Proceedings of the ECAI-04 Workshop on Agents in dynamic and real-time environments

Ubbo Visser
Hans-Dieter Burkhard
Patrick Doherty
Gerhard Lakemeyer
(editors)

Held on August 22, 2004 in conjunction with the 16th European Conference on Artificial Intelligence, Valencia, Spain

.

ECAI-04 Workshop on
Agents in dynamic and real-time environments

Held on August 22, 2004 in conjunction with the 16th European Conference on Artificial Intelligence, Valencia, Spain

Recent developments in multiagent systems (MAS) have been promising by achieving autonomous, collaborative behavior between agents in various environments. However, most of the agents, both software agents and physical agents, still have problems if the environment is dynamic and the agents have to act in real time. Examples are obstacle avoidance with moving obstacles or world models which are composed from egocentric views of numerous agents. Another aspect is the need for quick responses. In an environment where a number of agents build a team and both single agent decisions and team collaborative decisions have to be made methods have to be fast and precise. This workshop addresses various problems that occur with respect to these issues.

The main focus of this workshop will be methods from various areas such as world modeling, planning, learning, and communicating with agents in dynamic and real-time environments. Within this general theme we aim to bring together researchers to discuss the following topics:

- World modeling (quantitative, qualitative)
- Coaching (one agent gives advice to a group of agents)
- Planning with resources (especially time)
- Learning (both off- and on-line)
- Cooperation between agents (robot and/or humans)
- Communication between agents (implicit, non-verbal, or verbal one)
- Real-time systems software issues (often ignored but important if serious about real-time issues in robotics)
- Scalability and robotics interfacing issues (demands a great deal of support from the initial design of the system)

In the last decade, a lot of effort has been invested to develop methods that can be used with multi-agent systems. The language development in the area of communication between agents (ACL) might act as the first example. Speech acts serve as the basic principle and various protocols have been invented (e.g. auctions, contract-nets, etc.). Can we transfer these results to environments where quick decisions have to be made?

Consider planning as another example: there are promising methods for path planning, but do they still hold if the observed obstacles are moving? Learning is another example: we need on-line learning in a real-time scenario to give agents the option to learn more about their environment. Usually, learning takes a fair amount of time but sometimes this time is not available. Can we find methods which will consider these restrictions?

This workshop addresses researchers from various areas in AI who want to discuss the mentioned issues from their point of view. How can we develop new methods or adapt existing methods to meet these demands?

We had ten submissions for this workshop and each of the papers have been evaluated by three reviewers (except one paper with two reviewers). The results of the reviews were surprisingly close. After a discussion among the organizers we decided to accept all ten papers for oral presentation at the workshop.

The contributions can be roughly categorized into the following topics: *World modeling, planning, learning, and behavior/collaboration.* Please note that this is only a rough categorization and that there are a number of papers that belong to more than one topic.

World modeling: A qualitative spatial knowledge representation based on ordering information is proposed by *Wagner & Hübner*. They use this representation to navigate with physical robots taking into account the egocentric perspective of the robot. Their method also provides means to reason about the robots world model validity despite insufficient and uncertain sensory data. The method is based on spatial representations using landmarks.

The contribution by *Merke et al.* discusses important problems that occur with localization issues using physical agents that perform in real-time environments. Their proposed approach is based on a robust particle filter method using features found in a camera image. Those features are points on field lines and can be recognized reliably under natural light conditions with 30fps.

Planning: The contribution of *Domshlak & Lawton* deals with opportunistic planning and plan execution. They discuss how multi-agent systems can exploit shared knowledge for opportunistic predictive encoding using an approach based on an abstract plan representation called Partial Order Plan Graphs (POPGs). They also present several approaches for increasing system-level performance by improving the efficiency of the plans containing predictively encoded opportunities (e.g. planning with shortcuts, plan repair methods).

Le Gloannec et al. discuss planning issues that occur under uncertainty having multiple resources. They propose an approach to control the operation of an autonomous agent which operates under multiple resource constraints. They use a DAG of progressive tasks using an optimal policy obtained by an MDP. Computing an optimal policy for an MDP with multiple resources makes the search space large and therefore unusable at run-time. The authors thus propose a solution by decomposing a large MDP into smaller ones, compressing the state space, and constructing and recomposing local policies for the decomposed MDPs in order to obtain a near optimal global policy.

Learning: *Rettinger* proposes an idea that provides a scoring policy for simulated soccer agents. This method is able to be used in real-time in dynamic environments such as the

RoboCup Simulation League. The technique uses data obtained from prerecorded soccer games for supervised neural network learning.

The second paper that discusses a learning approach within the RoboCup Soccer Simulation proposes a symbolic learning method. *Konur et al.* learn decision trees for the selection of the agents next actions. The method is used to learn the action selection strategy of the whole team, that is, defenders, mid-fielders, and attackers, when a player is in ball possession. The authors state that the method can also be used in a different way. The learning method yielded a set of qualitative features to classify game situations, which are useful beyond reactive decision making.

Behavior and collaboration: *Lundh et al.* study teams of autonomous robotic agents where agents help each other out by offering information-producing resources and functionalities. Depending on the current situation and tasks, the team may need to change its functional configuration dynamically. The authors propose knowledge-based techniques to automatically synthesize new team configurations in response to changes in the situation or tasks.

Carrascosa et al. discuss the behavior of real-time agents with respect to reactivity and deliberation. They introduce the concept of *Reactivity Degree*. This concept implies some meta-reasoning capabilities to be available in the agent in order to dynamically decide the amount of resources which have to be assigned to deliberation and reaction. The paper also shows how to implement such a concept in the hard real-time, hybrid agent architecture called ARTIS.

Dorer proposes an approach where behavior networks can be extended to model behavior selection of agents in dynamic and continuous domains. The paper focusses on a mechanism for concurrent behavior selection by explicitly represent the resources which are used by a behavior. Dorer describes further how this process can be combined with behavior execution in continuous domains.

Shimony & Berler discuss local decision problems and how they can be solved using Bayesian knowledge bases. The authors state that collaboration of multiple intelligent agents on a shared task is a complex research issue. The problems become particularly difficult when communication is limited or impossible. They developed AWOL (Abstract World for Opportunistic Local decisions), an abstract framework with a disciplined treatment of opportunistic action, in the context of an existing joint plan.

Program Committee and Reviewers

We are grateful to the following members of the international program committee for helping us to make this a high quality workshop:

- Hans-Dieter Burkhard, Humboldt Universität, Berlin, GERMANY
- Thomas Christaller, AIS-Fraunhofer, St. Augustin, GERMANY
- Patrick Doherty, University of Linköping, SWEDEN
- Uli Furbach, Universität Koblenz, GERMANY
- Otthein Herzog, Universität Bremen, GERMANY
- Gerhard Lakemeyer, RWTH Aachen, GERMANY
- Paul Levi, Universität Stuttgart, Stuttgart, GERMANY
- John-Jules Meyer, Universiteit Utrecht, THE NETHERLANDS
- Frank Pasemann, AIS-Fraunhofer, St. Augustin, GERMANY
- Angel del Pobil, Jaume I, SPAIN
- Fiora Pirri, University La Sapienza, Rome, ITALY
- Alessandro Saffiotti, University of Örebro, SWEDEN
- Michael Thielscher, TU Dresden, GERMANY
- Ubbo Visser, Universität Bremen, GERMANY
- Thomas Wagner, Universität Bremen, GERMANY

Contents

An Egocentric Qualitative Spatial Knowledge Representation Based on Ordering Information for Physical Robot Navigation 9
Thomas Wagner and Kai Hübner

Line Based Robot Localization under Natural Light Conditions 19
Artur Merke, Stefan Welker, and Martin Riedmiller

Opportunistic Planning and Plan Execution 27
Carmel Domshlak and James H. Lawton

Planning under Uncertainty with Multiple Consumable Resources 37
Simon Le Gloannec, Abdel-Illah Mouaddib, and Francois Charpillet

Learning from Recorded Games: A Scoring Policy for Simulated Soccer Agents 43
Achim Rettinger

Learning Decision Trees for Action Selection in Soccer Agents 49
Savas Konur, Alexander Ferrein, and Gerhard Lakemeyer

Dynamic Configuration of a Team of Robots 57
Robert Lundh, Lars Karlsson, and Alessandro Saffiotti

Real-Time Agents: Reaction vs. Deliberation 63
Carlos Carrascosa, José Fabregat, Andrés Terrasa, and Vicente Botti

Extended Behavior Networks for Behavior Selection in Dynamic and Continuous Domains 71
Klaus Dorer

Abstract World for Opportunistic Local Decisions in Multi-Agent Systems Using Bayesian Knowledge Bases 77
Solomon Eyal Shimony and Ami Berler

.

An Egocentric Qualitative Spatial Knowledge Representation Based on Ordering Information for Physical Robot Navigation

Thomas Wagner [1] and **Kai Huebner** [2]

Abstract. Navigation is one of the most fundamental tasks to be accomplished by many types of mobile and cognitive systems. Most approaches in this area are based on building or using existing allocentric, static maps in order to guide the navigation process. In this paper we propose a simple egocentric, qualitative approach to navigation based on ordering information. An advantage of our approach is that it produces qualitative spatial information which is required to describe and recognize complex and abstract, i.e., translation-invariant behavior. In contrast to other techniques for mobile robot tasks, that also rely on landmarks it is also proposed to reason about their validity despite insufficient and uncertain sensory data. Here we present a formal approach that avoids this problem by use of a simple internal spatial representation based on landmarks aligned in an *extended panoramic representation* structure.

1 Introduction

Navigation is one of the most fundamental tasks to be accomplished by robots, autonomous vehicles, and cognitive systems. Most successful approaches in the area of robot navigation like potential fields (see [11] and [8]) are based on allocentric, static maps in order to guide the navigation process (e.g. [10]). This approach has an intuitive appeal and gains much intuition from cognitive science: the *cognitive map* (a good recent overview [17]). The main purpose is to build up a precise, usually allocentric, quantitative representation of the surrounding environment and to determine the robot's position according to this allocentric, quantitative map.

One difficulty results from the fact that the same spatial representation serves as a basis for different tasks often with heterogeneous requirements. For example, more abstract reasoning tasks like planning coordinated behavior, e.g., *counterattack* and *double pass*, and plan recognition usually rely on more abstract, qualitative spatial representations. Generation of qualitative spatial descriptions from quantitative data is usually a difficult task due to uncertain and incomplete sensory data. In order to fit heterogeneous requirements, we should be able to represent spatial qualitative description at different levels of granularity, i.e., invariant according to translation and/or rotation and based on different scalings.

Based on recent results from cognitive science (see, e.g., [33]), we present a formal, egocentric, and qualitative approach to navigation which overcomes some problems of quantitative, allocentric approaches. By the use of ordering information, i.e., based on a description of how landmarks can shift and switch, we generate an *extended panoramic representation* (EPR). We claim that our representation in combination with path integration provides sufficient information to guide navigation with reduced effort to the vision process. Furthermore the EPR provides the foundation for qualitative spatial descriptions that may be invariant to translation and/or rotation.

Since our approach abstracts from quantitative or metrical detail in order to introduce a stable qualitative representation between the raw sensor data and the final application, it can for example be used in addition to the well-elaborated quantitative methods.

2 Motivation

Modeling complex behavior imposes strong requirements on the underlying representations. The representation should provide several levels of abstraction for activities as well as for objects. For both types of knowledge, different representations were proposed and it was demonstrated that they can be used successfully. Activities can, e.g., be described adequately with hierarchical task networks (HTN) which provide clear formal semantics as well as powerful, efficient (planning-) inferences (see e.g. [4]). Objects can be described either in ontology-based languages (e.g., OWL [23]) or constraint-based languages (e.g., [9]). Both types of representations allow for the representation of knowledge at different levels of abstraction according to the domain and task specific requirements. In physically grounded environments, the use of these techniques requires an appropriate qualitative spatial description in order to relate the modeled behavior to the real world.

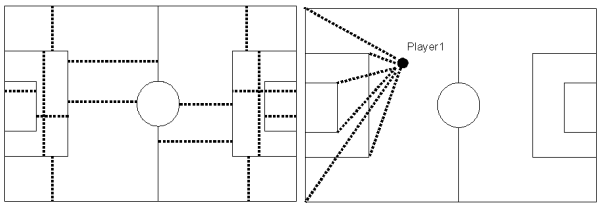
2.1 Allocentric and Egocentric Representations

In an egocentric representation, spatial relations are usually directly related to an agent by the use of an egocentric *frame of reference* in terms like, e.g., *left, right, in front, behind*. As a consequence, when an agent moves through an environment, all spatial relations need to be updated. In contrast, representations based on an allocentric frame of reference remain stable but are much harder to acquire. Additionally, the number of spatial relations which have to be taken into account may be much larger because we have to consider the relations between each object and all other objects in the environment, whereas the number of relations in egocentric representations can be significantly smaller (see Fig. 1)[3]. An interesting phenomenon, when looking into the didactic literature about, e.g., sports [13] we often

[1] Center for Computing Technologies (TZI), Bremen, Germany email: twagner@tzi.de

[2] Institute of Safe Systems (BISS), Bremen, Germany email: khuebner@tzi.de

[3] For reasons of clarity not all allocentric relations are drawn in diagram 1(a).



(a) Allocentric relations (b) Egocentric relations

Figure 1. Allocentric vs. egocentric spatial relations

find that (tactical and strategic) knowledge is described in both, egocentric and allocentric terms, whereas, e.g., the literature about driving lessons strongly relies on purely egocentric views. At least one of the reasons are that the latter representation seems to provide better support for acting directly in physically grounded environments, since perception as well as the use of actuators are directly based on egocentric representations. In addition, egocentric representations provide better support for rotation and translation invariant representations when used with a qualitative abstraction (see section 3.3 and 4 for more details).

3 Related Work

3.1 Cognition: Dynamic, Egocentric Spatial Representations

The fact that even many animals (e.g., rodents) are able to find new paths leading to familiar objects seems to suggest that spatial relations are encoded in an allocentric static *"cognitive map"*. This almost traditional thesis is supported by many spatial abilities like map navigation and mental movement that humans are able to perform (beginning with [28] and [15]). Nevertheless, recent results in cognitive science provide strong evidence for a different view ([33] among many others). Instead of using an allocentric view-independent map, humans and many animals build up a dynamic, view-dependent egocentric representation. Although the allocentric interpretation of the *cognitive map* seems to differ radically from the egocentric representation theory, both theories can account for many observations and differ mainly in two points: The allocentric, *cognitive map*-interpretation assumes that the spatial representation is view-independent and that therefore viewpoint changes do not have any influence on the performance of, e.g., spatial retrieval processes. Many recent experiments provide evidence for the opposite, they show that viewpoint changes can significantly reduce performance in terms of time and quality (e.g. pointing errors) (among others, [31] and [32]). The second main difference is concerned with the dynamic of the underlying representation. The egocentric interpretation assumes that all egocentric relations have to be updated with each egocentric movement of a cognitive system. The underlying assumption of a sophisticated series of experiments done by Wang ([31] and [32]) was that spatial relations have to remain stable in an allocentric, *cognitive map* independent from egocentric movements. When errors arise, e.g., because of path integration, the error rate (*„configuration error"*) should be the same for all allocentric relations; otherwise they rely on an egocentric representation. The results indicate clear evidence for egocentric representations and have been confirmed in a series of differently designed experiments[4], e.g., [3] and [5].

[4] Nevertheless, these results do not allow the strict conclusion that humans do not build up an allocentric cognitive map. On the contrary, e.g., Easton and Sholl [3] have shown that under very specific conditions it is possible to build up allocentric maps. Regardless these results indicate, that under more natural conditions human navigation relies on egocentric snapshots and a dynamic mapping between these.

3.2 Robot Navigation

Navigation and localization is the most fundamental task for autonomous robots and has gained much attention in the robotic research over the last decades. While several earlier approaches addressed this problem qualitatively [10], e.g., topological maps ([12], [16], [1]), more recent approaches focus very successfully on probabilistic methods. Famous examples are RHINO [25], MINERVA [24] and more recently [27]. Currently, the most promising techniques for robust mobile robot localization and navigation are either based on Monte-Carlo-Localization (MCL) (see [19] for RoboCup-application and the seminal paper [26]) or on various extentions of *Kalman-filters* (e.g., [14]) using probabilistic representations based on quantitative sensory data. MCL is based on a sample set of postures; the robot's position can be estimated by probabilities which allow to handle not only the *position tracking*- and the *global localization* problem but also the challenging *kidnapped robot* problem of moving a robot without telling it.

Furthermore, probabilistic methods based on quantitative data play a crucial role in handling the mapping problem, i.e., the SLAM-problem[5]. Very much the same is true for many robotic approaches to navigation, e.g., potential fields for avoiding obstacles by following the flow of superposed partial fields in order to guide the robot to a goal position (see [11] and [8] for a RoboCup-application) based on quantitative data.

According to the *spatial semantic hierarchy* (SSH) [10], these approaches try to address the problems related to robot navigation on the *control level*. Besides the strong computational resource requirements they usually do not address the problem of generating a discrete, qualitative spatial representation which for instance is required at more abstract levels, e.g., for describing complex coordinated tactical and strategic behavior either on individual- and on team level.

3.3 The Panorama Approach

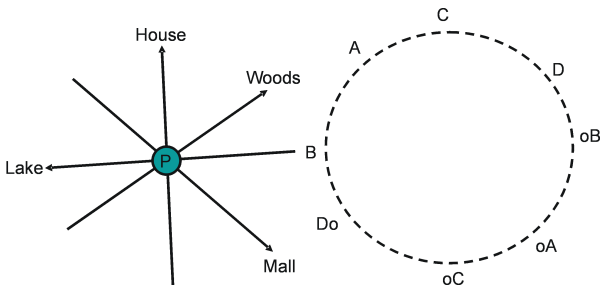
The concept of panorama representation has been studied extensively in the course of specialized sensors (e.g., omnivision, see, e.g., [34]). We present an extended approach based on the panorama approach by Schlieder ([20], [21]and [22]).

A complete, circular panorama can be described as a 360^o view from a specific, observer-dependent point of view. Let *P* in Fig. 2(a) denote a person, then the panorama can be defined as the strict ordering of all objects: *house, woods, mall, lake*. This ordering, however, does not contain all ordering information as described by the scenario. The *mall* is not only directly between the *woods* and the *lake*, but more specifically between the opposite side of the *house* and the *lake* (the tails of the arrows). In order to represent the spatial knowledge described in a panorama scenario, [20] introduced a formal model of a panorama.

Definition 3.1 (Panorama) *Let* $\Theta = \{\theta_1, \ldots, \theta_n\}$ *be a set of points* $\theta_i \in \Theta$ *and* $\Phi = \{\phi_1, \ldots, \phi_n\}$ *the arrangement of n-1 directed lines connecting* θ_i *with another point of* Θ*, then the clockwise oriented cyclical order of* Φ *is called the panorama of* θ_i*.*

As a compact shorthand notation we can describe the panorama in Fig. 2(b) as the string $< A, C, D, Bo, Ao, Co, Do, B >$. Standard

[5] This term is also directly connected to a set of algorithms addressing exactly this problem (e.g., [2])



(a) Concrete panorama (b) Abstract panorama

Figure 2. Panorama-views

letters (e.g., A) describe reference points, and letters with a following *o* (e.g., Ao) the opposite side (the tail side). As the panorama is a cyclic structure the complete panorama has to be described by n strings with n letters, with n being the number of reference points on the panorama. In our example, the panorama has to be described by eight strings. Furthermore, the panorama can be described as a set of simple constraints $dl(vp, lm_1, lm_2)$[6]. Based on this representation, [21] also developed an efficient qualitative navigation algorithm.

The panorama representation has an additional, more important property: it is invariant with respect to rotation and translation. But evidently, not every behavior can be described in such an abstract manner. In order to model complex, coordinated behaviors, often more detailed ordinal information is involved. Additionally, different metric information (e.g., distance) is required in some situations. In the following section, we show how the panorama can be extended in a way that more detailed ordinal and metric information can be introduced.

4 An Extended Panorama Representation

Instead of building an allocentric map we provide an egocentric snapshot-based approach to navigation. The most fundamental difference between both approaches is that an egocentric approach strongly relies on an efficient, continuous update mechanism that updates all egocentric relations in accordance with the players' movement. In this section we show that this task can be accomplished by strict use of a simple 1D-ordering information, namely an extended qualitative panorama representation (EPR).

This update mechanism has to be defined with respect to some basic conditions:

- Updating has to be efficient since egocentric spatial relations change with every movement, i.e., the updating process itself and the underlying sensor process.
- The resulting representation should provide the basis for qualitative spatial descriptions at different levels of granularity.
- The resulting representation should provide different levels of abstraction, i.e., rotation and/or translation invariance.
- The process of mapping egocentric views should rely on a minimum of allocentric, external information.

Due to the nature of ordering information, this task has to be divided into two subtasks: (1) updating within a given frame of reference (short notation: FoR), i.e., the soccer field and (2) updating of landmark representations from an external point of view, e.g., the penalty area. In section 4.1 we briefly discuss the key properties of the first task in relation to ordering information from a more theoretical point of view, whereas in section 5 these aspects are investigated in more detail. In section 4.2 we describe the theoretical framework underlying the mapping- and update-mechanism for egocentric views on external landmarks.

[6] Short for $direct - left(viewpoint, landmark_1, landmark_2)$.

4.1 Within a Frame of Reference

A crucial property of panoramic ordering information is that it does not change as long as an agent stays within a given FoR, i.e., the corners of a soccer field, do not change unless the player explicitly leaves the field (see Fig. 3(a)). So in order to use ordering information for qualitative self-localization we have to introduce an egocentric FoR. But even with an egocentric FoR the location within this FoR can only be distinguished into a few different qualitative states (e.g., ego-front between front-left and front-right corner of the field, see Fig. 3(a)). This way of qualitative self-localization is too coarse for many domains as well as for the different RoboCup-domains. In section 5 we demonstrate in more detail how angular distances can be used to overcome this problem[7].

A perhaps even more important property of spatial locations within a given FoR is that they can be used as a common FoR for the position of different landmarks in relation to each other (e.g., the position of the penalty area can be described in within-relation to the soccer field). This property is especially important for an egocentric snapshot-based approach to navigation since it provides the common frame that is required to relate different snapshots to each other (for a more detailed discussion see [29]).

4.2 Updating Outside-Landmark Representations

In a re-orientation task we can resort the knowledge about the previous position of a player. Therefore we concentrate on an incremental updating process, based on the following two assumptions: (1) It is known that the configuration of perceived landmarks $A, B, ... \in L$ either form a triangle- or a parallelogram configuration (e.g. either by vision or by use of background knowledge). (2) The position P_{t-1} of an agent A in relation to L at time step $t-1$ is known. The EPR (LP_T) of a triangle configuration can then be defined as follows (see also Fig. 3(b)):

Definition 4.1 (Triangle Landmark Panorama) *Let P_A denote the position of an agent A and $C_{T(ABC)}$ the triangle configuration formed by the set of points A, B, C in the plane. The line $L_{P_A/VP}$ is the line of view from P_A to* VP*, with* VP *being a fixed point within $C_{T(ABC)}$. Furthermore, $L_{Orth(P_A/VP)}$ be the orthogonal intersection of $L_{P_A/VP}$. The panoramic ordering information can be described by the orthogonal projection $P(P_A, VP, C_{T(ABC)})$ of the points ABC onto $L_{Orth(P_A/VP)}$.*

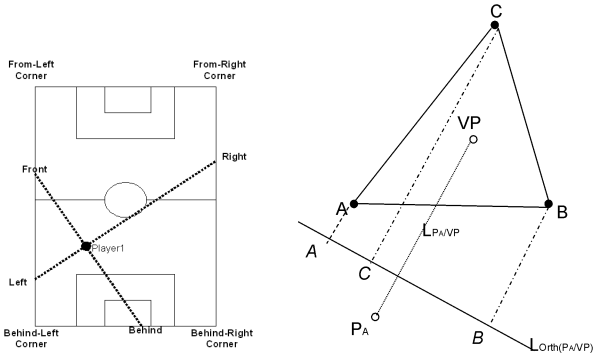
Therefore, moving around a triangle configuration $C_{T(ABC)}$ results in a sequence of panoramas which qualitatively describe the location of the observer position. A 360^o movement can be distinguished in six different qualitative states:

Observation 1 *(Triangle Landmark Panorama Cycle)*
The EPR resulting from the subsequent projection

[7] An additional approach is to introduce more landmarks that are easy to perceive or to introduce additional allocentric FoR when available (e.g., north, south, etc.)

$P(P_A, VP, C_{T(ABC)})$ *by counter-clockwise circular movement around* VP *can be described by the following ordered, circular sequence of panoramas:*
$(CAB), (ACB), (ABC), (BAC), (BCA), (CBA)$

For each landmark panorama the landmark panorama directly left as well as at the right differ in exact two positions that are lying next to each other (e.g., $(ABC), (BAC)$ differ in the position exchange between A and B). These position changes occur exactly when the view line $L_{P_A/VP}$ intersects the extension of one of the three triangle lines: L_{AB}, L_{AC}, L_{BC}. Starting with a given line (e.g., L_{AB}) and moving either clock- or counter-clockwise, the ordering of line extensions to be crossed is fixed for any triangle configuration (see Fig. 3(b)). This property holds in general for triangle configurations but not, e.g., for quadrangle configurations (except for some special cases as we will see below). Since (almost) each triplet of landmarks



(a) Use of Egocentric Frame of Reference (b) Triangle panorama construction by projection (result here: (ACB))

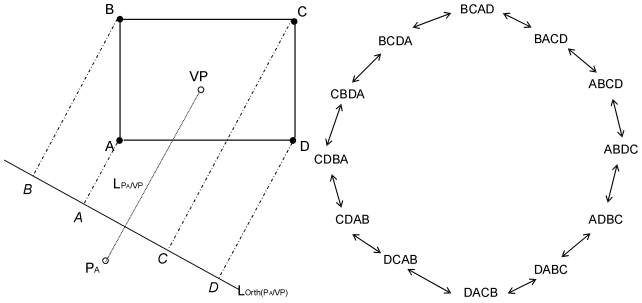
Figure 3. FoR and Triangle panorama.

can be interpreted as a triangle configuration, this form of qualitative self-localization can be applied quite flexibly with respect to domain-specific landmarks. The triangle landmark panorama, however, has (at least) two weaknesses: The qualitative classification of an agent's position into six areas is quite coarse and, triangle configurations are somewhat artificial constructs that are rarely found in natural environments when we consider solid objects[8]. A natural extension seems to be applying the same idea to quadrangles (see Fig. 4). The most direct approach is to interpret a quadrangle as a set of two connected triangles sharing two points by a common line so that each quadrangle would be described by a set of two triangle panoramas. With this approach, the space around a quadrangle would be separated into ten areas and therefore it would be more expressive than the more simple triangle panorama. It can be shown that eight of the resulting triangle landmark panorama (one for each triangle of the quadrangle) can be transformed into quadruple that results when we transform e.g. a rectangle directly into a landmark panorama representation (e.g., the above given tuple ((BCA)(CDA)) can be transformed into (BCDA) without loss of information)[9]. The expressiveness of the other two landmark panoramas is weaker: they have to be described as a disjunction of two quadruple tuples. Since the expressiveness is weaker and the landmark panorama representation of a quadruple tuple panorama representation is much more intuitive we focus on the latter one (see Fig. 4(a)).

Definition 4.2 (Parallelogram Landmark Panorama) *Let P_A denote the position of an agent A and $C_{P(ABC)}$ the parallelogram configuration formed by the set of points A, B, C, D in the plane. The line $L_{P_A/VP}$ is the line of vision from P_A to* VP*, with* VP *being a fixed point within $C_{P(ABCD)}$. Furthermore, $L_{Orth(P_A/VP)}$ be the orthogonal intersection of $L_{P_A/VP}$. The landmark panoramic ordering information can then be described by the orthogonal projection $P(P_A, VP, C_{P(ABCD)})$ of the points $ABCD$ onto $L_{Orth(P_A/VP)}$.*

Moving around a parallelogram configuration $C_{P(ABCD)}$ also results in a sequence of landmark panoramas which describe the location of the observer position qualitatively. A 360^o movement can be split into twelve different states:

Observation 2 *(Parallelogram Landmark Panorama Cycle)*
The panoramic landmark representations resulting from the subsequent projection $P(P_A, VP, C_{P(ABCD)})$ by counter-clockwise circular movement around VP *can be described by the following ordered, circular sequence of panoramas:*
$((BCAD), (BACD), (ABCD), (ABDC), (ADBC), (DABC), (DACB), (DCAB), (CDAB), (CDBA), (CBDA), (BCDA))$



(a) Use of Egocentric Frame of Reference (b) Triangle panorama construction by projection (result here: (ACB))

Figure 4. Parallelogram panorama

The two presented landmark panoramas can be mapped flexibly onto landmarks that can be found in natural environments like a penalty area. While solid objects often form rectangle configurations, irregular landmarks can be used in combination as a triangle configuration, since this approach is not strictly restricted to point-like objects. An interesting extension is to build up more complex representations by using landmark configurations as single points in larger landmark configurations. This allows us to build up nesting representations which support different levels of granularity according to the requirements of the domain.

5 Implementation

According to the described scenarios, the EPR is meant to be a qualitative fundament for tasks that are important for mobile robot exploration. The latter part described in 4.2 is not adequate for the RoboCup scenario because there is almost no structure for the robot to move around. Here, we will show some experimental extraction

[8] The triangle configuration can be applied generally to any triplet of points that form a triangle - also to solid objects. The connecting lines pictured in Fig. 3(b) and 4(a) are used to explain the underlying concept of position exchange (transition).

[9] The detailed proof will take too much space.

of EPR sequences to practically point up the idea presented in section 4.1 and the basic idea of building panoramic ordering information from the image data.

For our first experiments, we use the *RobotControl*/*SimRobot* [18] simulation environment for the simulation of one four-legged robot. This tool is shared with the GermanTeam, which is the German national robotic soccer team participating in the Sony four-legged league in the international RoboCup competitions. The EPR concept presented is not proposed to be restricted to this special domain, as discussed. The tool supports simulated image retrieval and motion control routines that are easy to use and portable to physical robots, while it is possible to encapsulate the EPR and adapted image feature extraction in distinct solutions, letting other modules untouched.

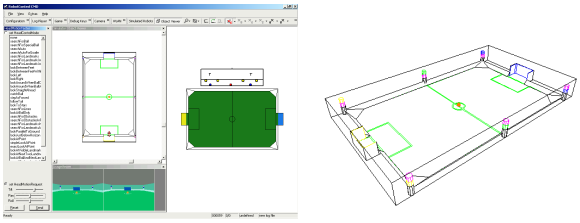


Figure 5. Simulation environment of the GermanTeam (left); the standard four-legged league field configuration (right).

5.1 Visual Feature Extraction

In order to expediently fill the EPR with information, the recognition of landmarks is necessary. Usually, the robot's viewing angle of 57.6^o degrees is not sufficient to get a reasonably meaningful EPR with the feature extraction of goals and flags supported by the *RobotControl* tool (see [19] for a description of these features).

Even if the scene is perceived from one goal directly to the other, there are just three landmarks that can be found. On the other hand, the standard configuration of all landmarks as can be seen in Fig. 5 is of an unfavorable kind for the EPR. The landmarks build a convex structure which the robot never can leave, thus the ideal EPR will never allow to reason about the environment by permuted landmarks (see section 4.1). For robots with common cameras, searching for localization markers additionally inhibits from concentration on essential game objects like the ball. Thus, it is only possible to either localize the robot or to capture the ball at a point of time, accordingly it would be more efficient to concentrate on the field for localization. A possible and more intelligent solution could be the extraction of field lines. We further introduced the symmetry line operator proposed by Huebner [7] to address these problems by extracting 2D field lines from the image data as additional features.

5.2 Symmetry Operator

Our line detection method is based on a compact 1-dimensional symmetry operator for arbitrary images [6]. For each pixel of the image, a qualitative value of reflective symmetry in horizontal or vertical direction is determined. *Vertical symmetry* is defined as symmetry of a vertical axis, thus only pixels in the same image row $R = [p_0, p_{w-1}]$ have to be considered for the detection of vertical symmetry of a pixel $p_i \in R$, where w is the width of the image. The same is applied for horizontal symmetry regarding only one column of the image. Furthermore, robot vision requires processing of real images. Because of the common image distortion in real images, an operator detecting exact, mathematic symmetry fails and offers erroneous symmetry images. Therefore, we propose the following qualitative symmetry operator based on a normalized mean square error function:

$$S(p_i, m) = 1 - \frac{1}{C \cdot m} \sum_{j=1}^{m} \sigma(j, m) \cdot g(p_{i-j}, p_{i+j})^2 \qquad (1)$$

where $m > 0$ is the size of the surrounding of p_i in which its value of symmetry shall be detected. Thus, the complete number of pixels considered is $2m$. C is a normalization constant depending on the used color space and on $\sigma(j, m)$, which is a radial weighting function. The difference between two opposing points p_{i-j} and p_{i+j} is determined by a gradient function $g(p_{i-j}, p_{i+j})$, which usually is the Euclidian distance of the corresponding color vectors $\overline{p}_{i-j}$ and $\overline{p}_{i+j}$. For all presented experiments, we used 8-bit gray-scale representation with

$$g(p_{i-j}, p_{i+j}) = \begin{cases} \|\overline{p}_{i-j} - \overline{p}_{i+j}\| & \text{if } p_{i-j} \in R \wedge p_{i+j} \in R \\ c & \text{otherwise} \end{cases} \qquad (2)$$

where c is the maximum error available (depending on color space), and a linear weighting function additionally depending on m

$$\sigma(j, m) = 1 - \frac{|j|}{m + 1} \qquad (3)$$

Important symmetry axes can be found at places where not necessarily high symmetry values but symmetry peaks can be detected. Though the extraction of maxima and minima of a symmetry image causes more distortion in resulting binary images, it is more significant than using a threshold value. Thresholds may vary from application to application or even from image to image. Additionally, appropriate thresholds are difficult to find for normalized symmetry. A symmetry value of 0 corresponds to hard black-white transitions between each pair of opposing points p_{i-j} and p_{i+j}, while a value of 1 corresponds to exact parity. Thus, high symmetry values are more frequent and much denser, which makes threshold setting very ineffective. Symmetry is more adapted for the application of local extrema, since it is a regional feature characterizing the local environment (in contrast to local features like edges). Since calculation of vertical symmetry in one row is independent of those in other rows or columns, maxima and minima can be detected line by line and column by column, respectively for horizontal symmetry. Results of this symmetry axes detection are presented in Fig. 6. Note that each result has been achieved by only using the symmetry operator and maximum detection, without any kind of pre- or post-processing like Gauss filtering, segmentation, or related techniques.

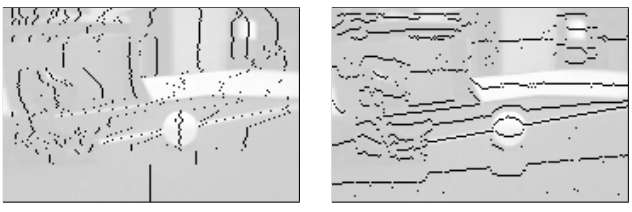


Figure 6. Symmetry maxima of a RoboCup image using $m = 5$ for vertical (left) and horizontal (right) symmetry axes extraction.

5.3 Symmetry Line Detection

Line extraction techniques usually need some preprocessing, e.g., edge detection, thresholding or thinning. Using symmetry, we can detect lines as a structure from arbitrary images. For example, a horizontal line is a structure where we should continuously detect a given A_1SA_2-pattern ($= A\text{symmetry}_1 - S\text{ymmetry} - A\text{symmetry}_2$) in each small vertical neighborhood along the line. Actually, A_1S is sufficient, because the symmetry axis S implies that there is another A_2 symmetric to A_1. An example for detection of this structure is shown in Fig. 7, where only horizontal symmetry axes using $m = 3$ were detected. If the specific AS-pattern can be found in the same environment, we can assume that it is part of a line.

