corresponding to the number of jobs. The final search path diagram represents the optimal solution and traces a path from one of the root nodes, where the minimum makespan is the value of h or f, to a terminal node where h = f = 0. The optimal job sequence can be read by recording the completed operations along the path from the root node to the terminal node.

2.4 IHSA* (initial version)

Step 1: At state level 0 expand the node identified by calculating the value of H_3 from (1), and move to the nodes at state level 1. If more than one node is identified then break ties randomly.

For example, if $H_3 = a_s + b_s + \sum_{i=1}^{n} c_i$ then the node with J_s and a_s recorded in the first cell is the node to be expanded.

Step 2: At the current state level if the heuristic estimate of one of the nodes has been updated by backtracking use the updated value as the heuristic estimate for that node and proceed to Step 3. Otherwise, calculate a heuristic estimate for each node at the current state level using Procedure 1 and proceed to Step 3.

Procedure 1 is described below.

Step 3: At the current state level select the node with the smallest heuristic estimate. If it is necessary then break ties randomly.

The smallest heuristic estimate is admissible and underestimates the minimum time required to complete all of the incomplete jobs on all of the machines.

Step 4: Calculate f = h + k where h is the smallest heuristic estimate found in Step 3 and k (edge cost) is the time that has elapsed since the preceding state transition occurred.

Step 5: If $f > h'$, where h' is the minimum heuristic estimate calculated at the preceding state level, then backtrack to that preceding state level and increase the value of h' at that preceding node to the current value of f and repeat Step 4 at that node.

Step 6: If $f \le h'$ then proceed to the next state level and repeat from Step 2.

Step 7: If f = 0 and h = 0 then Stop.

Procedure 1 is used in Step 2 to calculate a heuristic estimate for each node at the current state level:

(a) If cell 1 is labelled with J_i then the heuristic estimate h for the node is based on the operation in cell 1 and is given by,

$$h = \begin{cases} a_i + b_i + C_1; \text{ for } O_{i1} \text{ in the } \textit{not scheduled} \text{ state,} \\ p_{i1} + b_i + c_i + C_1; \text{ for } O_{i1} \text{ in the } \textit{in-progress} \text{ state,} \end{cases} \quad (2)$$

where C_1 is the sum of the values of c_k for all values of k such that O_{k1} is in the *not scheduled* state.

(b) If cell 1 is blank, and cell 2 is labelled with J_i then the heuristic estimate h for the node is based on the operation in cell 2 and is given by,

$$h = \begin{cases} b_i + C_2; \text{ for } O_{i2} \text{ in the } \textit{not scheduled} \text{ state,} \\ p_{i2} + c_i + C_2; \text{ for } O_{i2} \text{ in the } \textit{in-progress} \text{ state,} \end{cases} \quad (3)$$

where C_2 is the sum of the values of c_k for all values of k such that O_{k2} is in the *not scheduled* state.

(c) If cell 1 and cell 2 are blank, and cell 3 is labelled with J_i then the heuristic estimate h for the node is based on the operation in cell 3 and is given by,

$$h = \begin{cases} C_3; \text{ for } O_{i3} \text{ in the } \textit{not scheduled} \text{ state,} \\ p_{i3} + C_3; \text{ for } O_{i3} \text{ in the } \textit{in-progress} \text{ state,} \end{cases} \quad (4)$$

where C_3 is the sum of the values of c_k for all values of k such that O_{k3} is in the *not scheduled* state.

In Procedure 1 the calculation of a heuristic estimate for a node is based on an operation in only one of the cells at the node and operations in the other 2 cells are not taken into account. For example, if cell 1 is not blank then the estimate is based only on the operation in cell 1. If cell 1 is blank then the estimate is based only on the operation in cell 2. An operation in cell 3 is only considered if the other 2 cells are blank.

The following example illustrates the use of the initial version of IHSA* to solve the flow-shop problem given in Table 1. For instructional purposes the problem is deliberately simple with only 3 machines and 3 jobs because it is intended to provide readers with an opportunity to become familiar with the algorithm using an example that can be reworked easily by hand.

Table 1: Example of a Flow-shop Problem

Jobs/Machines	M_1	M_2	M_3
J_1	2	1	10
J_2	4	6	5
J_3	3	2	8

For illustrative purposes only the first search path diagram is presented in Figure 2.

At the start of the search H_3 = 26. In total 5 search path diagrams are required to find the optimal sequence $J_1J_2J_3$ with a minimum makespan of 26. Twenty nodes are expanded, 16 backtracking steps are required, and 43 steps of the algorithm are executed.

3 Modifications to the initial version of IHSA*

The first modification determines the best heuristic function to use for a given problem and affects Step 1 of the algorithm. The second modification affects Procedure 1 used in Step 2 and the third modification affects Step 7 and enables multiple optimal solutions to be found when they exist.

3.1 A modification to step 1

In the Appendix a set of 6 heuristic functions are derived for the case of 3 machines and proofs of the admissibility of these functions are presented. It is shown that among this set of 6 heuristic functions the one which

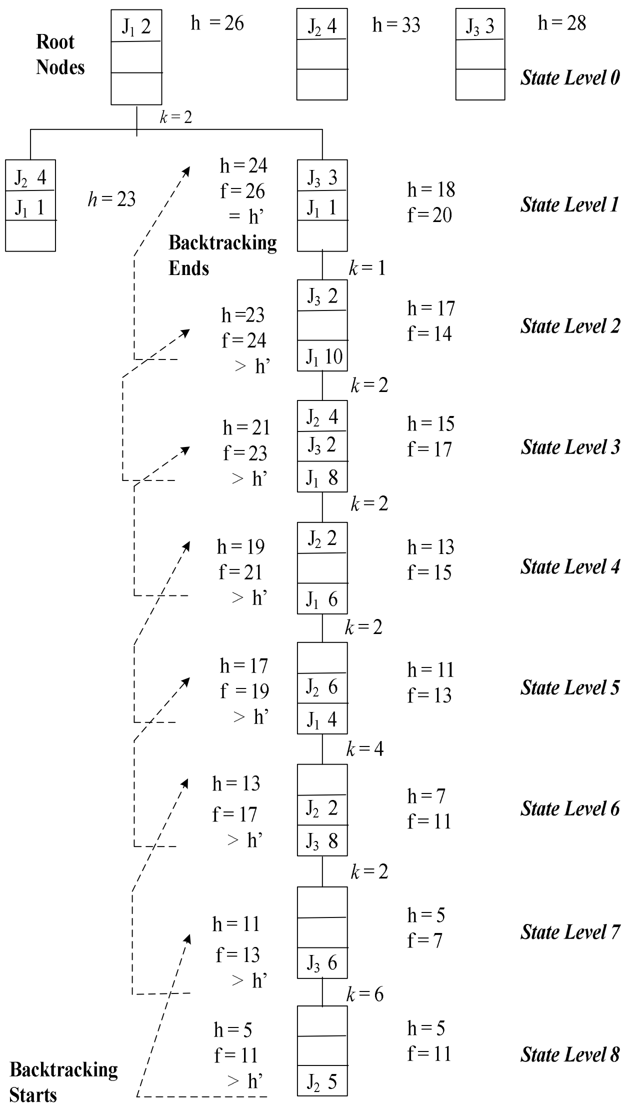


Figure 2: The First Search Path Diagram Using the Initial Version of IHSA*

is admissible and has a value which is closest to the minimum makespan will be the one among H_1, H_2, and H_3 which has the largest value where,

$$\left.\begin{aligned} H_1 &= \min[b_1 + c_1, b_2 + c_2, \ldots, b_n + c_n] + \sum_{i=1}^{n} a_i, \\ H_2 &= \min[a_1 + u_1, a_2 + u_2, \ldots, a_n + u_n] + \sum_{i=1}^{n} b_i, \\ H_3 &= \min[a_1 + b_1, a_2 + b_2, \ldots, a_n + b_n] + \sum_{i=1}^{n} c_i, \end{aligned}\right\} \quad (5)$$

where: $u_1 = \min[c_2, c_3, \ldots, c_n]$; $u_k = \min[c_1, c_2, \ldots, c_{k-1}, c_{k+1}, \ldots, c_n]$, for $2 \le k \le n - 1$; and $u_n = \min[c_1, c_2, c_3, \ldots, c_{n-1}]$.

Choosing the admissible heuristic function among H_1, H_2, and H_3 with the largest value in the first step of the algorithm ensures that the search begins with an estimate of the minimum makespan that is not greater than it but is the closest to it. This choice is expected to reduce the need for backtracking at a subsequent stage of the search since backtracking takes the search back to a previous node and increases the heuristic estimate at that node to a value which is admissible but closer to the minimum time required to complete all of the incomplete jobs on all of the machines.

Consequently, Step 1 in the initial version of IHSA* is modified and becomes:

Step1: At state level 0, from (5), choose the admissible heuristic function among H_1, H_2, and H_3 which has the largest value and if necessary break ties randomly. In the case where machine M_j dominates the other machines select H_j. Expand the node identified by the chosen admissible heuristic function and move to the nodes at state level 1. If more than one node is identified then break ties randomly.

The example in section 4 below illustrates the use of the modification to Step 1 in a simple problem which enables the reader to rework the example by hand. In the Appendix it is shown that in the particular case where machine M_j dominates the other machines then the best admissible heuristic function among H_1, H_2, and H_3 is H_j and it has a value which is greater than the value of either of the other two functions by at least $(n – 1)(n – 2)$ where n is the number of jobs. Thus in the case of a dominant machine in the first step of the algorithm there is no need to calculate each of the values of H_1, H_2, and H_3 in order to choose the one with the largest value. Instead, it is known that it will be H_j if machine M_j is the dominant machine.

In the case where there are m machines and $m > 3$ it is shown in the Appendix that the best admissible heuristic function to use in the first step of the algorithm is the one among $F_1, F_2, \ldots, F_m$ which has the largest value and if $m = 3$ then $F_1 = H_1$, $F_2 = H_2$, and $F_3 = H_3$. Experimental evidence of improvements in performance characteristics of the algorithm which result from using the modification to Step 1 is presented in the Appendix Table A1 and is discussed below in section 5.

3.2 A modification to step 2

The second modification to the initial version of IHSA* concerns the calculation of heuristic estimates at nodes on the search path when the search has commenced. It is based on the principle that when heuristic estimates for the nodes at the same state level are being calculated in Step 2 it is desirable to obtain the largest possible estimate at each of these nodes before selecting the node with the smallest estimate as the node to be expanded. The larger the value of this smallest estimate then the less likely it is that the search will need to backtrack and this is expected to improve the performance characteristics of the algorithm. As noted above, the use of Procedure 1 in Step 2 in the initial version of IHSA* gives a heuristic estimate for a node based on an operation in only one of the cells while operations in the other 2 cells are not taken into account.

The second modification affects Procedure 1 and involves calculating heuristic estimates h_1, h_2, and h_3 at a node for cells 1, 2, and 3, respectively. Then $\max[h_1, h_2, h_3]$ is used as the heuristic estimate (h) for the node. This is done for each node at the current state level and then, as before, in Step 3 the minimum estimate among these estimates identifies the node to be expanded. Consequently, Procedure 1 is replaced by the following Procedure 2:

In Step 2 of the algorithm for a node at the current state level,

$$\left.\begin{array}{l} \text{(a) For cell 1: If the cell is blank then } h_1 = 0. \\ \quad\text{Otherwise, } h_1 \text{ is given by (2).} \\ \text{(b) For cell 2: If the cell is blank then } h_2 = 0. \\ \quad\text{Otherwise, } h_2 \text{ is given by (3).} \\ \text{(c) For cell 3: If the cell is blank then } h_3 = 0. \\ \quad\text{Otherwise, } h_3 \text{ is given by (4).} \end{array}\right\} \qquad (6)$$

Procedure 2 refers to calculations specified in Procedure 1 which incorporate currently the heuristic function H_3. However, (2), (3), and (4) are easily changed to incorporate H_1 or H_2 for problems where one of these functions has been selected using the modification to Step 1 of the algorithm. For example, if H_2 is used then (3) and (4) are not changed but (2) becomes,

$$h = \begin{cases} a_i + w_i + B_1;\ \text{for } O_{i1} \text{ in the } \textit{not scheduled} \text{ state,} \\ p_{i1} + w_i + b_i + B_1;\ \text{for } O_{i1} \text{ in the } \textit{in-progress} \text{ state,} \end{cases}$$

where, B_1 is the sum of the values of b_k for all values of $k \neq i$ such that O_{k1} is in the *not scheduled* state and w_i is the smallest value of c_k for all values of $k \neq i$ such that O_{k1} is in the *not scheduled* state.

Procedure 2 produces the same heuristic estimate as Procedure 1 if and only if one of the following 3 conditions is satisfied: $h_1 = \max[h_1, h_2, h_3]$; $h_2 = \max[h_1, h_2, h_3]$ and cell 1 is blank; or $h_3 = \max[h_1, h_2, h_3]$ and cells 1 and 2 are blank. Under any other conditions Procedure 2 will produce a heuristic estimate at a node which is larger than the estimate given by Procedure 1. Consequently, using Procedure 2 will never produce an estimate that is less than the estimate produced by Procedure 1 and in practice the estimate using Procedure 2 is usually larger and leads to a reduction in backtracking.

Using Procedure 2 in the initial version of IHSA* modifies Step 2 and it becomes:

Step 2: At the current state level if the heuristic estimate of one of the nodes has been updated by backtracking use the updated value as the heuristic estimate for that node and proceed to Step 3. Otherwise, at each node use Procedure 2 to calculate h_1, h_2, h_3 and use $\max[h_1, h_2, h_3]$ as the heuristic estimate for the node.

If there are m machines and $m > 3$ then there are m cells at each node and an estimate is calculated for each cell by extending (6) and (2), (3), (4) to accommodate the heuristic function F_j (see Appendix) used in Step 1 of the algorithm.

The example in section 4 below illustrates the use of the modification to Step 2 in a simple problem which enables the reader to rework the example by hand. Experimental evidence of additional improvements in performance characteristics of the algorithm from using the modifications to Steps 1 and 2 together is presented in the Appendix Table A1 and is discussed below in section 5.

3.3 A modification to step 7

For some problems there are multiple optimal solutions and it is often important in practical situations to be able to find all of the optimal solutions since it may be required to find an optimal solution that also satisfies other criteria. For example, an optimal solution may be sought which also has the least waiting time for jobs that are queuing to be processed.

When IHSA* is implemented there are 2 situations which indicate the possible existence of multiple optimal solutions. The first situation occurs when there is more than 1 node at a state level with the smallest heuristic estimate. In this case the ties are broken randomly and one of the nodes is selected for expansion and the search continues and produces an optimal solution. At the completion of the search returning to that state level and selecting for expansion one of the other nodes which were not selected when the ties were broken may lead to a different optimal solution. The second situation occurs when at the completion of the search for an optimal solution one or more of the nodes at state level 0 has a heuristic estimate that is less than or equal to the minimum makespan. In this case returning to those nodes and beginning the search again may produce different optimal solutions.

The modification to the initial version of IHSA* that enables multiple optimal solutions to be determined affects Step 7. This modification is different from the previous 2 modifications in that it does not improve the performance characteristics of the algorithm but instead it is intended to find multiple optimal solutions if they exist.

Consequently, Step 7 becomes:

Step 7: If $f = 0$ and $h = 0$ then an optimal solution has been found. If along the path representing the optimal solution there is a node which was selected for expansion by breaking ties randomly among nodes at the same state level with the same minimum heuristic estimate then return to that state level and repeat from Step 2 ignoring any node that was selected previously for expansion as a result of breaking ties. If any of the values of h at root nodes (state level 0) is less than or equal to the minimum makespan then return to state level 0 and repeat from Step 2 ignoring root nodes that lead to a previous optimal solution. Otherwise, Stop.