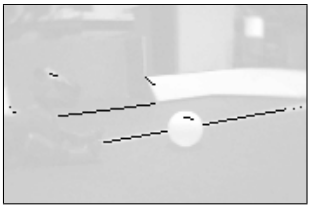


Figure 7. Filtered horizontal line points of Fig. 6 by AS pattern.

In the following, two approaches are presented to extract lines from the images resulting from the proposed symmetry line filter. The first one is a modified Hough approach using the Wallace Muff space [30], which represents a line by its start and end point on the image border rectangle. The Muff parameter space (see Fig. 8) shows that there are two lines which lead from the left to the right side of the symmetry line image. The results seem quite acceptable, but several adaptation were required steps, due to the difficulty to extract maxima in Muff space. There are further disadvantages of this approach, for example, a line now is represented by its image border points, thus information about line segments is lost. Additionally, curve segments may also be detected as lines with this method, which in itself is complex enough without a modified Hough transform for circle detection.

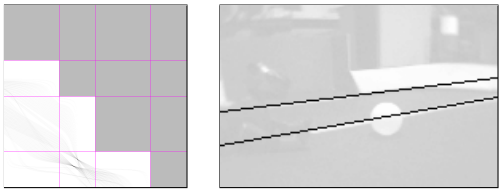


Figure 8. Muff parameter space of line points in 7(left). Lines found in Muff space (right).

Because of these disadvantages, we developed another approach which takes advantage of the fact that most feature points of the symmetry line image only have one or two neighboring feature points. Simply using the number of feature points in the 3×3-neighborhood of a point p, each feature point can be classified as one of the following types:

- A: if p has **no** neighbor, it is not interesting for line extraction.
- B: if p has **one** neighbor, it is start or end point of a line.
- C: if p has **more** neighbors, it is part of a line.

Thus, we only have to search for feature points of type B (a line's start point) and recursively search the next neighboring point of type C, until we find another point of type B (the line's end point). Therefore, we use the search patterns described in Fig. 9 which are rotational invariant:

Figure 9. The two search patterns for line segmentation.

Suppose X and Y are points of type B or C, and Y has been detected as the neighbor of X. Now we can start searching the neighboring fields as proposed, until we find a new feature point. If no feature point is found, Y is the end point of the current line, otherwise we proceed with Y in the same manner. Note that the fields left empty can not be occupied by feature points because of the symmetry maxima detection.

Each line segment can now be represented as the list of feature points found by this method. Based on this representation, we can access further information about the line, e.g. the variance of each point to the line described by start and end point. This measure is very useful to easily distinguish curves from straight lines, because the maximum variance will probably not exceed a few pixels in the case of a straight line (see Fig. 10 for an example). As proposed, the performance of this method is quite acceptable. Additionally, it is more compact and faster than the Muff space approach. It needs less adaptation, but offers extraction of line segments and classification of curves. In each presented case, we had to search for thin white

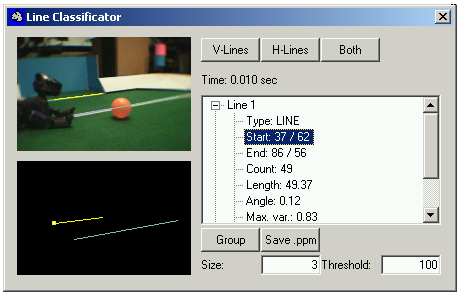


Figure 10. Screenshot of the Line Classificator Dialog.

horizontal lines. Thus, we applied the horizontal symmetry operator using $m = 3$ and included an illumination threshold neglecting those feature points having a gray-scale value smaller than 100 in the source image. Additionally, we disregarded lines shorter than a given threshold and implemented a heuristic to combine line segments, in the case that they seem to belong to the same field line, but are disrupted by occluding objects.

5.4 Landmarks of the EPR

The proposed method for line extraction is simple, robust, and works without plenty of parametrization. Additionally, it offers the opportunity to test the approach with natural landmarks (lines) instead of ar-

tifacts (colored beacons). After processing the images, lines are distinguished from curves and represented by their start and end point in the image.

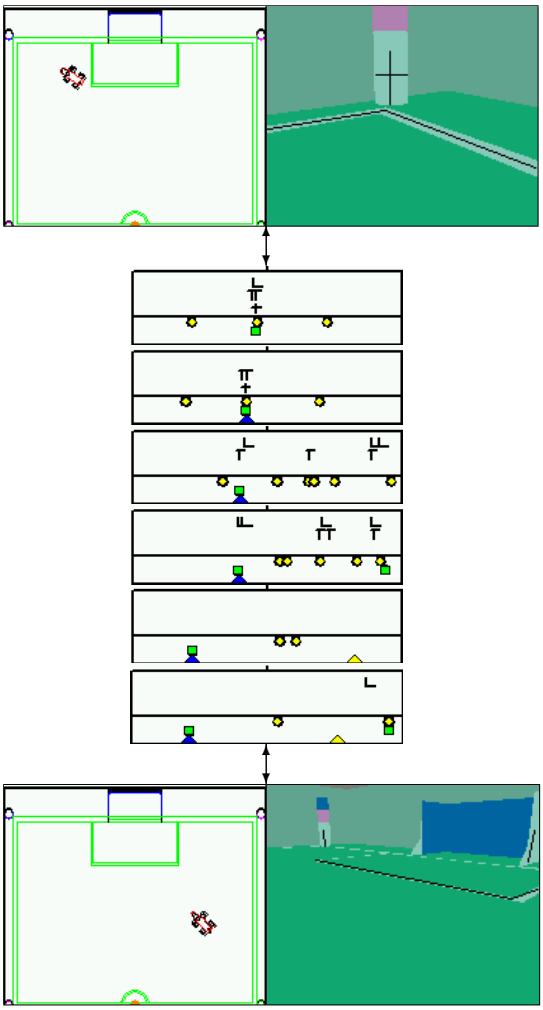


Figure 11. Landmarks for the EPR. Center column: Landmarks extracted (for six representation between given start position (left) and goal position (right): “L” for L-junctions, “T” for T-junctions, “X” for X-junctions; horizontal lines (yellow), vertical lines(green), goals(red) and flags(blue).

Those lines can be put into the EPR by adopting these points or the center point, for example. Anyway, a classification of edge types is more efficient with respect to the subsequent need of recovering landmarks. To support the panorama with a broader range of landmark types which ideally are points on the field, we can classify each pair of lines extracted from an image into different line pair types. In our experiment, we extracted L-junctions, T-junctions and X-junctions (see Fig. 11). These edge-extracted features represent the additional landmarks that are used for the EPR.

5.5 Qualitative Representation

The simulated environment for the experiment corresponds to the standard four-legged league field configuration with lines instead of sideboards. One robot is instructed to move a certain path presented by a given sequence of EPRs. Using the EPR representation and a qualititative conversion of the feature angles, we can establish a qualitative EPR sequence of detected landmark configurations for a path. Some samples of such sequences might look like the following, corresponding to the EPR of Fig. 11:

```
[(T_JUNC,FAR);(L_JUNC, SAME);(T_JUNC,SAME);
 (X_JUNC,SAME);]

[(T_JUNC,FAR);(X_JUNC, SAME);(FLAG,SAME);
 (T_JUNC,SAME);]

[(T_JUNC,FAR);(FLAG,SAME);(L_JUNC,CLOSE);
 (T_JUNC,MEDIUM);(T_JUNC,MEDIUM);(L_JUNC,SAME);
 (L_JUNC,CLOSE);]

[(FLAG,FAR);(L_JUNC,SAME);(L_JUNC,CLOSE);
 (T_JUNC,MEDIUM);(L_JUNC,SAME);(T_JUNC,CLOSE);
 (L_JUNC,MEDIUM);(T_JUNC,SAME);]

[(FLAG,MEDIUM);(GOAL,FAR);]
[(FLAG,MEDIUM);(GOAL,FAR);(L_JUNC,CLOSE);]
```

As can be seen in this example, the line landmarks appear and disappear frequently in the robot’s view. This is caused by the landmark feature extraction working on insufficient simulated image data. We are optimistic that real images are more comfortable for the extraction of lines because they are not supposed to be fragmented like those in simulated images. Although this is error-prone in this regard, we claim to deal with this problem using the EPR. The representation can generally be useful for this re-orientation task, where the agent knows at least to some extent where it has been. Based on this information, the circular panorama landmark representation can tell us which hypotheses are plausible according to previous information.

The same panoramic representation is additionally used in our simulation soccer team *Virtual Werder*. Although sensor problems are neglectable since the world model is more comprehensive and detailed, it provides a simple and intuitive interface for the generation of qualitative descriptions.

5.6 Experiments on Real Images

Finally, some experiments have been made to test the proposed feature extraction and EPR construction on real images (see Fig. 12)[10] using one Sony AIBO ERS-7 model inside a common four-legged league scenario. Without plenty of adaptation, the results are as good

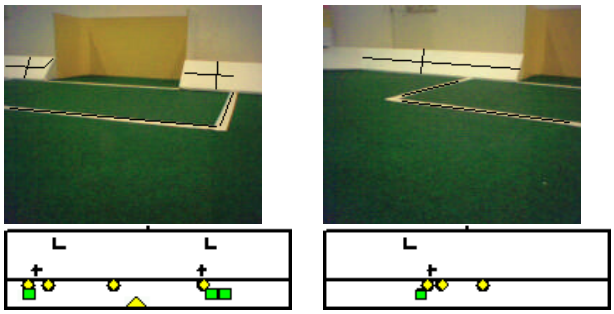


Figure 12. Landmarks for the EPR on real images. Top row: image data and extracted field / border lines. Bottom row: Landmarks extracted.

as those in the simulation examples. Problems appearing by the line

[10] The difference of size in the corresponding images is caused by the different image sizes of the old AIBO model ERS-210 to the new ERS-7.

extraction technique (e.g. side walls as lines, lines found over horizon, optional grouping of lines to handle occlusions) will be addressed in future work to increase robustness and performance.

6 Conclusion and Future Work

Navigation, localization, planning, and reasoning for physically grounded robots imposes strong but heterogeneous requirements on the underlying spatial representation in terms of abstraction and precision. In contrast to many other approaches to this topic which try to generate *allocentric* maps, we proposed a new *egocentric* approach based on recent results from cognition. The qualitative EPR is dynamic in a predictable way for outside landmarks as stated in the two observations described above. This representation, however, provides also interesting properties for navigation inside fixed landmarks (e.g., navigating within a room).

Besides the re-orientation task mentioned in the last section, the landmark panorama can help to focus perception in a qualitative self-allocation task. During the transition of one panorama landmark into another exactly one position change is performed. Therefore, in this case the perception of further landmarks is without any use for updating the qualitative position of the agent. Additionally, the panorama landmark representation is not only useful for position updating but also for re-orientation without knowledge about the previous position. The perception of a partial landmark panorama of a triangle configuration is sufficient to provide us with two hypotheses about the current position. In order to validate which hypothesis holds we just have to find out where another landmark appears in the panoramic structure. Addionally, a landmark panorama provides a stable basis for qualitative, spatial descriptions (e.g. left of, right of), since it is, obviously, sensitive to rotation but invariant to transition, it is also interesting for several outstanding applications based on qualitative information.

Although a detailed analysis of the relation to the recent cognitive results is out of the scope in this paper, we want to mention that the EPR shows several properties which are observed in recent experiments: e.g., translation tasks seem to be performed more easily and accurately than rotation tasks.

Several tasks remain to be done. We are currently extending our landmark-based (re-)orientation vision module so that it is not only able to track EPRs but also allows active snapshot-based navigation (first results are available). Thereby we implement the concept of outside-landmarks that formally describes how landmarks can shift and switch during movement (see section 4.2). This should also allow to detect the geometric structure of previously unseen objects. After validating our extended panorama representation in the RoboCup-domain, we consider to transfer this method of the EPR into an omnidirectional vision module for mobile robot tasks.

7 Acknowledgements

The presented work is being funded by the German Research Foundation (DFG) within the project *Automatic Plan Recognition and Intention Recognition of Foreign Mobile Robots in Cooperative and Competitive Environments* as part of the Priority Research Program SPP-1125 *Cooperative Teams of Mobile Robots in Dynamic Environments*.

REFERENCES

[1] D. Busquets, C. Sierra, and R. L. De Mantaras, 'A multiagent approach to qualitative landmark-based navigation', *Autonomous Robots*, **15**(1), 129–154, (2003).
[2] H. Durrant-Whyte, S. Majumder, S. Thrun, M. de Battista, and S. Schelling, 'A bayesian algorithm for simulaneous localization and map building', in *Proceedings of the 10'th International Symposium on Robotics Research (ISRR'01)*, Lorne, Australia, (2001). AAAI Press/MIT Press.
[3] R. D. Easton and M. J. Sholl, 'Object-array structure, frames of reference, and retrieval of spatial knowledge', *Journal of Experimental Psychology: Learning, Memory and Cognition*, **21**(2), 483–500, (1995).
[4] Kutluhan Erol, James Hendler, and Dana S. Nau, 'HTN planning: Complexity and expressivity', in *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, volume 2, pp. 1123–1128, Seattle, Washington, USA, (1994). AAAI Press/MIT Press.
[5] B. Garsoffky, S. Schwan, and F. W. Hesse, 'Viewpoint dependency in the recognition of dynamic scenes', *Journal of Experimental Psychology: Learning, Memory and Cognition*, **28**(6), 1035–1050, (2002).
[6] K. Huebner, 'A 1-Dimensional Symmetry Operator for Image Feature Extraction in Robot Applications', *The 16th International Conference on Vision Interface*, 286–291, (June 2003).
[7] K. Huebner, *A Symmetry Operator and its Application to the RoboCup*, 7th International Workshop on RoboCup 2003 (Robot World Cup Soccer Games and Conferences). Lecture Notes in Artificial Intelligence, Springer Verlag, 2004. To appear.
[8] S. Johansson and A. Saffiotti, 'Using the Electric Field Approach in the RoboCup Domain', in *RoboCup 2001: Robot Soccer World Cup V*, eds., A. Birk, S. Coradeschi, and S. Tadokoro, number 2377 in LNAI, 399–404, Springer-Verlag, Berlin, DE, (2002).
[9] Ulrich John, 'Solving large configuration problems efficiently by clustering the conbacon model', in *IEA/AIE 2000*, pp. 396–405, (2000).
[10] Benjamin Kuipers, 'The spatial semantic hierarchy.', *Artificial Intelligence*, **119**(1-2), 191–233, (2000).
[11] J. C. Latombe, *Robot Motion Planning*, volume 18, Kluwer Academic Press, 1991.
[12] T. S. Levitt and D. T. Lawton, 'Qualitative navigation for mobile robots', *Artificial Intelligence*, **44**, 305–360, (1990).
[13] Massimo Lucchesi, *Choaching the 3-4-1-2 and 4-2-3-1*, Reedswain Publishing, edizioni nuova prhomos edn., 2001.
[14] C. Martin and S. Thrun, 'Online acquisition of compact volumetric maps with mobile robots', in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, (2002).
[15] J. O'Keefe and A. Speakman, *The Hippocampus as A Cognitive Map*, Oxford, Clarendon Press, 1978.
[16] T. J. Prescott, 'Spatial representation for navigation in animats', *Adaptive Behavior*, **4**(2), 85–125, (1996).
[17] A.D. Redish, *Beyond the Cognitive Map - From Place Cells to Episodic Memory*, The MIT Press, Cambridge, Mass., 1999.
[18] T. Roefer, *An Architecture for a National RoboCup Team*, 417–425, RoboCup 2002: Robot Soccer World Cup VI. Lecture Notes in Artificial Intelligence, Springer Verlag, 2003.
[19] T. Roefer and M. Juengel, 'Vision-Based Fast and Reactive Monte-Carlo Localization', in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA-2003)*, pp. 856–861, Taipei, Taiwan, (2003).
[20] C. Schlieder, *Ordering information and symbolic projection*, 115–140, Schlieder, C. (1996). Ordering information and symbolic projection. In S. K. Chang, E. Jungert and G. Tortora (Eds.), Intelligent image database systems. Singapore: World Scientific Publishing.
[21] C. Schlieder, 'Representing visible locations for qualitative navigation', 523–532, In: Qualitative reasoning and decision technologies, N. Piera-Carrete & M. Singh (Eds.) CIMNE Barcelona [http://www.iig.uni-freiburg.de/cognition/members/cs/cs-pub.html], (1993).
[22] Christoph Schlieder, *Anordnung und Sichtbarkeit - Eine Charakterisierung unvollständigen räumlichen Wissens*, Ph.D. dissertation, University Hamburg, Department for Computer Science, 1991.
[23] Michael K. Smith, Chris Welty, and Deborah L. McGuinness, 'Owl web ontology language guide', W3c candidate recommendation 18 august 2003, (2003). http://www.w3.org/TR/owl-guide/.
[24] S. Thrun, M. Beetz, M. Bennewitz, W. Burgard, A. B. Cremers, F. Dellaert, D. Fox, D. Hähnel, C. Rosenberg, N. Roy, J. Schulte, , and D. Schulz, 'Probabilistic algorithms and the interactive museum tour-

guide robot minerva', *International Journal of Robotics Research*, **19**(11), 972–999, (2000).
[25] S. Thrun, A. Bücken, W. Burgard, D. Fox, T. Fröhlinghaus, D. Henning, T. Hofmann, M. Krell, and T. Schmidt, *AI-based Mobile Robots: Case Studies of Successful Robot Systems*, chapter Map learning and high-speed navigation in RHINO, MIT Press, 1998.
[26] S. Thrun, D. Fox, W. Burgard, and F. Dellaert, 'Robust monte carlo localization for mobile robots', *Artificial Intelligence*, **128**(1-2), 99–141, (2000).
[27] S. Thrun, D. Hähnel, D. Ferguson, M. Montemerlo, R. Triebel, W. Burgard, C. Baker, Z. Omohundro, S. Thayer, and W. Whittaker, 'A system for volumetric robotic mapping of abandoned mines', in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, (2003).
[28] E. C. Tolman, 'Cognitive maps in rats and men', *Psycholoical Review*, **55**, 189–208, (1948).
[29] Thomas Wagner, Christoph Schlieder, and Ubbo Visser, 'An extended panorama: Efficient qualitative spatial knowledge representation for highly dynamic enironments', *IJCAI-03 Workshop on Issues in Desgning Physical Agents for Dynamic Real-Time Environments: World Modeling, Planning, Learning, and Communicating*, 109–116, (2003).
[30] R.S. Wallace, 'A Modified Hough Transform For Lines', *IEEE International Conference on Computer Vision and Pattern Recognition*, **85**, 665–667, (1985).
[31] R. F. Wang, 'Representing a stable environment by egocentric updating and invariant representations', *Spatial Cognition and Computation*, **1**, 431–445, (2000).
[32] R. F. Wang and E. S. Spelke, 'Updating egocentric represenations in human navigation', *Cognition*, **77**, 215–250, (2000).
[33] R. F. Wang and E. S. Spelke, 'Human spatial representation: Insights from animals', *Trends in Cognitive Science*, **6**(9), 176–182, (2002).
[34] J.Y. Zheng and S. Tsuji, 'Panoramic representation for route recognition by a mobile robot', *International Journal of Computer Vision*, **9**(1), 55–76, (1992).

.

Line Based Robot Localization under Natural Light Conditions

Artur Merke[1] and **Stefan Welker**[2] and **Martin Riedmiller**[3]

Abstract. In this paper we present a framework for robot selflocalization in the robocup middle size league. This framework comprises an algorithm for robust selflocalization and a set of benchmarks which can be used offline to test other algorithms and to compare their outcomes with our results. The algorithm is part of our competition team Brainstormers-Tribots which won the Robocup German Open 2004. This is a multi agent real time environment, therefore our algorithm is prepared to work with 30 frames per second, leaving enough time for other tasks like robot control or path planning. Our approach uses a particle filter method relying on features found in the image. The features are points on field lines. They can be recognized reliably under natural light conditions, so the is no longer a need for a well defined and constant light source. Also color coded landmarks or goals are not required for a stable selflocalization. We present results for different runs on our benchmark suite, which is an outdoor soccer field with the size of 16x10m. This field size is bigger then in current competitions and anticipates the trend of using larger fields in future competitions.

1 Introduction

Since 1996, when the first Robocup competition took part, there was a steady pursuit of making the Robocup environment more realistic and less artificial. Certainly one can argue that there were not enough changes in this direction, as the games are still conducted under well defined artificial floodlight. Also different color coded landmarks are used for self-localization on the field. The use of such color coded landmarks strongly relies on color classifiers, which are very sensitive to external light conditions. So to get a system which works under natural light condition one has to extract more shape oriented features from the images.

In this paper we present a method which relies on easily extractable shape information, which can be robustly recognized under different light conditions. Our method uses particle filtering and is a significant extension of the method used in our Brainstormers-Tribots team in 2003. The old method worked well and our team scored 5 wins, 1 draw and conceived 2 defeats (scoring 26:8 goals altogether) in the world championships in Padova 2003. But there were also foreseeable limitations. The old method relied on color coded poles and goals. For example distant poles were easily overlooked (they can be few pixel large due to the geometry of the mirror) or other colored objects could be mistaken for poles or goals.

In our new method we only detect points on lines along rays radially arranged around the center of the omni-directional camera image. As such points can be recognized without highly tuned color classification (see section 4 for more details) we were able to conduct different benchmarks under natural light conditions. Also such features are insensitive to varying surroundings, so for example the robot cannot get misled by different colored objects in the audience. Another advantage is that we can now self localize on larger fields, as we don't rely on specific distant and therefore small features. Using particle filter methods enables us to estimate the position of the robot very exactly on a 16x10 meters large outdoor field (see figure 1 in section 2). For example driving and turning the robot (using omni-directional drive) for 30 seconds across the field with a speed of 1.5 m/s, the self-localization deviates only approximately 20 cm on average from the reference path (with maximal deviation of 50 cm). See section 5 for further results.

The second most important aspect of this paper is the reproducibility of the presented results. We compiled a set of 25 runs of our robot on an 16x10 meters large outdoor field. Each run consists of all image information (e.g. 30 fps), the gathered odometry data, and externally measured reference robot positions during the whole run. These reference positions were measured with a laser scanner positioned outside the field, and can be used to evaluate the accuracy of the used algorithms.

To our knowledge it is the first benchmark in the Robocup middle size league for self localization. We hope this will encourage other researchers (also such which haven't yet participated in Robocup, but are active in the machine vision field) to compare their algorithms against our benchmark. The setting of our benchmark excels current rules of the middle size league (larger field, natural light). We hope that this is a chance to test current algorithms for coming requirements, and that this also will accelerate the process of making the conditions in the middle size league more realistic.

2 Environment

In this section we describe the setting of our experiments. All experiments for this paper were conducted on an outdoor field with the length of 16 meters and the width of 10 meters, see figure 1. We have chosen a field covered with tartan, be-

[1] Universität Dortmund, Germany, artur.merke@udo.edu
[2] Universität Dortmund, Germany, stefan.welker@udo.edu
[3] Universität Osnabrück, Germany, martin.riedmiller@uos.de

Figure 1. Outdoor field, 16 meters long, 10 meters wide

cause on surfaces like asphalt or concrete the omni-directional wheels of our robot do not have enough grip and also fret extremely. Originally we looked for a field with the dimensions 16x12 meters but could not find a tartan field in such size without disturbing lines. As the fields in Lisboa in 2004 will have the dimensions of 12x8 meters we are still much ahead of the current Robocup requirements.

It was very important to us to use an outdoor field, so that we could demonstrate self-localization under natural light conditions. As it was quite difficult to find a large enough tartan field without disturbing lines it was even more difficult to find one covered with green tartan. Here we decided for a compromise, weighting the outdoor conditions and large size more then a specific surface color. Our field is therefore colored dark red with slight color intensity variations (the tartan is quite old and soiled). Meanwhile we consider the different surface color as an additional challenge, actually real soccer is also played on different fields not always lawn covered, but also covered with red ash (at least in the lower soccer leagues).

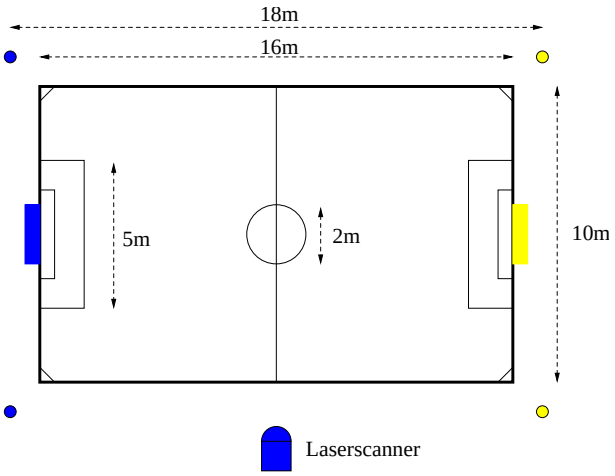


Figure 2. Field dimensions

The dimensions of the field and positions of lines, goals and poles conform to the current Robocup rules for the year 2004 [2], see figure 2. The poles and goals are not needed in our algorithm, but we positioned them on the field such that groups with other approaches could also use our benchmarks. The outer lines are 12 cm thick, all inner lines are 6 cm thick.

Outside the field we stationary positioned a laser range scanner. This enables us to scan the robot path for each run. Afterwards the scanned positions (each position has an unique time stamp) of the robot can be used to verify the obtained algorithmic results.

Figure 3. Tribot, omni-directional drive and camera

For all runs we used our standard competition robot - the Tribot. See figure 3 for a picture of the Tribot. It has an omni-directional drive and can for example drive towards the ball and rotate at the same time (see [1] for more details). During a run the robot records camera images and odometry data. In the collected odometry data we abstract from this particular drive, and only collect the velocity in direction x and y and the rotation velocity ϕ of the robot. So using the benchmark one is not confined to this particular robot drive.

The images recorded by the robot are omni-directional, because the camera captures the reflections of the field produced by a hyperbolical mirror (see [1] for more details). Due to a calibration process one can compute for every pixel in the image the corresponding real world distance. This matches the reality as long the recognized objects stand on the ground. The accuracy of the distance estimation decreases significantly with the distance from the robot. In figure 4 one can see two separate images from the camera, where one can also clearly recognize the different light conditions.

3 Benchmarks

At the moment our benchmark suite consists of 25 different robot runs. Currently each run is at least 11 and at most 35 seconds long. Using a frame rate of 30 frames per seconds and storing the images uncompressed in VGA resolution it requires between 200 and 600 megabyte disk space per run.

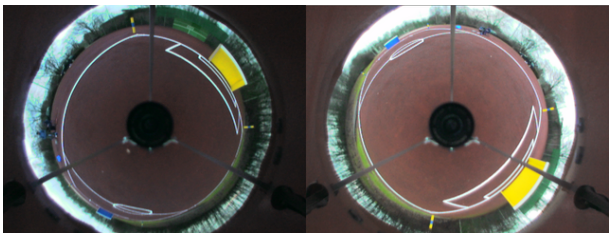


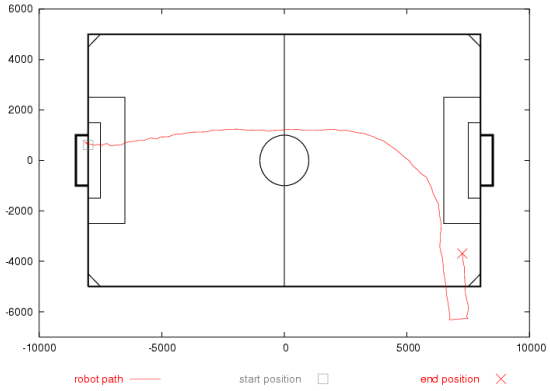
Figure 4. Two camera images showing different light conditions.

In figure 5 we depicted four exemplary robot runs, see [8] for a complete list. For example in figure 5 (a) the robot starts in front of the blue (=left) goal, drives across the whole field almost crossing the right penalty area, exits the field near the right bottom pole and enters it again driving toward the yellow goal. The sequence is 23 seconds long, consisting of 350 raw images (approx. 200 megabyte) taken with the frame rate of 15 frames per second. During the run approximately 4 laser measurements per second were recorded, resulting altogether in approx. 100 reference positions. These externally measured reference positions can be used to measure the quality of the deployed algorithms. As another example the run in figure 5 (d) is 35 seconds long, consists of 1050 raw images (because of the doubled frame rate) with approximately 160 reference positions.

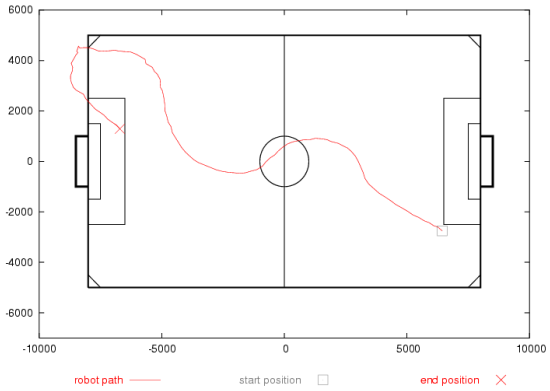
The collection of benchmarks is supposed to grow in the future. The current set of 25 benchmarks can be found at [8]. At this URL we also provide source code for reading and showing the raw images and related data. We hope that this will encourage other teams to use our benchmark suite and maybe also to contribute in extending the existing data base.

4 Algorithm

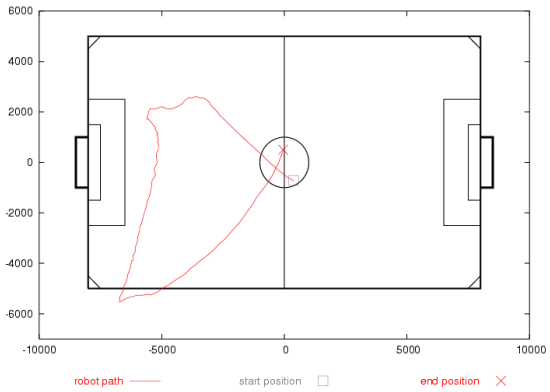
For localizing our robot in the Robocup environment we essentially rely on a camera as the primary sensor. A camera image provides very significant input data, a human can easily estimate the position of a robot by looking at the image. On the other hand a camera image can contain a lot of useless, even obstructive data such as image noise, light artifacts , color shift, brightness or camera shutter issues. It can be a difficult task to find an algorithm that recognizes meaningful features in an image, which can be used for localization. Most of the time a trade-off has to be made between speed and reliability. In the past most approaches were based on recognizing the color coded landmarks in the Robocup environment. Due to lightning conditions it can be hard to classify pixels safely to different color classes such as blue, yellow, green, orange or black. Therefore it is more robust to rely on shape oriented features. We decided to use the white field lines for localization, which can be recognized under varying light conditions. As our incremental algorithm makes do with even a small number of such features, lines occluded by obstacles and overlooked distant lines do not present a problem for our approach.



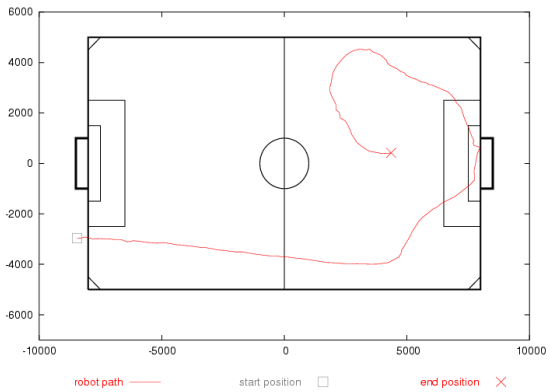
(a) speed 1.5 m/s, 15 fps



(b) speed 1.5 m/s, 15 fps



(c) speed 1.1 m/s, 30 fps



(d) speed 1.5 m/s, 30 fps

Figure 5. Example of different recorded robot paths.

4.1 Vision Architecture

With our vision system we present a fast detection method for pixels in the image that belong to the field lines. Our algorithm does not recognize the location and direction of the lines and corners in the image. Also color coded landmarks like poles and goals are not necessary. We do not use these additional features because our self-localization algorithm uses particle filtering which is a probabilistic method and the points on white lines are sufficient to get good and robust localization results.

To gather samples of points on lines in the image, we scan the image along several scan lines. These lines are radially arranged around the center of the omni-directional camera image. See figure 6 for an arrangement of these scan lines.

To recognize a line crossing along a scan line we search for significant variations in the color values. The variations are measured using an euclidean distance function in the YUV color space. By applying a threshold to these distances, we detect possible color transitions. Two consecutive transitions are recognized as a line transition if they are in close real world proximity to each other and the color before and after the transition show only a small color distance. This process gathers all kinds of line transitions in the image. To sort out transitions that do not belong to field lines an additional color validation is conducted. The deployed color classifier does not need to be non-ambiguous. The color classes may overlap and can therefore be tolerant enough cope with changes in light conditions during the classifying process.

The recognition of line transitions along the scan lines has the additional advantage of using only small amounts of computational resources. The image does not have to be segmented as a whole. This allows to run the system with 30 frames per second at a resolution of 640 x 480 pixels.

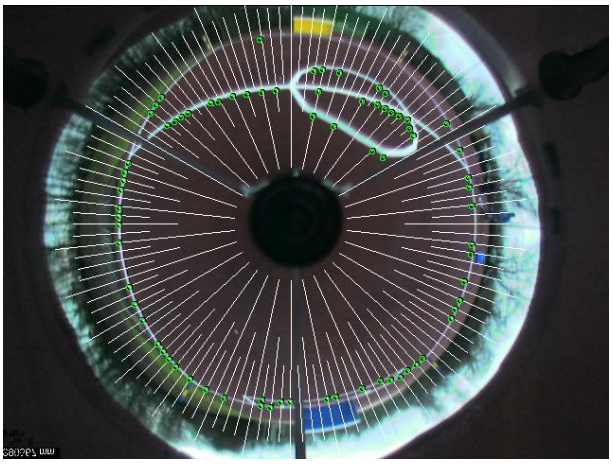


Figure 6. Analysis of an image with scan lines and recognized line transitions

Each line transition is a possible sensor measurement. To make it meaningful, we need its distance and angle to the center of the robot. To get the position of a pixel in the image in real world coordinates, we calibrate a distance mapping $D(r, \varphi)$ into every direction φ of the robot. The process of calibration is partially automated. A distance calibration environment has to be set up before the process. It consists of several color coded markings that are located at defined distances from the robot. While turning slowly, the robot recognizes these markers to gather data about the real distances of pixels in the image. With this data a complete distance mapping can be calculated for every pixel of the image. This mapping is only meaningful for objects that are located on the ground.

4.2 Self-Localization

In order to localize a robot correctly an appropriate estimate of the robot position $x_t \in R^n$ at time t has to be found. In our setting the position consists of the coordinates of the robot on the field and its relative orientation ($n = 3$). We utilize a sequential Monte Carlo method to generate the posterior probability distribution $\pi_{t|t}$ of the robot state x_t with regard to the prior distribution $\pi_{t-1|t-1}$. The former estimate $\pi_{t-1|t-1}$ is incrementally updated using new odometry data a_t and sensor values y_t. This is done in two stages.

First a *prediction* step is conducted using the odometry:

$$\pi_{t|t-1}(\cdot) = \int_{R^{n_x}} \pi_{t-1|t-1}(dx_{t-1}) K(\cdot|x_{t-1}, a_t) \qquad (1)$$

where $K(\cdot|x_{t-1}, a_t)$ denotes the Markov transition kernel for *action* a_t.

Afterwards an *update* step is performed using the current sensor values:

$$\pi_{t|t}(\cdot) = [\int_{R^{n_x}} g(y_t|x_t)\pi_{t|t-1}(dx_t)]^{-1} * g(y_t|x_t)\pi_{t|t-1}(\cdot) \quad (2)$$

where $g(y_t|x_t)$ is the conditional probability density of the observed sensor values with respect to the estimated position. See [4] for more details.