4 The final version of IHSA*

The final version of IHSA* incorporates each of the 3 modifications:

Step1: At state level 0, from (5), choose the admissible heuristic function among H_1, H_2, and H_3 which has the largest value and if necessary break ties randomly. In the case where machine M_j dominates the other machines select H_j. Expand the node identified by the chosen admissible heuristic function and move to the nodes at state level 1. If more than one node is identified then break ties randomly.

Step 2: At the current state level if the heuristic estimate of one of the nodes has been updated by backtracking use the updated value as the heuristic estimate for that node and proceed to Step 3. Otherwise, at each node use Procedure 2 to calculate h_1, h_2, h_3 and use max[h_1, h_2, h_3] as the heuristic estimate for the node.

Step 3: At the current state level select the node with the smallest heuristic estimate. If it is necessary then break ties randomly.

Step 4: Calculate $f = h + k$ where h is the smallest heuristic estimate found in Step 3 and k (edge cost) is the time that has elapsed since the preceding state transition occurred.

Step 5: If $f > h'$, where h' is the minimum heuristic estimate calculated at the preceding state level, then backtrack to that preceding state level and increase the value of h' at that preceding node to the current value of f and repeat Step 4 at that node.

Step 6: If $f \le h'$ then proceed to the next state level and repeat from Step 2.

Step 7: If $f = 0$ and $h = 0$ then an optimal solution has been found. If along the path representing the optimal solution there is a node which was selected for expansion by breaking ties randomly among nodes at the same state level with the same minimum heuristic estimate then return to that state level and repeat from Step 2 ignoring any node that was selected previously for expansion as a result of breaking ties. If any of the values of h at root nodes (state level 0) is less than or equal to the minimum makespan then return to state level 0 and repeat from Step 2 ignoring root nodes that lead to a previous optimal solution. Otherwise, Stop.

If there are m machines and $m > 3$ then Steps 1 and Step 2 need to be modified in accordance with the discussion of this case presented in sections 3.1 and 3.2 above.

The simple instructive example which was used to illustrate the initial version of IHSA* (see Table 1) is used again to illustrate the final version of IHSA*. For this problem, from (5), $H_3 = 26 > H_1 = 19 > H_2 = 16$ and using the modification to Step 1 of the algorithm H_3 is used in Step 1 of the algorithm. Since no backtracking is necessary an optimal solution for the problem requires only 1 search path diagram which is shown in Figure 3 where the optimal solution has a minimum makespan of 26 and a job sequence J_1, J_2, J_3. At each node the search path diagram shows the estimates h_1, h_2, h_3 and the heuristic estimate for the node h = max[h_1, h_2, h_3] which result from the use of the modification to Step 2 of the algorithm.

It is noted that the minimum heuristic estimate at state level 1 is 24 at both of the nodes at that state level. In Step 3 of the algorithm the tie was broken randomly and the node at which job J_2 is scheduled on machine M_1 was selected for expansion. In Step 7, although for simplicity a second search path diagram has not been drawn, the search returns to state level 1 and instead the

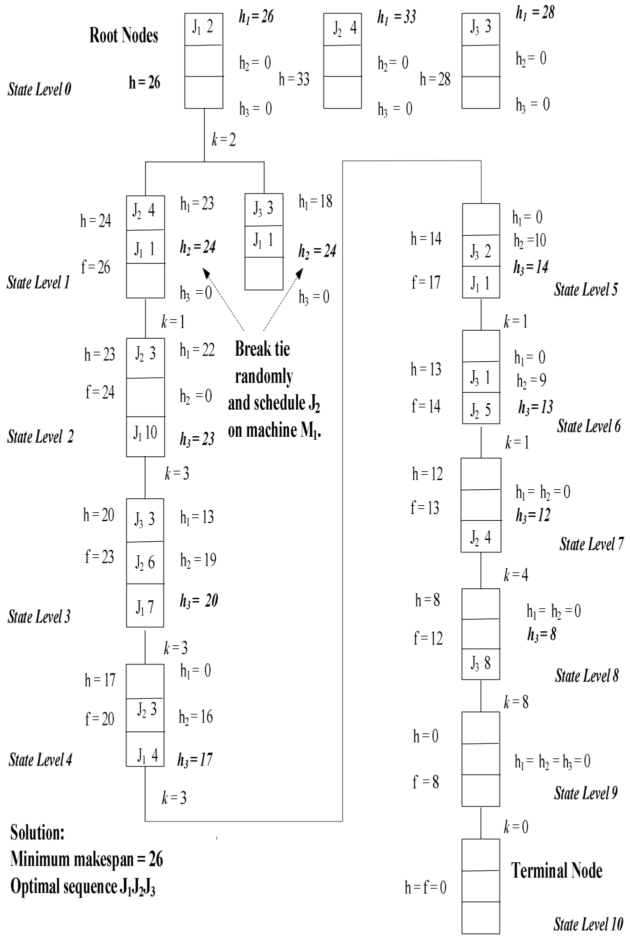


Figure 3: Search Path Diagram Using the Final Version of IHSA*

node at which job J_3 is scheduled on machine M_1 is expanded. This gives a second optimal solution where the job sequence is J_1, J_3, J_2.

5 Experimental evidence of improvements in performance

Experimental evidence of improvements in performance characteristics of IHSA* using the modifications to Steps 1 and 2 is presented in Appendix Table A1. The characteristics considered are: the number of nodes expanded; the number of backtracking steps required; and the number of steps of the algorithm executed.

In total 14 problems are considered involving: 3, 5 and 10 machines; and 3, 4, 10, 15, and 40 jobs. Each problem involving 3 machines was solved using the heuristic functions H_1, H_2, and H_3 in (5) which are the same as F_1, F_2, and F_3, respectively, when m = 3. Problems involving 5 and 10 machines were solved using their corresponding heuristic functions F_1, F_2, F_3, F_4, F_5 and F_1, F_2, F_3, …, F_{10}, respectively. The solutions enabled improvements in the performance characteristics resulting from the use of only the modification to Step 1 to be assessed. In addition, for each problem the solution was obtained using the modification to Step 1 together with the modification to Step 2. The performance characteristics associated with each of these solutions enabled an assessment of any further improvements in performance characteristics resulting from the inclusion of the modification to Step 2.

From Table A1 it is seen that for each problem regardless of the number of jobs and machines the modification to Step 1, which involves using the heuristic function with the largest value in Step 1, leads to improvements in all of the performance characteristics. Furthermore, in each problem using the modification to Step 1 together with the modification to Step 2, which affects the calculation of heuristic estimates as the search progresses, leads to further improvements in the performance characteristics.

6 Conclusion

Three modifications to the initial version of a new intelligent heuristic search algorithm (IHSA*) have been described. The algorithm guarantees an optimal solution for flow-shop problems involving an arbitrary number of jobs and machines provided the job sequence is the same on all of the machines.

The first modification affects Step 1 of the algorithm and concerns the choice of an admissible heuristic function which is as close as possible to the minimum makespan for the problem. For problems with an arbitrary number of jobs and 3 machines (M_1, M_2, M_3) a set of 6 possible functions is derived (H_1, H_2, …, H_6) and their admissibility is proved. It is shown that the function which has a value that is closest to the minimum makespan and is the best function to use in Step 1 of the algorithm is the function among H_1, H_2, and H_3 which has the largest value. In the particular case where one of the machines (M_j) dominates the other 2 machines the best function is H_j and there is no need to calculate the values of the other 2 functions. Furthermore, its value is greater than the value of either of the other 2 functions by at least $O(n^2)$ where n is the number of jobs. More generally, for problems with more than 3 machines (M_1, M_2, …, M_m) the best admissible heuristic function to use is the one among F_1, F_2, …, F_m with largest value and if machine M_j dominates the other machines then F_j is the best heuristic function. The proofs of these more general results may be obtained following the methods used in the proofs presented in the Appendix of the corresponding results for H_1, H_2, and H_3.