In particle filtering the real probability distribution $\pi_{t|t}$ is represented by a discrete probability measure using a set of N (currently about 200) weighted particles. The steps from equation 1 and 2 imply the following procedure of sequential importance sampling and resampling steps. This process consists of

1. Predicting new positions for particles while incorporating action information i.e. odometry
2. Updating the particle probability weights by estimating sensors input probability
3. Normalizing the probability weights of the particles
4. Resampling from the particle distribution to get the posterior distribution

In the following we will elaborate on the particularities of the steps 1, 2 and 4 in our approach.

Step 1. To predict the particles position by action we add the odometry reading a_t that consists of $(x_{odo}, y_{odo}, \varphi_{odo})$ for each particle. To represent the uncertainty of the odometry reading, gaussian noise is added to the particle location, proportional to the length of the odometry.

Step 2. We estimate the probability of obtaining the captured camera image at the location of every particle. This is done using an approximation which only relies on the detected

line transitions. For this approximation a product of all single transition probabilities is a reasonable estimate. To compute the probability of a single line transition, the location of the transition is mapped to a point p in the global coordinate system using the orientation and position of the considered particle. The minimal distance of this point p to the existing lines determines the probability value for the transition.

As this has to be done for every line transition and every particle, it can be very resource consuming process. Therefore we use a precalculated two dimensional look-up table of the field that provides the distance of every location on the field to the lines in $O(1)$.

Additionally the probability value for the transition depends on the distance of the point p to the position of the particle. This is because the distance measurement error of the vision system increases significantly for distant objects.

Step 4. To make the algorithm work the particles have to be resampled. Resampling statistically multiplies or discards particles at each time step to adaptively concentrate particles on regions of high posterior probability. This process consists of drawing N new Particles from the existing ones according to the particle weights using a multinomial distribution. In general this requires $O(n \log n)$ but can be done in $O(n)$ according to [5].

With these steps we can achieve robust incremental knowledge of the robot position by determining the average of the particle positions and headings.

5 Results

In this section we present the results for the sequences introduced in section 3. In this test we assume that the initial position of the robot is known, but our algorithm also solves the global localization (kidnapped robot) problem.

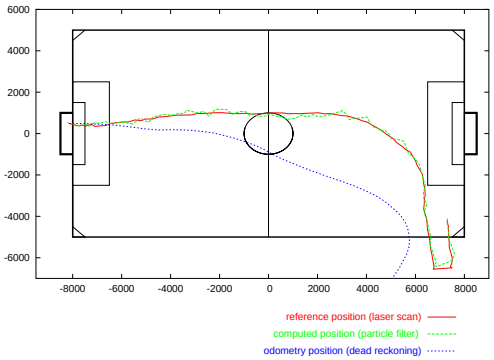


Figure 7. Sequence a, 15 frames/sec, 23 seconds

Sequence (a) (see Figure 7) shows the robot starting at the blue goal, driving across the whole field with a speed of 1.5 meters per second. The run is 23 seconds long and was captured at 15 frames per second. At the end of the run it leaves the field turns around and enters the field again. During this run the average deviation of the particle filter position to the laser scan position was only 18.1 cm, the maximum absolute error was 59.7 cm. In comparison to the large field size we only deviate by 1.8% of the field width and 1.1% with respect to the field length. This high quality of the result can be also seen in Figure 7 as the computed path lies very close to the reference position path. Also the deviation of the self-localization is uniformly accurate across the whole field, not only in regions close to lines. In contrast to the self-localization by particle filter the dead reckoning position obtained by the odometry measurements deviates very quickly from the real path and cannot be relied on for self-localization purposes.

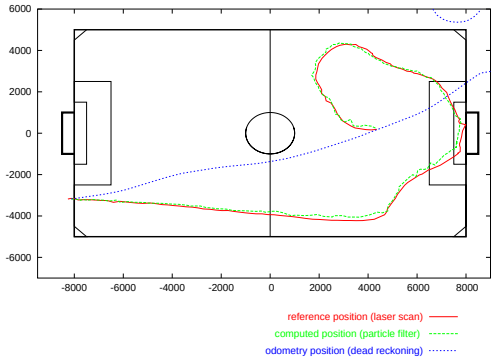


Figure 8. Sequence b, 30 frames/sec, 35 seconds

Sequence (d) (see Figure 8) shows a run with 30 frames per second and 35 seconds length. The robot starts at the side of the blue goal, drives across the field into the yellow goal region and back into the field while turning. Again the average error is only 17.4 cm and the maximum deviation 51.6 cm. This test run performed a little bit better which may be because of the increased frame rate. This shows that even when only processing half of the available image data the localization is still sufficiently precise. In Table 1 we summarized all results

Table 1. Evaluation for test runs in figure 5

sequence	length	avg. error	max. error
run (a)	23 sec	0.152 m	0.359 m
run (b)	23 sec	0.390 m	0.847 m
run (c)	35 sec	0.177 m	0.500 m
run (d)	35 sec	0.205 m	0.502 m

for the benchmarks presented in section 3. In all test runs the particle filter shows similar small deviations from the real path.

5.1 Results under reduced view range

The above results were obtained on a field without obstacles. Our algorithm also does work with partly occluded lines,

which happens in real world applications, where obstacles can cover significant parts of the image. We could prove this in the Robocup German Open 2004, where our team won the competition (winning all its 8 games, scoring 44:3 goals). The robots didn't delocalize during the matches, although the lines were covered by 7 other robots (3 teammates and 4 opponents) and a human referee.

As our current benchmark suite does not yet include sequences with obstacles (such sequences will be included in near future), we simulate an occluded view range using artificial black areas in the images. To this end we use different bitmap masks which reduce the original image information. A mask with four black areas can seen in figure 10. This mask constantly occludes two thirds of the image.

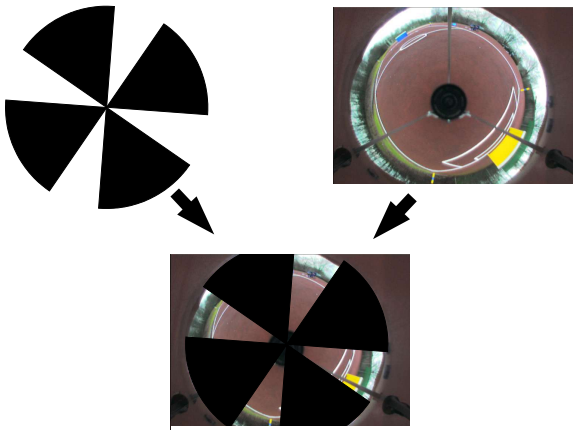


Figure 9. Reduction of the view range

As our algorithm also makes do with few lines, the results are only slightly worse than in the case without occlusions. This can be seen in table 2.

Table 2. Evaluation for test runs in figure 5 with occluded view range using mask shown in figure 9

sequence	length	avg. error	max. error
run (a)	23 sec	0.161 m	0.354 m
run (b)	23 sec	0.393 m	0.877 m
run (c)	35 sec	0.195 m	0.466 m
run (d)	35 sec	0.231 m	0.537 m

5.2 Simulating a directed view range

An occlusion mask as used in the above section, can also be used to simulate the view range of a directed camera. This can be seen in figure 10.

Also the results for this case are very good concerning the fact that only the lines and no other landmarks were used. This is remarkable because due to the big size of the field

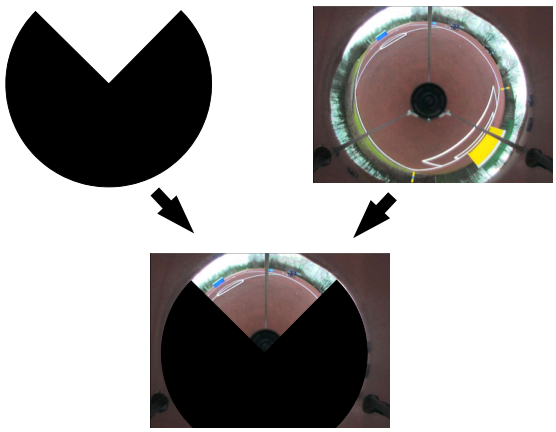


Figure 10. Simulation of directed view range

there are times where no lines at all are detected. But this short periods are compensated by the odometry. In figure 11 one can see such a period, where the robot leaves the left penalty area.

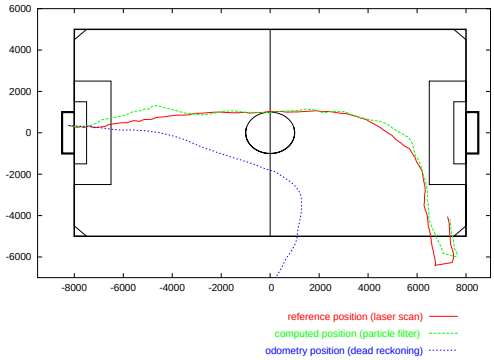


Figure 11. Sequence a, using directed view range

In table 3 we summarized the results for the directed view range case.

Table 3. Evaluation for test runs in figure 5 using directed view range from figure 10

sequence	length	avg. error	max. error
run (a)	23 sec	0.308 m	0.775 m
run (b)	23 sec	0.856 m	0.1764 m
run (c)	35 sec	0.274 m	0.687 m
run (d)	35 sec	0.352 m	0.837 m

6 Conclusions

In this paper we presented a framework for self-localization in the Robocup middle size league. This framework consists of

- a benchmark suite for testing self-localization algorithms and
- a newly developed algorithm for self-localization.

Using these freely available and reproducible benchmarks we presented the results for our algorithm. The obtained results show that our algorithm is very well suited for self-localization under natural light conditions. This is achieved without relying on color coded features like goals or poles and does work a 16x10 meters large outdoor field. Beside the presented benchmark results the algorithm was deployed in our competition team Brainstormers-Tribots, which won the Robocup German Open 2004.

The reason for the good performance of our algorithm lies in the combination of sequential Monte Carlo methods (particle filter) and the robust extraction of line features from the image data.

Both mentioned parts of the presented work are innovative. Benchmarks are extensively used in many fields of machine learning, but to our best knowledge our benchmark is the first considering self-localization of autonomous robots in the Robocup environment. The presented benchmarks are considered as a starting point and will be extended in the future (hopefully for non Robocup specific environments as well).

With respect to our algorithm and its outdoor deployment there are similar but distinct approaches in the literature. In [6] natural light conditions are considered, but only for color classification, no resulting self-localization performance tests were presented. In our work we do not heavily rely on color classification, as the main features are obtained from strong thresholds in color values which appear on line crossings. Color classification is also used, but just for validation of the obtained line crossings and can therefore be more fuzzy.

The work in [11] presents a robust self-localization algorithm for the middle size league. It relies on the detection of more complex line features in an homogeneously colored field. Our algorithm makes do with less structured features which we consider as one of the reasons of its robustness. Also the use of a particle filter distinguishes our work from [11]. It would be interesting to test the algorithm from [11] under the conditions of our benchmark suite.

Maybe the algorithm presented in [9] is most similar to ours. The extraction of line features is different, as we for example do not rely on separate detection of a horizon line. Also the deployment of the particle filter and the computation of sensor probabilities differ partially from our approach. The results presented in [9] were obtained on a quite small field (due to the Sony legged league limitations) and under artificial light conditions, therefore it would be interesting to see it's performance in our framework.

There are other approaches in the literature, see for example [10]. The methods used therein can also be distinguished from the work presented by us, but no qualitative comparisons were possible until now. We hope that our benchmark suite will be helpful in making such qualitative comparisons in the future. Also by using our challenging extensions of the current Robocup environment (natural light, large field), we hope to accelerate the progress in the Robocup environment.

The work presented in this paper would not be possible without the foundations created by the Brainstormers-Tribots team [1] in 2003. We would also like to thank the CoPS team from Stuttgart [3] for their support with the laser range scanner. Finally we thank our local sport facilities for providing the environment for our experiments.

REFERENCES

[1] M. Arbatzat, S. Freitag, M. Fricke, R. Hafner, C. Heermann, K. Hegelig, A. Krause, J. Krüger, M. Lauer, M. Lewandowski, A. Merke, H. Müller, M. Riedmiller, J. Schanko, M. Schulte-Hobein, M. Theile, S. Welker, and D. Withopf, 'Creating a robot soccer team from scratch: the brainstormers-tribots', in *RoboCup-2003 - Proceedings of the International Symposium*, (2003).

[2] M. Asada, T. Balch, A. Bonarini, A. Bredenfeld, S. Gutmann, G. Kraetzschmar, P. Lima, E. Menegatti, T. Nakamura, E. Pagello, F. Ribeiro, T. Schmitt, W. Shen, H. Sprong, S. Suzuki, and Y. Takahashi. Middle size robot league rules and regulations for 2004, http://www.tcsi.de/ROBOCUP/_DOCUMENTS/MSL/msl-rules-2004.pdf.

[3] T. Buchheim, G. Kindermann, R. Lafrenz, H. Rajaie, M. Schanz, F. Schreiber, and P. Levi, 'Team description - cops stuttgart', in *RoboCup-2003 - Proceedings of the International Symposium*, (2003).

[4] D. Crisan and A. Doucet, 'A survey of convergence results on particle filtering methods for practitioners', in *IEEE Transactions on Signal Processing*, (2002).

[5] Arnaud Doucet. On sequential monte carlo sampling methods for bayesian filtering, 1998.

[6] G. Mayer, G. K. Kraetzschmar, and H. Utz, 'Playing robot soccer under natural light: A case study', in *7th International Workshop on RoboCup 2003*, Lecture Notes in Artificial Intelligence. Springer, (2004).

[7] A. Merke and M. Riedmiller, 'Karlsruhe brainstormers - a reinforcement learning way to robotic soccer ii', in *RoboCup-2001: Robot Soccer World Cup V, LNCS*, eds., A. Birk, S. Coradeschi, and S. Tadokoro, 322–327, Springer, (2001).

[8] A. Merke, S. Welker, and M. Riedmiller. Benchmark suite for self-localization, http://lrb.cs.uni-dortmund.de/~merke/robocup/sloc.

[9] Th. Röfer and M. Jüngel, 'Fast and robust edge-based localization in the sony four-legged robot league.', in *7th International Workshop on RoboCup 2003*, Lecture Notes in Artificial Intelligence. Springer, (2004).

[10] E. Schulenburg, T. Weigel, and A. Kleiner, 'Self-localization in dynamic environments base on laser and vision data', in *International Conference on Intelligent Robots and Systems (IROS)*, volume 18, pp. 998–1004, (2003).

[11] F. v. Hundelshausen and R. Rojas, 'Tracking regions', in *7th International Workshop on RoboCup 2003*, Lecture Notes in Artificial Intelligence. Springer, (2004).

.

Opportunistic Planning and Plan Execution

Carmel Domshlak [1] and **James H. Lawton** [2]

Abstract. *Multi-agent opportunism* refers to the ability of agents operating in a multi-agent system (MAS) to recognize and respond to potential opportunities for mutual assistance in achieving individual goals. Two major potential obstacles in operationalizing multi-agent opportunistic assistance in real-world systems are (i) low amounts of knowledge shared between the agents, and (ii) limited ability of the agents to re-plan dynamically. We have previously shown that even under these limiting conditions, systems of agents can benefit from multi-agent opportunism. In this work we discuss how multi-agent systems can exploit shared knowledge for opportunistic predictive encoding using an approach based on an abstract plan representation called Partial Order Plan Graphs (POPGs). Further, we present several approaches for increasing system-level performance by improving the efficiency of the plans containing predictively encoded opportunities, as well as the results of an empirical analysis of their impact on the system performance.

1 Introduction

Single-agent opportunism is the ability of an agent to alter a preplanned course of action to pursue a different goal, based upon a change in the environment or in the agent's internal state – an *opportunity* [8, 11]. Extending this notion, multi-agent opportunism refers to the ability of agents operating in a multi-agent system (MAS) to assist one another by recognizing and responding to potential opportunities for each other's goals [4, 12]. Naturally, multi-agent opportunistic behavior is feasible only when the agents have sufficient knowledge about one another. Unfortunately, in real-world (and especially heterogeneous) MASs, the agents may not possess a significant degree of shared knowledge (such as, *e.g.*, shared plans [7] or even shared goals [2]). Therefore, the two fundamental practical questions regarding multi-agent opportunism are (i) *can multi-agent opportunism be helpful at all in situations where the amount and type of the shared knowledge are very limited?*, and, if so, (ii) *what is the best way to exploit this information both offline (in planning) and online (during the execution)*?

We consider multi-agent opportunism in systems where the agents are required to perform non-trivial planning tasks. The planning and execution scheme for the agents used in this study was developed especially to support opportunistic behavior of agents in various settings of shared knowledge. To preserve generality, this scheme does not assume that the agents are using any particular, or even the same, planning methodology. This is achieved by using a generic form of plan representation, with which we can represent artifacts of most (if not all) techniques used in the area of classical (STRIPS-based) AI domain-independent planning [18]. Further, the selected planning and execution scheme readily supports a computationally efficient form of opportunism based on *predictive encoding* [16], in which potential opportunities are pre-computed and associated with existing plan elements.

In our work, we examine both whether and how different forms of shared knowledge can be exploited for multi-agent opportunistic behavior, and the actual impact that such exploitation may have on performance improvements in an MAS due to multi-agent opportunism. In particular, we have focused on variants of two basic types of knowledge that we believe could be shared even in most heterogeneous and complex MASs: knowledge of agents' principal capabilities (*i.e.*, what goals can possibly be assigned to each of the agents), and knowledge of the goals that have been actually assigned to the agents. The results of our focused evaluation show that adopting opportunistic behavior for planning agents does not come for free, and that its efficiency depends significantly on the way the shared knowledge is exploited by the agents.

On the positive side, we show that multi-agent opportunism can improve the overall performance of an MAS, even in extreme situations where the amount and type of the shared knowledge are very limited, and when the agents have little or no ability to re-plan. However, the basic mechanism used to achieve multi-agent opportunism often produced inefficient plans, occasionally resulting in reduced system performance. Therefore, we introduce a set of extensions to our original approach to multi-agent opportunistic planning and execution aiming to improve the efficiency of the plans and the system performance. We formally describe these extensions, and present the results of a comparative empirical analysis.

2 Computational Model of Planning and Execution

Our abstract MAS model is similar to those used in [15] and [17], but has been extended to support opportunistic behavior. We model an MAS as a finite set of benevolent agents $\{\mathbb{A}_1, \cdots, \mathbb{A}_n\}$, where each agent $\mathbb{A}_i$ is associated with a set of capabilities $C_i = \{c_{i_1}, \cdots, c_{i_l}\}$, and a set of resources $R_i = \{r_{1_i}, \cdots, r_{m_i}\}$. To avoid confusion due to overloading of these two notions in the literature, in our model *the capabilities set C_i corresponds to the goals that in general can be assigned to $\mathbb{A}_i$*. For example, in a team of three planetary rovers $\mathbb{A}_1$, $\mathbb{A}_2$ and $\mathbb{A}_3$, where both $\mathbb{A}_1$ and $\mathbb{A}_2$ are equipped with cameras, while $\mathbb{A}_3$ is not, the goal "have picture of location `L1`" can possibly be in C_1 and C_2, but not in C_3. In contrast, the resources stand for the physical means consumed by the agent's actions: For $1 \leq j \leq m$, we have $r_{j_i} \in \mathit{Dom}(\mathbf{r}_j)$, where $\mathbf{r}_j$ is a certain type of resource (e.g., time, energy, etc.), and $\mathit{Dom}(\mathbf{r}_j)$ is the corresponding domain of $\mathbf{r}_j$.

In addition to the acting agents $\{\mathbb{A}_1, \cdots, \mathbb{A}_n\}$, the system contains a task broker $\mathbb{B}$ [10]. The primary job of $\mathbb{B}$ is simply to dispatch the goals of the system to the acting agents. Note that we use the task broker only to simplify the description of the infor-

[1] Cornell University, Ithaca, NY, USA. Email: dcarmel@cs.cornell.edu
[2] US Air Force Research Laboratory, Inforation Directorate, Rome, NY, USA. Email: lawton@ai.rl.af.mil

mation flow in the system: The decision process behind $\mathbb{B}$, as well as its actual implementation, are tangential to our work and thus are not within its scope. The only thing we assume about $\mathbb{B}$ is that it will assign goal g to $\mathbb{A}_i$ only if $g \in C_i$. Given a set of goals $G_i = \{g_{i_1}, \cdots, g_{i_k}\} \subseteq C_i$, agent $\mathbb{A}_i$ plans for this set of goals, and executes the generated plan $\mathcal{P}_i$. However, during the actual execution of $\mathcal{P}_i$, several aspects of the world could change, impacting the relative attractiveness of $\mathcal{P}_i$. For instance, any of the following may occur:

(a) $\mathbb{A}_i$ is assigned an additional goal $g_{i_{k+1}}$ by $\mathbb{B}$.
(b) Some other agent $\mathbb{A}_j$ in the group fails to accomplish one of its assigned goals $g \in G_j$.
(c) The value of some $g \in G_i$ has been changed (positively or negatively).
(d) Some of the goals in G_i becomes unreachable with respect to $\mathcal{P}_i$.
(e) Resource consumption by the part of $\mathcal{P}$ executed so far has been significantly different (positively or negatively) from what it was expected during planning.

In such cases, we would like $\mathbb{A}_i$ to revisit its current course of action, possibly updating its set of active goals, and *suspending* goals it determines are no longer feasible. Normally, these suspended goals are returned to the broker for redistribution to other agents in the multi-agent system. In our model, though, $\mathbb{A}_i$ may attempt to satisfy these goals opportunistically by fitting them into its current plan, or into the current plan of another agent in the multi-agent system, with as little re-planning as possible (if any).

Below we describe our scheme for planning and execution (originally introduced in [4]), as well as its conceptual extension to support multi-agent opportunism as in [13].

2.1 Planning

The qualitative part of the problem is assumed to be described using the standard propositional STRIPS formalism in which both positive and negative preconditions are allowed (which is exactly the formalism used for the first level of the annual International Planning Competition [5]). Each agent is associated with a description of its initial state (represented as a conjunct of all the propositions valid in this state), a set of goal propositions to be achieved, and a set of all types of actions this agent is able/allowed to perform. In turn, the quantitative part of the problem is described by the resource consumptions of agents' actions and the relative desirability associated with each goal. Since in most applications resource consumption is not necessarily deterministic, it is modeled via *consumption distributions* associated with each action. Likewise, the desirability of each goal g is modeled via a *value function* V_g, allowing conditioning goal desirability on deadlines reached, costs involved, etc.

Following the practical methodology developed in [3], in the planning stage for agent $\mathbb{A}_i$ we first solve the qualitative, propositional planning problem for all the goals in G_i while ignoring the issues of resource consumption and variance in the importance of the goals. The main difference between our framework and that in [3] is that the plan representation required in [3] commits planning to a certain methodology, while our approach is planner-independent. The latter is achieved by representing plans using a novel structure, called a *partial order plan graph* (or *POPG*, for short).

Structurally, POPGs have some properties of both plan graphs [1] and partial order plans [14]: As with plan graphs, POPGs consist of two types of nodes (proposition and action nodes) interrelated via causal links. Unlike plan graphs, however, POPGs are not leveled graphs, and (like partial order plans) the alternative total-order schedules of the actions in a POPG are captured by the ordering constraints between the actions. An example POPG capturing an episode of a plan for the `Rovers` domain [5] is shown in Figure 1. The borderless and rectangular nodes represent proposition and action nodes, and the solid and dashed edges represent the causal links and ordering constraints, respectively. The propositions $\texttt{at}(\texttt{p})$, $\texttt{hs}(\texttt{p})$ and $\texttt{hp}(\texttt{p})$ respectively stand for "located at", "have rock sample from" and "have picture from" location `p`. The causal links from $\texttt{at}(\texttt{L1})$ to $\texttt{SampleRock}(\texttt{L1})$, and from $\texttt{SampleRock}(\texttt{L1})$ to $\texttt{hs}(\texttt{L1})$ capture that $\texttt{at}(\texttt{L1})$ and $\texttt{hs}(\texttt{L1})$ are the preconditions and effects of the action $\texttt{SampleRock}(\texttt{L1})$, respectively. The ordering constraint between $\texttt{SampleRock}(\texttt{L1})$ and $\texttt{Navigate}(\texttt{L1},\texttt{L2})$ captures that the former action cannot be performed after the latter. The (relevant part of the) initial state of the agent is $\texttt{at}(\texttt{L1}) \wedge \neg\texttt{hs}(\texttt{L1}) \wedge \neg\texttt{hp}(\texttt{L1}) \wedge \neg\texttt{hs}(\texttt{L2})$, while the goals are $\texttt{hs}(\texttt{L1})$, $\texttt{hp}(\texttt{L1})$, and $\texttt{hs}(\texttt{L2})$.

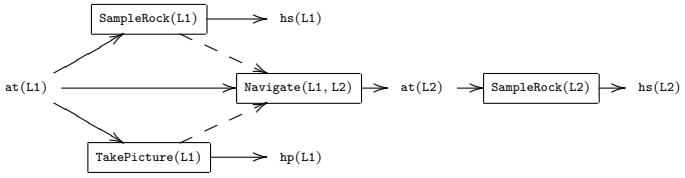


Figure 1. A fragment of a partial order plan graph (POPG) for the `Rovers` example.

For further technical details on POPGs and the precise relation of POPGs to plan graphs and partial order plans, we refer the reader to [4]. The only thing we would like to highlight is that POPGs can be easily derived from any form of plan representation used in the planning community, making our framework essentially planner-independent. Finally, after constructing a (POPG-represented) "skeleton" plan for the qualitative part of the problem, the quantitative information is added into this structure: The actions are annotated with their resource consumption distributions and the goal nodes are annotated with their value functions. The resulting structure is ready to be used in the execution stage.

2.2 Execution

Given an initial state, a set of resources, and a POPG structure of the plan annotated with the information about resource consumption of the actions in the plan and goal values, the agent starts executing its plan. At each intermediate state of the execution, s, the agent must make a decision about the next action to perform. As the agent is provided with a (partial order) POPG, there may be more than one action applicable in the state s. For instance, in the initial state of the running example, both actions $\texttt{SampleRock}(\texttt{L1})$ and $\texttt{TakePicture}(\texttt{L1})$ are applicable. In addition, observe that the action $\texttt{Navigate}(\texttt{L1},\texttt{L2})$ can be performed in the initial state as well. Clearly, if resources are not an effective limitation, performing this action will be irrational, as the agent will loose its ability to achieve the goals $\texttt{hs}(\texttt{L1})$ and $\texttt{hp}(\texttt{L1})$. However, if the resources are limited and the goal $\texttt{hs}(\texttt{L2})$ is very important, it might be the case that the right thing to do is to forget about $\texttt{hs}(\texttt{L1})$ and $\texttt{hp}(\texttt{L1})$, and to perform $\texttt{Navigate}(\texttt{L1},\texttt{L2})$, trying to achieve $\texttt{hs}(\texttt{L2})$ with as little risk as possible. In general, for an agent to decide which action among a set of alternative applicable actions should be performed, requires an estimate of how much "ex-

$\mathsf{Refine}(\mathcal{P}, a, s)$

1. Remove a from $\mathcal{P}$, together with all its outgoing edges.
2. Iteratively remove:
   - All the proposition nodes p (together with their outgoing edges), such that $p \notin \sigma(s, a)$, and the node p has no incoming edges, and
   - All the action nodes a', such that for at least one of the preconditions $q \in \mathsf{prec}(a')$ there is no proposition node associated with q and having an outgoing edge to a'.

Figure 2. Procedure for updating POPG $\mathcal{P}$ after performing action a.

pected value" could be gained by performing each of these actions. Computing these values exactly is intractable, as it requires taking into account not only the probability of certain resource consumption by each action to be executed in the future, but also capturing in the model all possible results of potential future failures. However, adopting the way that resource consumption distributions are abstracted in [3] (see below), we can perform an approximated value estimation.

Informally, at every decision point s the agent:

1. Eliminates from its current plan $\mathcal{P}$ all the actions that are not executable with the current resources ρ.
2. Estimates the expected value $U(\mathcal{P}, s, \rho)$ of $\mathcal{P}$ in the current state s with the current resources ρ.
3. Chooses the most cost-effective action $a \in \mathcal{P}$ from the currently applicable actions that actually provides $U(\mathcal{P}, s, \rho)$.
4. Performs a (resulting in a new state s', and some remaining resources $\rho' \leq \rho$), and updates its plan $\mathcal{P}$ to reflect the result of executing a.

More specifically, let

$$\mathsf{actions}(\mathcal{P}, s, \rho) = \{a \in \mathcal{P} \mid \mathsf{prec}(a) \in s \ \wedge \ \rho \geq \min(a)\} \qquad (1)$$

be the set of actions in $\mathcal{P}$ that are executable in state s with ρ amount of resource available. The value $U(\mathcal{P}, s, \rho)$ represents our estimate of how much value could be gained by executing plan $\mathcal{P}$ with ρ amount of resource, starting at the state s. This value is specified by Eq. 3 via (i) the value of the plans that the agent will have after performing one of the actions $a \in \mathsf{actions}(\mathcal{P}, s, \rho)$, and (ii) the value of the goals achieved directly by the action a; these quantities are specified in Eq. 2 by $\alpha(\mathcal{P}, a, s, \rho)$ and $\beta(a, \rho)$, respectively. The part of the plan $\mathcal{P}$ (= subgraph of POPG $\mathcal{P}$) remaining after performing action a in state s is constructed by the procedure $\mathsf{Refine}(\mathcal{P}, a, s)$, which appears in Figure 2. The value of each such "sub-plan" generated by the Refine procedure is evaluated with the initial state $\sigma(s, a)$, which results from executing action a in state s, and with the amount of resource that is expected to remain after executing a.

$$\begin{aligned} \alpha(\mathcal{P}, a, s, \rho) &= U\left(\mathsf{Refine}(\mathcal{P}, a, s), \sigma(s, a), \rho - \mu(a)\right) \\ \beta(a, \rho) &= \sum_{g \in \mathsf{effects}(a)} V_g(\rho - \mu(a)) \end{aligned} \qquad (2)$$

where $\mu(A)$ stands for the expected resource consumption of the action a. Putting things together, we have:

$$\begin{aligned} U(\emptyset, s, \rho) &= 0 \\ U(\mathcal{P}, s, \rho) &= \max_{a \in \mathsf{actions}(\mathcal{P}, s, \rho)} [\alpha(\mathcal{P}, a, s, \rho) + \beta(a, \rho)] \end{aligned} \qquad (3)$$

Finally, notice that in step (3) of the decision process above, the agent is selecting one of the most cost-effective actions among those maximizing the expected value achievement. This is done to distinguish between different courses of action not only on the basis of their expected value, but also with respect to the costs involved in achieving this value. Thus, each sub-plan is implicitly associated with its cost, and the costs of the optimal sub-plans are back-propagated in attachment to the values of these sub-plans calculated by the dynamic programming procedure as in Eq. 3.

To illustrate possible outcomes of this process, consider the POPG in Figure 1. Suppose that the properties of the actions with respect to the resource are abstracted as in the table below, where $\mu(a)$ and $\min(a)$ stand for the expected resource consumption of a and the minimal amount of energy with which execution of a is permitted, respectively [3].

a	$\mu(a)$	$\min(a)$
SampleRock(L1)	3	3
TakePicture(L1)	2	2
Navigate(L1, L2)	10	15
SampleRock(L2)	5	7

Likewise, let the value functions of the goals to be constant: $V_{\mathtt{hs(L1)}} = 2$, $V_{\mathtt{hp(L1)}} = 2$, and $V_{\mathtt{hs(L2)}} = 10$. If we have $\rho = 19$, then we will have $U(\mathcal{P}, s, \rho) = 12$ and the action to be executed is TakePicture(L1), as the estimated best course of action is to perform first TakePicture(L1), then Navigate(L1, L2), and finally SampleRock(L2). However, if $\rho = 18$, then we will have $U(\mathcal{P}, s, \rho) = 10$ and the action to be executed is Navigate(L1, L2), as we estimate that performing any other action will prevent us from achieving (very valuable) hs(L2).

Observe that sudden unreachability of goals, as well as uncertainty in resource consumption by the agent's actions, is captured by the model implicitly. Likewise, suppose that the value of some of the (still reachable) goals that the agent had planned to achieve have changed. The only thing that the agent has to do is update the value functions associated in its plan with the corresponding goals — All of the subsequent decisions will implicitly take into account this change in the agent's objectives. The only part of dynamics that seems to be problematic is assigning a new goal to an agent (i.e. a goal that is not captured by the current plan $\mathcal{P}$). Such a goal can be either completely new to the multi-agent group, or one of the goals that has been suspended by some other agent in the group. Clearly, a complete re-planning for the extended set of goals will solve the problem, and in many domains such a painful solution might be unavoidable. However, below we show that, at least for some practical domains, we can slightly extend the above model of planning and execution in a way that will require no re-planning at all.

3 Multi-Agent Opportunism and Shared Knowledge

The general scheme for planning and execution described above provides an agent with flexibility in selecting its course of action. In this section we show that this flexibility would in turn allow agents to better adapt to dynamic environments by exploiting multi-agent opportunism, even in settings that severely limit potential cooperation between the agents. We first discuss some general issues that should be addressed while considering adopting multi-agent opportunism in practice.[3] We then describe how those issues can be handled in our scheme for multi-agent planning and execution, and provide some experimental results from [13] for the case without re-planning.

[3] For a detailed discussion on issues involved in multi-agent opportunism, see [11, 12]

3.1 Some General Concerns (Our Motivation)

In theory, multi-agent systems can clearly benefit from the ability of agents to act opportunistically, adapting to changing environments, unexpected events, etc. In practice, however, taking advantage of an opportunity is far from trivial. The opportunity must first be recognized, the course of action it facilitates must be determined, and the agent must decide whether or not it is appropriate to pursue this course of action at the current time [6]. (To use an extreme example, it would probably not be appropriate to stop for a drink while being chased by a bear, no matter how thirsty you are.) Multi-agent opportunism (*i.e.*, exploiting opportunities at an inter-agent level) is even more complicated: The agents should be capable of recognizing whether a given event or situation may be an opportunity for a goal of *another* agent in the system, and of responding appropriately to these recognized opportunities. Two key issues below can make the potential practical attractiveness of multi-agent opportunism somewhat questionable.

First, both recognizing opportunities and responding to them should have low computational complexity, otherwise the MAS will be more "socially friendly" than useful. Theoretically, if agent $\mathbb{A}_i$ has information that another agent in the system could benefit from $\mathbb{A}_i$ achieving goal g, then $\mathbb{A}_i$ could try to adjust its plan to achieve $G_i \cup \{g\}$. The problem is that, in some domains, even small adjustments to the plan can be computationally hard [20] and thus infeasible during the execution. Likewise, in the case of physical agents such as robots or sophisticated hardware controllers, the qualitative part of an agent's plan must often be carefully verified off-line (sometimes even by human operators [19]), and thus the agent is not allowed to consider completely new courses of action. This discussion boils down to a very basic question: *Can multi-agent opportunism be effective if only minimal or no replanning at all is allowed during execution?*

The second issue is that, in order for an agent to recognize potential opportunities for other agents, the agent clearly has to know something about what these other agents are doing. However, we believe that especially in heterogeneous MASs, this "something" may turn out to be very limited. For instance, the agents may know only the goals that some agents can no longer achieve (*i.e., suspended goals*), while knowing only little about these other agents *a priori*. If so, *can we hope that multi-agent opportunism will be effective in such cases of extremely limited shared knowledge?*

We address these two questions using our scheme for multi-agent planning and execution as a platform for the analysis. To make the analysis as conclusive as possible, here we focus on somewhat "least permitting" conditions, taking the system to an extreme in which multi-agent opportunism is least likely to contribute. First, we assume that no online replanning is allowed (and/or possible) whatsoever. Second, we consider two (probably the most basic) settings of shared knowledge. In both settings, the agents communicate only information about their suspended goals. In addition, the agents are assumed to have *a priori* only very limited knowledge about the other agents in the group: In the less informative settings, the agents know only the "types" of the other agents in the MAS, *i.e.*, their individual capabilities. In the more informative settings, the agents know about the individual goals that have been assigned to the agents by the broker. In the reminder of this section, we show how multi-agent opportunism can be supported in our scheme of multi-agent planning and execution under these conditions, and briefly present the results of our empirical evaluation from [13]. In the later sections, we present our complementary work on improving the effectiveness of this planning and execution methodology.

3.2 Making Execution Opportunistic

Consider an MAS $\{\mathbb{A}_1, \cdots, \mathbb{A}_n\}$ as in Section 2 executing plans $\mathcal{P}_1, \cdots, \mathcal{P}_n$ for the goal sets $G_1, \cdots, G_n$, respectively. Suppose that, at some point agent $\mathbb{A}_i$ suspends a goal $g \in G_i$, notifying the rest of the agents that it can no longer satisfy g. Since we are assuming the agents in the MAS are benevolent, we would like the other agents $\mathbb{A}_j$ (such that $g \in C_j$) to at least consider whether they can achieve g themselves and whether the corresponding changes in their course of action would be worth it. However, if the agents are prevented from replanning, the plans $\mathcal{P}_1, \cdots, \mathcal{P}_{i-1}, \mathcal{P}_{i+1}, \cdots, \mathcal{P}_n$ cannot be changed, and thus any opportunistic assistance to $\mathbb{A}_i$ (if possible at all) must be based on them as they are.

Notice, though, that even without replanning, it might be the case that g is present, and thus potentially achievable, in one of the POPG-represented plans $\mathcal{P}_j$ (*e.g.*, as a side-effect of $\mathbb{A}_j$'s primary activities). To opportunistically adopt g as a new goal, $\mathbb{A}_j$ needs only to properly increase the value of g, updating the value function V_g in $\mathcal{P}_j$. *While considering actions in the future, the execution module of $\mathbb{A}_j$ will implicitly adjust its intention with respect to this update.*

Let us consider some properties of this extension to our planning and execution scheme to support multi-agent opportunism. First, because we are using predictive encoding, the runtime computational complexity of some agent $\mathbb{A}_j$ providing opportunistic assistance to another agent $\mathbb{A}_i$ is, as desired, kept low. In our model, updating a value function V_g is all that is required to determine a potential opportunity for a suspended goal g. This process is linear in the size of POPG $\mathcal{P}_j$ in the worst case.

Second, the value of g is automatically leveraged against the value of other goals in G_j. As the choice of action in our scheme of execution is based on maximizing expected value, achieving g will not come at the expense of other goals in G_j unless g is justifiably considered to contribute more to the MAS. Finally, even if agent $\mathbb{A}$ is not itself capable of single-agent opportunism, it may still be able to provide opportunistic support for other agents, as long as the goals suspended by these other agents are still reachable in the POPG of $\mathbb{A}$. This is interesting because we had initially considered multi-agent opportunism strictly as an extension of single-agent opportunism.

Returning to the discussion on our approach to opportunistic execution, the alert reader may rightfully say that if g was assigned as a goal to $\mathbb{A}_i$, it is not very likely that g will also appear in the plan of another agent. (In our Mars rovers example, indeed, why would a rover plan to sample rocks at a certain location if it was not assigned to it?) It again appears as if multi-agent opportunism without dynamic replanning is not very promising. However, this is not necessarily the case.

Observe that nothing in our scheme for planning and execution prevents agents from planning for goals that they were *not* assigned to, *i.e.*, goals having *zero* value from the local perspective of these agents. Since the decision mechanism behind the execution takes into account not only the value of the goals to be achieved, but also the risk behind the various courses of action (encountered via cumulative resource consumption), achieving a goal with a zero value will *automatically* be postponed. Similarly, if one of the goals that the agent has planned for becomes irrelevant, instead of removing this goal from the plan, the agent could simply zero its value function. Now, recall that in our model each agent is characterized by a set of capabilities representing all the goals that can possibly be assigned to the agent. In general, instead of planning for the set of goals that

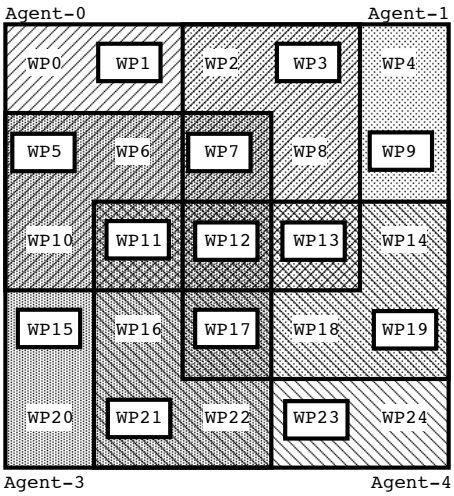


Figure 3. Workspace partitioning for MAS.

have been actually assigned to the agent, one can consider *planning for the whole set of capabilities* and reasoning about the best course of action during the execution, when the value of different capabilities is known better than during the off-line planning. It is not hard to see that planning for capabilities would allow an agent to readily adapt to newly assigned goals, without re-planning, at runtime. Perhaps more importantly, though, is that planning for capabilities could significantly improve the performance of multi-agent opportunism, since more opportunities may be discovered that would otherwise go unnoticed. Of course, nothing prevents the set of capabilities from being orders of magnitude larger than an average set of goals the agent is actually assigned. For such cases, planning for capabilities should be *selective*, restricted only to "most promising" capabilities to plan for.

3.3 Empirical Evaluation

Suppose that agent $\mathbb{A}_i$ considers planning not only for its assigned goals G_i, but also for a limited set of its other, opportunity-wise "most promising", capabilities. Can $\mathbb{A}_i$ estimate whether the purported overhead in planning is worth the potential contribution to the overall achievements of its MAS? What makes certain capabilities more promising than others?

In attempt to address these questions, we have implemented an evaluation testbed for MASs with our planning and execution scheme. The simulation platform used in the experiments is a discrete-event simulator, used to put the activities of different agents on a single time scale. The benchmark problems we have used in the evaluation are based on the standard planning benchmark domain `Rovers` (inspired by the planning problems for NASA's Mars Rovers), used in International Planning Competition (IPC-2002) [5].

In a nutshell, the working area of a set of rovers consists of 25 waypoints, schematically arranged at the cells of a 5×5 grid (see Figure 3). Each rover can navigate between 12 waypoints, and perform various scientific tasks such as soil sampling, rock sampling, and taking pictures of objects of interest if they are visible from the rover's current location. There are total of 65 different goals that can be assigned to the rovers by the broker, namely 13 rock samplings (at the emphasized locations), 13 soil samplings, and 13 objects to be photographed, where each picture can be taken at three different levels of quality.

The evaluation was performed on problem instances involving teams of 4 agents, $\mathbb{A}_1, \ldots, \mathbb{A}_4$, with partially overlapping capabilities: The working area of the team was divided into 4 partially overlapping regions (see Figure 3), and each rover could navigate only within its designated area. The scientific tasks that a rover $\mathbb{A}_i$ can perform constitute its capabilities C_i, and these are schematically restricted to a sub-area in which this rover can navigate. This way, each agent is capable of performing some 35 (of the 65) goals. At the beginning of each planning/execution cycle, each agent is assigned by the broker a set of goals G_i, such as (i) $G_i \subseteq C_i$, (ii) for $1 \leq i \leq n$, we have $|G_i| = k$, and (iii) $\bigcap_i G_i = \emptyset$. The agents start with planning for their individual sets of goals using a domain-independent planning methodology. In our experiments the agents use the `FF` planner [9], but any "off-the-shelf" planner capable of producing plans for the IPC-2002 domains should be applicable.

For the results reported here, an evaluation was performed on 100 randomly generated problem instances. Given a problem instance for which the agents have generated individual plans $\mathcal{P}_1, \ldots, \mathcal{P}_4$, let $EC(\mathcal{P}_i)$ be the expected amount of energy required for $\mathbb{A}_i$ to fulfill its plan $\mathcal{P}_i$ completely. Each agent is allocated with a fraction δ_i of $EC(\mathcal{P}_i)$, where δ_i is randomly chosen from a uniform distribution within $[0.5, 1.5]$. In addition to the k assigned goals, each agent was allowed to choose and plan for another k' capabilities, basing its choice on the knowledge available about the other agents. Recall that the idea is to select goals that might get suspended by other agents at runtime, thus predictively encoding potential opportunities. Since the primary focus of the evaluation has been on potential contributions of exploiting severely limited shared knowledge, we conducted two sets of experiments corresponding to two different levels of such knowledge:

1. Individual Capabilities (CK): The agents have complete knowledge about each other's capabilities. Therefore, each agent $\mathbb{A}_i$ chooses for itself k' goals from $\mathcal{C}_i = \left(\bigcup_j (C_i \cap C_j)\right) \setminus G_i$, (*i.e.*, from the capabilities that $\mathbb{A}_i$ shares with other agents).
2. Individual Goals (GK): The agents have complete knowledge about the goals that have been assigned to every agent by the broker. Thus, each agent $\mathbb{A}_i$ chooses for itself k' goals from $\mathcal{G}_i = \left(\bigcup_j (C_i \cap G_j)\right) \setminus G_i$, (*i.e.*, from the assigned goals that happen to be in the capabilities of $\mathbb{A}_i$).

The experiments for CK and GK were performed under several *choice functions* from $\mathcal{C}_i$ and $\mathcal{G}_i$, respectively. First, consider the case of CK and let $\mathcal{C}_i = \bigcup_{j=1}^{n-1} \mathcal{C}_i^j$ be a disjoint partition of $\mathcal{C}_i$, such that $\mathcal{C}_i^j$ consists of the capabilities of $\mathbb{A}_i$ that are also part of the capabilities of exactly j other agents in the system. The first $n-1$ (in our experiments, three) choice functions $\chi_1, \ldots \chi_{n-1}$ correspond to randomly choosing k' extra goals from $\mathcal{C}_i^1, \ldots, \mathcal{C}_i^{n-1}$, respectively. An additional choice function for the case of CK, denoted as χ_{norm}, picks k' extra goals from $\mathcal{C}_i$ at random, where the random choice is not uniform, but normalized with respect to the above partition of $\mathcal{C}_i$. Specifically, let $\gamma = \sum_{k=1}^{N-1} |\mathcal{C}_i^j|/j$. The probability of choosing $g \in \mathcal{C}_i^j$ is given by $1/\gamma k'$.

Since GK is strictly more informative than CK, we would expect that the better choice functions for GK would be based on the additional information carried with GK. One obvious function like this, $\xi_{\max}$, corresponds to $\mathbb{A}_i$ selecting k' capabilities with the greatest value in $\mathcal{G}_i$. Two additional choice functions, $\xi_{\min}$ and ξ_{med}, correspond to choosing capabilities with the lowest and median values in $\mathcal{G}_i$, respectively.

Figure 4. Initial results (normalized).

The results of this evaluation are shown in Figure 4, where the baseline corresponds to expected performance with no multi-agent opportunism whatsoever. These results show that opportunistic predictive encoding in constrained environments can be effective after all. However, Figure 4 shows that adopting multi-agent opportunism was only moderately effective at the level of knowledge of individual goals (GK), while actually being *harmful* at the level of knowledge of individual capabilities only (CK).

At first view, given the decision-theoretic nature of the execution module, this is a somewhat unexpected result: Since each agent acts to maximize its (and thus global) expected payoff, having more potentially valuable goals in a plan should only increase its flexibility, guaranteeing an improvement in the expected performance. Thus planning for extra goals, using either choice function, should lead to performance at least as good as when planning for only the assigned goals. The pitfall here is that this claim is sound only under an "all else being equal" assumption, *i.e.*, only if we compare two qualitatively identical plans. In the case of plan-based predictive encoding, however, achieving k assigned goals using a plan created for these and some other k' goals can be far more complicated (and thus more risky and resource consuming) than achieving the k goals using a plan created only for them. Seeking improvement, in the next section we address these planning shortfalls by considering various ways to improve the result of the planning process.

4 Making POPGs More Efficient

In the previous section we noted that planning for opportunities produced less than overwhelming results because the plans created for the $k + k'$ goals were inefficient for accomplishing the assigned k goals. This is because such plans may lead an agent to execute unneeded actions on the chance that an extra goal would be suspended by another agent. To make the plans more efficient, we need to find a way to create a plan that can opportunistically satisfy extra goals as they arise, yet efficiently satisfy the assigned goals when no opportunities are present.

In this section we examine three approaches to this problem.

1. *Planning with shortcuts:* Each agent first generates a plan for *all* of its (assigned and extra) $k + k'$ goals, and then augments the structure of that plan by adding "shortcuts" to the assigned goals. Shortcuts are actions (or short sequences of actions) that bypass the segments of the plan devoted strictly to support predictively encoded extra goals.
2. *Predictive plan repair:* Each agent creates a plan for just its k assigned goals, and then repairs that plan by creating and adding subplans that take a form of "side-loops" to and from the core plan, devoted to accomplish the extra k' goals.
3. *Reactive plan repair:* Similar to predictive plan repair, but differs by postponing updating the core plan to the execution phase, namely to the points in time when the agents learn of other agents' suspended goals.

4.1 Planning with Shortcuts

As with our basic approach to multi-agent opportunism described in Section 2, we assume that each agent $\mathbb{A}$ is assigned k goals by the broker $\mathbb{B}$, and that $\mathbb{A}$ selects k' additional goals to plan for on the expectation that they may lead to opportunistic execution. Also like in the basic approach, $\mathbb{A}$ creates a plan $\mathcal{P}$ for all these $k + k'$ goals. Since the external planner cannot differentiate between the assigned and extra goals, it will produce a plan that satisfies all of the goals in a manner that it considers "efficient" – often based on minimizing the number of actions. We would, however, like the agents to be able dynamically skip those actions that do not contribute to achieving any of the currently valuable goals.

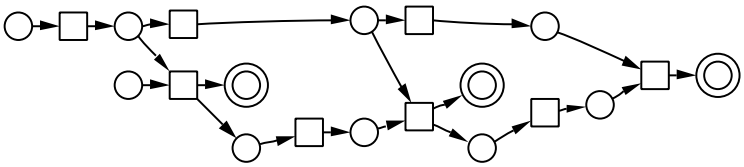
Consider, for example, the small POPGs shown in Figure 5, where the circled and rectangular nodes stand for proposition and action nodes, respectively, and doubly-circled propositions stand for goals. Figure 5(a) shows a base plan for $k = 3$ assigned goals, while Figure 5(b) shows a plan $\mathcal{P}$ for these k and some extra chosen k' goals. The gray nodes in Figure 5 represent the additional actions and conditions needed to accomplish these $k' = 2$ extra goals. Figure 5(c) shows the plan $\mathcal{P}_s$, constructed from $\mathcal{P}$ by automatically adding shortcuts devoted to bypass achieving the extra goals. The validity of adding shortcut nodes to a POPG, creating an *Extended POPG* (or EPOPG), is discussed in [4], and thus we will not go into it further. The real question is, however, how do we determine where to place the shortcut actions? Our approach is to trace through a *total order* of the plan $\mathcal{P}$, finding the start and end nodes of "skippable" sections. For each pair of start and end nodes, we can generate a plan fragment that bypasses the corresponding section of the core plan.

To formally specify the notion of skippable sections of a plan, assume that an agent $\mathbb{A}$ is assigned k goals $G_a = \{g_1, g_2, \ldots, g_k\}$ (with $V(g_i) > 0$ for $1 \leq i \leq k$), and that it also selects additional k' goals $G_e = \{g_{k+1}, g_{k+2}, \ldots, g_{k+k'}\}$ (with $V(g_i) = 0$ for $k + 1 \leq i \leq k + k'$). Further, let $\mathcal{P}$ be the plan generated for all $k + k'$ goals, and $\mathcal{A} = \{a_1, a_2, \ldots, a_n\}$ be a total order of its n actions consistent with the ordering induced by the POPG. For each pair of actions $a_i, a_j \in \mathcal{A}$, we say that $a_i < a_j$ if $i < j$. This sequence of actions is implicitly associated with a sequence of $n + 1$ states $\{s_0, s_1, \ldots, s_n\}$, such that s_0 is the initial state, and for $1 \leq i \leq n$, a_i moves the agent from state s_{i-1} to state s_i. Let $a(g)$ be the action that actually satisfies some goal $g \in G_a \cup G_e$, *i.e.* $a(g) = a_m$ is the action such that $g \in s_m$, but for $0 \leq i < m$, $g \notin s_i$. Consider the assigned goals $\{g_1, g_2, \ldots, g_k\}$ numbered according to their achievement along $\mathcal{A}$. That is, for $1 \leq i < j \leq k$, if $a(g_i) \neq a(g_j)$, we have $a(g_i) < a(g_j)$.

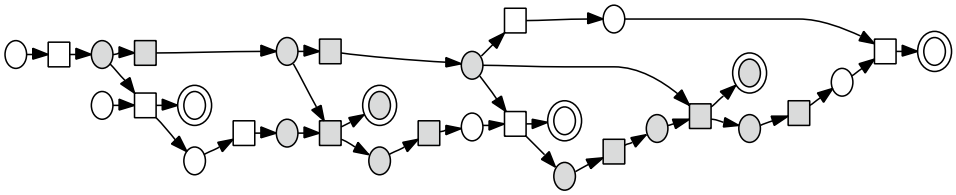
Suppose that for some pair of consequent goals $g_i, g_{i+1} \in G_a$, we have an extra goal $g \in G_e$, such that $a(g_i) < a(g) < a(g_{i+1})$. Let $a(g_i) = a_j$ and $a(g_{i+1}) = a_l$. To allow bypassing the actions needed *only* for achieving g, we can first create a plan fragment $\mathcal{P}'$ with $s'_0 = s_j$ and the goal conjunct $G_{\mathcal{P}'} = s_{l-1}$, and then attach $\mathcal{P}'$ to $\mathcal{P}$, properly grounded at the appropriate proposition nodes in s_j and supporting the proposition nodes of s_{l-1}. This will create an alternate path around the skippable section, as illustrated in Figure 5(c) by the segments with solid black nodes. As formalized by Proposition 1, adding such shortcuts preserves the completeness of the plan with respect to the assigned goals G_a.

Proposition 1 *The structure resulting in replacing the actions $a_{j+1}, \ldots, a_{l-1}$ in $\mathcal{P}$ with the plan fragment $\mathcal{P}'$ is a valid partial order plan that, given unbounded amount of resources, achieves all the assigned goals G_a.*

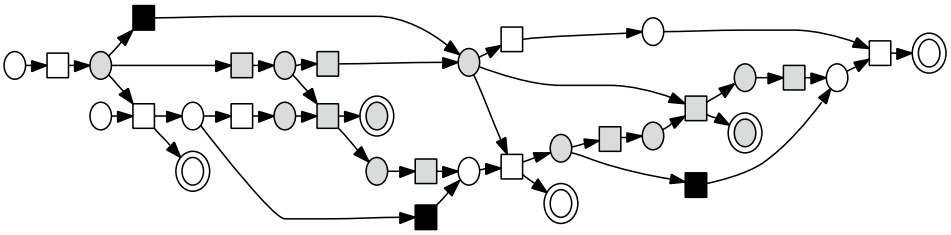
Observe further that the total set of propositions in $G_{\mathcal{P}'}$ can be large, as it corresponds to the entire state s_{l-1}. However, in general there is no need to plan for all the propositions of s_{l-1}, since many



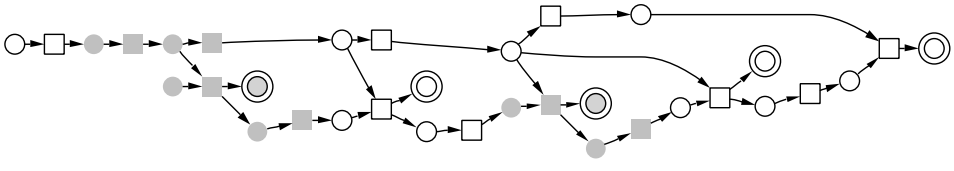
(a)



(b)



(c)



(d)

Figure 5. Example POPGs: (a) Base plan; (b) Extended plan; (c) Extended plan with shortcuts; (d) Base plan with repairs.

of them may have no effect on the applicability of the remaining part $\{a_l, \ldots, a_n\}$ of $\mathcal{P}$. Therefore, without loss of either soundness or completeness, we can assign $G_{\mathcal{P}'}$ to contain only the propositions of s_{l-1} that act as pre-conditions of some actions in $\{a_l, \ldots, a_n\}$. That is:

$$G_{\mathcal{P}'} = \bigcup_{a \geq a_l} s_{l-1} \cap \mathsf{prec}(a) \tag{4}$$

4.2 Predictive Plan Repair

For predictive plan repair, we again assume that each agent $\mathbb{A}$ is assigned k non-zero-valued goals $G_a = \{g_1, g_2, \ldots, g_k\}$, and that it also selects additional k' (zero-valued) goals $G_e = \{g_{k+1}, g_{k+2}, \ldots, g_{k+k'}\}$. Unlike in planning with shortcuts, however, the agent initially generates a plan $\mathcal{P}$ only for its k assigned goals G_a, and then *expands* $\mathcal{P}$ to include actions that accomplish the goals in G_e. For illustration, such a plan $\mathcal{P}$ for achieving $k = 3$ assigned goals as depicted in Figure 5(a). Figure 5(d) shows an expansion $\mathcal{P}_r$ of $\mathcal{P}$, achieved by augmenting $\mathcal{P}$ to include extra actions (shown in borderless gray) to satisfy the additional $k' = 2$ goals.

As with the formalism for planning with shortcuts, let $\mathcal{A} = \{a_1, a_2, \ldots, a_n\}$ be a total order of the actions of the current plan $\mathcal{P}$. To expand $\mathcal{P}$ with respect to an extra goal $g \in G_e$, the agent selects from its action set an action a that provides g and has the best support in $\mathcal{P}$ among all such actions. That is:

$$a = \underset{a' \text{ s.t. } g \in \mathsf{effects}(a')}{\operatorname{argmax}} \left\{ \max_{1 \leq i \leq n} \{|s_i \cap \mathsf{prec}(a')|\} \right\} \tag{5}$$

and with s_i being the corresponding state supporting a, i.e.:

$$s_i = \underset{s_j \in s_0, \ldots, s_n}{\operatorname{argmax}} \{|s_j \cap \mathsf{prec}(a)|\} \tag{6}$$

To preserve the opportunistic nature, the agent should avoid predictive encoding of extra goals having insufficient correlation with the current plan. For instance, in our evaluation discussed in the next section, s_i is required to meet at least half of a's preconditions, otherwise g is not considered to be a potential opportunity.

Given the core plan $\mathcal{P}$, and an action/state pair a and s_i as above, the agent starts by creating a plan fragment $\mathcal{P}'$ for $s'_o = s_i$ and $G_{\mathcal{P}'} = \mathsf{prec}(a)$. Let s' be the state resulting from applying $\mathcal{P}'$ in s_i, and s'' be the state resulting from applying a in s'. Next, the agent creates another plan fragment $\mathcal{P}''$ for $s''_0 = s''$ and $G_{\mathcal{P}''} = s_i$, and concatenates $\mathcal{P}'$, a, and $\mathcal{P}''$. Again, as with planning with shortcuts, the resulting "side-loop" $\mathcal{P}_r$ is attached to $\mathcal{P}$ by linking it to the appropriate proposition nodes in s_i, and executed as usual. Proposition 2 states that, since the execution mechanism always picks the most cost-effective course of action among those achieving maximal expected value, if at runtime the value of one of these extra goals $g \in G_e$ remains zero, the "side-loop" added for achieving g will be pruned by the execution mechanism.

Proposition 2 *If during the execution of $\mathcal{P}_r$ all the extra goals G_e will remain zero valued, the execution of $\mathcal{P}_r$ will be equivalent to executing the core plan $\mathcal{P}$.*

Finally, here as well $G_{\mathcal{P}''} = s_i$ can be replaced without any loss of generality with the specification as in Eq. 4. However, as the extra goals G_e are considered one by one, the whole process of plan repair is incremental with respect to G_e. Therefore, $G_{\mathcal{P}''}$ as in Eq. 4 should be based on the *current* plan (i.e. the core plan with all the previously considered repairs), and not on the core plan for the k assigned goals only. This is necessary to ensure new plan repairs do not prevent previously added repairs from satisfying their extra goals if the corresponding opportunities arise.

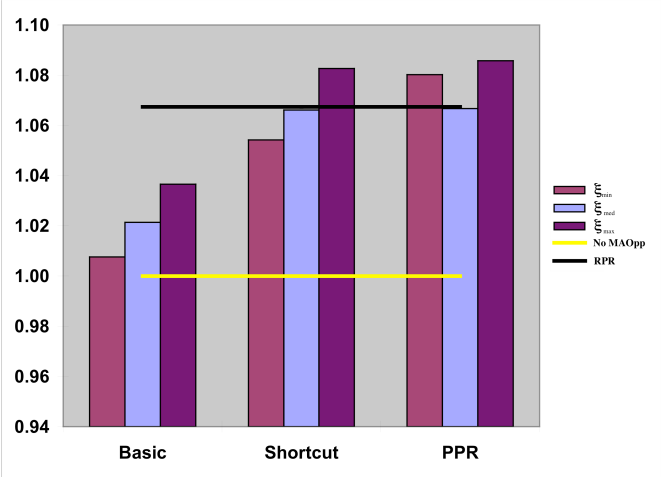


Figure 6. Plan Efficiency Experiments results (normalized), Goal-based selection.

4.3 Reactive Plan Repair

Planning with shortcuts and predictive plan repair are two forms of predictive encoding that can be adopted by agents which are not capable and/or not allowed to adjust their plans at execution time. If, however, some degree of online plan adjustment is possible, the agents can adjust their intentions as they learn about the suspended goals of other agents, without having to "guess" and plan for any extra goals in advance.

In this case, one may consider a reactive variant of predictive plan repair. Following the reactive plan repair approach, here again we have each agent generating a plan $\mathcal{P}$ for its k assigned goals. Unlike in purely offline approaches to predictive encoding, however, the agents do *not* select any additional goals. Rather, when an agent is notified about some other agent's suspended goal, it uses the plan repair mechanism described in Section 4.2 to fit the goal (if possible) in to the remaining portion of its current plan.

A significant advantage of this approach is that the agents only plan for goals that actually get suspended. They do not waste any effort preparing for goals on the chance they might lead to opportunistic execution. Rather, they can focus their resources on considering opportunities for goals that they know cannot otherwise be satisfied. The reason we consider reactive plan repair in our analysis is twofold. First, this approach corresponds to what we expect is the minimal form of online re-planning, preserving the qualitative core of the plan generated off-line. Second, as such, this approach provides us with yet another reference point for evaluating attractiveness of purely offline forms of predictive encoding, the main interest of our work here.

5 Evaluation

To examine the impact of the various plan enhancement approaches discussed in Section 4, we have repeated the experiments discussed in Section 3 for each of the three techniques as above. This evaluation

has been performed on the same 100 problem instances [4] involving teams of 4 agents with partially overlapping capabilities as shown in Figure 3.

The results of this evaluation for our two settings of shared knowledge GK and CK are shown in Figures 6 and 7, respectively. The left-most group of bars labeled "Basic" simply replicates the results depicted in Figure 4 for offline predictive encoding with no efficiency adjustments to the plan. The next two groups of bars depict the total value achieved on these problem instances under planning with shortcuts and predictive plan repair (PPR), respectively. In both Figure 6 and Figure 7, the lower, light-colored horizontal line depicts the total value obtained on these problem instances using the standard execution mechanism without any multi-agent opportunism. As in Figure 4, this represents our baseline, and thus the results for all other planning and execution strategies are normalized against it. Finally, the upper, dark horizontal line in both graphs shows the total value obtained when the agents are allowed to adjust their plans at runtime using the reactive plan repair (RPR) mechanism.

From Figures 6 and 7 it is easy to see that all three plan enhancement techniques lead to an improvement in performance both over the baseline plan execution mechanism with no opportunism and over our original (Basic) approach to offline predictive encoding. Considering Figure 6, none of the three advanced techniques seem to dominate the other two, despite the significant differences between them (*e.g.*, one-shot vs. incremental planning, predictive encoding online vs. offline, etc.) One difference, however, between these techniques is in time complexity of the corresponding planning and execution. In our experiments, the average time each agent spent per problem instance was 7.57 minutes with planning with shortcuts, 22.87 minutes with predictive plan repair, and only 0.43 minutes with reactive plan repair[5]. The dramatically shorter execution time for the reactive plan repair approach is not surprising, since the plans are modified only for goals that are known to be suspended. This leads to smaller plans to execute, with fewer contingency branches to consider, which in turn allows for faster execution. However, recall that reactive plan repair is feasible only if some degree of online replanning is allowed. Otherwise, our results favor the use of planning with shortcuts.

In some ways, the results obtained with plan enhancement techniques while exploiting the less informative CK knowledge (see Figure 7) may be considered even more impressive than the GK results. Recall that our basic approach to multi-agent opportunism actually produced a reduction in system performance as compared to not adopting opportunism at all (see the Basic group of bars in Figure 7). The results for CK with plan enhancements, however, show that we can significantly reduce the overhead involved in opportunistic planning, making adopting multi-agent opportunism attractive even in cases of extremely limited shared knowledge.

As with the GK results, neither planning with shortcuts or predictive plan repair was dominant when exploiting CK knowledge. Both produced moderate performance improvements compared to the baseline of not exploiting opportunities (and average of ≈3% for planning with shortcuts, and ≈4% for predictive plan repair). But again the time complexity of these approaches (and average of 12.77 minutes per problem instance for planning with shortcuts, and 26.65 minutes for predictive plan repair) suggests a preference for planning with shortcuts. Of course, if the agents are able to replan at execution

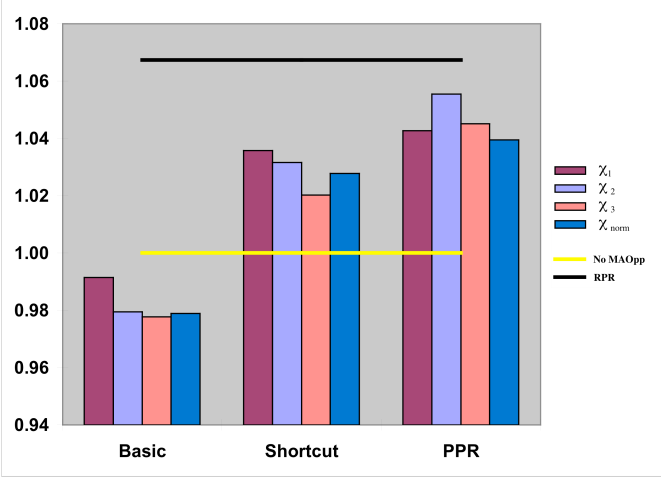


Figure 7. Plan Efficiency Experiments results (normalized), Capability-based selection.

time, reactive plan repair would again be the preferred choice.

It might be argued that, even with our plan enhancement techniques, the improvements obtained by using multi-agent opportunism may not be worth the additional computational burden incurred. Indeed, Figure 6 shows that predictive encoding results in ≈7% average improvement over the baseline, non-opportunistic planning and execution, while in Figure 7 the improvement is only ≈3% on average. However, notice that each problem instance has a total of 16 goals assigned to the system of 4 agents. The value of each goal is selected at random from the uniform distribution between 1 and 100. Thus, the expected value of a goal is 50, and the expected total value of the 16 goals (if all were satisfied) is 800. Hence, a 7% improvement means an average increase of 56, or the equivalent of 1 goal that would have otherwise not been accomplished. Even a 4% improvement would provide an average increase of 32, which would likely indicate the accomplishment of an additional, lesser-valued goal that otherwise would have been unsatisfied. Our statistical analysis verifies that such qualitative improvements do in fact take place, and that they are statistically significant [6].

6 Summary

Through our study we have shown that with limited shared knowledge, and even with no re-planning or plan repair capabilities, real-world systems of heterogeneous agents can assist one another opportunistically in accomplishing their goals. Conversely, we have also shown that adopting opportunistic behavior for planning agents does not come for free, and that its efficiency depends significantly on the way the shared knowledge is exploited by the agents.

We have presented a general scheme for multi-agent opportunistic planning and execution, which is based on selective predictive encoding of opportunities and the principle of planning for capabilities. To overcome some computational limitations, we have presented three techniques for plan enhancement that allow the agents to avoid performing unneeded actions. In particular, we have examined two post-planning methods of enriching the core structure of a plan: one that

[4] At the time of this submission, the CK/PPR experiments were still running. The results reported use only the first 75 of the 100 problem instances.

[5] Since these experiments were run on comparable, but not identical computers, the precise relation between these numbers may slightly vary.

[6] As measured with a paired t-Test. For the worst case (χ_3, Shortcut), $p < 0.15$, thus a significant difference cannot be claimed with reasonable confidence. However, for the others in CK experiments $p < 0.06$, and in the GK experiments $p < 0.0006$ in the worst case.

adds "shortcuts" bypassing the segments of the plan devoted strictly to support predictively encoded extra goals, and one that predictively repairs the core plan to include subplans achieving the extra goals. We have also examined an online approach that assumes the agents possess limited runtime plan repair capabilities. Using this approach, the agent attempts to enhance its core plan only at the time it learns of a goal suspended by another agent.

Finally, we have presented the results of an empirical analysis of our plan enhancement approaches. These results demonstrate that when we take simple measures to augment our core plans, multi-agent opportunism is indeed feasible in that it produces results as least as good as, and often better than, not using multi-agent opportunism. Further, this improvement can be obtained even when the agents have only very limited knowledge of each other's capabilities, and even when the agents have no ability to re-plan at runtime.

ACKNOWLEDGEMENTS

James Lawton would like to thank Elise Turner and Roy Turner of the University of Maine for supporting his work in this project. The work of Carmel Domshlak was supported by the Intelligent Information Systems Institute, Cornell University (AFOSR grant F49620-01-1-0076). The views and conclusions contained herein are those of the author and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the U.S. Government.

REFERENCES

[1] A. Blum and M. Furst, 'Fast planning through planning graph analysis', *Artificial Intelligence*, **90**, 281–300, (1997).
[2] P. Cohen and H. Levesque, 'Intention is choice with commitment', *Artificial Intelligence*, **42**(3), 213–261, (1990).
[3] R. Dearden, N. Meuleau, S. Ramakrishnan, D. Smith, and R. Washington, 'Contingency planning for planetary rovers', in *3rd Int. NASA Workshop on Planning & Scheduling for Space*, Houston, (2002).
[4] C. Domshlak and J.H. Lawton, 'On planning for multi-agent opportunistic execution', in *IJCAI-03 Workshop on Issues in Designing Physical Agents for Dynamic Real-Time Environments*, Acapulco, Mexico, (2003).
[5] M. Fox and D. Long, 'The third international planning competition: Temporal and metric planning', in *Proc. of AIPS-02*, pp. 333–335, (2002).
[6] A. Francis, *Memory-Based Opportunistic Reasoning*, Ph.d. thesis proposal, College of Computing, Georgia Institute of Technology, 1995. `ftp.cc.gatech.edu/pub/ai/students/centaur/proposal.ps.Z`.
[7] B. Grosz and C. Sidner, 'Plans for discourse', in *Intentions in Communication*, eds., P. Cohen, J. Morgan, and M. Pollack, MIT Press, (1990).
[8] K. Hammond, 'Opportunistic memory', *The Journal of Machine Learning*, **10**(3), (March 1993).
[9] Jörg Hoffmann and Bernhard Nebel, 'The FF planning system: Fast plan generation through heuristic search', *Journal of Artificial Intelligence Research*, **14**, 253–302, (2001).
[10] M. Klusch and K Sycara, 'Brokering and matchmaking for coordination of agent societies: A survey', in *Coordination of Internet Agents*, ed., A.Omicini et al. Springer Verlag, (2001).
[11] J.H. Lawton, 'Opportunism in planning agents', in *The Seventh World Multiconference on Systemics, Cybernetics and Informatics (SCI-03)*, Orlando, Florida, (July 2003).
[12] J.H. Lawton and C. Domshlak, 'Towards multi-agent opportunism with planning agents', in *AAMAS-03 Workshop on Autonomy, Delegation, and Control*, Melbourne, (2003).
[13] J.H. Lawton and C. Domshlak, 'On the role of knowledge in multi-agent opportunism', in *Third International Joint Conference on Autonomous Agents and Multi-Agent Systems*, New York, (2004).
[14] D. McAllester and D. Rosenblitt, 'Systematic nonlinear planning', in *Proc. of AAAI-91*, pp. 634–639, (1991).
[15] V. Ogston and S. Vassiliadis, 'Matchmaking among minimal agents without a facilitator', in *Proc. of Agents-01*, pp. 608–615, Montreal, (2001).
[16] A. Patalano, C. Seifert, and K. Hammond, 'Predictive encodings: Planning for opportunities', in *Proc. of the 15th Conf. of the Cognitive Science Society*, pp. 800–805, (1993).
[17] O. Shehory and S. Kraus, 'Formation of overlapping coalitions for precedence-ordered task execution among autonomous agents', in *Proc. of ICMAS-96*, pp. 330–337, (1996).
[18] D. Smith, ed. *Special Issue on the 3rd International Planning Competition*, volume 20, 2003.
[19] B. Williams and P. Nayak, 'A reactive planner for a model-based executive', in *Proc. of IJCAI-97*, pp. 1178–1185, Nagoya, Japan, (1997).
[20] Q. Yang, D.S. Nau, and J. Hendler, 'Merging separately generated plans with restricted interactions', *Computational Intelligence*, **8**(2), 648–676, (1992).