The second modification changes the procedure used in Step 2 of the initial version of the algorithm to determine heuristic estimates at nodes on the search path. The initial version determines a heuristic estimate at a node by considering an operation in only one of the cells at the node while operations in the other cells are not taken into account. The modified procedure determines a heuristic estimate at a node by selecting the largest of the separate estimates calculated for each cell at the node. The modified procedure never produces an estimate for a node that is smaller than the estimate produced by the procedure used in the initial version of the algorithm and in many cases it will be larger.

The first and second modifications ensure that at the start of the search and as the search progresses heuristic estimates are admissible and are as close as possible to the minimum time needed to complete all of the incomplete operations on all of the machines. This reduces the chance that the search will backtrack and improves the performance characteristics of the algorithm. Experimental evidence from problems involving various numbers of machines and jobs indicates that although the first modification produces improvements in performance characteristics of the algorithm these improvements are enhanced when the second modification is included.

The third modification relates to Step 7 of the algorithm and concerns problems where there are multiple optimal solutions. It enables all of the optimal solutions to be found and this is convenient for situations where additional criteria may need to be satisfied by an optimal solution.

This article has focussed on describing the development of the final version of IHSA*. However, there are several areas for future investigation including a comparison of the performance of the algorithm with other methods such as branch- and-bound methods and methods for pruning the search tree in order to improve memory management during implementation.

References

[1] Blum, C., Roli, A. "Metaheuristics in combinatorial optimization: overview and conceptual comparison," *ACM Comput. Surv.*, 35, 2003, 268-308.

[2] Chen, C.L., Neppalli, R.V., Aljaber, N. "Genetic algorithms applied to the continuous flow shop problem," *Computers and Industrial Engineering* 30: (4), 1996, 919-929.

[3] Cleveland, G.A., Smith, S.F. "Using genetic algorithms to schedule flow shop," Proceedings of 3rd Conference on Genetic Algorithms, Schaffer, D.(ed.), San Mateo: Morgan Kaufmann Publishing, 1989, 160-169.

[4] Conway, R.W., Maxwell, W.L., Miller, L.W. Theory of scheduling, Addison-Wesley, Reading Massachusetts, 1967.

[5] Eitler, O., Toklu, B., Atak, M., Wilson,J. "A genetic algorithm for flowshop scheduling problems," J. Oper. Res. Soc., 55, 2004, 830-835.

[6] Fan, J.P.-O. "The development of a heuristic search strategy for solving the flow-shop scheduling problem," Proceedings of the IASTED International Conference on Applied Informatics, Innsbruck, Austria, 1999, 516-518.

[7] Fan, J.P.-O. "An intelligent search strategy for solving the flow-shop scheduling problem," Proceedings of the IASTED International Conference on Software Engineering, Scottsdale, Arizona, USA, 1999, 99-103.

[8] Fan, J.P.-O. An intelligent heuristic search method for flow-shop problems, doctoral dissertation, University of Wollongong, Australia, 2002.

[9] Framinan, J.M, Ruiz-Usano, R., Leisten, R. "Sequencing CONWIP flow-shops: analysis and heuristic," Int. J. Prod. Res., 39, 2001, 2735-2749.

[10] Gheoweth, S.V., Davis, H.W. "High performance A* search using rapidly growing heuristics," Proceedings of the International Joint Conference on Artificial Intelligence, Sydney, Australia, 1991, 198-203.

[11] Grabowski, J., Wodecki, M. "A very fast tabu search algorithm for the permutation flowshop problem with makespan criterion," Comput. Oper. Res., 31, 2004, 1891-1909.

[12] Hart, P.E., Nilsson, N.J., Raphael, B. "A formal basis for the heuristic determination of minimum cost paths," IEEE Transactions on Systems Science and Cybernetics, Vol. SSC-4: (2), 1968, 100-107.

[13] Hong, T.P., Chuang, T.N. "Fuzzy scheduling on two-machine flow shop," Journal of Intelligent & Fuzzy Systems, 6: (4), 1998, 471-481.

[14] Hong, T.P., Chuang, T.N. "Fuzzy CDS scheduling for flow shops with more than two machines," Journal of Intelligent & Fuzzy Systems, 6: (4), 1998, 471-481.

[15] Hong, T.P., Chuang, T.N. "Fuzzy Palmer scheduling for flow shops with more than two machines," Journal of Information Science and Engineering, Vol.15, 1999, 397-406.

[16] Hong, T.P., Wang, T.T. "A heuristic Palmer-based fuzzy flexible flow-shop scheduling algorithm," Proceedings of the IEEE International Conference on Fuzzy Systems, Vol. 3, 1999, 1493-1497.

[17] Hong, T.P., Huang, C.M., Yu, K.M. "LPT scheduling for fuzzy tasks," Fuzzy Sets and Systems, Vol. 97, 1998, 277-286.

[18] Hong, T.P., Wang, C.L., Wang, S.L. "A heuristic Gupta-based flexible flow-shop scheduling algorithm," Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, Vol. 1, 2000, 319-322.

[19] Ignall, E., Schrage, L.E. "Application of the branch and bound technique to some flow shops scheduling problems," Operations Research, Vol. 13: (3), 1965, 400-412.

[20] Johnson, S.M. "Optimal two- and three-stage production schedules with setup times included," Naval Research Logistics Quarterly, 1: (1), 1954, 61-68.

[21] Kamburowski, J. "The nature of simplicity of Johnson's algorithm," Omega-International Journal of Management Science, 25: (5), 1997, 581-584.

[22] Korf, R.E. "Depth-first iterative-deepening: an optimal admissible tree search," Artificial Intelligence, Vol. 27, 1985, 97-109.

[23] Korf, R.E. "Iterative-deepening A*: an optimal admissible tree search," Proceeding of the 9th International Joint Conference on Artificial Intelligence, Los Angeles, California, 1985, 1034-1036.

[24] Korf, R.E. "Real-time heuristic search," Artificial Intelligence, Vol. 42, 1990, 189-211.

[25] Korf, R.E. "Linear-space best-first search," Artificial Intelligence, 62: (1), 1993, 41-78.

[26] Lai, T.C. "A note on heuristics of flow-shop scheduling," Operations Research, 44: (6), 1996, 648-652.

[27] Lee, G.C., Kim, Y.D., Choi, S. W. "Bottleneck-focused scheduling for a hybrid flow-shop," Int. J. Prod. Res., 42, 2004, 165-181.

[28] Liu, B., Wang, L., Jin, Y-H. "An effective PSO-based memetic algorithm for flow shop scheduling, "IEEE T. Syst. Man. CY. B.," 37, 2007, 18-27.

[29] Lomnicki, Z. "A branch and bound algorithm for the exact solution of three machine scheduling problem," Operational Research Quarterly, 16: (1), 1965, 89-100.

[30] McMahon, C.B., Burton, P.G. "Flow-shop scheduling with the branch and bound method," Operations Research, 15: (3), 1967, 473-481.

[31] Nawaz, M., Enscore Jr. E., Ham, I. "A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem," Omega-Int. J. Manage. S., 11, 1983, 91-95.

[32] Ogbu, F.A., Smith, D.K. "The application of the simulated annealing algorithm to the solution of the n/m/Cmax flowshop problem," Comput. Oper. Res., 17, 1990, 243-253.

[33] Onwubolu, G.C., Davendra, D. "Scheduling flow-shops using differential evolution algorithm," Eur. J. Oper. Res., 171, 2006, 674-692.

[34] Osman, I., Potts, C. "Simulated annealing for permutation flow shop scheduling," OMEGA, 17, 1989, 551-557.

[35] Pan, C.H. "A study of integer programming formulations for scheduling problems," International Journal of System Science, 28: (1), 1997, 33-41.

[36] Pan, C.H., Chen, J.S. "Scheduling alternative operations in two-machine flow-shops," Journal of the Operational Research Society, 48: (5), 1997, 533-540.

[37] Ravendran, C. "Heuristic for scheduling in flowshop with multiple objectives," Eur. J. Oper. Res., 82, 1995, 540-555.

[38] Ruiz, R., Maroto, C., Alcaraz, J. "Two new robust genetic algorithms for the flowshop scheduling problem," Omega-Int. J. Manage. S., 34, 2006, 461-476.