Planning under uncertainty with multiple consumable resources

Simon Le Gloannec[1] and **Abdel-Illah Mouaddib**[1] and **François Charpillet**[2]

Abstract. Most work on planning under uncertainty in AI assumes rather simple action models, which do not consider multiple resources. This assumption is not reasonable for many applications such as planetary rovers which much cope with uncertainty about the duration of tasks, the energy, and the data storage necessary. In this paper, we outline an approach to control the operation of an autonomous rover which operates under multiple resource constraints. We consider a directed acyclic graph of progressive processing tasks with multiple resources, for which an optimal policy is obtained by solving a corresponding Markov Decision Process (MDP). Computing an optimal policy for an MDP with multiple resources makes the search space large. We cannot calculate this optimal policy at run-time. The approach developed in this paper overcomes this difficulty by combining: decomposition of a large MDP into smaller ones, compression of the state space by exploiting characteristics of the multiple resources constraint, construction of local policies for the decomposed MDPs using state space discretization and resource compression, and recomposition of the local policies to obtain a near optimal global policy. Finally, we present first experimental results showing the feasibility and performances of our approach.

1 Introduction

There has been considerable work in AI on planning under uncertainty. However, this work generally assumes an extremely simple model of action that does not consider continuous time and multiple resources[3]. These assumptions are not reasonable for many application domains such as space mission and planetary rovers which much cope with uncertainty about the duration of tasks, the energy required, the data storage necessary and limited communication capacity. Limited communication capacity combined with multiple resource constraints require that the remote spacecraft or planetary rover operates autonomously. The need of autonomy and robustness in the face of uncertainty will grow as rovers become more capable and as missions explore more distant planets.

Planning systems that have been developed for planetary rovers and similar applications typically use a deterministic model of the environment and action effects. Such a planning system produces a deterministic sequence of actions to achieve a set of tasks under nominal conditions. These current planning systems, which rely on re-planning to handle uncertainty , are myopic and do not model the uncertainty in the planetary applications. As the mission complexity (the set of tasks grows) and communication constraints grow, the weakness of these approaches will become critical.

[1] GREYC-Université de Caen Campus II - BP 5186 F-14032 Caen Cedex
[2] LORIA, BP 239 F-54506 Vandœuvre-lès-Nancy

Decision Theory is a framework for reasoning under uncertainty, rewards, and costs. This framework allows to find a tradeoff between uncertainty on the multiple resources consumption, the value gained when achieving a goal and the cost of consuming resources. This framework combined with the progressive processing that allows a rover to trade off execution resources against the quality of the result similar to resource-bounded reasoning provides a suitable framework. The objective of this paper is to complete previous decision-theoretic approaches on controlling progressive processing to overcome new requirements of the rover applications that we illustrate by an example of plans in the next section. These plans present several new requirements that have not been previously addressed:

• **Task inter-dependency:** Task execution may depend on the outcome of previous actions.

• **Multiple resources:** The controller must optimize its operation with respect to multiple resources.

The contribution of this paper is twofold. First, we generalize previous decision-theoretic control techniques [6, 4] to handle multiple resources under uncertainty by using a Markov Decision Process using multidimensional utility and value functions. Second, we examine the effect of increasing the size of plans (the acyclic graph) on the MDP. We address the problem of the large size of the MDP by using classical techniques of decomposition of the large MDP into smaller ones that are easy to solve and then recompose the local policies to obtain an optimal or near optimal global policy. The remaining of the paper describes these different steps of our approach that consisting of formalizing the problem of planning under multiple resources constraints with a Markov Decision Process with multidimensional value and utility functions, examining the complexity of solving the obtained MDP to construct an optimal policy and then tackling this difficulty : (1) decomposing a large MDP into smaller ones [7], (2) compressing the states of multiple variables that are not significantly different, that is, that have the same transition probabilities to other states, and thus the same expected return, under the optimal policy, (3) constructing local policies of small MDPs under the model of discretization of state space of multiple resources using a minimal partition of the state multiple resources using point 2 and (4) re-composing local policies to obtain a near optimal global policy [5]. The rest of the paper describes in detail all these steps.

2 Progressive task planning

2.1 General definitions

Definition 1 *An **exploration graph** $\mathcal{G}$ is a directed acyclic graph of P progressive processing units $PRU_1, PRU_2, \ldots, PRU_P$.*

in the rest of the paper, we assign p or PRU_p as the p^{th} PRU in the exploration graph (figure 1)

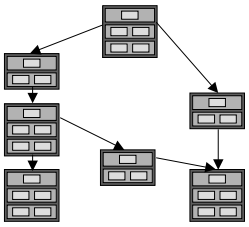


Figure 1. An exploration graph

Definition 2 *A **progressive processing unit** (PRU), PRU_p consists of a set of L_p levels, $\mathcal{L}_p = \{l_{p,1}, l_{p,2}, \ldots, l_{p,L_p}\}$.*

Definition 3 *A **level** $l_{p,l}$ in PRU_p consists of a set of $M_{p,l}$ modules, $\mathcal{M}_{p,l} = \{m_{p,l,1}, m_{p,l,2}, \ldots, m_{p,l,M_{p,l}}\}$.*

A level corresponds to a specific task. The PRU_p in figure 3 is divided into 3 levels ($L_p = 3$), which can be for example $l_{p,1}$: arm camera, $l_{p,2}$: take a picture and $l_{p,3}$: save the picture.

Definition 4 *The **module** $m_{p,l,m}$ of the level $l_{p,l}$ in PRU_p consists of a quality $Q_{p,l,m}$ and a probability distribution $\Pi_{p,l,m}$ over the consumed resources when the module is executed*

A module is a specific way to execute a level task. For example, $m_{p,2,1}$: take a low resolution picture and $m_{p,2,2}$: take a high resolution picture ($M_{p,2} = 2$) in the level $l_{p,2}$ of PRU_p in figure 3, and which means $Q_{p,2,1} < Q_{p,2,2}$. An example of module is given in figure 2.

2.2 Extension to multiple resources

The main contribution of our work is the extension of progressive task planning to multiple resources. The problem is that we do not know exactly how many resources will be consumed by a specific task. We represent this uncertainty with a resource probability distribution. A task can for example consumes between 8 and 12 units of energy.

2.2.1 Resource dependency

In this problem, we must take into account all resources. There consumption may be sometimes dependent. We distinguish two kinds of dependency : internal and external dependencies. We can have an internal dependency when the energy consumption depends on the time consumption for a specific task. For example, the more the robot digs a hole, the more it takes time and energy. We can easily express this with a dependency mapping f such that $energy = f(time)$.

An external dependency happens when a resource consumption for a given task affects a later task. For example, if the rover takes a picture during the night, it has to use the flash, which consumes energy, but he can also wait until the day. It has to make a choice between two resources, that can have consequences for the rest of the plan.

The rover will evolve in a real environment, and resource consumption may also depend on some external factors like temperature or sunshine. For the moment, we do not take these factors into account.

2.2.2 Resource vector

We have chosen to work with resource vector, for example if we are dealing with energy and time, we denote it as $\overrightarrow{R} = \{energy, time\}$.

In general, for r resources, we note $\overrightarrow{R} = \{R_1, R_2, ...R_r\}$, where R_ρ the ρ^{th} resource, $1 \leq \rho \leq r$. We assume that we know in real time the amount of remaining resource and we note it $\overrightarrow{R}_{rem}$.

Definition 5 *To model the evolution of the remaining resource vector, we introduce the following notation :*

$$\overrightarrow{R}_{rem} \leq \overrightarrow{R}'_{rem} \quad \Leftrightarrow \quad \forall \rho \in [1; r],\ R_\rho \leq R'_\rho \tag{1}$$

Definition 6 *It is not physically possible to have a single resource with negative value. In such situation we denote*

$$\overrightarrow{R}_{rem} = \overrightarrow{R}_\emptyset \quad \Leftrightarrow \quad \exists\, \rho \in [1; r],\ R_\rho < 0 \tag{2}$$

The resources we are dealing with, like time and energy, are continuous variables. The better for us would be to take into account this continuous dimension as long as we can. But realistically we need to use a discretization.

Definition 7 *A **discrete probability distribution** of the possible consumption of the resources vector for the module $m_{p,l,m}$ is :*
$\Pi_{p,l,m} = \{(P_{p,l,m,1}, \overrightarrow{R}_{p,l,m,1}), \ldots, (P_{p,l,m,X}, \overrightarrow{R}_{p,l,m,X})\}$ where X is the number of possible consumed resource vector, $\overrightarrow{R}_{p,l,m,x}$ is a consumed resources vector, and $\sum_{x=1}^{X} P_{p,l,m,x} = 1$,

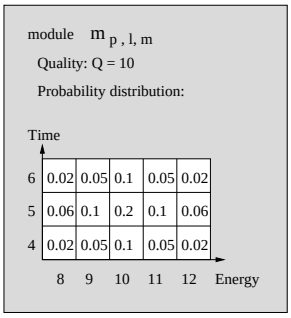


Figure 2. A module descriptor

The module described in figure 2 has a quality of 10 and consumes two resources, energy and time. The execution of this module can for example consume 9 units of energy and 5 units of time with probability 0.1. We denote it as ($P_{p,l,m,x} = 0.1$, $\overrightarrow{R}_{p,l,m,x} = \{9, 5\}$). In this example $X = 3 \times 5 = 15$

2.3 Task selection

Definition 8 *The **progressive processing control problem** is to select at run-time the set of tasks that maximizes the global utility.*

When the rover has executed a module for a given level $l_{p,l}$, he has to choose between two actions :

- select a module of the next level $l_{p,l+1}$ and execute it,
- skip the remaining levels in the PRU_p to move to another $PRU_{p'}$ accessible from PRU_p in the exploration graph.

The optimal decision is the one that maximizes the global utility. The sequence of decisions will determine the set of modules to be executed. The global utility is the cumulative reward of the executed modules (reward is measured by the qualities associated to modules). Since the rover decision process only depends on the quality of the next modules and the current remaining resources, the problem of module selection respect the Markov property. We can control the rover with a Markov Decision Process (MDP).

3 Markov Decision Process Controller

3.1 Definitions

An MDP is a quadruplet $\{\mathcal{S}, \mathcal{A}, \mathcal{T}, R\}$ where $\mathcal{S}$ is a set of states $\mathcal{A}$ is a set of actions, $\mathcal{T}$ is a set of transitions, R is a reward function. In the following, we define what $\{\mathcal{S}, \mathcal{A}, \mathcal{T}, R\}$ means in our context.

Definition 9 *The **accumulated quality** Q_{acc} is the sum of the previously executed module quality $Q_{p,l,m}$ in the current PRU.*

Definition 10 *A **state**, $s = [l_{p,l}, Q_{acc}, \overrightarrow{R}_{rem}]$, consists of the last executed step, the accumulated quality and the remaining resources vector $\overrightarrow{R}_{rem}$.*

Definition 11 *there are two different kinds of terminal states*

- *A **failure state** $[l_{p,l}, Q_{acc}, \overrightarrow{R}_{\emptyset}] = [failure, \overrightarrow{R}_{\emptyset}]$ is reached when one resource is totally consumed(see* definition 6*)*
- *A **final state** is reached when a task of a terminal PRU has been executed (a PRU with no successor in the exploration graph)*

Definition 12 *There are two possible **actions** (see figure 3): execute $\boldsymbol{E}^m_{p,l+1}$ and move $\boldsymbol{M}_{p\rightarrow p'}$. The action $\boldsymbol{M}_{p\rightarrow p'}$ moves the MDP to the $PRU_{p'}$. The action $\boldsymbol{E}^m_{p,l+1}$ execute the m^{th} module $m_{p,l+1,m}$ of level $l_{p,l+1}$ (if $l < L_p$).*

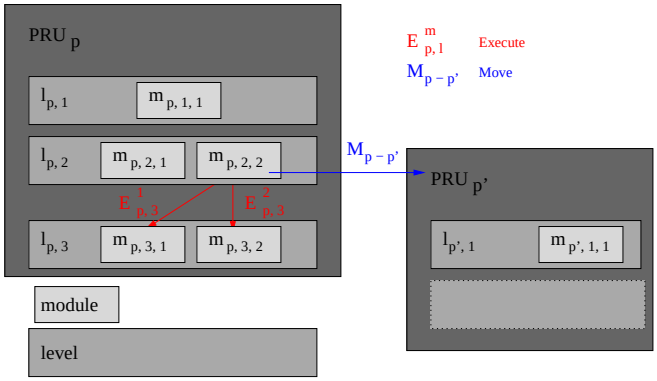


Figure 3. 2 PRU and the two possible actions **E** and **M**

Definition 13 *the **transition model** is a mapping from $\mathcal{S} \times \{\boldsymbol{E}, \boldsymbol{M}\}$ to a discrete probability distribution over $\mathcal{S}$. The move action is deterministic :*

$$\Pr([l_{p',0}, 0, \overrightarrow{R}] | [l_{p,l}, Q_{acc}, \overrightarrow{R}], \boldsymbol{M}_{p\rightarrow p'}) \quad = \quad 1$$

*The execution action is probabilistic, the distribution is given by the module descriptor $\Pi_{p,l,m}$ (*Definition 4 and 7*),*
***if** $\forall r_\rho \in \overrightarrow{R}', r_\rho \geq 0$*

$$\Pr([l_{p,l+1}, Q'_{acc}, \overrightarrow{R}'] | [l_{p,l}, Q_{acc}, \overrightarrow{R}], \boldsymbol{E}^m_{p,l+1}) = P_{l+1,m,l}$$

where $Q'_{acc} = Q_{acc} + Q_{p,l+1,m}$ and $\overrightarrow{R}' = \overrightarrow{R} - \overrightarrow{R}_{l+1,m,l}$,
***else** $\forall \overrightarrow{R}', \exists r_\rho \in \overrightarrow{R}', r_\rho < 0$:*

$$\Pr([failure, \overrightarrow{R}'] | [l_{p,l}, Q_{acc}, \overrightarrow{R}], \boldsymbol{E}^m_{p,l+1}) = \sum_{\overrightarrow{R}' \leq \overrightarrow{R}_{\emptyset}} P_{l+1,m,x}$$

Definition 14 *Rewards are associated with each state based on the quality gain by executing the most recent module $M_{p,l,m}$.*

$$Rew([l_{p,l}, Q_{acc}, \overrightarrow{R}]) = \begin{cases} 0 & if l < L \\ Q_{acc} & if l = L \end{cases} \quad (3)$$

3.2 State's value

We adapt the Bellmann equation to our problem

$$V(s = [l_{p,l}, Q_{acc}, \overrightarrow{R}]) = Rew(s) + \max_a \sum_{s'} \Pr(s'|s,a).V(s') \quad (4)$$

$$= \max \begin{cases} Rew(s) + \max_m (a = \mathbf{E}^m_{p,l+1}) & \sum_{x=1}^{X} \Pr(s'|s, \mathbf{E}^m_{p,l+1}).V(s') \\ Rew(s) + \max_{p'} (a = \mathbf{M}_{p\rightarrow p'}) & 1.V(s'') \end{cases}$$

$$with \begin{cases} s' = [l_{p,l+1}, Q_{acc} + Q_{p,l+1,m}, \overrightarrow{R}_{rem} - \overrightarrow{R}_{p,l+1,m,x}] \\ s'' = [l_{p',1}, 0, \overrightarrow{R}] \end{cases}$$

However s' could be the state $[failure, \overrightarrow{R}]$ that does not appear in the equation because it's value is 0. For terminal states :

$$V([l_{p,L_p}, Q_{acc}, \overrightarrow{R}]) = Rew([l_{p,L_p}, Q_{acc}, \overrightarrow{R}]) \quad (5)$$

$$V([failure, \overrightarrow{R}]) = Rew([failure, \overrightarrow{R}]) = 0 \quad (6)$$

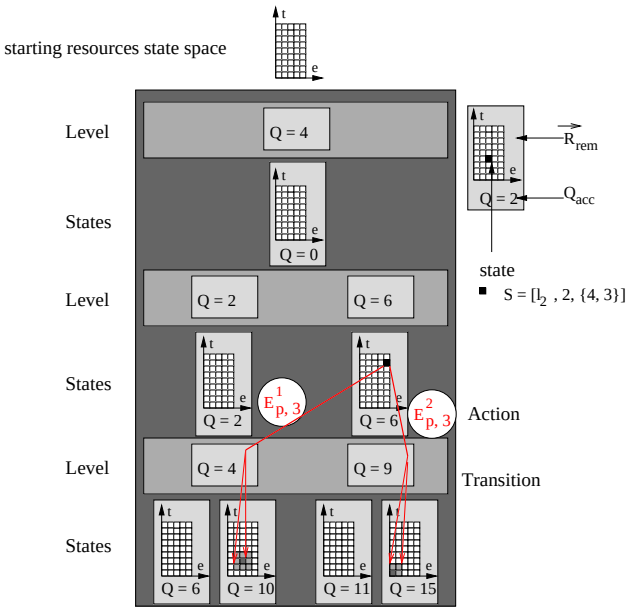


Figure 4. State representation for a single PRU

3.3 Optimal policy computation

We are dealing with a finite-horizon MDP with no loops. Transitions with **E** and **M** move forwards in the state space by always incrementing level or unit number. Although there are a lot of states due to the number of resources, this kind of MDP can easily be solved because the value function is calculated in one sweep (backward chaining, beginning with terminal states); But we have two problems:

- The MDP is large, i.e there is a lot of states and transitions. Although we calculate each state value only once, the computation time increases exponentially in the number of resources with the number of PRU. Therefore we cannot calculate the global optimal policy at run-time.
- We cannot add ore remove any PRU to the exploration graph without recalculating the entire new MDP. This global approach is not adapted to our problem.

The rover has to choose his action at run-time. But it can not calculate at run time the entire MDP values : when the number of PRU increases , the number transitions corresponding to a **M**ove action increases too. We decide to approximate the MDP, to allow dynamic task selection. In the next section, we address the problem of solving this large MDP.

4 Solving the large MDP

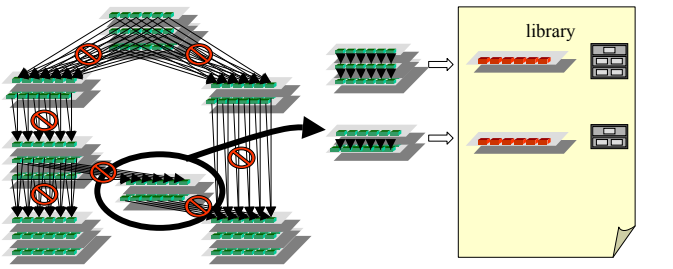
4.1 Principle

To overcome these difficulties we decompose the large MDP into smaller ones, that we later recombine to construct a nearly optimal policy. The goal of the decomposition is to avoid calculation of the MDP states values that are not directly accessible from the current state. We just want to focus on the states that are in the current PRU. The states in the next PRU are not evaluated at run-time, to avoid combinatorial explosion. We evaluate all PRU initial states before run-time. At run-time the agent evaluates the states values in the current PRU, and chooses the best action between **M** and **E**. We explain now the way we decompose the global MDP.

Since the transition corresponding to an action **M** is deterministic, a natural way to decompose the MDP is to calculate a local policy for each PRU(figure 5.b). We keep only the action **E** and we obtain as many local policies as there are different PRU in the graph. For one PRU, we do not store the entire state space, but only the starting level state space (see the top of figure 4), because an action **M** can only reach the starting level of the next PRU. Once all the states have been pre-evaluated, we store this starting state space and its values for each PRU(figure 5.c) in a library, before execution time.

At run-time, we dynamically recompose the MDP. The question is to decide if it is better to remain or to leave the current PRU. We examine the local policy of the current PRU, we compare the expected value for the best execution action $V_{\mathbf{E}}$ to the value for the best move action $V_{\mathbf{M}}$. Since it is not feasible to calculate $V_{\mathbf{M}}$ at run-time, we make an approximation that we denote $V_{\mathbf{M}_{dec}}$ (for decomposition)(figure 6.c).

The remaining of this section explain how we managed to calculated $V_{\mathbf{M}_{dec}}$. We also have problem with the number of resources, with which the starting state space we store for each PRU grow. We find a way to improve the state space storage. Finally, in section (5), we compare $V_{\mathbf{M}}$ to $V_{\mathbf{M}_{dec}}$.

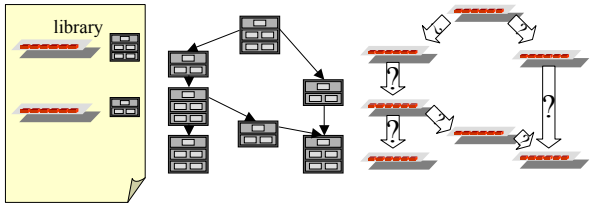


(a)Decomposition (b)Pre computation (c) Compression + Data storage

Figure 5. Decomposition

4.2 Decomposition

To estimate a PRU we calculate the estimated value for all the states in its starting level (figure 7), $\mathcal{S} = \{[l_{p,0}, Q_{acc} = 0, \overrightarrow{R}_{rem}]\}$(figure 5.b). We calculate also the estimated consumed ressources ($\overrightarrow{R}_{est}$)



(a) Stored data (b) Exploration graph (c) Recomposition

Figure 6. Recomposition

for each possible starting state. The algorithm we use to estimate resource consumption is the same that we use for estimating the state value (see section 3.2). To do it, we accumulated the total consumed ressources $\overrightarrow{R}_{cons}$ as we did with Q_{acc} previously. $V_{est} = V$.

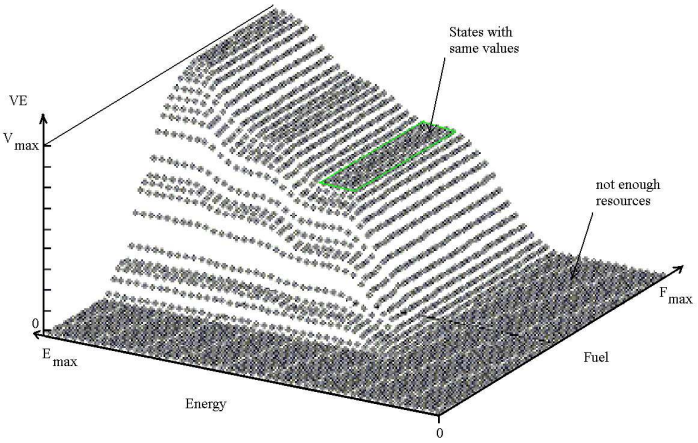


Figure 7. Starting states values for a PRU with 2 ressources

$$\overrightarrow{R}_{est} = \sum_{\overrightarrow{R}' \geq \overrightarrow{R}_{\emptyset}} \Pr(s' = [l_{l+1}, Q'_{acc}, \overrightarrow{R}'] | s, \mathbf{E}^m_{p,l+1}).\overrightarrow{R'}_{est}$$

$$+ \sum_{\overrightarrow{R}' \leq \overrightarrow{R}_{\emptyset}} \Pr(failure | s, \mathbf{E}^m_{p,l+1}).(\overrightarrow{R}_{p,l+1,m,x} + \overrightarrow{R}_{cons})$$

and for the last level l_L: $\overrightarrow{R}_{est} = \overrightarrow{R}_{cons}$.

property :

$$\forall s,\ s',\ V_{est}(s) = V_{est}(s') \Leftrightarrow \overrightarrow{R}_{est} = \overrightarrow{R}'_{est} \qquad (7)$$

4.3 State data storage and space compression

We are dealing with multiple ressources and facing a new problem: how can we **store** the starting state space for each PRU (figure 5.c)? We can however quickly **find** the expected value for a given state, which is necessary for the run-time recomposition. This section explain first how the transformation $T : \mathcal{S} \rightarrow \mathcal{S}_{stored}$ works. In a second time, we explain $T^{-1} : \mathcal{S}_{stored} \rightarrow \mathcal{S}$. A starting state correspond to $s = [l_{p,0}, Q_{acc} = 0, \overrightarrow{R}]$, only $\overrightarrow{R}$ varies. so we project the starting state space on $[\overrightarrow{R}]$. This state space correspond to a r-dimensionnal cartesian product with one dimension for each resource

that T tranforms into a 1-dimensionnal space (corresponding to values). Since we have a monotonic function $\forall s,\ 0 \leq V(s) \leq V_{max} = V([\overrightarrow{R}_{max}] = \{(r_{max})_1, \ldots, (r_{max})_r\})$, we project the state space on the value space (figure 8 is the projection of figure 7). For each value, we make groups of states with the same value in $\mathcal{S}_v$. And in each group we only keep the states with minimal resources in $(\mathcal{S}_v)_{min}$. Notice that there can be several states with minimal resources in a $\mathcal{S}_v$ set. Formally :

$$\mathcal{S}_v = \{s = [\overrightarrow{R}]; V(s) = v\} \quad (8)$$

$$(\mathcal{S}_v)_{min} = \{s \in \mathcal{S}_v, \nexists\, s' \in \mathcal{S}_v, \forall \rho \in [1; r],\ r'_\rho \leq r_\rho\} \quad (9)$$

and we finally store the set of groups :

$$\mathcal{S}_{stored} = \{s \in (\mathcal{S}_v)_{min},\ 0 \leq v \leq V_{max}\} \quad (10)$$

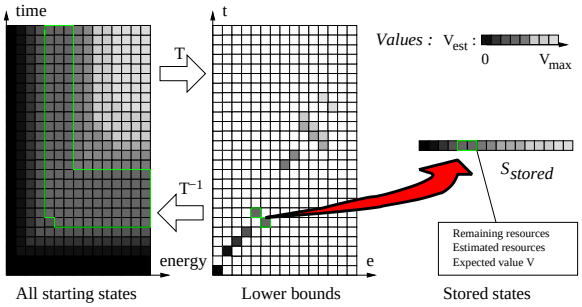


Figure 8. multi resource data compression

The transformation T^{-1} is very simple. We want to know a state value, given $\overrightarrow{R}$. We just have to pass through the $\mathcal{S}_{stored}$ vector and stop when we reach a state with more resources. The last encountered state give us the value. We managed to transform $\mathcal{S} = \{[\{0,0\}], ..., [\{29,29\}]\}$(size = 900) into $\mathcal{S}_{stored}$ with a length of 20.

4.4 Recomposition

The goal of this paragraph is to recompose dynamically at run-time a policy that approximate the optimal one. To do it, we need to calculate the expected value $V_{\mathbf{M}_{dec}}$ for a **M** action from a state, given the exploration graph and the stored data. Therefore we have to recompose the MDP (figure 6). We calculate $V_{\mathbf{M}_{dec}}$ by maximizing the sum of the expected state values in last PRU. The first thing to do is to calculate each PRU depth in the exploration graph, keep for each depth the PRU with the best global expected value in a queue. Then, we sort the PRU queue, we put the best PRU to the top. From a state $s = [l, Q, \overrightarrow{R}]$, we take all the available resources $\overrightarrow{R}$, we add the value given by the first PRU. Then we remove the estimated resources $\overrightarrow{R} = \overrightarrow{R} - \overrightarrow{R}_{est}$. We continue with the next PRU until the queue is empty. In fact, we just make d additions, where d is the size of the queue, or the depth of the graph.

$$V_{\mathbf{M}_{dec}}(s = [p, \overrightarrow{R}]) = V_{est}(s) + (V_{\mathbf{M}_{dec}}([p+1, \overrightarrow{R} - \overrightarrow{R}_{est}]))$$

and for the last PRU :

$$V_{\mathbf{M}_{dec}}(s = [p, \overrightarrow{R}]) = V_{est}(s) \quad (11)$$

We are currently searching for new heuristics to improve this recomposition algorithm, but it already works well, as we will see.

5 Experiments

5.1 Methods comparison

We compare the optimal policy with the policy obtained by decomposition. The advantages of the second method are the short computation time, and the small state space. However, we need to analyze how good the policy obtained by decomposition approaches the optimal policy.

To do so we calculate two error values: the mean error e_{mean}, and the decision error. The mean error indicates if the values obtained by decomposition and recomposition are close to the optimal policy. The decision error counts the times the error exceeds a fixed threshold. The measure our result in term of graph depth, so we experiment only on queues of PRU. We show in the next section that the mean error is small and that the decision error converge toward zero.

5.2 Error measure

We denote $\overrightarrow{R}_{max}$ as the amount of resources required to execute all tasks for the whole plan. We denote $V_{opt}(\overrightarrow{R})$ as the value obtained with the optimal policy for a state s with $\overrightarrow{R}$ remaining resources, and $V_{dec}(\overrightarrow{R})$ the value obtained with the decomposition method.

$$e(\overrightarrow{R}) = \frac{|V_{opt}(\overrightarrow{R}) - V_{dec}(\overrightarrow{R})|}{V_{opt}(\overrightarrow{R})} \quad (12)$$

The mean and the max error for $\mathcal{R} = \{\{0, \ldots, 0\}, \ldots, \overrightarrow{R}_{max}\}$ are

$$e_{mean} = \frac{\sum_{\overrightarrow{R} \in \mathcal{R}} e(\overrightarrow{R})}{Card(\mathcal{R})} \quad \text{and} \quad e_{max} = \max_{\overrightarrow{R} \in \mathcal{R}} e(\overrightarrow{R})$$

For the decision error, we take a threshold of 20%, considering that the rover could make a bad decision if the error exceeds this value.

5.3 Results

We first experiment on a queue with identical $PRUs$.

Graph depth	1	2	3	4	5	10	20	40
Mean error (%)	0	7.4	7.1	6.7	7.1	8.0	8.0	8.4
Max error (%)	0	28	28	28	28	28	28	28
Decision error (%)	0	1.6	4.4	3.3	2.6	1.3	0.6	0.3

We also used queues with different $PRUs$.

Graph depth	1	2	3	4	5	10	20	40
Mean error (%)	0	2.6	2.8	2.7	2.8	2.7	3.9	2.5
Decision error (%)	0	5.0	5.8	4.4	5.5	2.7	1.3	0.6

5.4 Discussion

Errors are made when the PRU queue is short, but the goal of our algorithm is to treat very large PRU queues. For short PRU queues, we can calculate the optimal policy without any approximation.

In each case the mean error stays constant. The more resources and PRU are left, the better is our approximation. The decision error decreases toward zero. The max error is high, and it stays constant. This is the main problem of our algorithm. Errors are locally high, so we intend to search for better approximation methods to reduce them.

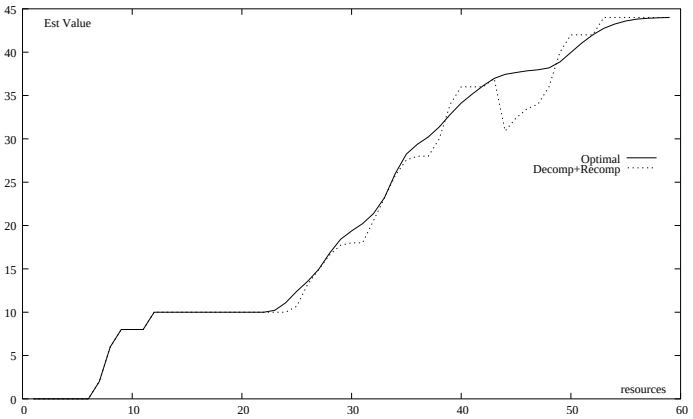


Figure 9. Values for 4 $PRUs$ and 1 resource

6 Conclusion and perspectives

We have presented a solution to the problem of planning under uncertainty with multiple resources. This approach relies on solving a corresponding MDP, generalizes earlier work on MDP controller [6, 4]; it permits for the first time to deal with multiple resources. Moreover, our approach addresses effectively the problem of limited amount of memory required for creating and solving the large obtained MDP and storing the resulting policy. Using decomposition of the large MDP into smaller ones, we managed to reduce the size of MDPs to create and to solve, using a compression data technique, we managed to store the resulting policies and using re-composition approach based on an approximation technique of value functions, we managed the construction of a global policy with a small loss decision quality (less than 1% for large problems). Future works will allow us to develop more precise estimates of the value function. This will include state decomposition, policy decomposition, and action decomposition [1] [2]. We are also developing new techniques to deal with dynamic operation by recomposing local policies of individual PRUs when new PRUs should be added or removed.

REFERENCES

[1] C. Boutilier, R. Brafman, and C. Geib. Prioritized goal decomposition of markov decision processes: Toward a synthesis of classical and decision theoretic planning. In *IJCAI 97*, 1997.

[2] C. Boutilier, T. Dean, and S. Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. In *Journal of AI Research (JAIR)*, pages 11:1–94, 1999.

[3] J. Bresina, R. Dearden, N. Meuleau, S. Ramakrishnan, D. Smith, and R. Washington. Planning under continuous time and resource uncertainty : A challenge for ai. In *UAI*, 2002.

[4] S. Cardon, A. Mouaddib, S. Zilberstein, and R. Washington. Adaptive control of acyclic progressive processing task structures. In *IJCAI*, pages 701–706, 2001.

[5] N. Meuleau, M. Hauskrecht, K. Kim, L. Peshkin, L. Kealbling, T. Dean, and C. Boutilier. Solving very large weakly coupled markov decision processes. In *UAI-98*, 1998.

[6] A.-I. Mouaddib and S. Zilberstein. Optimal scheduling for dynamic progressive processing. In *ECAI-98*, pages 499–503, 1998.

[7] R. Parr. Flexible decomposition algorithms for weakly coupled markov decision process. In *UAI-00*, 2000.

Learning from Recorded Games: A Scoring Policy for Simulated Soccer Agents

Achim Rettinger[1]

Abstract.

This paper outlines the implementation of a new scoring policy for the agents of the Simulated Robot Soccer team from the University of Koblenz, called RoboLog. The applied technique is capable of acting in real time in the dynamic environment of the RoboCup Simulation League and uses data obtained from prerecorded soccer games for supervised neural network learning. The benchmark used for testing this approach is the Optimal Scoring Problem stated as finding the point in the goal where the probability of scoring is the highest when the ball is shot to this point in a given situation. Goalshot situations from numerous logfiles are extracted and employed for the training of two independent multi layered perceptrons. Beside the usage as training patterns the gained data is evaluated statistically and provides interesting general insights into goalshots carried out lately in Simulated Robot Soccer.

The results obtained after extensive testing of the new policy are presented. Furthermore, general issues of learning from observed logfile data and starting points for future work are discussed.