[39] Stutzle, T. "Applying iterated local search to the permutation flowshop problem," AIDA-98-04, TU Darmstadt, FG Intellektik, 1998.

[40] Taillard, E. "Some efficient heuristic methods for the flow shop sequencing problem," Eur. J. Oper. Res., 47, 1990, 65-74.

[41] Taillard, E. "Benchmarks for basic scheduling problems," Eur. J. Oper. Res., 64, 1993, 278-285.

[42] Wang, C.G., Chu, C.B., Proth, J.M. "Efficient heuristic and optimal approaches for N/2/F/SIGMA-C-I scheduling problems," International Journal of Production Economics, 44: (3), 1996, 225-237.

[43] Ying, K.C., Liao, C.J. "An ant colony system for permutation flow-shop sequencing," Comput. Oper. Res., 31, 2004, 791-801.

[44] Zamani, M.R., Shue, L.Y. "Developing an optimal learning search method for networks," Scientia Iranica, 2: (3), 1995, 197-206.

[45] Zamani, R., Shue, L.Y. "Solving project scheduling problems with a heuristic learning algorithm," Journal of the Operational Research Society, 49: (7), 1998, 709-716.

[46] Zamani, M.R. "A high performance exact method for the resource-constrained project scheduling problem," Computers and Operations Research, 28, 2001, 1387-14.

[47] Zobolas, G.I., Tarantilis, C.D., Ioannou, G. "Minimizing makespan in Permutation Flow Shop scheduling problems using a hybrid metaheuristic algorithm," Computers and Operations Research, 2008, doi:10.1016/j.cor.2008.01.007.

Appendix

Derivation of heuristic functions

The purpose is to develop heuristic functions suitable for use in IHSA*. In each case the objective is to develop a function which underestimates the minimum makespan (i.e. admissible). Six functions are developed and the proof of their admissibility is presented in the next section.

From Figure 1, $S(\phi_{st}) \geq \max\,[b_t +, a_s + \sum_{i=1}^{n} b_i]$ and $T(\phi_{st}) \geq \max\,[S(\phi_{st}) + c_t\,,\ a_s + b_s + \sum_{i=1}^{n} c_i]$ which means that:

$$T(\phi_{st}) \geq a_s + b_s + \sum_{i=1}^{n} c_i \text{ or,} \qquad \text{(A1)}$$

$$T(\phi_{st}) \geq S(\phi_{st}) + c_t \geq b_t + c_t + \sum_{i=1}^{n} a_i \text{ or,} \qquad \text{(A2)}$$

$$T(\phi_{st}) \geq a_s + c_t + \sum_{i=1}^{n} b_i\,. \qquad \text{(A3)}$$

From (A1) two heuristic functions H_3 and H_6 are proposed:

$H_3 = \min[a_1 + b_1, a_2 + b_2, \ldots, a_n + b_n] + \sum_{i=1}^{n} c_i$ and

$H_6 = \min[a_1, a_2, \ldots, a_n] + \min[b_1, b_2, \ldots, b_n] + \sum_{i=1}^{n} c_i\,.$

The rationale for the development of H_3 is: select the job that will be finished on M_2 at the earliest possible time if it is placed first in the job sequence. When this job is finished on M_2 $\min[a_1 + b_1, a_2 + b_2, \ldots, a_n + b_n]$ units of time have elapsed and the additional time needed to complete all of the jobs on all of the machines will be at least $\sum_{i=1}^{n} c_i$ units of time. Since min[$a_1 + b_1, a_2 + b_2, \ldots, a_n + b_n$] ≥ min[$a_1, a_2, \ldots, a_n$] + min[$b_1, b_2, \ldots, b_n$] it follows that $H_3 \geq H_6$, which is therefore also a plausible heuristic function.

H_1 and H_5 are derived from (A2):

H_1 = min[$b_1 + c_1, b_2 + c_2, \ldots, b_n + c_n$] + $\sum_{i=1}^{n} a_i$ and

H_5 = min[$b_1, b_2, \ldots, b_n$] + min[$c_1, c_2, \ldots, c_n$] + $\sum_{i=1}^{n} a_i$.

The rationale for the development of H_1 is: select the job which requires the least total amount of time on machines M_2 and M_3 (i.e. min[$b_1 + c_1, b_2 + c_2, \ldots, b_n + c_n$] units of time) and suppose that it is placed last in the job sequence which means that the earliest time that it can start on M_2 is after $\sum_{i=1}^{n} a_i$ units of time. Since min[$b_1 + c_1, b_2 + c_2, \ldots, b_n + c_n$] ≥ min[$b_1, b_2, \ldots, b_n$] + min[$c_1, c_2, \ldots, c_n$] it follows that $H_1 \geq H_5$, which is therefore also a plausible heuristic function.

H_2 and H_4 are derived from (A3):

H_2 = min[$a_1 + u_1, a_2 + u_2, \ldots, a_n + u_n$] + $\sum_{i=1}^{n} b_i$ and

H_4 = min[$a_1, a_2, \ldots, a_n$] + min[$c_1, c_2, \ldots, c_n$] + $\sum_{i=1}^{n} b_i$,

where: u_1 = min[$c_2, c_3, \ldots, c_n$]; u_k = min[$c_1, c_2, \ldots, c_{k-1}, c_{k+1}, \ldots, c_n$] for $2 \leq k \leq n - 1$; and u_n = min[$c_1, c_2, c_3, \ldots, c_{n-1}$]. The rationale for the development of H_2 is: consider each job in turn and suppose that it is placed first in the job sequence and then from among all of the other jobs select the one which requires the least amount of time on M_3. Now for each pair of jobs selected in this manner determine the pair that gives the least total time on M_1 and M_3. This total time plus the minimum total time required to finish all of the jobs on M_2 is the value of H_2. Also, min[$a_1 + u_1, a_2 + u_2, \ldots, a_n + u_n$] ≥ min[$a_1, a_2, \ldots, a_n$] + min[$u_1, u_2, \ldots, u_n$] = min[$a_1, a_2, \ldots, a_n$] + min[$c_1, c_2, \ldots, c_n$] and it follows that $H_2 \geq H_4$, which is therefore also a plausible heuristic function.

Admissibility

Results and selected proofs related to the admissibility of the heuristic functions H_1,
H_2, H_3, H_4, H_5, and H_6 are presented:

***R_1*.** $H_3 \geq H_6$ and both are admissible.

***R_2*.** $H_2 \geq H_4$ and both are admissible.

***R_3*.** $H_1 \geq H_5$ and both are admissible.

Only a proof for ***R2*** is given since the remaining proofs may be constructed in the same manner.

From (A3), $T(\phi_{st}) \geq a_s + c_t + \sum_{i=1}^{n} b_i$ for s, t = 1, 2, …, n with s ≠ t and so in particular, $T(\phi_{1t}) \geq a_1 + c_t + \sum_{i=1}^{n} b_i$, $T(\phi_{2t}) \geq a_2 + c_t + \sum_{i=1}^{n} b_i$, …, $T(\phi_{nt}) \geq a_n + c_t + \sum_{i=1}^{n} b_i$.

Hence, if $T^*(\phi_{st})$ denotes the earliest time at which any job sequence which starts with job J_s is completed on M_3 then $T^*(\phi_{1t}) \geq$ min[$a_1 + c_2, a_1 + c_3, \ldots, a_1 + c_n$] + $\sum_{i=1}^{n} b_i$,

$T^*(\phi_{2t}) \geq$ min[$a_2 + c_1, a_2 + c_3, \ldots, a_2 + c_n$] + $\sum_{i=1}^{n} b_i$, …,

$T^*(\phi_{nt}) \geq$ min[$a_n + c_1, a_n + c_2, \ldots, a_n + c_{n-1}, \ldots, a_n + c_n$] + $\sum_{i=1}^{n} b_i$ and the minimum makespan T^* = min[$T^*(\phi_{1t})$, $T^*(\phi_{2t}), \ldots, T^*(\phi_{nt})$] ≥ min[$a_1 + u_1, a_2 + u_2, \ldots, a_n + u_n$] + $\sum_{i=1}^{n} b_i$ = H_2 ≥ min[$a_1, a_2, \ldots, a_n$] + min[$c_1, c_2, \ldots, c_n$] + $\sum_{i=1}^{n} b_i$ = H_4. Consequently, $H_2 \geq H_4$ and both are admissible.

From the results *R1*, *R2*, and *R3* it is seen that the heuristic functions H_1, H_2, H_3, H_4, H_5, and H_6 are all admissible. However, in order to select the heuristic function among these that is the closest in value to the minimum makespan (i.e. the best to use in Step1 of IHSA*) the choice should be made from among only H_1, H_2, and H_3 because the function among these 3 which has the largest value is admissible and has a value which is larger than any of the other 5 admissible functions. Consequently, in Step1 of IHSA* the values of H_1, H_2, and H_3 are calculated and the function with the largest value is selected for use.

Dominance

Machine M_1 dominates the other 2 machines if min[$a_1, a_2, \ldots, a_n$] ≥ max[$b_1, b_2, \ldots, b_n$] and min[$a_1, a_2, \ldots, a_n$] ≥ max[$c_1, c_2, \ldots, c_n$] and similar definitions apply if machine M_2 or machine M_3 is dominant.

In the case of a dominant machine results *R5*, *R6*, and *R7* identify immediately which heuristic function among H_1, H_2, and H_3 has the largest value and is the best to use in IHSA*. Also, from *R8* it is seen that the best heuristic function has a value which is greater than the value of either of the other functions by $O(n^2)$ where n is the number of jobs.

***R5*.** If machine M_1 dominates then H_1 is the heuristic function with the largest value,

***R6*.** If machine M_2 dominates then H_2 is the heuristic function with the largest value,

***R7*.** If machine M_3 dominates then H_3 is the heuristic function with the largest value.

***R8*.** If a machine is dominant then the best heuristic function has a value which is greater than the value of either of the other 2 functions by at least $(n - 1)(n - 2)$ where n is the number of jobs and $n \geq 3$.

The proofs for *R5* and *R8* are given noting that proofs for the other results may be constructed in the same manner. Throughout these proofs $\min(a_i) = \min[a_1, a_2, \ldots, a_n]$, $\min(b_i) = \min[b_1, b_2, \ldots, b_n]$, $\min(c_i) = \min[c_1, c_2, \ldots, c_n]$, $\max(a_i) = \max[a_1, a_2, \ldots, a_n]$, $\max(b_i) = \max[b_1, b_2, \ldots, b_n]$, and $\max(c_i) = \max[c_1, c_2, \ldots, c_n]$.

Suppose machine M_1 dominates and for $i = 1, 2, 3, \ldots, n$: $a_i \in [r_1, r_1 + w - 1]$; $b_i \in [s_1, s_1 + l - 1]$; and $c_i \in [t_1, t_1 + d - 1]$ are distinct non negative integers from intervals of widths w, l, and d, respectively, each greater than or equal to n (the number of jobs).

It follows that the minimum values of $\sum_{i=1}^{n} a_i$, $\sum_{i=1}^{n} b_i$, $\sum_{i=1}^{n} c_i$ are $nr_1 + 0.5n(n-1)$, $ns_1 + 0.5n(n-1)$, and $nt_1 + 0.5n(n-1)$, respectively, and when these minimum values are attained $\min(a_i) = r_1$, $\min(b_i) = s_1$, $\min(c_i) = t_1$, $\max(a_i) = r_1 + n - 1$, $\max(b_i) = s_1 + n - 1$, and $\max(c_i) = t_1 + n - 1$ for $i = 1, 2, 3, \ldots, n$.

Also, the maximum values of $\sum_{i=1}^{n} a_i$, $\sum_{i=1}^{n} b_i$, $\sum_{i=1}^{n} c_i$ are $n(r_1 + w) - 0.5n(n+1)$, $n(s_1 + l) - 0.5n(n+1)$, and $n(t_1 + d) - 0.5n(n+1)$, respectively, and when these maximum values are attained $\min(a_i) = r_1 + w - n$, $\min(b_i) = s_1 + l - n$, $\min(c_i) = t_1 + d - n$, $\max(a_i) = r_1 + w - 1$, $\max(b_i) = s_1 + l - 1$, $\max(c_i) = t_1 + d - 1$.

Now, $H_1 = \min[b_1 + c_1, b_2 + c_2, \ldots, b_n + c_n] + \sum_{i=1}^{n} a_i$

$$\geq \min(b_i) + \min(c_i) + \min\left(\sum_{i=1}^{n} a_i\right) \qquad \text{(A4)}$$

and similarly,

$$H_2 \leq \max(a_i) + \max(c_i) + \max\left(\sum_{i=1}^{n} b_i\right) \qquad \text{(A5)}$$

$$\text{and } H_3 \leq \max(a_i) + \max(b_i) + \max\left(\sum_{i=1}^{n} c_i\right). \qquad \text{(A6)}$$

If (A4), (A5), (A6) are all true then,

$H_1 \geq s_1 + l - n + t_1 + d - n + nr_1 + 0.5n(n-1)$, (A7)

$H_2 \leq r_1 + n - 1 + t_1 + d - 1 + n(s_1 + l) - 0.5n(n+1)$, (A8)

$H_3 \leq r_1 + n - 1 + s_1 + l - 1 + n(t_1 + d) - 0.5n(n+1)$. (A9)

From (A7) and (A8),

$s_1 + l - n + t_1 + d - n + nr_1 + 0.5n(n-1) - r_1 - n + 1 - t_1 - d + 1 - n(s_1 + 1) + 0.5n(n+1) = s_1 - ns_1 + l - n_l + nr_1 - r_1 + n^2 - 3n + 2 = (n-1)[r_1 - (s_1 + l) + n - 2] \geq (n-1)(n-2) \geq 0$, for $n \geq 2$, and so H_1 is greater than H_2 by a value which is at least $(n-1)(n-2)$, for $n \geq 3$.

In a similar manner it follows from (A7) and (A9) that H_1 is greater than H_3 by a value which is at least $(n-1)(n-2)$, for $n \geq 3$ and this completes the proof of *R5* and *R8*.

The best admissible heuristic function for an arbitrary number of machines

For the case where there are more than 3 machines there is a need to change the notation used previously to represent the time that each operation $O_{i,j}$ requires on each machine so that $t_{i,j}$ is the number of units of time required by job J_i on machine M_j.

If there are m machines then the best admissible heuristic function will be the one with the largest value among the set of m functions $F_1, F_2, F_3, \ldots, F_m$ where,

$$F_j = \begin{cases} \min\left[\sum_{i=2}^{m} t_{1,i}, \sum_{i=2}^{m} t_{2,i}, \ldots, \sum_{i=2}^{m} t_{n,i}\right] + \sum_{i=1}^{n} t_{i,1}, \text{ for } j=1, \\ \min\left[\sum_{i=1}^{j-1} t_{1,i} + u_{j,1}, \sum_{i=1}^{j-1} t_{2,i} + u_{j,2}, \ldots, \sum_{i=1}^{j-1} t_{n,i} + u_{j,n}\right] \\ \quad + \sum_{i=1}^{n} t_{i,j}, \text{ for } 2 \leq j \leq m, \end{cases}$$

where, for $2 \leq j \leq m - 1$,

$$u_{j,k} = \begin{cases} \min\left[\sum_{i=j+1}^{m} t_{2,i}, \sum_{i=j+1}^{m} t_{3,i}, \ldots, \sum_{i=j+1}^{m} t_{n,i}\right], \text{ for } k = 1, \\ \min\left[\sum_{i=j+1}^{m} t_{1,i}, \sum_{i=j+1}^{m} t_{2,i}, \ldots, \sum_{i=j+1}^{m} t_{k-1,i}, \sum_{i=j+1}^{m} t_{k+1,i}, \ldots, \right. \\ \quad \left. \sum_{i=j+1}^{m} t_{n,i}\right], \text{ for } 2 \leq k \leq n-1, \\ \min\left[\sum_{i=j+1}^{m} t_{1,i}, \sum_{i=j+1}^{m} t_{2,i}, \ldots, \sum_{i=j+1}^{m} t_{n-1,i}\right], \text{ for } k = n, \end{cases}$$

and $u_{m,k} = 0$, for $k = 1, 2, 3, \ldots, n$.