1 INTRODUCTION

Scoring goals is essential for winning games not only in real soccer but also in the RoboCup Simulated Soccer League. The purpose of the RoboCup Simulated Soccer League is to provide a standardized problem domain for Artificial Intelligence research based on a soccer simulation called the RoboCup Soccer Server [2]. Teams of soccer agents programmed by researchers from all over the world can compete with each other by using this simulator.

This paper outlines the implementation of a new scoring policy for the RoboLog team from the University of Koblenz. Searching for a scoring policy is a comparably simple task. Although the properties of the environment provided by the RoboCup Soccer Server are inaccessible, non-deterministic, dynamic and continuous (see [6]), the success of a goalshot can directly be evaluated. In most other problems within the RoboCup domain the outcome of actions cannot be estimated as simple because the actions only result in intermediate and therefore not easily evaluable states. Contrary to that, a goal is definitely a success for the attacking team and final reward can be assigned. This makes the Optimal Scoring Problem a well suited benchmark for various techniques.

Accordingly, we chose the Optimal Scoring Problem for evaluating the innovative use of supervised learning from existing data. The data needed for this kind of inductive learning was obtained by analyzing relevant situations in prerecorded games. The automatically learned heuristic was intended to replace the analytical algorithm applied so far in the RoboLog team which based its decision whether to shoot and where solely on human consideration. In the old approach, manually adjusted thresholds gave the striker the positions, relative to goal and opponents, in which he was supposed to shoot.

[1] University of Koblenz, Koblenz, Germany email: achim@uni-koblenz.de

1.1 Problem statement

The Optimal Scoring Problem is stated as follows (see [4]): *"Find the point in the goal where the probability of scoring is the highest when the ball is shot to this point in a given situation.*"

When observed in more detail another side of this problem appears to be essential for finding an optimal scoring policy: *Given the point to shoot, determine the probability of scoring if the ball is shot to this point in a given situation.* Although, this heuristic is especially interesting for deciding whether to shoot or not, it is not mandatory for finding the point to shoot in our approach.

Both problems can be correlated to each other. If you can solve the first problem you know which point to test for the second problem. But if you can solve the second problem you can also find a good solution to the first problem by comparing numerous different points and taking the one with the highest probability of scoring. Thus, the second problem seems to be an intermediate step to solve the first statement of the Optimal Scoring Problem.

In this paper solutions to both problems will be presented which are not dependent on each other.

1.2 Related work

As the Optimal Scoring Problem is well suited for Machine Learning techniques previous work has been carried out in this area.

A detailed implementation of the scoring policy used by the UvA Trilearn 2001 team is described in [4]. Here, data is generated from repeated experiments where a striker is placed somewhere in front of the goal, the opponent goalie somewhere in the goal. Then the ball is shot to some position in the goal. The outcome of this shot is evaluated statistically. In the end, a function is presented that can calculate the probability of scoring if shot to a given point in the goal. Finally, the best point to shoot at is determined by computing the probability for some discretized scoring points on the goal line and by choosing the global maximum of the results.

In comparions to [4] our approach differs in three major points. First, the training data is not generated by simulating situations but by extracting already existing data from prerecorded soccer games (logfiles). Second, far more influencing factors of a goalshot situation, not just one forward and one goalie are taken into account. Third, two separate modules are developed to solve both in section 1.1 mentioned problems independently from each other. Thus, there is no need for testing discretized shooting points.

In [1] high level actions are based on Neural Networks which are trained to learn success rates. In this case the ”shoot2goal” action will compute the probability of scoring which is later on used for decision support by ranking the success rates of all actions in a priority list. This paper does not mention how to find the best point to shoot at. Again, training data was obtained by repeated generations of situations.

In contrast to that, a tool for the analysis of games played by a certain soccer team is presented in [5]. Special game situations (like goalshots) are identfied in logfiles on this selected team only. The patterns obtained are fed into a decision tree induction algorithm resulting in a set of rules which describe classes of successful scoring attempts and classes of unsuccessful attempts, respectively. Afterwards, those rules are used for a perturbation analysis that can give recommendations for changes in the goalshot heuristic used in this certain team.

Although logfiles were used for obtaining data in [5] and, among others, goalshot situations were extracted the crucial difference to the method outlined here is that the knowledge obtained was used for recommending changes to an already existing behavior (like the scoring policy) of a certain team. In contrast to that, we intended to find a universal and optimal scoring policy from scratch.

By combining data acquisition from logfiles with neural network learning two promising techniques are combined in the approach described in this paper. In addition to that, not only are success rates learned, but the best point in the goal to aim at is determined directly by a module independent from the success rate module. This redundantizes the test of several different scoring points as done in previous work.

2 APPLICATION

The application of our approach can be separated in three phases. First obtain the training data by extraction from logfiles, second analyze this data by supervised neural network learning and last evaluate the performance of the heuristic, in this case the feed forward networks.

2.1 Extraction of data

To obtain training samples, goalshot situations must be identified in logfiles. It is not enough to find successful scoring attempts because positive and negative training samples are required for classifying the success rate. The characteristics of a potential goalshot, identifiable from logfile data, are:

- A forward has kicked the ball.
- The forward is in a reasonable distance to the opponent goal.
- The shot has the potential to reach the opponent goal (reasonable power and direction).

Even if all those conditions apply, further tests need to be done to make sure that it is a valid goalshot and to obtain information about the outcome of this scoring attempt. To determine that, the successive cycles are scanned and checked individually:

- Can the situation be classified as goal, out, goalie catch or offsite? In this case it is a valid shot and the outcome is known.
- But, if the ball was kicked by another player it could also be classified as passing (if kicked by a player from the own team) or dribbling (if kicked by the same striker again) and thus not as a scoring attempt. If kicked by an opponent defender or opponent goalie though, it is interpreted as a valid but unsuccessful goalshot.

total		ratio	
games analyzed	996		
goalshots extracted	9315	shots/game	9.352
successful			
successful goalshots	3745	goals/game	3.760
unsuccessful			
intercepted by goalie	4305	goalie/game	4.322
intercepted by defender	993	defender/game	0.997
out	203	out/game	0.204
other reasons for miss	69	others/game	0.069

Table 1. Statistical evaluation of analyzed goalshots

All those heuristics can be no guarantee for identifying and classifying all scoring attempts correctly, as the internal state of the striker cannot be reconstructed from logfiles precisely. It is impossible to restore the intentions of a player in a specific situation only by observing the visual outcome of its actions. Nevertheless evaluation by hand showed that most of the shots a human observer would classify as scoring attempts were equally categorized by the automatic extractor. Besides that, the classification accuracy is, in that case, not essential for the purpose of neural network learning as long as the action holds valuable information. On this account, successfull shots, never intended to be scoring attempts (but e.g. passes), are important as well.

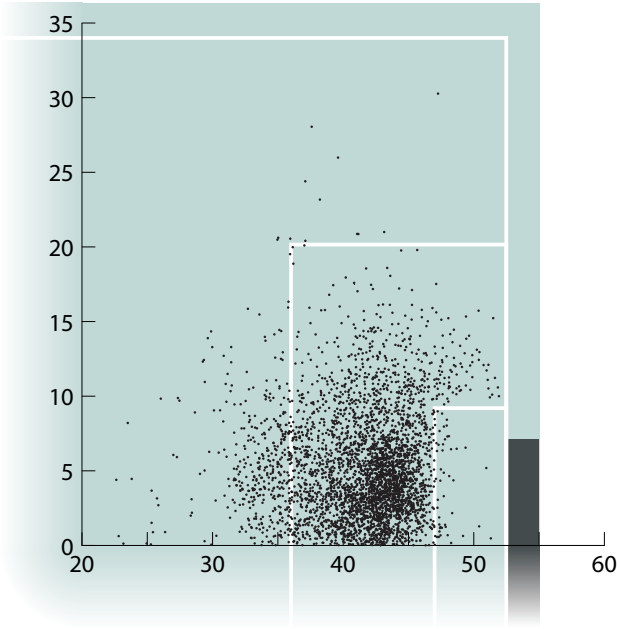


Figure 1. Position of striker while kicking; successful shots

Those heuristics were finally applied to all recorded games from the last RoboCup in Padua (2003) and games from the Simulated Soccer Internet League. Plenty of interesting information can be

gained from statistical analysis of the obtained goalshot situations. An overview is given in Tab 1.

It is interesting to know that 40.2% of the identified scoring attempts were successful and 77.3% of the unsuccessful ones were caught by the goalie. As expected, the goalie is the main factor in intercepting goalshots but it also becomes apparent that the opponent defenders should not be neglected. After all, they are responsible for 17.8% of the inhibited attempts.

The extracted goalshot data can give even more interesting insights. Fig 1 shows the upper right quarter of a soccer field when looked at in top view and landscape format. One half of the opponent goal is drawn as a filled black rectangle in portrait format at the lower right part of the figure. Accordingly, one of the corners is presumed in the top right. The white lines denote parts of the goal line, the side line, the goal area and the penalty area, respectively. The axis refer to the coordinates used in the Soccer Server. The scattered black dots indicate the position of the forward at that point in time when the successful goal kick was carried out.

In contrast to that, Fig 2 marks the position of the forward at the moment of a goal kick that turned out to be unsuccessful. Obviously the dots are spread more widely as expected.

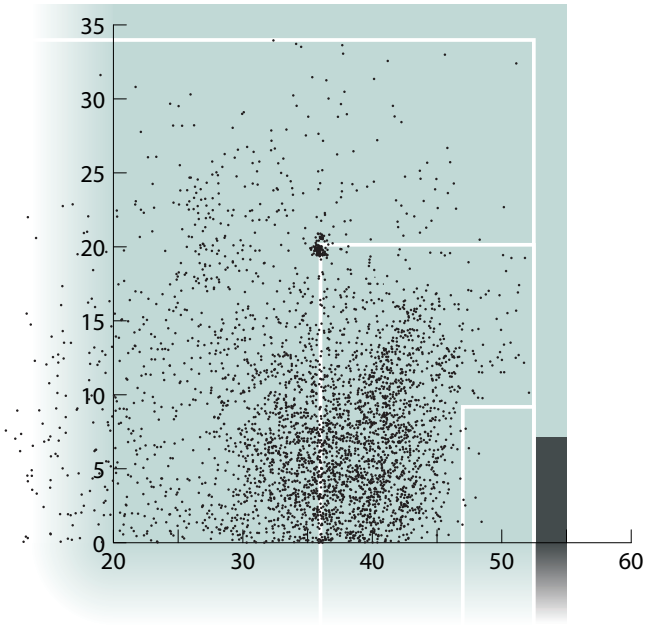


Figure 2. Position of striker while kicking; unsuccessful shots

Note that all goalshot situations are mirrored to this upper right quarter of the soccer field not only for visualization reasons, but mirroring is also essential for avoiding the aliasing problem. While training, the network could get confused if apparently different patterns have the same outcome, if mirrored.

Fig 3 shows where successful goalshots crossed the goal line. Darker areas denote more crossings. This time the goal is drawn in landscape format as a white rectangular boundary; scaling and mirroring is applied accordingly. As it can be easily seen, most of the shots were aimed at the corners of the goal, especially to the goal pole which was closer to the attacker.

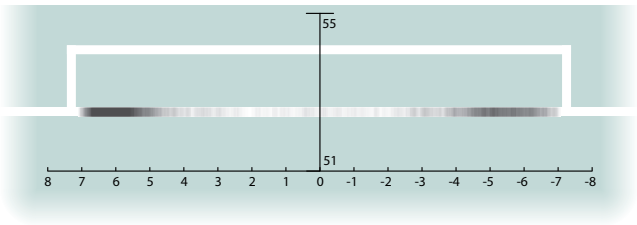


Figure 3. Goal line crossings: dark areas denote more crossings

2.2 Learning

As mentioned before, two basic 3-layered backpropagation neural networks were trained to solve the two tasks. The first network is required for predicting the point to shoot at that maximizes the likelihood of scoring in a specific situation. The second network should be able to classify the success rate of scoring, given a specific situation and the point to shoot. Diverse issues need to be addressed concerning the pragmatics of neural learning.

In the following, some considerations of the decisions that needed to be made shall be presented. One main issue is whether to use objective world data taken from the logfiles directly (accessible environment) instead of trying to simulate the subjective world model of a specific soccer agent (inaccessible environment). In the later case, the objective world data from logfiles like the exact ball position would have to be reduced und altered according to the limited subjective world model of an agent. On the one hand, it seems reasonable to use incomplete and noisy data for training because in a real simulated soccer game an agent would only get incomplete data as well. There is already previous work providing a method for estimating the internal state of RoboLog agents in a specific situation from logfile data only. Thus, it would be easy to use this data as input to the machine learning technique, every agent could be prepared with a specific decision module for its specific procedure of constructing its world knowledge. Unfortunately, it is still impossible to make sure that the reconstructed subjective world model precisely matches the original model from the recorded situation. Thus it is likely that the recorded action is not appropriate to the interpreted world model. Additionally, there is another fundamental shortcoming of using subjective data. As soon as the way a player constructs his world knowledge is changed, all the training needs to be redone. Therefore objective world data was used for learning to take advantage of this more general approach.

Another issue is the question which format of the input data would be the most suited one for this kind of problem. A polar representation of the positions was favored over a Cartesian representation because polar coordinates implicitly express relations between objects which could be more useful for the networks to generalize over the seen examples.

Besides that, the search for the most significant relations in the data remains a challenge, independent from the representation. As most design decisions involved in neural learning are still considered an empirical art (see [3]), the final selection and representation of inputs was found by comparing the results of numerous trained networks using three set cross-validation. A visualization of the final inputs is given in Fig 4. The indices refer to Fig 6. The attacker is drawn in yellow, the goalie in dark grey, the ball is a white circle. Defender 1 to Defender 3 (marked blue) are the three opponents which

can reach the ball first[2]. All variables were scaled to range between 0 and 1 and assigned to one input node each.

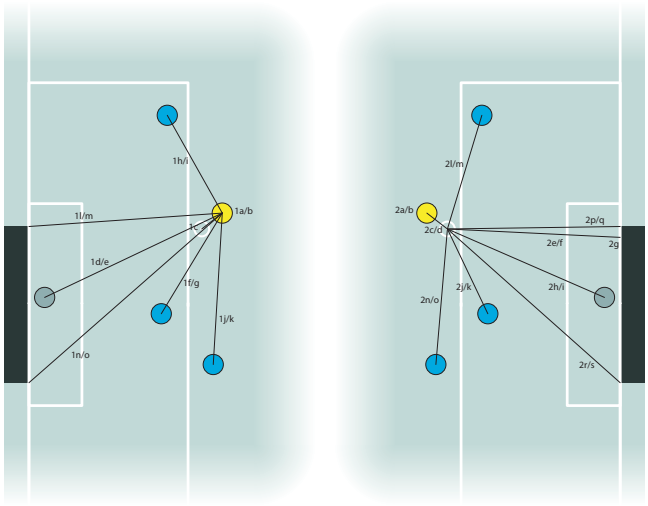


Figure 4. Visualization of input variables; left: best-scoring-point net, right: success-rate net

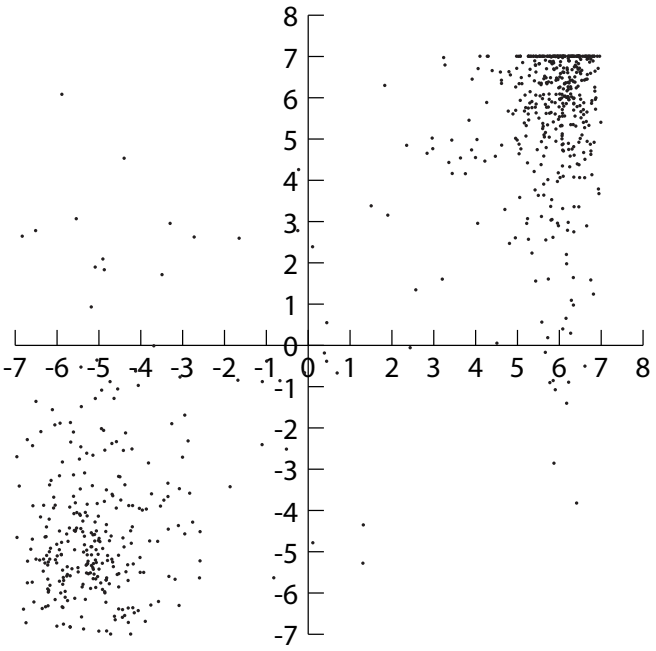


Figure 5. performance of the best-scoring-point net

The target value of the best-scoring-point network is the y-coordinate on the goal interval. The output, goal or no-goal, of the success-rate network is a binary class variable. As the classes are separable, two output nodes - one for goal and the other one for no goal - are used. In the end, both values are combined again to get a success rate between 0 and 1 by using the simple formula: $(((\text{output for no goal}) - (\text{output for goal}))/2) + 0.5$.

The final topology of the best-scoring-point network and the success-rate network derived from cross validation, is 15-53-1 and 19-80-2, respectively. For the first network only positive samples were used partitioned into three data sets for cross validation purposes (sample size: 2060, 936 and 748). The stratified sets of the second network contained equal proportions of positive and negative samples (sample size: 4494, 1693 and 1302). The stopping criteria is based on a calibration interval of 200 (total of training patterns processed per event), where training stops when the last minimum on the testing set (second data set) has occurred 50000 events ago.

After tweaking the various parameters involved in neural network learning, the prediction accuracy of both networks on the evaluation set (third data set) showed promising results.

The best-scoring-point network gave a mean average error of 1.4 units, which is reasonable if taken into account that the goal is more than 14 units wide. So the deviation on average is 10%. Fig 5 visualizes the performance using a scattered graph. The horizontal axis denotes the actual value and the vertical axis the interpreted output. Optimal predictions would result in a line from bottom left to top right. It can be observed that there is no bias towards a certain point in the predictions. The success-rate network achieved a remarkable 85.4% classification accuracy of the goal/no-goal patterns.

Calculating the contribution factor for each input variable is another way to get information about the networks' performance[3]. Fig 6 exemplarily shows the contribution factors for the success-rate network. "X" denotes a Cartesian x-coordinate and "Y" denotes a Cartesian y-coordinate, respectively. A "D" denotes distance in polar coordinates and "A" angle in polar coordinates. This nomenclature also corresponds to Fig 4.

Most of the values correspond to common sense. For instance the most important input is the angle and the distance between the ball and the goalie. Regarding the opponent defenders, the distance is more important than the angle and the closest defender has the biggest influence.

2.3 Evaluation

A feed forward version of both networks, using the learned weights, was finally integrated in the RoboLog framework. Two methods are provided to the soccer agents. If an agent decides that he is in a position where it makes sense to consider a goalshot he makes use of both methods. The first is required to find the best-scoring-point and the second to get the probability of scoring with this shot.

The threshold indicating whether an agent risks a shot or not has to be found by experiment. An observation that thereby needs to be taken into account is the following: The closer the striker gets to the goal the more unlikely it is that he can improve his position for a goalshot. This is because the resistance of the defenders is more concentrated around the goal. This fact cannot be taken into account by a neural network as used here. There is no temporal component that could give feedback about the quality of future states. Consequently, a region model was introduced. From the plot of the coordinates of successful goalshots (see Fig 1) we specified three regions according to the number of goalshots. Region 1 is closest to the goal and most of the goals were scored here. Region 2 is more spacious but still

[2] Those three defenders are determined by using a method from the RoboLog code, based on a Newton iteration.

[3] Contribution factors are a rough measure of the importance of a variable in predicting the network's output, relative to the other input variables in the same network.

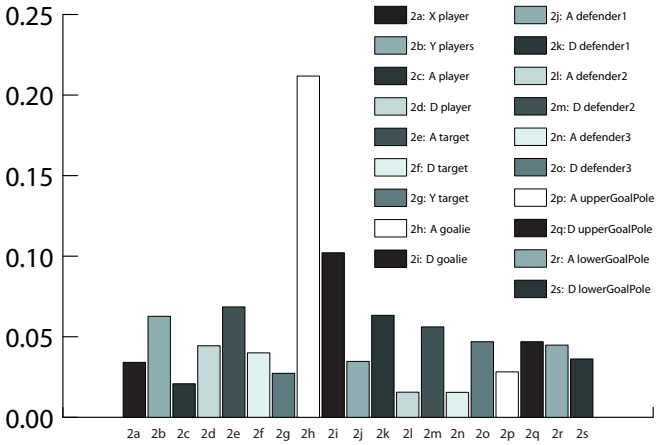


Figure 6. Contribution factors of the success-rate net

covers lots of goalshots. Region 3, finally, is the most wide-spread only containing a few positive data patterns (see Fig 7). From Region 3 down to Region 1 we gradually decreased the threshold that is used for the final decision whether to shoot or not. We observed that these rules could successfully prevented the agent from trying to score a goal when he is far from the goal and there is enough time and space to improve its scoring position.

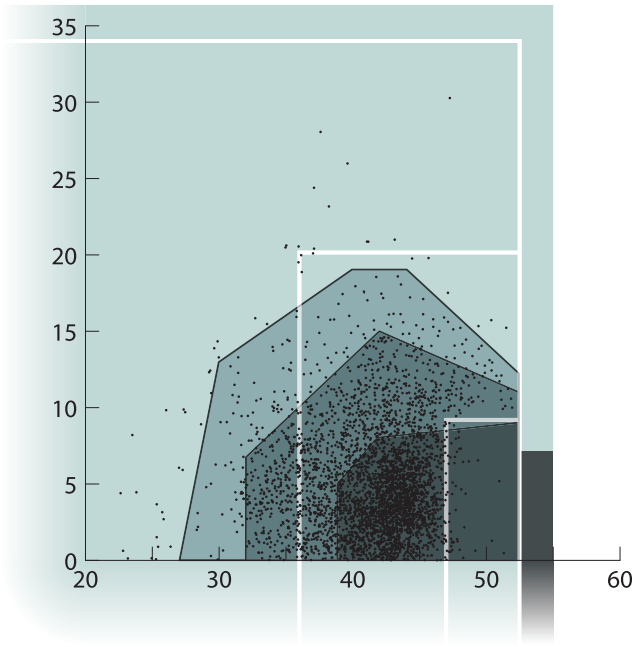


Figure 7. Zone model

For the purpose of finetuning and evaluating the performance, 400 test games were played. To allow for a meaningful comparison this was done by letting the conventional RoboLog team play against a RoboLog team with the new neural modules. As the performance of an agent is dependent on the available computational power the goalshot extraction heuristic described in section 2.1 was once again used for automatically evaluating all games played. Thus, the ratio of scored goals to scoring attempts could also be calculated and so the real performance without the deviation of server related performance losses could be determined.

Tab 2 shows the average performance over the 440 test games played. Statistics of all games played are summarized on the left half and the final 60 games on the right half of the table. This distinction is due to the fact, that we tried different parameter-settings for the networks and the zone model in the first 380 games. This resulted in fluctuations of the performance. The best set of parameters was finally used for the last 60 games.

	average test		average final	
	conventional	ANN	conventional	ANN
games total	440		60	
games won	121	130	11	21
ratio won/total	0.275	0.295	0.183	**0.35**
shots total	578	556	75	83
shots goals	172	216	23	33
ratio goals/shots	0.298	0.388	0.307	**0.398**

Table 2. Results for test phase and final settings

The ratio of successful shots to scoring attempts is significantly better for the team with the neural networks. Although most of the games still were a draw, in the end the new goalshot module could clearly outperform the conventional module by winning twice as often.

Another indication for the potentials of this approach is the performance of the RoboLog team at the RoboCup German Open 2004 where the new module was used in a competition for the first time. Even though, the overall results were not good and therefore don't look promising on the first sight the performance of the module becomes obvious after having a closer look at the logfiles. There were hardly any chances to score for the RoboLog team because the RoboLog strikers rarely got close to the opponent goal. So once more the ratio of goals to scoring attempts was calculated by using the goalshot extractor described in section 2.1. This more signifcant benchmark turned out to be exactly 50% which means that every second shot was successfull. In addition to that, a RoboLog agent for the first time managed to score against the Brainstormers04 team. Brainstormers04 ranked third in the end and conceded only two more goals in the whole competition.

3 CONCLUSION

This paper outlined a technique that uses data obtained from prerecorded soccer games for supervised neural network learning. The benchmark used for testing this approach is the Optimal Scoring Problem. The problem was tackled by decomposing it into two sub problems which where both individually addressed with one multilayered perceptron each, resulting in a variety of applications. The results show that observational learning from logfiles using neural networks delivers a promising performance while providing a series of advantages compared to previous work on this field.

- The time consuming step of generating training data from repeated experiments is not required.

- Training patterns obtained from prerecorded games provide universal information about the observed situations. The data is not limited by a specific agent used for data generation.
- The features used for training can be easily extended. There is no fundamental limitation due to complexity if further input variables are added to the networks.

Besides that, the extensive statistical analysis of goalshots provided in this paper should arouse interest of everyone dealing with simulated robot soccer.

3.1 Future work

It is obvious that the presented work is not able to provide a really optimal scoring policy.

First of all, there are starting points to improve the module without making fundamental changes. More sophisticated inputs to the networks, like speed, acceleration and body/view-angle of moving objects on the field, would most likely result in more accuracy. In addition, putting more effort into the learning process and providing more training data would also help the networks to better generalize over the seen samples.

But it becomes apparent, that all those straightforward improvements will never be able to result in an optimal scoring policy. To achieve that all future situations on the soccer field and the strategies of both teams would have to be taken into account. The work introduced in this project would in that case only provide a solution to a sub task in a broad decision support system. For a holistic solution, a prediction of the next actions of the opponent team (opponent modeling) and the own team needs to be made in order to set up an optimal scoring strategy. Only then, would it be possible to decide whether there will be a better position to score if the attacker performs some action, like dribbling first, and then tries to score instead of shooting right away. This cannot be achieved by a simple zone model with thresholds.

Steps in this direction could be to rank success rates of various actions of all team-mates (see [1]) or to use an auction protocol to decide on the next action to be carried out. However, it is very difficult to take all relevant future actions into account in this rapidly changing environment. But a one-step optimization has not the potential of being optimal.

Besides that, there are other interesting aspects to think about. Obtaining data by generation as done in previous work has the capability of finding situations not observed in prerecorded games. This could help to round off areas with sparse logfile training data or test unconventional strategies. Thus, if possible, a combination of both kinds of data acquisition seems to be the most promising approach.

Finally, a policy more specific to an opponent team could be advantageous and achieved in two ways. For one thing, training data from shots against one specific team could be emphasized in the training process to bias the network. This would result in a specific network for each opponent team. For another thing, online learning could most dynamically handle new situations and is as promising as challenging.

ACKNOWLEDGEMENTS

I would like to thank Oliver Obst for all his motivating support.

REFERENCES

[1] Sebastian Buck and Martin Riedmiller. *Learning Situation Dependent Success Rates Of Actions In A RoboCup Scenario*. In R. Mizoguchi and J. Slaney, editors, PRICAI 2000 Topics in Artificial Intelligence, number 1886 in Lecture Notes in Artificial Intelligence, Springer Verlag, page 809, 2000.

[2] Mao Chen, Klaus Dorer, Ehsan Foroughi, Fredrik Heintz, ZhanXiang Huang, Spiros Kapetanakis, Kostas Kostiadis, Johan Kummeneje, Jan Murray, Itsuki Noda, Oliver Obst, Pat Riley, Timo Steffens, Yi Wang and Xiang Yin: *Users Manual, RoboCup Soccer Server, for Soccer Server Version 7.07 and later*; February 11, 2003

[3] Vojislav Kecman. *Learning and Soft Computing: Support Vector Machines, Neural Networks, and Fuzzy Logic Models*, page 267, Bradford Book, MIT Press, Cambridge, Massachusetts, 2001.

[4] Jelle Kok, Remco de Boer and Nikos Vlassis. *Towards an optimal scoring policy for simulated soccer agents*. In M. Gini, W. Shen, C. Torras, and H. Yuasa, editors, Proc. 7th Int. Conf. on Intelligent Autonomous Systems, pages 195-198, IOS Press, California, March 2002. Also in G. Kaminka, P.U. Lima, and R. Rojas, editors, RoboCup 2002: Robot Soccer World Cup VI, Fukuoka, Japan, pages 296-303, Springer-Verlag, 2002.

[5] Tayler Raines, Milind Tambe and Stacy Marsella. *Towards Automated Team Analysis: A Machine Learning Approach*. Third international RoboCup competitions and workshop, 1999.

[6] Michael Wooldridge. *Intelligent Agents*. In Gerhard Weiss, editor, Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence, page 30, MIT Press, Cambridge, Massachusetts, 2000.

Learning Decision Trees for Action Selection in Soccer Agents

Savas Konur [1] and **Alexander Ferrein** and **Gerhard Lakemeyer** [2]

Abstract.

In highly-dynamic domains such as robotic soccer it is important for agents to take action rapidly, often in the order of a fraction of a second. This requires, a possible longer-term planning component notwithstanding, some form of reactive action selection mechanism. In this paper we report on results employing decision-tree learning to provide a ball-possessing soccer agent in the SIMULATION LEAGUE with such a mechanism. The approach has payed off in at least two ways. For one, the resulting decision tree applies to a much larger set of game situations than those previously reported and performs well in practice. For another, the learning method yielded a set of qualitative features to classify game situations, which are useful beyond reactive decision making.

1 Introduction

In highly-dynamic domains like robotic soccer it is important for agents to take action rapidly, often in the order of a fraction of a second. This is especially true in the application domain considered in this paper, the ROBOCUP SIMULATION LEAGUE with 11 players per team on a 2D playing field. Such tight time constraints require, a possible longer-term planning component notwithstanding, some form of reactive action selection mechanism. By *reactive* we mean, roughly, that decisions are made solely on the basis of a description of the current situation or world model. In particular, this precludes any evaluation of different possible courses of actions as in planning.

When presented with a game situation in the SIMULATION LEAGUE, humans are usually quite capable of choosing a reasonable action for, say, the ball-possessing agent. However, it is not at all easy to encode this "expert" knowledge in a way suitable for an artificial soccer agent for at least two reasons:

1. It is not clear what the salient features of a game situation are, which determine the action to be chosen. Presumably, these features would include qualitative descriptions such as the team member or opponent closest to the ball. But what the relevant ones?
2. Even if we were given those features, it is not clear how to translate them into rules for decision making. We could try to hand-code them, but this approach is likely error-prone, not to mention the difficulty of eliciting the rules from the human expert.

Perhaps the best way to overcome these problems is to use machine-learning techniques. Deciding what action to take next in robotic soccer can be thought of as a classification problem, where a game situation is classified according to the best next action. Machine-learning techniques suitable for classification are decision-tree learning such as ID3 [8] or C4.5 [9]), neural networks [12, 18] and reinforcement learning [15, 18, 16].

For our work we have chosen decision-tree learning, in particular, C4.5, as it is capable to deal well with both issues raised above. For one, given a sufficiently large set of training examples, the system automatically builds a decision tree, which encodes the rules for action selection. Compared to other techniques like neural networks, decision trees also have the well-known benefit that they can be inspected and understood by humans. For another, it is not necessary to decide beforehand what the relevant features are for classification. All that is needed is that the system is given a sufficiently large set. The relevant features are produced as a side-effect of building the decision tree in the sense that only those features or attributes that eventually appear as nodes in the decision tree are thought of as relevant.

We remark that we applied learning to all types of players (except the goalie) anywhere on the field, but we restricted ourselves to players in ball possession.

We believe that our results are noteworthy for at least the following reasons. For one, the resulting decision tree covers a much wider range of game situations and actions than in previous work such as [7, 17]. For another, as we will see in the discussion of experimental results, a team using this decision tree, but which is otherwise not optimized at all, performs surprisingly well. Finally, as already noted above, while decision-tree learning by itself does not come up with qualitative world descriptions, it is nevertheless useful in pruning irrelevant attributes from a given set.

This rest of the paper is organized as follows. In Section 2 we briefly discuss existing learning methods applied to robotic soccer. In Section 3, we describe our approach to decision-theoretic learning of action selection for a soccer agent in the SIMULATION LEAGUE, followed by a discussion of experimental results in Section 4. The paper ends with a brief summary and concluding remarks.

2 Related Work

In this section we present some of the work on applying machine learning techniques to robotic soccer and action selection. One focus is learning of basic agent skills such as *dribbling*, *passing*, and *intercepting*. [11], for example, use reinforcement learning for this purpose. In [13, 17] a form of so-called *Layered Learning* is proposed. It provides a bottom-up hierarchical approach to learning agent behaviors. In this framework, the learning at each level is directly used in the learning at the next higher level. The bottom layer considers low-level individual agent skills such as *ball interception* or *drib-*

[1] Free University of Amsterdam Artificial Intelligence Department, Amsterdam, The Netherlands, skonur@cs.vu.nl,

[2] RWTH Aachen, Knowledge-Based Systems Group, Aachen, Germany, {ferrein,gerhard}@cs.rwth-aachen.de

bling. In contrast to [11], the behaviors are learned using a neural network. At higher levels, action selection of the ball-possessing agent is learned using multi-agent reinforcement learning. We remark that the authors consider only eight kicking actions, which is much more limited than in our case. (A comparison of multi-agent reinforcement learning methods in the soccer domain can be found in [**?**].) In earlier work, Matsubara et al. [7] considered action selection using neural networks. There the scope was even more limited, as they restrict themselves to the decision of whether to shoot directly to the goal or to pass to a better positioned player. Decision-tree learning has been applied in robotic soccer as well. For example, Visser and Weland [19] recently applied C4.5 to learning aspects of the strategy of the opposing team in the SIMULATION LEAGUE.

Outside of the soccer domain, action selection for robots is often addressed using reinforcement learning. For example, [**?**] proposes hierarchical Q-learning for action selection, where the control task of a robot is divided into a set of simpler problems each learned separately. Another reinforcement learning approach to the action selection problem was proposed by Humphrys [**?**]. Each behavior module proposes an action with a certain weight of which the action with the highest weight is executed. The weights of the actions are modified based on the difference between the weight of the action being executed and the action a behavior module proposed using a form of reinforcement learning. The application domain presented in [**?**] is a simulated environment of a house keeping robot.

3 Learning the Decision Tree

In this section we present how we applied C4.5 to our SIMULATION LEAGUE agent. We start with an overview of the categories which should be learned, i.e. the action which the agent should perform. In Section 3.2, we present the attributes which turned out to be appropriate for the SIMULATION LEAGUE before we show how we instructed the agent in Section 3.3. The consulting procedure in on-line games is represented in Section 3.4.

3.1 Skill Hierarchy and Meta-Level Actions

As C4.5 cannot deal with parameterized categories to be learned [9], we implemented special behaviors which are to be selected by the decision tree. Figure 1 gives an overview of the skill hierarchy we use in our reactive soccer agent. The low-level action layer comprises basic actions like *dashing to a position*, *accelerating the ball to a certain velocity*, or *freezing the ball*. Those commands are translated into the SOCCERSERVER commands, such as *dash*, *kick*, *turn*, etc. The intermediate action layer defines actions like *moving to a position*, or *kicking the ball to a certain point*, which are based on the low-level action layer. High-level actions use the intermediate actions for the desired behavior. *dribbling* and *passing the ball to a teammate* are two examples of high-level actions taken from [1].

C4.5 requires that output values (categories) of decision trees must be discrete and specified in advance. This means action categories which we use in the learning process should not take any argument in order to satisfy the C4.5 requirements. Therefore, we cannot directly use the high-level actions in the learning process since they require some arguments in order to be executed. For that reason, we need new actions which should have the form of a argument-free discrete category. In order to accomplish this purpose we have introduced the meta-level actions which use the high-level actions to generate the desired behavior. These skills are parameterless encapsulations of skills from the other layers suitable to deal as a category for C4.5.