For a problem with m machines where $m > 3$ and the job sequence is the same on each machine the function among $F_1, F_2, F_3, \ldots, F_m$ with the largest value is selected in Step 1 of $IHSA^*$.

If $m = 3$ then using $t_{1,i} = a_i$, $t_{2,i} = b_i$, and $t_{3,i} = c_i$ for $i = 1, 2, 3, \ldots, n$ and representing $u_{j,1}$, $u_{j,k}$, and $u_{j,n}$ simply by u_1, u_k, and u_n, respectively, the 3 admissible heuristic functions H_1, H_2, and H_3 in (5) which have been used throughout the description of the development of $IHSA^*$ are given by F_1, F_2, and F_3, respectively.

Experimental evidence of improvements in performance characteristics

Table A1: Performance of IHSA[*]: Modification to Step 1 compared to Modifications to Steps 1 and 2.

Note: For each problem: (a) the highlighted first row, associated with the use of the modification to Step 1, indicates the performance characteristics using the best heuristic function; (b) the highlighted last row, associated with the use of modifications to Steps 1 & 2, indicates the performance characteristics when the best heuristic function is used together with the modification to Step 2.

No. of Machines	Number of Jobs	Problem	Modification Used	Heuristic Function	Value of Heuristic Function	Performance Characteristics			Minimum Makespan
						Nodes Expanded	Backtracks	Algorithm Steps	
3	3	1	1	**H1**	**17**	**13**	**0**	**10**	17
				H3	10	22	11	33	
				H2	9	21	12	34	
			1&2	**H1**	**17**	**10**	**0**	**7**	
		2	1	**H2**	**24**	**13**	**6**	**22**	24
				H3	15	36	39	88	
				H1	13	36	37	84	
			1&2	**H2**	**24**	**8**	**2**	**17**	
		3	1	**H3**	**28**	**16**	**1**	**14**	28
				H2	20	17	3	16	
				H1	18	17	3	16	
			1&2	**H3**	**28**	**13**	**0**	**10**	
	4	4	1	**H1**	**23**	**35**	**20**	**52**	25
				H3	19	50	30	72	
				H2	14	57	37	82	
			1&2	**H1**	**23**	**35**	**15**	**42**	
		5	1	**H2**	**22**	**32**	**27**	**66**	23
				H3	19	79	70	152	
				H1	15	80	79	170	
			1&2	**H2**	**22**	**31**	**21**	**54**	
		6	1	**H3**	**24**	**66**	**53**	**109**	27
				H2	21	69	63	138	
				H1	20	71	64	140	
			1&2	**H3**	**24**	**63**	**43**	**96**	
	15	7	1	**H1**	**26**	**20**	**2**	**22**	28
				H3	18	24	10	30	
				H2	16	26	15	36	
			1&2	**H1**	**26**	**18**	**0**	**10**	
		8	1	**H2**	**28**	**20**	**9**	**38**	31
				H3	20	38	40	90	
				H1	16	40	45	98	
			1&2	**H2**	**28**	**16**	**4**	**25**	
		9	1	**H3**	**29**	**19**	**2**	**22**	30
				H2	20	22	4	28	
				H1	19	25	4	29	
			1&2	**H3**	**29**	**16**	**0**	**18**	
	40	10	1	**H1**	**43**	**50**	**12**	**70**	45
				H3	38	53	30	105	
				H2	35	70	38	113	
			1&2	**H1**	**43**	**44**	**6**	**55**	
		11	1	**H2**	**42**	**49**	**30**	**60**	43
				H3	40	50	32	85	
				H1	37	66	45	115	
			1&2	**H2**	**42**	**32**	**18**	**52**	
		12	1	**H3**	**45**	**68**	**46**	**115**	50
				H2	38	77	58	152	
				H1	35	90	93	205	
			1&2	**H3**	**45**	**54**	**30**	**84**	
5	15	13	1	**F1**	**37**	**42**	**12**	**65**	39
				F2	30	60	15	93	
				F3	27	72	18	104	
				F4	25	82	25	126	
				F5	23	97	32	150	
			1&2	**F1**	**37**	**30**	**5**	**50**	
10	10	14	1	**F3**	**50**	**35**	**20**	**63**	52
				F2	48	37	23	83	
				F4	46	41	28	89	
				F1	45	43	30	91	
				F6	42	45	33	95	
				F5	40	52	34	107	
				F9	38	60	40	123	
				F7	32	69	45	135	
				F8	32	72	48	137	
				F10	30	81	51	151	
			1&2	**F3**	**50**	**30**	**2**	**40**	

Table A1: Performance of IHSA[*]

CONTENTS OF *Informatica* Volume 32 (2008) pp. 1–467

Papers

PARAMESWARAN, L. & , K. ANBUMANI . 2008. Content-Based Watermarking for Image Authentication Using Independent Component Analysis. Informatica 32:299–306.

REN, Y. & , D. GU. 2008. Efficient Hierarchical Identity Based Encryption Scheme in the Standard Model. Informatica 32:207–211.

RUPNIK, J. & , M. GRČAR, T. ERJAVEC. 2008. Improving Morphosyntactic Tagging of Slovene Language through Meta-tagging. Informatica 32:437–444.

SALEHI, M.A. & , H. DELDARI, B.M. DORRI. 2008. Balancing Load in a Computational Grid Applying Adaptive, Intelligent Colonies of Ants. Informatica 32:327–335.

SAMBT, J. & , M. ČOK. 2008. Demographic Pressure on the Public Pension System. Informatica 32:103–109.

SCHUSTER, A. & . 2008. DNA Algorithms for Petri Net Modeling. Informatica 32:421–427.

SHERMEH, A.E.Z. & , R. GHADERI. 2008. An Intelligent System for Classification of the Communication formats Using PSO. Informatica 32:213–218.

ŠIMOŇÁK, S. & , Š. HUDÁK, Š. KOREČKO. 2008. APC Semantics for Petri Nets. Informatica 32:253–260.

ŠKULJ, D. & , V. VEHOVAR, D. ŠTAMFELJ. 2008. The Modelling of Manpower by Markov Chains - A Case Study of the Slovenian Armed Forces. Informatica 32:289–297.

TAŠKOVA, K. & , P. KOROŠEC, J. ŠILC. 2008. A Distributed Multilevel Ant Colonies Approach. Informatica 32:307–317.

VOR DER BRÜCK, T. & , S. HARTRUMPF, H. HELBIG. 2008. A Readability Checker with Supervised Learning Using Deep Indicators. Informatica 32:429–435.

WOTAWA, F. & , M. NICA. 2008. On the Compilation of Programs into their Equivalent Constraint Representation. Informatica 32:359–371.

ŽENKO, B. & . 2008. Learning Predictive Clustering Rules. Informatica 32:95–96.

ŽIBERT, J. & , B. VESNICER, F. MIHELIČ. 2008. A System for Speaker Detection and Tracking in Audio Broadcast News. Informatica 32:51–61.

Editorials

MALAČIČ, J. & , J. JÓŹWIAK, A. FÜRNKRANZ-PRSKAWETZ. 2008. Introduction. Informatica 32:101–102.

BADICA, C. & , M. GANZHA, M. PAPRZYCKI . 2008. Editor's Introduction to the Special Issue: Intelligent Systems. Informatica 32:340–340.

JOŽEF STEFAN INSTITUTE

Jožef Stefan (1835-1893) was one of the most prominent physicists of the 19th century. Born to Slovene parents, he obtained his Ph.D. at Vienna University, where he was later Director of the Physics Institute, Vice-President of the Vienna Academy of Sciences and a member of several scientific institutions in Europe. Stefan explored many areas in hydrodynamics, optics, acoustics, electricity, magnetism and the kinetic theory of gases. Among other things, he originated the law that the total radiation from a black body is proportional to the 4th power of its absolute temperature, known as the Stefan–Boltzmann law.

The Jožef Stefan Institute (JSI) is the leading independent scientific research institution in Slovenia, covering a broad spectrum of fundamental and applied research in the fields of physics, chemistry and biochemistry, electronics and information science, nuclear science technology, energy research and environmental science.

The Jožef Stefan Institute (JSI) is a research organisation for pure and applied research in the natural sciences and technology. Both are closely interconnected in research departments composed of different task teams. Emphasis in basic research is given to the development and education of young scientists, while applied research and development serve for the transfer of advanced knowledge, contributing to the development of the national economy and society in general.

At present the Institute, with a total of about 800 staff, has 600 researchers, about 250 of whom are postgraduates, nearly 400 of whom have doctorates (Ph.D.), and around 200 of whom have permanent professorships or temporary teaching assignments at the Universities.

In view of its activities and status, the JSI plays the role of a national institute, complementing the role of the universities and bridging the gap between basic science and applications.

Research at the JSI includes the following major fields: physics; chemistry; electronics, informatics and computer sciences; biochemistry; ecology; reactor technology; applied mathematics. Most of the activities are more or less closely connected to information sciences, in particular computer sciences, artificial intelligence, language and speech technologies, computer-aided design, computer architectures, biocybernetics and robotics, computer automation and control, professional electronics, digital communications and networks, and applied mathematics.

The Institute is located in Ljubljana, the capital of the independent state of **Slove**nia (or S♡nia). The capital today is considered a crossroad between East, West and Mediterranean Europe, offering excellent productive capabilities and solid business opportunities, with strong international connections. Ljubljana is connected to important centers such as Prague, Budapest, Vienna, Zagreb, Milan, Rome, Monaco, Nice, Bern and Munich, all within a radius of 600 km.

From the Jožef Stefan Institute, the Technology park "Ljubljana" has been proposed as part of the national strategy for technological development to foster synergies between research and industry, to promote joint ventures between university bodies, research institutes and innovative industry, to act as an incubator for high-tech initiatives and to accelerate the development cycle of innovative products.

Part of the Institute was reorganized into several high-tech units supported by and connected within the Technology park at the Jožef Stefan Institute, established as the beginning of a regional Technology park "Ljubljana". The project was developed at a particularly historical moment, characterized by the process of state reorganisation, privatisation and private initiative. The national Technology Park is a shareholding company hosting an independent venture-capital institution.

The promoters and operational entities of the project are the Republic of Slovenia, Ministry of Higher Education, Science and Technology and the Jožef Stefan Institute. The framework of the operation also includes the University of Ljubljana, the National Institute of Chemistry, the Institute for Electronics and Vacuum Technology and the Institute for Materials and Construction Research among others. In addition, the project is supported by the Ministry of the Economy, the National Chamber of Economy and the City of Ljubljana.

Jožef Stefan Institute
Jamova 39, 1000 Ljubljana, Slovenia
Tel.:+386 1 4773 900, Fax.:+386 1 251 93 85
WWW: http://www.ijs.si
E-mail: matjaz.gams@ijs.si
Public relations: Polona Strnad

INFORMATICA
AN INTERNATIONAL JOURNAL OF COMPUTING AND INFORMATICS
INVITATION, COOPERATION

Submissions and Refereeing

Please submit an email with the manuscript to one of the editors from the Editorial Board or to the Managing Editor. At least two referees outside the author's country will examine it, and they are invited to make as many remarks as possible from typing errors to global philosophical disagreements. The chosen editor will send the author the obtained reviews. If the paper is accepted, the editor will also send an email to the managing editor. The executive board will inform the author that the paper has been accepted, and the author will send the paper to the managing editor. The paper will be published within one year of receipt of email with the text in Informatica MS Word format or Informatica LaTeX format and figures in .eps format. Style and examples of papers can be obtained from http://www.informatica.si. Opinions, news, calls for conferences, calls for papers, etc. should be sent directly to the managing editor.

QUESTIONNAIRE

☐ Send Informatica free of charge

☐ Yes, we subscribe

Please, complete the order form and send it to Dr. Drago Torkar, Informatica, Institut Jožef Stefan, Jamova 39, 1000 Ljubljana, Slovenia. E-mail: drago.torkar@ijs.si

Since 1977, Informatica has been a major Slovenian scientific journal of computing and informatics, including telecommunications, automation and other related areas. In its 16th year (more than ten years ago) it became truly international, although it still remains connected to Central Europe. The basic aim of Informatica is to impose intellectual values (science, engineering) in a distributed organisation.

Informatica is a journal primarily covering the European computer science and informatics community - scientific and educational as well as technical, commercial and industrial. Its basic aim is to enhance communications between different European structures on the basis of equal rights and international refereeing. It publishes scientific papers accepted by at least two referees outside the author's country. In addition, it contains information about conferences, opinions, critical examinations of existing publications and news. Finally, major practical achievements and innovations in the computer and information industry are presented through commercial publications as well as through independent evaluations.

Editing and refereeing are distributed. Each editor can conduct the refereeing process by appointing two new referees or referees from the Board of Referees or Editorial Board. Referees should not be from the author's country. If new referees are appointed, their names will appear in the Refereeing Board.

Informatica is free of charge for major scientific, educational and governmental institutions. Others should subscribe (see the last page of Informatica).

ORDER FORM – INFORMATICA

Name: ....

Title and Profession (optional): ....

....

Home Address and Telephone (optional): ....

....

Office Address and Telephone (optional): ....

....

E-mail Address (optional): ....

Signature and Date: ....

Informatica WWW:

http://www.informatica.si/

Referees:

Informatica

An International Journal of Computing and Informatics

Web edition of Informatica may be accessed at: http://www.informatica.si.

Subscription Information Informatica (ISSN 0350-5596) is published four times a year in Spring, Summer, Autumn, and Winter (4 issues per year) by the Slovene Society Informatika, Vožarski pot 12, 1000 Ljubljana, Slovenia.
The subscription rate for 2008 (Volume 32) is
– 60 EUR for institutions,
– 30 EUR for individuals, and
– 15 EUR for students
Claims for missing issues will be honored free of charge within six months after the publication date of the issue.

Typesetting: Borut Žnidar.
Printing: Dikplast Kregar Ivan s.p., Kotna ulica 5, 3000 Celje.

Orders may be placed by email (drago.torkar@ijs.si), telephone (+386 1 477 3900) or fax (+386 1 251 93 85). The payment should be made to our bank account no.: 02083-0013014662 at NLB d.d., 1520 Ljubljana, Trg republike 2, Slovenija, IBAN no.: SI56020830013014662, SWIFT Code: LJBASI2X.

Informatica is published by Slovene Society Informatika (president Niko Schlamberger) in cooperation with the following societies (and contact persons):
Robotics Society of Slovenia (Jadran Lenarčič)
Slovene Society for Pattern Recognition (Franjo Pernuš)
Slovenian Artificial Intelligence Society; Cognitive Science Society (Matjaž Gams)
Slovenian Society of Mathematicians, Physicists and Astronomers (Bojan Mohar)
Automatic Control Society of Slovenia (Borut Zupančič)
Slovenian Association of Technical and Natural Sciences / Engineering Academy of Slovenia (Igor Grabec)
ACM Slovenia (Dunja Mladenič)

Informatica is surveyed by: Citeseer, COBISS, Compendex, Computer & Information Systems Abstracts, Computer Database, Computer Science Index, Current Mathematical Publications, DBLP Computer Science Bibliography, Directory of Open Access Journals, InfoTrac OneFile, Inspec, Linguistic and Language Behaviour Abstracts, Mathematical Reviews, MatSciNet, MatSci on SilverPlatter, Scopus, Zentralblatt Math

The issuing of the Informatica journal is financially supported by the Ministry of Higher Education, Science and Technology, Trg OF 13, 1000 Ljubljana, Slovenia.

Volume 32 Number 4 November 2008 **ISSN 0350-5596**

Informatica

An International Journal of Computing and Informatics

Informatica **32** (2008) Number 4, pp. 339–467