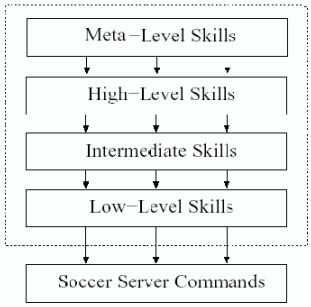


Figure 1. Skill hierarchy

The meta-level actions calculate necessary arguments before calling the corresponding high-level actions. In our current implementation, we have defined 15 meta-level actions (see below). An example for such an action is the dribble action depicted in Fig. 2. Some decisions like with which angle and speed the agent should dribble are made. For the supervision process it is very important that the supervisor has the semantics of the respective meta-level action in mind in order to give the right advice. The high-level dribble action in turn is responsible for correctly determining when to kick and intercept the ball in order to move player and the ball to the demanded position on the field.

```
dribble()
if ball is not in kickable margin then
   return intercept()
else
   if path toward opponent goal is free then
      ang ← direction to opponent goal
      type ← DRIBBLE_FAST
   else if path toward goal is fairly free then
      ang ← direction to opponent goal
      type ← DRIBBLE_SLOW
   else
      ang ← getDirectionOfWidestAngle()
      if ang = wide then
         type ← DRIBBLE_FAST
      else
         type ← DRIBBLE_SLOW
      end if
   end if
end if
return dribble(ang, type)
```

Figure 2. The dribble action which is executed by the decision tree

In the following, we give an overview of the categories (meta-level actions) which were used.

- *Outplay Opponent* The ball is played into the opponent's back followed by an intercept action.
- *Dribble* calculates the angle relative to the agent where it should dribble to. A second argument is the speed with which the agent should dribble. Two different speeds are distinguished: slow and fast.

- *Direct Pass* comprises several actions. It is distinguished between direct passes of an attacker and between a defensive player and a midfielder. Moreover, there exists a pass action for back passes and passes in front of the player. We have to split the direct pass action because the different aspects (playing a pass to a player in front or to the back) does not fit in one pass action model. C4.5 is not able to determine the differences in the semantics only by looking at the attributes. Each different instance of a direct pass share the calculation of the "least congested team-mate". Heuristically, this team-mate is chosen. The heuristics is based on the number of opponents in a certain distance around the player, there exists a free pass-way to that respective player, and some more.
- A *Through Pass* is a pass which is played behind the opponent defense. A free space behind the defensive line is found where a team-mate is able to receive.
- A *Leading Pass* is a pass in the run-way of the respective team-mate. It is calculated if a team-mate can intercept the ball after the pass in a certain amount of time.
- The *Shoot at Goal* action calculates a point on the opponent goal line with maximum distance to the opponent goal keeper.
- With *Clear Ball* the player simply kicks the ball as far away as possible. For instance, if a defender is not able to dribble or pass the ball to a team-mate it seem reasonable to bring the ball as far away from the own goal as possible.
- *Turn to Opponent Goal* When the agent is in ball possession and cannot see the opponent goal in order to perceive the opponent's goal keeper position, this action enables the agent to turn towards the goal without leaving the ball.

3.2 Constructing the Attribute Set

In order to generate a good classification by the C4.5 algorithm choosing an appropriate attribute set is a crucial task. Having irrelevant attributes in the attribute set is the main reason for *overfitting* [8, 9]. Another difficulty for finding an appropriate set lies in the nature of the soccer domain. As there are different player types and situations during a game where each player has to react in different ways according to its type and location on the field, we have to account for this by dividing the field into different regions. One such possible division is depicted in Figure 3 which was proposed in [1]. One approach to the problem could have been to learn different trees for different player types, such as *attacker*, *midfielder*, *defender*, by constructing the test sets with only the relevant information. However, this approach raises several problems: (1) it is not trivial to recognize all relevant regions; statically dividing the field into defense, midfield and attack is not sufficient because also a defender might sometimes be located in a midfield region, (2) a separate construction of the training set for each region and player type is a tedious task and available data from the LOGPLAYER[3] consists of whole game information, (3) different decision trees for each player type according to the game situations demand a selection mechanism that tells which tree should be consulted; this would take the same problem to a higher level.

Therefore, we decided to use only one decision tree containing the distinguishing features like player types and playing region as attributes. We also restrict the consulting of the decision tree to game situations where players are in ball possession.

From the considerations above and from many experiments we arrived at 35 attributes, which can be summarized as follows:

[3] The LOGPLAYER is a tool coming with the SIMULATION LEAGUE simulation server to replay recorded games.

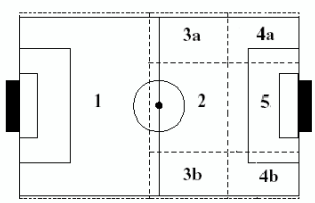


Figure 3. Possible regions a player can be in.

- *Type of player* is a discrete attribute and distinguishes between defender, midfielder, and attacker.
- *Playing region* is a discrete attribute representing in which region the player is located. The different regions are depicted in Figure 3.
- *Closest teammate to ball* is a boolean attribute denoting if the player is the closest player to the ball.
- *Distance and angle to ball, goals and opponent goal keeper* are continuous attributes determining the agent's distance and angle to the ball, the own as well as the opponent goal, and to the opponent goal keeper.
- *Distances and angles to the visible teammates and opponents* are a number of attributes denoting the visible teammates and opponents of an agent.
- *Closest team to the ball* is a boolean attribute which is true if one player of our team is the closest player to the ball and false otherwise.
- *Ball possessing team* takes three values corresponding to whether the ball is in kickable range for our team, for the opponent team, or for none.
- *Ball distances and angles to both goals* are continuous attributes representing the distances and angles of the ball to both of the goals.
- *Opponent goalie's distances and angles to its goal posts* is a number of attributes representing the the distance and angle of the opponent goalie to the opponent goal posts.

The reader should note that it was the decision-tree learning algorithm that ultimately decided that these are the relevant attributes of a game situation for a player in ball possession, as only these attributes were used in the decision tree. For example, it turns out that only the *five* nearest players to the ball are ever considered relevant. One possible explanation for this are the restrictions due to two dimensions of the current SIMULATION LEAGUE, where passes across the opponent defense are impossible. Other attributes which were used during tests were not contained in the resulting tree and therefore deemed irrelevant.

We also remark that the choice of attributes may likely be different for players other than the one in ball possession. For example, one would not expect the goalie to care much about the distance to the opponent's goal.

The reader should note that many of the above attributes have a continuous domain. We make use of C4.5's ability to discretize continuous attributes given the training set. This discretization sometimes results in wrong classifications during the consulting phase, as hard bounds on the attributes are drawn. Nevertheless these errors seem acceptable in practice.

```
...
MyPlayerType = 2:
|   BallDistanceToOpponentGoal <= 18.2609 :
|   |    MyCurrentPlayingRegion = 4: 9 (6.0/1.2)
|   |    MyCurrentPlayingRegion = 5:
|   |    |   MyDistanceToOpponentGoal <= 12.4953 :
|   |    |   |    MyDistanceToOurGoal <= 92.6233 : 1 (3.0/2.1)
|   |    |   |    MyDistanceToOurGoal > 92.6233 :
|   |    |   |    |    MyDistanceToOpponentGoalie > 18.5529 : 9 (3.0/2.1)
|   |    |   |    |    MyDistanceToOpponentGoalie <= 18.5529 :
|   |    |   |    |    |   OpponentGoalieGoalRightCornerAngle <= 47.7002 : 11 (40.0/2.6)
|   |    |   |    |    |   OpponentGoalieGoalRightCornerAngle > 47.7002 :
|   |    |   |    |    |   |   MyAngleToBall <= 26.746 : 11 (4.0/1.2)
|   |    |   |    |    |   |   MyAngleToBall > 26.746 : 10 (2.0/1.0)
|   |    |   |    MyDistanceToOpponentGoal > 12.4953 :
|   |    |   |    |    MyDistanceToOpponentGoalie <= 6.47349 :
|   |    |   |    |    |    OpponentGoalieGoalRightCornerDistance > 16.9349 : 9 (5.0/2.3)
|   |    |   |    |    |    OpponentGoalieGoalRightCornerDistance <= 16.9349 :
|   |    |   |    |    |    |    MyDistanceToSecondVisibleOpponent <= 5.27978 : 4 (3.0/1.1)
|   |    |   |    |    |    |    MyDistanceToSecondVisibleOpponent > 5.27978 : 11 (11.0/2.5)
|   |    |   |    |    MyDistanceToOpponentGoalie > 6.47349 :
|   |    |   |    |    |    MyAngleToOpponentGoal <= -103.167 : 10 (3.0/1.1)
|   |    |   |    |    |    MyAngleToOpponentGoal > -103.167 :
|   |    |   |    |    |    |    MyDistanceToThirdVisibleOpponent <= 8.71111 :
|   |    |   |    |    |    |    |   MyDistanceToOpponentGoal <= 14.6303 : 9 (2.0/1.0)
|   |    |   |    |    |    |    |   MyDistanceToOpponentGoal > 14.6303 :
|   |    |   |    |    |    |    |   |   MyAngleToFirstVisibleTeammate > 53.8028 : 8 (5.0/2.3)
|   |    |   |    |    |    |    |   |   MyAngleToFirstVisibleTeammate <= 53.8028 :[S7]
|   |    |   |    |    |    MyDistanceToThirdVisibleOpponent > 8.71111 :
|   |    |   |    |    |    |   ClosestTeamToBall = 0: 1 (0.0)
|   |    |   |    |    |    |   ClosestTeamToBall = 2: 11 (2.0/1.0)
|   |    |   |    |    |    |   ClosestTeamToBall = 1:
|   |    |   |    |    |    |   |   MyDistanceToOurGoal > 95.9754 : 9 (5.0/2.3)
|   |    |   |    |    |    |   |   MyDistanceToOurGoal <= 95.9754 :
|   |    |   |    |    |    |   |   |    MyDistanceToThirdVisibleTeammate <= 14.797 :[S8]
|   |    |   |    |    |    |   |   |    MyDistanceToThirdVisibleTeammate > 14.797 :[S9]
|   BallDistanceToOpponentGoal > 18.2609 :
|   |   MyAngleToOpponentGoal <= 98.1456 :
|   |   |    MyAngleToOpponentGoal <= -89.512 :
|   |   |    |    MyDistanceToFirstVisibleOpponent <= 1.87341 : 8 (21.0/2.5)
|   |   |    |    MyDistanceToFirstVisibleOpponent > 1.87341 :
|   |   |    |   MyAngleToOpponentGoal <= -118.357 :
|   |   |    |   |   OpponentGoalieGoalRightCornerAngle <= 12.9107 : 10 (53.0/3.8)
|   |   |    |   |   OpponentGoalieGoalRightCornerAngle > 12.9107 : 8 (3.0/1.1)
|   |   |    |   MyAngleToOpponentGoal > -118.357 :
|   |   |    |   |   MyAngleToFirstVisibleTeammate <= -113.017 : 4 (7.0/2.4)
|   |   |    |   |   MyAngleToFirstVisibleTeammate > -113.017 :
|   |   |    |   |   |   MyDistanceToThirdVisibleTeammate <= 13.8264 :
|   |   |    |   |   |   |    MyDistanceToThirdVisibleOpponent <= 14.2218 : 8 (9.0/2.4)
|   |   |    |   |   |   |    MyDistanceToThirdVisibleOpponent > 14.2218 : 1 (4.0/2.2)
|   |   |    |   |   |   MyDistanceToThirdVisibleTeammate > 13.8264 :
|   |   |    |   |   |   |    MyDistanceToOurGoal <= 74.9802 : 10 (15.0/4.7)
|   |   |    |   |   |   |    MyDistanceToOurGoal > 74.9802 : 4 (2.0/1.8)
|   |   |    MyAngleToOpponentGoal > -89.512 :
|   |   |    |    BallAngleToOpponentGoal <= 32.6327 :
|   |   |    |   ClosestTeamToBall = 0: 1 (0.0)
...
```

Figure 4. Excerpt from the resulting decision tree.

3.3 Gathering the Training Data

For the supervision process we extended the LOGPLAYER to be able to associate the actions described in Section 3.1 to players. This *supervisor monitor* generates the training examples by storing the output category (actions) together with the input categories (world model information).

It is important to note that while calculating the attribute values we cannot use the global information from the LOGPLAYER directly. Instead, we must calculate the supervised agent's relative world model from the global information and derive the attribute values from it. This is important because while the agent consults the decision tree in on-line games, the world model information comes from the SOCCERSERVER, which supplies the agent with relative information about the world model. Therefore, in the supervision process by calculating the relative information from global view, it is guaranteed that our training and test data are almost from the same distribution.

Another important point to be noted is that the supervisor should have a good idea of how soccer is played in order to give advice to the agent. For humans it is easier to classify a given situation including qualitative measures and give advices of what to do than to formalize a good action selection scheme. In the supervision process, the idea to specify the action classification of a play situation was that a player should select the most suitable action which provides him with the best advantage among alternatives actions. In this case, we can say that each action has a priority which depends on the player type and the region the agent plays in. Below is some part of the scheme that we used in the supervision process while advising the agents:

```
if scoring prob. is very high then
  goalshot
else if agent in defensive region close to our goal then
   if no opponent close and agent faces our goal then
     turn to opponent goal
    else if there is a very free teammate ahead then
      pass ball directly to this teammate
    else if trajectory to opponent goal is fairly free then
      dribble forward
    else
      clear ball
    endif
. . .
  else if agent in wings close to opponent goalie then
. . .
```

One might ask why we simply did not implement the above scheme instead of using a learned decision tree. As motivated in the introduction human beings are good in classifying the world into qualitative categories but encoding this as agent control software is much harder. As one can see from the scheme it uses qualitative statements like "very" or "fairly". When supervising "fairly" is evaluated by the supervisor in the complex situation the agent is in. On the other hand, by having a qualitative world model it would be interesting to compare an agent using the supervision scheme as action selection with the decision tree learned by C4.5.

Naturally, the supervisor makes mistakes in the classification or decides on the border line, giving contradictory advices. But as C4.5 is very robust against such mistakes they do not matter that much as they would in a hand-coded variant of the scheme.

3.4 Consulting the Decision Tree

In the previous sections, we considered how the attribute and data sets are gathered through the supervision process (the *training phase*). After the training phase the *model generation phase* starts in which these input files are passed through the C4.5 system, and the decision tree model is produced by executing the C4.5 program. That is, at the end of these phases we have acquired a model which can be used by an agent to classify unseen cases. In the ROBOCUP context, classification means offering a convenient action to the agent as it is playing an on-line game in the SOCCERSERVER. This task is done in a different process which we call the *consult phase* in which an agent consults the resulting decision tree to select an action in a play situation. An agent consults the decision tree model when the ball is kickable for him. In this case, a new process is started in which the attribute values are calculated according to the world situation. Based on these values the decision tree offers an action category which will be performed by the agent in this particular world situation, and the consult process halts for this time instance. Whenever the agent is in the ball-kickable margin, this process is started again. This process repeats until the game finishes. The hierarchical relationship between these phases is shown in Figure 5.

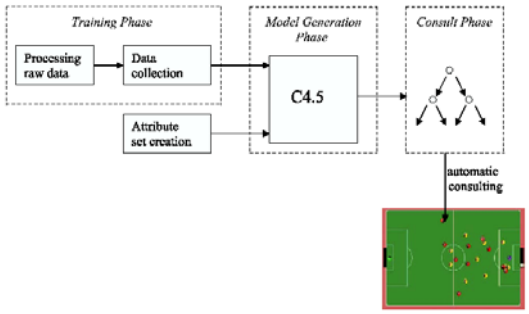


Figure 5. Overview of processing the *Reactive Component* for our soccer agents

Figure 4 shows parts of the resulting decision tree for a midfield player. Attributes are the nodes in this tree, e.g. *MyPlayerType* or *BallDistanceToOpponent Goal*. The leaves of the tree can be identified by numbers which correspond to a respective meta-level action, e.g. action 1 stands for *dribble*, 4 represents a *through pass*, and 11 means *shoot to goal*. The pair which follows an action shows the number of training instances and the number of misclassifications. The numbers in square brackets represent another subtree which is not shown here for readability.

4 Empirical Results

For assessing the quality of the learned decision tree we conducted several experiments.

The first question of interest was the accuracy of our training data. In total, we collected 3000 training examples and grouped them in training sets in steps of 500 examples up to the largest set containing the whole number of training examples (see Table 1). For each set size we built several instances choosing randomly from the whole set of training data. Table 2 shows the classification error rates. The column *Tree size* reflects the number of nodes the tree contains. Based on this table we can make the following observations:

First, the results show a (slightly) decreasing error rate with an increasing number of examples. The fact that we are left with an error rate of almost 9 % even before pruning has at least two reasons. One reason is that the supervisor makes mistakes giving contradictory examples. The other is that we use a large number of continuous attributes. For a continuous attribute, C4.5 finds a split value which maximizes information gain for the respective attribute. This discretization leads to misclassifications.

Second, in each category, we see the error rate of the pruned tree is higher than that of the original tree. Actually, this result is expected since in the pruning process some branches of the tree are replaced by a leaf node, yielding misclassification of some of the examples which were previously classified correctly.

Finally, it should be noted that the size of the trees gradually increases as the size of training data gets bigger, since C4.5 adds new branches to the decision trees in order to correctly classify the data instances.

Category	1	2	3	4	5	6
Set Size	500	1000	1500	2000	2500	3000

Table 1. Sizes of the test sets.

Cat.	Before Pruning		After Pruning	
	Tree size	Error(%)	Tree size	Error
1	174	8.40	144	9.74
2	353	7.82	296	9.40
3	493	8.76	422	9.94
4	672	8.60	588	9.75
5	846	8.20	722	9.50
6	979	8.02	847	9.30

Table 2. Error rates for the training set.

The next interesting question was how good the decision tree classifies unseen examples. We therefore played a large number of games against several teams with a different tree for each category from our training set (For each category we collected 1500 test examples). For assessing "ground truth" we classified for each logged game the situations according to the supervision scheme we presented in Section 3.3. The results over the training games are presented in Table 3 and Figure 6

Category	1	2	3	4	5	6
Classification Ratio (%)	35.1	46.3	59.8	64.1	66.8	66.5

Table 3. Results of training games.

By looking at the table and figure we can see that there is a sharp increase in the correctness between the Category 1 and Category 3. However, the performance increases only slightly between Categories 3 and 5. In the last category we even see a small reduction in the correctness of the classification. This suggests that the optimal size of the training set is reached at around 2500 examples.

The highest ratio of correct classifications we obtained is 66.8 % (Category 5). If we take the RoboCup's domain characteristics and

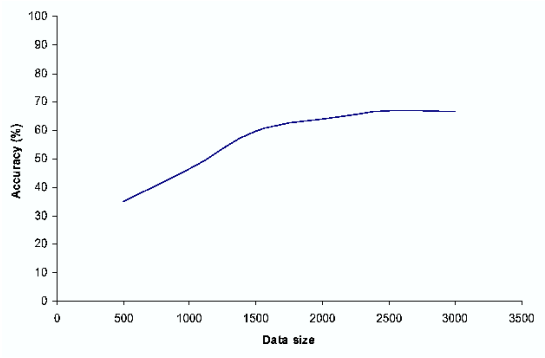


Figure 6. Learning curve of the agent

restrictions into account, we can say that this value is quite reasonable. Especially our results seem to compare favorably with other case studies. For example, Matsubara *et. al.* [7] performed an experiment, in which only the simple situation of two attackers attempting to score a goal against a single opponent is examined. In this experiment, the attackers learned when to select either 'pass' or 'shoot' actions. The ratio of the correct classification that the results showed was 68 %. Note also that the choices in this experiment are far simpler than in our case where we consider all skills for all types of players.

AllemaniACs: Robolog	2 : 0
AllemaniACs: VirtualWerder	1 : 0
AllemaniACs : UvaTrilearn	0 : 9
AllemaniACs : WrigthEagle	0 : 0

Table 4. Some test game results

We played several games against SIMULATION LEAGUE teams from 2003 showing the performance of the learned decision tree. Table 4 shows the results of some of these games. Against mediocre teams like Robolog or Virtual Werder we are able to win. Against the world champion Uva Trilearn our approach leaves room for improvement. One has to note that for these games the agent used the decision tree when a player was in ball possession and executed some standard behavior like "move to strategic position" or "search ball" otherwise. The agent was not highly tuned as we wanted to see the performance of the decision tree.

5 Conclusions

In this paper we described an application of the decision-tree learning method C4.5 to RoboCup's SIMULATION LEAGUE. The method was used to learn the action selection strategy of the whole team, that is, defenders, midfielders, and attackers, when a player is in ball possession. We were able to obtain decision trees which performed surprisingly well in real game situations. Moreover, the method is suitable for selecting the relevant attributes from a given set of qualitative world descriptors.

While this paper focusses on reactive action selection, we believe that cooperative team-play cannot be achieved by reactive control alone, taking only the current game situation into account. Consider, for example, the situation where a wing-change would be necessary

because one side of the field is blocked by opponents. A good choice would be to shift the game to the other wing of the field. It is hard to imagine how such behavior could come about without some form of deliberation where different possible courses of action are considered and evaluated.

For this purpose we have developed an architecture which provides for reactive control as well as a *deliberative component* using the logic-based language Golog [6]. Golog is a language for reasoning about actions and change and is based on the situation calculus [14]. Recent extensions like dealing with a continuously changing world [5] and the integration of a form of decision-theoretic planning [2] to account for the uncertainty arising in the soccer domain makes it a suitable language to reason about scenarios like a wing-change and to coordinate the agents accordingly (cf. [3] for an example in the soccer domain).

When using deliberation one needs a symbolic representation of the environment. Therefore, we are interested in building up a qualitative world model which can be used for the deliberative component. One of the central problems is finding the appropriate attributes to describe the environment in a qualitative way. Recently, Dylla et al. [4] approached this problem by looking at the issue of specifying soccer moves based on the knowledge from a domain expert's (from [10] in their case) for different ROBOCUP leagues. As soccer theory is described in a very abstract fashion, qualitative descriptions clearly seem important, but the theory itself does not answer the question of which qualitative descriptions are most suitable.

The present paper can perhaps be thought of as one step in this direction. As we saw, one interesting outcome is that for the player in ball possession only the five nearest team-mates and opponents seem to matter. Applying the presented approach also for other players like the goal keeper one probably can learn more about the relevant information in robotic soccer.

We believe that the proposed method for reactive action selection is not restricted to the ROBOCUP domain. Highly dynamic domains have in common that actions must be performed rapidly, even if those actions seem to be sub-optimal. Applying decision-tree learning yields one method for implementing a reactive action selection mechanism. In future work we will apply this approach to other dynamic real-time domains, for instance to soft-bots in computer games, to get comparable results. Also the suitability of decision-tree learning for achieving good attribute sets for qualitative world modeling will be further investigated.

Acknowledgments

This work was supported by the German National Science Foundation (DFG) in the Priority Program 1125, *Cooperating Teams of Mobile Robots in Dynamic Environments*. We would like to thank the anonymous reviewers for their comments.

REFERENCES

[1] R.de Boer and J.Kok, *The Incremental Development of a Synthetic Multi-Agent System: The UvA Trilearn 2001 Robotic Soccer Simulation Team*, Master's thesis, University of Amsterdam, 2002.

[2] Craig Boutilier, Ray Reiter, Mikhail Soutchanski, and Sebastian Thrun, 'Decision-theoretic, high-level agent programming in the situation calculus', in *Proc. of AAAI-00*, pp. 355–362. AAAI Press, (July 30– 3 2000).

[3] F. Dylla, A. Ferrein, and G. Lakemeyer, 'Specifying multirobot coordination in ICPGolog – from simulation towards real robots', in *Proc. of the Workshop on Issues in Designing Physical Agents for Dynamic Real-Time Environments: World modeling, planning, learning, and communicating (IJCAI 03)*, (2003).

[4] Frank Dylla, Alexander Ferrein, Gerhard Lakemeyer, Jan Murray, Oliver Obst, Thomas Röfer, Frieder Stolzenburg, Ubbo Visser, and Thomas Wagner, 'Towards a League-Independent Qualitative Soccer Theory for Robocup'. accepted at 8th RoboCup International Symposium as poster.

[5] Henrik Grosskreutz and Gerhard Lakemeyer, 'On-line execution of cc-Golog plans', in *Proc. of IJCAI-01*, (2001).

[6] H.Levesque, R.Reiter, Y.Lesperance, F.Lin, and R.Scherl, 'Golog: A logic programming language for dynamic domains', *Journal of Logic Programming*, (1997).

[7] H.Matsubara, I.Noda, and K.Hiraki., 'Learning of cooperative actions in multi-agent systems: a case study of pass play in soccer', *In S. Sen, editor, AAAI Spring Symposium on Adaptation, Coevolution and Learning in multi-agent systems*, (1996).

[8] J.Quinlan, 'Induction of decision trees', *Machine Learning, Kluwer Academic Publishers*, (1986).

[9] J.Quinlan, *C4.5 Programs for Machine Learning*, Morgan Kaufmann, 1993.

[10] Massimo Lucchesi, *Coaching the 3-4-1-2 and 4-2-3-1*, Reedswain Publishing, 2001.

[11] M.Riedmiller, A.Merke, D.Meier, A.Hoffman, A.Sinner, O.Thate, and R.Ehrmann, 'Karlsruhe brainstormers - a reinforcement learning approach to robotic soccer', in *RoboCup 2000*, Lecture Notes in Artificial Intelligence, Springer-Verlag, (2001).

[12] P.Antognetti and V.Milutinovic, *Neural Networks: Concepts, Applications, and Implementations, Vol. II*, Printice Hall, 1991.

[13] P.Stone, *Layered Learning in Multiagent Systems: A Winning Approach to Robotic Soccer (Intelligent Robotics and Autonomous Agents)*, MIT Press, 2000.

[14] R. Reiter, *Knowledge in Action*, MIT Press, 2001.

[15] R.Sutton and A.Barto, *Reinforcement Learning: An Introduction*, MIT Press, 1998.

[16] P.Norvig S.Russell, *Artificial Intelligence: A Modern Approach-Second Edition*, Printice Hall, 2002.

[17] S.Whiteson and P.Stone, 'Concurrent layered learning', in *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pp. 193–200. ACM Press, (2003).

[18] T.Mitchell, *Machine Learning.*, McGraw-Hill, 1997.

[19] U.Visser and H.G.Weland, 'Using online learning to analyze the opponent's behavior', in *RoboCup 2002*, Lecture Notes in Artificial Intelligence, Springer-Verlag, (2003).

.

Dynamic Configuration of a Team of Robots

Robert Lundh and **Lars Karlsson** and **Alessandro Saffiotti**[1]

Abstract. We study teams of autonomous robotic agents in which agents can help each other by offering information-producing resources and functionalities. Depending on the current situation and tasks, the team may need to change its functional configuration, that is, which agents provide which functionalities to whom. We propose to use knowledge-based techniques to automatically synthesize new team configurations in response to changes in the situation or tasks. This note summarizes our approach, and reports our preliminary steps in this direction.

1 Introduction

Consider a society of autonomous robotic systems embedded in a common environment. By an *autonomous robotic system* we mean here any computer-controlled system able to sense the environment, take decisions about actions to perform in the environment, and perform those actions. These include mobile robots, like the one pictures in Fig. 1, as well as simpler devices like domestic appliances or monitoring apparatuses. We do not assume that the systems in the society are homogeneous: they may have different sensing, acting, and reasoning capacities.

From an abstract point of view, this society can be seen as one distributed robotic system. The system usually includes a number of *functionalities* organized in some way, for instance, in a generic two-layer hybrid architecture like the one shown in Fig. 2 (left). In these architectures, the top layer implements higher cognitive processes for world modeling (M) and for planning and deliberation (D). The bottom layer implements sensori-motor processes for sensing and perception (P) and for motion control (C), which are connected to a set of sensors (S) and actuators (A).

In practice, the above functionalities can be distributed across different physical units in the society (robots, devices, etc). Each unit includes several functionalities in each one of the {P, M, D, C, S, A} classes, which it can use to perform the tasks assigned to it. In addition, each unit may use functionalities from other units in order to compensate for some one that it is lacking, or to improve its own. Consider for example the following scenario involving a pair of outdoor robots, A and B, equipped with pan-tilt stereo cameras. Robot A needs to perform the action to cross a gate in a metalic fence, as shown in Fig. 1. To do so, it must have a P functionality to measure the relative position and orientation of the gate, since this information is needed by the controller. Robot A can use its stereo camera to observe the edges of the gate during the crossing, but the measure obtained when these edges are near is not very reliable. Robot B, however, could observe the entire scene from a distance and compute a better estimate of the relative position and orientation between robot A and the gate. Robot B can therefore offer this functionality to A so that A can perform its task more reliably — see Fig. 2 (right).

In general, the same task can be performed by using different functionalities in different robots and connected in different ways. For example, the previous gate-crossing task can be achieved by either (1) connecting the camera functionality in A to the gate-crossing functionality in A, or (2) connecting the camera functionality in B to the gate-crossing functionality in A. We informally call *configuration* any way to allocate and connect the functionalities of a distributed multi-robot system. Note that we are interested in functional software configurations, as opposed to the hardware configurations usually considered in the field of reconfigurable robotics (e.g., [7, 12]). Clearly, which configuration should be preferred depends on the task, situation and resources. This suggests that the system should be able to switch to a new configuration whenever these conditions change.

The focus of our work is to study configurations of a society of robotic agents. Our objective is threefold:

1. To define the concept of functional *configuration* of a robot society: which robot is providing which functionalities to which one, and how.
2. To study how to *dynamically change* the configuration of a robot society in response to changes in the environment, in the tasks, or in the available resources.
3. To use knowledge-based techniques (e.g., planning and monitoring) to automatically *detect* when a configuration is not adequate any more, and *synthesize* a new one for the current situation.

Figure 1. An outdoor robot about to cross a gate in a fence.

[1] Center for Applied Autonomous Sensor Systems, University of Örebro, Sweden. {robert.lundh, lars.karlsson, alessandro.saffiotti}@aass.oru.se

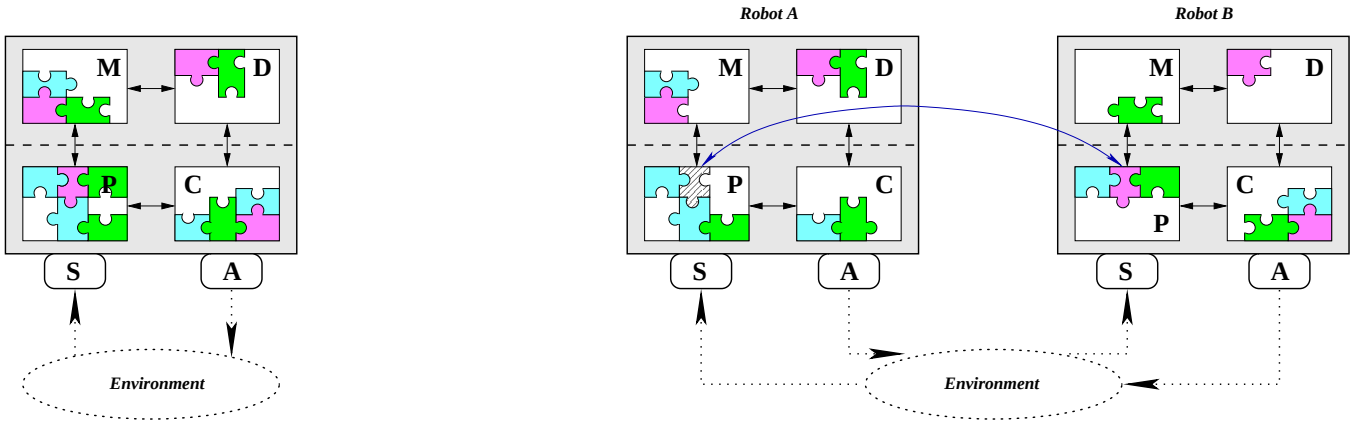


Figure 2. Left: abstract view of a team as a distributed robotic system. Right: a simple team configuration consisting of two-robots: Robot B is providing a missing perceptual functionality to Robot A.

At this initial stage of our work, we focus on the first objective: to define a concept of configuration which is adequate from the point of view of the other two objectives above. The rest of this paper outlines our first steps in this direction.

2 Related work

There are several relevant areas from which one might take inspiration to address the above objectives. In the area of multi-robot systems, much work has been done on the problem of multi-robot task allocation, that is, how to allocate a number of tasks to a number of robots taking into account that different robots may be differently adequate for different tasks (see, e.g., [9] for an overview and analysis). Some examples are the ALLIANCE architecture [13] and Local Eligibility approach [25] based on local utility estimates, and the M+ [2] and MURDOCH [8] approaches. Closely related to task allocation are the issues of robotic team configuration and of dynamic role assignment [23, 21, 11]. Chaimowicz *et al* [3] consider roles as the part of an individual agent in a cooperative task. They define a role as a control mode in a hybrid automaton, and a role assignment is a transition in that automaton. The approach that we propose in this paper departs from the above works since we focus on the distribution and — in particular — the interconnection of atomic (action and perception) functionalities. These are combined to form behaviors, which achieve tasks.

The problem of distributing the performance of a task across a number of agents according to their respective capabilities has been widely addressed in the Distributed AI (DAI) and in the Multi-Agent Systems (MAS) communities. Early work in DAI considered distributed problem solving settings with a precedence order among sub-tasks [6]. Later work has included the notion of coalitions between sub-groups of more closely interacting agents [17]. The notions of team-work [14], capability management [22] and norms [1] have also been used in the MAS community to account for the different forms of interactions between the sub-tasks performed by the agents in a team. These works, however, typically assume software agents, and are not concerned with issues of physical action, mobility, and perception, which play a central role in our work.

Another area of interest is program supervision, where program modules are combined, tuned and evaluated in order to solve specific computational tasks such as image processing, often using planning techniques [10, 4, 18]. Our work adds several dimensions to program supervision since we deal with multiple physical agents with both sensing and acting capabilities.

In a paper more similar in spirit to this one, Simmons et al [20] consider a task involving a heterogeneous team of robots — a crane, a robot with a manipulator, and a robot with stereo cameras — solving a construction task where a beam is placed on top of a stanchion. This task requires tight cooperation between the robots involved. Cooperation between the robots is hand-coded, although the authors declare their intention to use planning techniques to set up the cooperation. For specifying tasks, they use TDL (task description language) [19], an imperative language which is a superset of C++. This language does not appear to be adequate for automatic reasoning about configurations by, e.g., a planner.

3 Framework

The first goal in our research program is to develop a definition of configuration that is adequate for the three objectives presented in the introduction. In general, a configuration of a team of robots may include interconnected functionalities of two types: information providing functionalities, that is, functionalities that change the internal state by providing or processing information; and action executions, that is, functionalities that change the state of the environment. (Some functionalities can have both aspects.) In the work presented here we focus on the information providing functionalities, since these are a less studied aspect in the planning literature. The extension of our framework to include action functionalities is left as a second step.

To define our notion of configurations, a clarification of the three concepts of functionality, resource and channel is in order.

3.1 Functionality

A *functionality* is an operator that uses information provided by other functionalities to produce additional information. Each instance of a functionality is located in a specific robot (or other agent). The functionality consists of:

- a specification of inputs, to be provided by other functionalities. For each input, it contains information about domain (e.g. video images) as well as timing information (e.g. every 100 ms).

- a specification of outputs, to be provided for other functionalities. They also contain domain and timing information.
- a specification of relations between inputs to outputs.
- a set of causal preconditions, that is conditions in the environment that have to hold in order for the functionality to be operational.
- a set of causal postconditions, that is conditions in the environment which the functionality is expected to achieve.
- possibly also a specification of costs in terms of e.g. computation and energy.

3.2 Resource

A *resource* is a special case of a functionality. There are two different types of resources: sensing resources and action resources. As mentioned previously, only sensing resources will be considered in this paper. A *sensing resource* has no input from other functionalities, and is typically a sensor that gives information about the current state of the surrounding environment (e.g., a camera) or perhaps information about the internal state of the robot.

3.3 Channel

A *channel* transfers data from one functionality to another. A channel can be in terms of either inter-robot or intra-robot communication, and be on different mediums (radio, network, internal connections). A channel may have requirements of band width, speed and reliability.

3.4 Configuration

A *configuration* is the set of functionalities and the set of channels that connects functionalities to each other. Each channels connects the output of one functionality to the input of another functionality.

In the context of a specific world state, a configuration is *admissible* if the following conditions are satisfied:

- each input of each functionality is connected via an adequate channel to an output of another functionality with a compatible specification (information admissibility).
- all preconditions of all functionalities hold in the current world state (causal admissibility).
- the combined requirements of the channels can be satisfied.

3.5 Examples

In order to illustrate the above concepts, we consider a concrete example inspired by the scenario described in the introduction. In order to more easily test the example on real robots (see next section), we consider an indoor office building. A robot is assigned the task of moving from one room to another one by crossing a door between the two rooms. The "cross-door" action requires information about position and orientation of the door with respect to the robot performing the action. The resources available are two indoor robots (including the one crossing the door) each one equipped with a camera and a compass. The door to cross is equipped with a wide-angle camera.

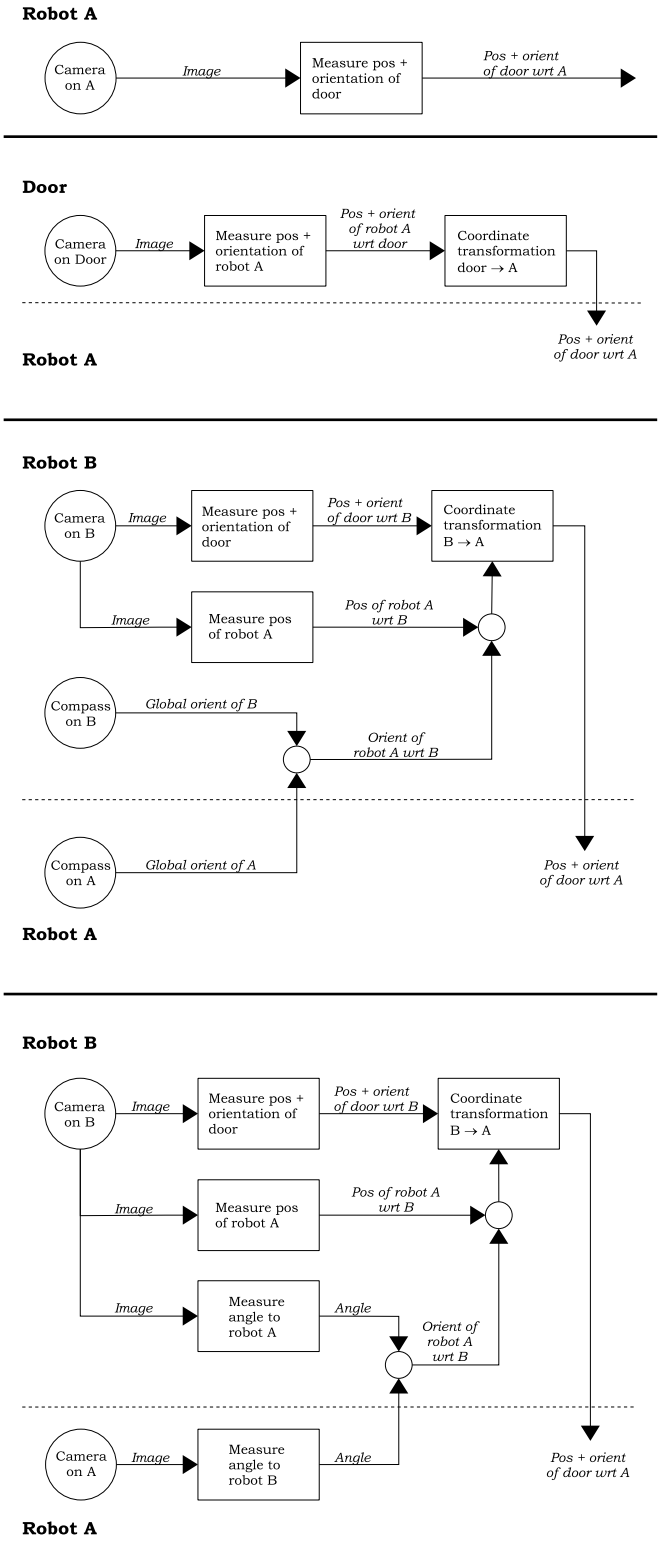


Figure 3. Four different configurations that provide the position and orientation of a given door with respect to robot A. See explanation in the text.

Two functionality operators from this scenario are shown below:

```
(Op Measure_Door (?Y)
    Inputs:    Image (?X)
    Outputs:   Door position & orientation
               of ?Y relative to ?X
    Preconds:  Door ?Y fully visible
               in image from ?X
    Postconds: -
    Transform: measuring door procedure
    )

(Op Camera (?X)
    Inputs:    -
    Outputs:   Image (?X)
    Preconds:  CameraOn
    Postconds: -
    Transform: image retrieval procedure
    )
```

The input and output of a functionality represent the data flow associated with the functionality. In the `Measure_Door` example we have an image taken by camera `?X` as input and from that we are able to compute the position and orientation of the door `?Y` relative to `?X` as output. The second example is an operator for a camera. Output from `Camera` is an image taken by camera `?X`. Since `Camera` is a sensing resource no input is specified. There are also certain conditions that need to be satisfied in order for the functionality to operate, and conditions that will be satisfied if the functionality is executed. This causal flow is represented as preconditions and post-conditions in the operator. For instance the precondition for `Measure_Door` is that the door `?Y` is fully visible in the input image and the precondition for `Camera` is that the camera is switched on. The body of the operator describes the computations performed on the input in order to generate the specified output provided that the preconditions are satisfied. Notice that the output of `Camera` matches the input of `Measure_Door`. Intuitively, this means that a channel between these two functionalities can legally be created.

Fig. 3 illustrates four different (admissible) configurations that provide the information required by the action "cross-door", which include the functionalities above.

The first configuration involves only the robot performing the action. The robot is equipped with a panoramic camera that makes it possible to view the door even during the passage. The camera produces information to a functionality that measures the position and orientation of the door relative to the robot.

The second configuration in Fig. 3 shows the other extreme, when all information is provided by the door that the robot is crossing and the robot is not contributing with any information. The door is equipped with a camera and functionalities that can measure the position and orientation of the robot relative to the door. This information is transformed into position and orientation of door with respect to the robot before it is delivered to robot A.

The third and fourth configurations in Fig. 3 consist of two robots (A and B), each with its own set of resources and functionalities.

In the third configuration, robot A (the robot performing the "cross-door" action) only contributes with one resource, a compass. Robot B's resources are a compass and a camera. The camera provides information to two functionalities: one that measures the distance and orientation to the door, and another one that measures the distance to robot A. All these measurements are computed relative to robot B. In order to compute the position and orientation of the door relative to robot A, we need to use a coordinate transformation. This in turn requires that we know the relative position and orientation of robot A relative to B. The relative position is obtained from the camera information. The relative orientation can be obtained by comparing the absolute orientations of the two robots, measured by their two on-board compasses.

The fourth configuration in Fig. 3 is similar to the third one, except that the orientation of robot A relative to B is obtained in another way, i.e., no compasses are used. Both robots are equipped with cameras and have a functionality that can measure the bearing to an object. When the robots are looking to each other, each robot can measure the bearing to the other one. By comparing these two measurements, we obtain the orientation of robot A relative to robot B.

4 A Simple Experiment

In order to test whether sharing of functionalities in different configurations would actually allow us to solve simple coordination examples, we have conducted a series of experiments using a pair of real robots equipped with different sensors. These experiments were also aimed at assessing the mechanisms for the switching between configurations. In these first experiments, the configuration generation and configuration switches were hand-coded. We intend to eventually make both aspects automatic.

We present here a simple experiment based on the third and fourth configurations in Fig. 3. The platform used were two Magellan Pro robots from iRobot, shown in Fig. 4. Each robot runs an instance of the layered hybrid architecture Thinking Cap [16].

Both robots are equipped with compasses and fixed color cameras. They have additional sensors (e.g., sonars, laser, and an electronic nose) not used in our experiments. Since the cameras are fixed, they can only measure distances to objects further away than 2 meters. The environment consists of two rooms (R1 and R2) with a door connecting them. The door and the robots have been marked with uniform colors in order to simplify the vision task (see Fig. 4).

The following scenario describes how the two configurations were used, and demonstrates the importance of being able to reconfigure dynamically. Robot A and robot B are in room R1. Robot A wants to go from room R1 to room R2. Since the camera can only measure distances to objects further away than 2 meters, robot A is not be able to perform the action on its own. Robot B is equipped with the same sensors as robot A, but since robot B is not crossing the door it is able to observe both the door and robot A from a distance during the whole procedure. We therefore configure our team according to the third configuration in Fig. 3, and execute the task. Robot A continuously receives information about the position and orientation during the execution of "cross-door".

When robot A enters room R1 it signals that the task is accomplished. This signal is received by robot B and the current configuration is played out. Next, robot B is assigned the task of going from room R1 to room R2. The same configuration as before is used to solve this task, but with the roles exchanged — i.e., robot A is now guiding robot B. This time, however, during the execution of the "cross-door" behavior a compass fails due to a disturbance in the magnetic field. This makes the current configuration not admissible, and a reconfiguration is necessary to proceed. The fourth configuration in Fig. 3 is still admissible even with no compass, and we therefore use this one to carry out the remaining part of the task. Fig. 5 shows the trajectories performed by the robots in a sample run of this experiment. In the picture, robot A is standing still at the observing position and robot B has just accomplished its task.

Figure 4. Robot B is guiding robot A through the door.

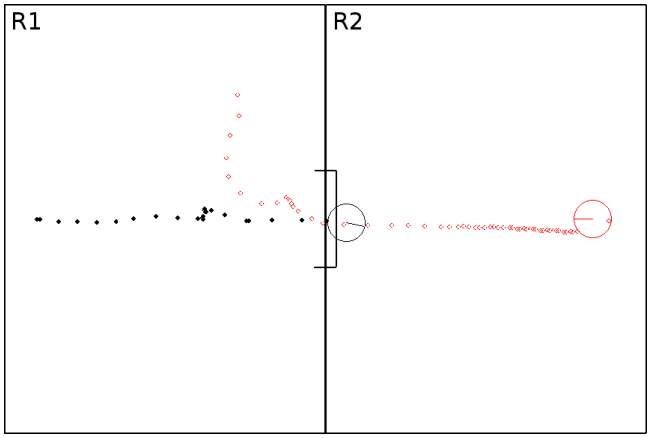


Figure 5. Robot A and B have both reached room R2. Circles show robot A's trajectory and dots show robot B's trajectory.

5 Conclusions

This paper has shown the first steps toward our goal to automatically synthesize a team configuration using knowledge-based techniques. Differently from most current works on team formation, our atomic unit of decomposition is not a task or a role, but a single functionality that a robot can make available to another one. The preliminary experiments indicate that we can achieve flexible forms of cooperations in this way. Moreover, we are able to describe information- and action-producing functionalities as abstract operators similar to the ones which are customary in planning systems. This suggests the possibility to build a system that uses planning techniques [24, 5] to automatically create team configurations given the current tasks, resources, and situation. We would like this system to be able to determine the most adequate configuration in terms of a set of criteria, like efficiency, reliability, or cost of resources and communication. Our research efforts are in this direction.

The distributed nature of our configuration is expected to be an important aspect in configuration planning. For instance, the cost and unreliability of inter-robot communications should be taken into account when evaluating alternative configurations. Moreover, configuration planning may have to be done in a distributed manner, or if centralized it must be preceded and followed by information exchanges between the robots.

Automatic re-configuration of a robotic team will be important as the task, environment and maybe also the composition of the team change dynamically. An important issue to consider here is how to decide when it is time to change configuration. A reconfiguration may be needed if the available functionalities change (e.g., due to malfunctioning), if the external conditions change, if the current task is completed, or if a new task is given. From an experimental perspective, we intend to apply automatic reconfiguration online on robots in increasingly complex and dynamic environments (e.g. 4-legged robotic soccer [15]).

ACKNOWLEDGEMENTS

This work was supported by the Swedish KK Foundation, the Swedish National Computer Graduate School in Computer Science (CUGS), and the Swedish Research Council.

REFERENCES

[1] G. Boella, 'Norms and cooperation: Two sides of social rationality', in *Agent Autonomy*, eds., H. Hexmoor, C. Castelfranchi, and R. Falcone, Kluwer, Boston/Dordrecht/London, (2003).
[2] S. Botelho and R. Alami, 'M+: a scheme for multi-robot cooperation through negotiated task allocation and achievement', in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1234–1239, (1999).
[3] L. Chaimowicz, M. Campos, and V. Kumar, 'Dynamic role assignment for cooperative robots', in *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, pp. 293–298, (2002).
[4] S. A. Chien and H. B. Mortensen, 'Automating image processing for scientific data analysis of a large image database.', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **18**(8), 854–859, (1996).
[5] M. E. desJardins, E. H. Durfee, C. L. Ortiz, Jr, and M. J. Wolverton, 'A survey of research in distributed continual planning', *AI Magazine*, **20**(4), 13–22, (1999).
[6] E.H. Durfee, V.R. Lesser, and D.D. Corkill, 'Coherent cooperation among communicating problem solvers', in *Readings in Distributed AI*, eds., A.H. Bond and L. Gasser, 268–284, Morgan Kaufmann, San Mateo, CA, (1988).
[7] T. Fukuda and S. Nakagawa, 'Approach to the dynamically reconfigurable robot systems', *Intelligent Robtics Systems*, **1**, 55–72, (1988).
[8] Brian P. Gerkey and Maja J Matarić, 'Sold!: Auction methods for multi-robot coordination', *IEEE Transactions on Robotics and Automation*, **18**(5), 758–768, (October 2002).
[9] Brian P. Gerkey and Maja J Matarić, 'Multi-Robot Task Allocation: Analyzing the Complexity and Optimality of Key Architectures', in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Taipei, Taiwan, (May 2003).
[10] L. Gong and A.C. Kulikowski, 'Composition of image analysis processes through objectcentered hierarchical planning', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **17**(10), (1995).
[11] J. S. Jennings and C. Kirkwood-Watts, 'Distributed mobile robotics by the method of dynamic teams', in *Proc. of the Intl. Symp. on Distributed Autonomous Robotic Systems (DARS)*, Karlsruhe, Germany, (1998).
[12] F. Mondada, M. Bonani, S. Magnenat, A. Guignard, and D. Floreano, 'Physical connections and cooperation in swarm robotics', in *Proc. of the 8th Int. Conf. on Intelligent Autonomous Systems (IAS8)*, pp. 53–60. IOS Press, (2004).
[13] L. Parker, 'ALLIANCE: An architecture for fault tolerant multi-robot cooperation', *IEEE Trans. on Robotics and Automation*, **14**(2), (1998).
[14] D.V. Pynadath and M. Tambe, 'Automated teamwork among heterogeneous software agents and humans', *Journal of Autonomous Agents and Multi-Agent Systems*, **7**, 71–100, (2003).
[15] A. Saffiotti, A. Björklund, S. Johansson, and Z. Wasik, 'Team Sweden', in *RoboCup 2001*, eds., A. Birk, S. Coradeschi, and S. Tadokoro, Springer-Verlag, Germany, (2002).

[16] A. Saffiotti, K. Konolige, and E. H. Ruspini, 'A multivalued-logic approach to integrating planning and control', *Artificial Intelligence*, **76**(1-2), 481–526, (1995).
[17] O. Shehory and S. Kraus, 'Methods for task allocation via agent coalition formation', *Artificial Intelligence*, **101**, 165–200, (1998).
[18] C. Shekhar, S. Moisan, R. Vincent, P. Burlina, and R. Chellappa, 'Knowledge-based control of vision systems', *Image and Vision Computing*, **17**, 667–683, (1998).
[19] R. Simmons and D. Apfelbaum, 'A task description language for robot control', in *Proceedings of the Conference on Intelligent Robotics and Systems*, Vancouver Canada, (October 1998).
[20] R. Simmons, S. Singh, D. Hershberger, J. Ramos, and T. Smith, 'First results in the coordination of heterogeneous robots for large-scale assembly', in *Proceedings of the International Symposium on Experimental Robotics (ISER)*, Honolulu Hawaii, (December 2000).
[21] P. Stone and M. Veloso, 'Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork', *Artificial Intelligence*, **110**(2), 241–273, (1999).
[22] I. J. Timm and P-O Woelk, 'Ontology-based capability management for distributed problem solving in the manufacturing domain', in *Multiagent System Technologies – Proceedings of the First German Conference, (MATES 2003)*, eds., M. Schillo and et al., pp. 168–179. Springer Verlag, (September 2003).
[23] D. Vail and M. Veloso, 'Multi-robot dynamic role assignment and coordination through shared potential fields', in *Multi-Robot Systems*, eds., A. Schultz, L. Parker, and F. Schneider, Kluwer, (2003).
[24] D. S. Weld, 'Recent advances in AI planning', *AI Magazine*, **20**(2), 93–122, (1999).
[25] B. B. Werger and M. J Matarić, 'Broadcast of local eligibility for multi-target observation', in *Distributed Autonomous Robotic Systems*, eds., L. E. Parker, G. Bekey, and J. Barhen, pp. 347–356. Springer Verlag, (2000).

Real-Time Agents: Reaction vs. Deliberation[1]

Carlos Carrascosa, José Fabregat, Andrés Terrasa and Vicente Botti
Deptartamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
Camino de Vera s/n, 46022, Valencia, Spain
{carrasco,jfabregat,aterrasa,vbotti}@dsic.upv.es

Abstract. In agents theory, it is commonly accepted that reactivity is one of the main features of an agent. Reactivity can be defined as the capability of an agent to respond to significant changes in its environment. Traditionally, reactivity has been confronted with the agent's capability of deliberation, in the sense that the most reactive an agent is, the least time it spends deliberating (and vice-versa). Agent architectures normally present a fixed proportion between reaction and deliberation, normally implemented by assigning a given amount of resources to each of them at design time, with no possibility of further adaptation at run time. In this way, the agent may work well for certain environments/problems, but it can poorly adapt this feature to changes in such initial conditions.

Therefore, if the agent could accommodate its reactivity to the current situation of the environment, its adaptability would be considerably enhanced and its behavior would be closer to humans. Furthermore, if the agent has real-time requirements, the agent's ability to adapt its reactivity becomes essential, because the environment will typically undergo periods of different stress conditions. In this sense, this paper introduces the concept of *Reactivity Degree*. This concept implies some meta-reasoning capabilities to be available in the agent, in order to dynamically decide the amount of resources which have to be assigned to deliberation and reaction. The paper also shows how to implement such concept in a hard real-time, hybrid agent architecture named ARTIS, as well as some experimental results which demonstrate the usefulness of this new concept.

1 Introduction

Reactivity is a general feature of live organisms, at both organic and cell level. It consists of producing a reaction to every physical or chemical stimulus of the environment. This reaction allows for the survival and development of such beings.

In the context of software systems, a reactive system is considered as a system that interacts with some independent "environment", which can be either physical (belonging to the real world) or simulated by another software system. In any case, the reactive system will need some sensors to detect changes in the environment, to which they have to produce responses of some kind. Normally, these responses will modify the internal state of the system and/or the environment itself.

In agents theory, it is commonly accepted that reactivity is one of the main features of an agent. Reactivity can be defined as the capability of an agent to respond to significant changes in its environment. This feature is present in all the alternative agent definitions in the literature, from the simplest [13] to the most complex ones [17]. Traditionally, reactivity has been confronted with the agent's capability of deliberation, in the sense that the most reactive an agent is, the least time it spends deliberating (and vice-versa). In fact, these confronted concepts have been used to classify the agent architectures in the literature into three groups (as established in [17]): reactive, deliberative and hybrid. Reactive and deliberative agent architectures are in opposite extremes, defining agents which either react to stimuli immediately (without deliberation) or spend all their time elaborating the best possible answer, with little or no computational and time restrictions.

Hybrid architectures, on the other hand, do consider an agent to possess both characteristics (reaction and deliberation). The proportion of reaction and deliberation in the agent definition may vary in different agent architectures, but, in every one of them, this proportion is a fixed trait in the agent. Such architectures normally implement this fixed proportion by assigning some given computational resources to the agent's reactive and deliberative layers at design time, with no possibility of dynamically reallocate such resources at run time. In this sense, each agent architecture is better suited to deal with certain problem domains and environments, depending on the amount of deliberation and reaction required, but they are all unable to adjust this feature to changes in the initial conditions of the problem.

As a result, if the agent could accommodate its reactivity to the current situation (including both its internal state and the status of the environment), its adaptability would be considerably enhanced. In this sense, it would behave closer to humans. For example, if a person sees an important paper starting to burn, she can spend a moment to think how to avoid the paper to be consumed by the flames; however, if her own hand is burning, she will not even think of what to do, she will just take the hand away from the fire immediately. This basic example presents a human being solving the same problem in two scenarios with different reactivity requirements.

Moreover, if the agent has to deal with a real-time environment, the ability to adapt its reactivity becomes essential, since some (or all) of the agent responses may have hard real-time restrictions. In such cases, the amount of time the agent may spend in deliberating is not only strictly bounded, but it may greatly vary depending on how stressed is the environment at the present moment. If the agent is statically designed to cope with the most stressed conditions at all times, it will waste its resources when such conditions do not apply. Conversely, underestimating the resources for dealing with the worst case may result in the agent missing a vital deadline because it spent

[1] Work partially funded by grants DPI2002-04434-C04-02 and TIC2003-07369-C02-01 of the Spanish government.

too much time calculating the solution.

With all this, the adaptive chances of the agent would be greatly improved if its reactivity was considered as a dynamic feature, which could be adapted to the current environmental situation. In this sense, this paper introduces the concept of *Reactivity Degree*. This concept implies some meta-reasoning capabilities to be available in the agent, in order to dynamically decide the amount of resources which have to be assigned to deliberation and reaction. The paper also shows how to implement such concept in a hard real-time, hybrid agent architecture named ARTIS, as well as some experimental results which demonstrate the usefulness of this new concept.

The rest of the paper is structured as follows: next section presents the concept of *real-time agent* and its base in the real-time systems field; Section 3 presents an overview of an specific real-time agent architecture, the ARTIS agent architecture; after that, Section 4 presents the concept of meta-reasoning and its application in the ARTIS architecture is presented, paying special attention to the concept of *reactivity degree*; Section 5 shows an application example of the new functionalities of the ARTIS architecture; last section presents the conclusions of the paper and some ongoing work.

2 Real-Time Agents

Real-time agents are agent-based software systems which have real-time requirements. In the Artificial Intelligence area, it is a common misunderstanding that a real-time system is a system which is continuously connected to some real environment, calculating its solution in non-simulated time; another typical misinterpretation of the term consider a real-time system as a system which has to provide a *fast* result. In fact, a real-time system (RTS) is a system which correction depends not only of the computation results, but also of the moment at which these results are produced [15]. In a RTS, the usual way of specifying the valid interval for a solution is by assigning a *deadline* to the task which is calculating this solution, meaning that if the solution is provided after that deadline, the solution loses quality or even it is completely invalid. In this sense, a *RTS* must try to achieve its objectives while also meeting its deadlines.

There are two types of *RTS* [15]:

Hard Real-Time Systems (HRTS). These are systems which have strict time requirements. In systems of this kind, if a task misses a single of its deadlines, the consequences may be grave (typically involving the integrity of the system itself, monetary loss or human lives).

Soft Real-Time Systems (SRTS): These are systems in which missing a deadline normally supposes a degradation in the system's quality response, but not intolerable loss.

According to this, real-time agents may be classified into hard and soft real-time agents, depending on whether they have strict temporal requirements or not.

In the RTS field, some research lines have been conducted toward the objective of adjusting the behavior of the system to changes in the environment (or in the system's internal state). This research line includes developments such as the *mode changes* [11], which propose a safe mechanism by which the system may change the set of tasks at run time without missing any hard deadline. An example of such techniques in the field of real-time agents can be found in the CIRCA/SA-CIRCA architecture [7, 8, 6], where a simple synchronous mode change is used. The problem with mode changes is that they are very drastic, in the sense that maybe it is not necessary to change the entire set of tasks.

3 ARTIS Agent ($\mathcal{AA}$)

This section provides a short description of the ARTIS Agent ($\mathcal{AA}$) architecture, for hard real-time environments (a more detailed description can be found in [3] [14] [4]). In accordance with existing agent architectures [17], the AA architecture could be labeled as a vertical-layered, hybrid architecture with added extensions to work in a hard real-time environment [3].

The ARTIS agent architecture guarantees an agent response that satisfies all the critical temporal restrictions of the system while also trying to obtain the best answer for the current environment status. This is due to its capacities for problem-solving, adaptability and proactivity, which have been added to the architecture.

The architecture of an $\mathcal{AA}$ can be viewed from two different perspectives: the user model (high-level model) [3] and the system model (low-level model) [16]. The user model offers the developer's view of the architecture, while the system model is the execution framework used to construct the final executable version of the agent.

To translate the user model' specification into the system model a toolkit, called *InSiDE* [14], is used. This toolkit allows the agent's designer to define the $\mathcal{AA}$'s user model and to convert this model to the corresponding system model automatically [5]. The result is an executable $\mathcal{AA}$.

3.1 User Model

From the **user model** point of view, the $\mathcal{AA}$ architecture is an extension of the blackboard model [9] which has been adapted to work in hard real-time environments. This model is formed by the following elements:

1. A set of **sensors** and **effectors** allowing the agent to interact with the environment. Due to the environment's restrictions, the perception and action processes are typically time-bounded.
2. A set of **in-agents** which model the $\mathcal{AA}$ behavior. The main reason to split the whole problem-solving method into smaller entities is to provide an abstraction which organizes the problem-solving knowledge in a modular and gradual way (see figure 1). There exists a CLIPS-like *language of entities* allowing the designer to specify these **in-agents**.

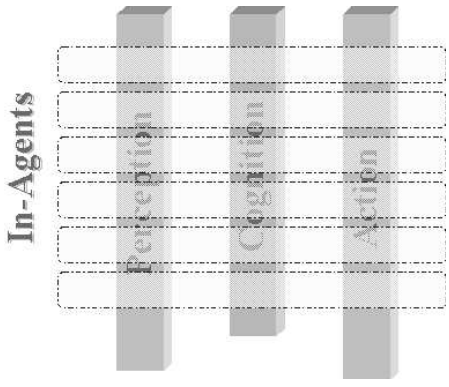


Figure 1. Modular division of an $\mathcal{AA}$ into in-agents

Each **in-agent** periodically performs an specific task. Each **in-agent** has to solve a particular subproblem, but all the **in-agents**

of a particular $\mathcal{AA}$ cooperate to control the entire problem, and an **in-agent** may use information provided by other **in-agents**.
In-agents can be classified into *critics* and *acritics*. The first ones are in charge of solving the essential problems of the $\mathcal{AA}$, so their execution are guaranteed at least for calculating a minimal quality answer. The last ones are in charge of solving non-essential problems of the $\mathcal{AA}$ in order to improve its performance quality.
A **critic in-agent** is characterized by a period and a deadline. The available time for the **in-agent** to obtain a valid response is bounded and, in this time, it must guarantee a basic response to the current environment situation.
From a functional point of view, an **in-agent** consists of two layers (see Figure 2): the reflex layer and the real-time deliberative layer.

The *reflex layer* assures a minimal quality response (an off-line schedulability analysis of the $\mathcal{AA}$, considering all the **in-agents** in the $\mathcal{AA}$, guarantees that this reflex layer will be fully executed). The reflex layer of all the **in-agents** make up the $\mathcal{AA}$ mandatory layer.

The *real-time deliberative layer* tries to improve this response (this layer will be executed in slack time). The real-time deliberative layers of all the **in-agents** form the $\mathcal{AA}$ optional layer. An **acritic in-agent** only has the real-time deliberative layer.

3. A set of **believes** comprising a world model (with all the domain knowledge which is relevant to the agent) and the internal state, that is, the mental states of the agent. This set is stored in a frame-based blackboard [2]. In a similar way as the **in-agents**, there is a CLIPS-like *language of classes* to specify this set.
4. A **control module** that is responsible for the real-time execution of the **in-agents** that belong to the AA. The temporal requirements of the two **in-agent** layers (reflex and deliberative) are different. Thus, the control module must employ different execution criteria for each one. In fact, the control module is divided into two submodules [4], *Reflex Server –RS–* and *Deliberative Server –DS–*, in charge of the reflex and deliberative parts of the AA respectively.

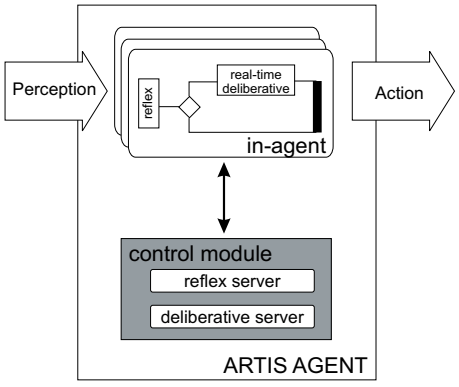


Figure 2. ARTIS Agent architecture

One of the main features of the $\mathcal{AA}$ architecture is its hard real-time behaviour. It guarantees the execution of the entire system's specification by means of an off-line analysis of the specification. This analysis is based on well-known feasibility analysis techniques in the RTS community, and it is described in [5].

3.2 System Model

The **system model** provides a software architecture for the $\mathcal{AA}$ that supports all the high level features expressed in the user model. The main features of this model are [5]:

A task Model that guarantees the critical temporal restrictions of the environment. So, the user model's **in-agents** are translated into system model's tasks. In this model, a generic task may have three parts:

- Initial: It is in charge of checking the system state with regards to the subproblem it knows. It also calculates a first answer to its problem, with low quality, but in a bounded time.
- Optional: It improves the quality of the answer calculated in the initial part, but this improvement may use time-unbounded methods.
- Final: It carries out the best answer calculated.

According to this model, only the initial and final parts of a critical task will have hard real-time restrictions, and also will be in charge of the interaction with the environment (by means of the sensor and effectors).
Off-line schedulability analysis. This off-line analysis only ensures the schedulability of real-time tasks (corresponding to **in-agents** with critical restrictions). However, it does not build a plan with the task execution sequence. There is a scheduling policy, compatible with the off-line analysis, that is used to decide the next task to be executed at run-time. This allows the $\mathcal{AA}$ to adapt itself to environment changes, and to take advantage of the tasks using less time than their *worst-case execution time* (*wcet*).
Some slack extraction method to on-line calculate the available time for executing the real-time deliberative layer.
A set of extensions to the Real-Time Operating System incorporating features for managing real-time capabilities.

It is very important to emphasize that there exists an automatic translation process between the user and system models, according to the correspondence that can be seen in figure 3.

The integration of intelligence in an $\mathcal{AA}$ lies in the effective management of the slack time by the control module.

The current version of the system model of an $\mathcal{AA}$ is implemented in RT-Linux 3.2-pre1 over Linux kernel 2.4.18.

4 Meta-Reasoning

In its most general meaning, *meta-reasoning* is any process interested in the operation of other computational process inside the same entity [12]. According to the nomenclature used in [10], an agent should be able to make two different types of decisions:

Meta or Macro-level decisions managed by the meta-level controller. This controller must be designed to take quick and cheap decisions about how many resources are dedicated to domain actions and how many to control actions.
Scheduling or Micro-level decisions managed by the domain-level controller.

The main purpose of the meta-level control activities is to optimize the agent execution. To do this, it allocates, in the proper moment and quantity, the processor and other resources between the control and domain activities.

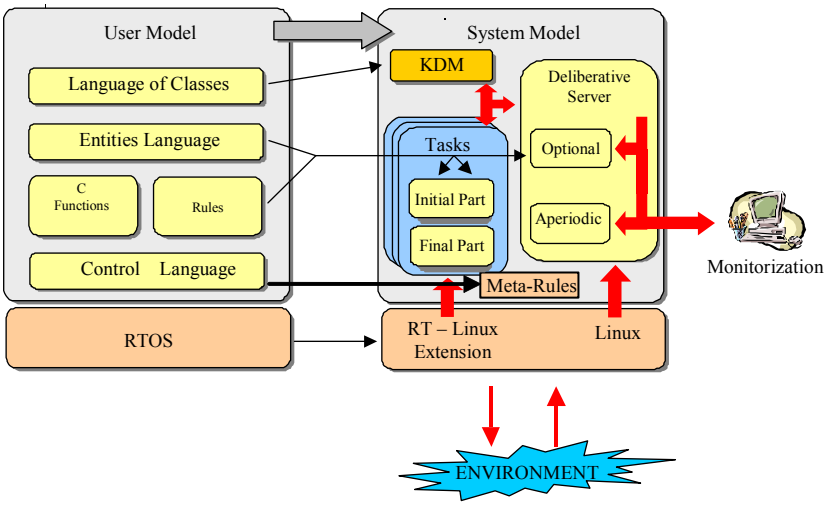


Figure 3. Correspondence between User and System Models

In the $\mathcal{AA}$ architecture, the Control Module includes all these controllers above mentioned, but also has to deliver with the peculiarities of this architecture. Due to its two reasoning layers, there are two domain-level controllers, that is, two schedulers:

First-Level Scheduler (FLS), in charge of the reflex layer. It is part of the *Reflex Server*.
Second-Level Scheduler (SLS), in charge of the real-time deliberative layer. It is part of the *Deliberative Server*.

This Control Module also incorporates the meta-level controller that must decide, among other things, how many time is dedicated to macro-level decisions as one of its possible meta-level decisions. All these decisions are carried out by a set of meta-rules that may be specified at design time by means of a *control language*. This language is also CLIPS-like to look like the rest of languages of the $\mathcal{AA}$'s user model.

4.1 Reactivity Degree

As it was presented in previous sections, the *Reactivity* must be seen as a degree rather than a trait. In this way, the *Reactivity Degree* of an agent must be defined.

The *Reactivity Degree* of an agent is a feature that indicates how much effort is going to dedicate such agent to deliberate. This degree defines two extreme situations with infinite intermediate states. These extreme situations are:

If the reactivity degree is zero, the agent works in **reflex mode**: it doesn't spend time to deliberate, to improve the first answer it has got.
If the reactivity degree is 1, the agent works in **deliberative mode**: it dedicates all its time to calculate the best answer to its problem.

As the reactivity degree is closer to 0 the agent is more sensitive to changes in the environment, whilst if the reactivity degree is closer to 1, it is more self-centered.

So, the reactivity degree allows an agent to *move* along the Arkin's robot control spectrum [1] to get the proper features to face the current situation.

4.1.1 $\mathcal{AA}$'s Reactivity Degree

This point centers the question of how to manage the reactivity degree of an agent, in the particular case of an $\mathcal{AA}$.

Focusing on the $\mathcal{AA}$ system model, after executing an initial part of a task, if there is available slack, the DS takes control of the $\mathcal{AA}$ during this slack time. One of the DS functions is to use the available time to schedule the execution of all the active optional parts (the ones whose initial parts have been executed and their final parts have not yet been executed).

If the DS finishes its operations before all the slack time is consumed, it returns the execution control to the RS. In this case, the RS is able to get the most out of this available time bringing forward part of the critical tasks execution[2].

This same method will be used to modify the *Reactivity Degree* of an $\mathcal{AA}$.

The *Reactivity Degree* of an $\mathcal{AA}$ is defined as the maximum available slack percentage that may be used for the agent to improve its answer (to deliberate) (figure 4). If the reactivity degree is zero, the agent works in **reflex mode** (it doesn't spend time to deliberate, to improve the first answer it has got), whilst if the reactivity degree is 1, the agent works in **real-time deliberative mode** (all its available slack time is dedicated to improving its answers).

Though it will have a default value, the reactivity degree is a dynamic value. So, for instance, an initial part of a task could detect if the agent must go to an emergency mode where it has to act immediately. In this case, the slack should be temporally annulated, and the whole system's execution would be bringed forward (suppressing the DS and optional parts execution).

[2] This method is even used by some second level scheduling policies used by the DS to join several slack gaps to improve the possibilities of optional parts execution

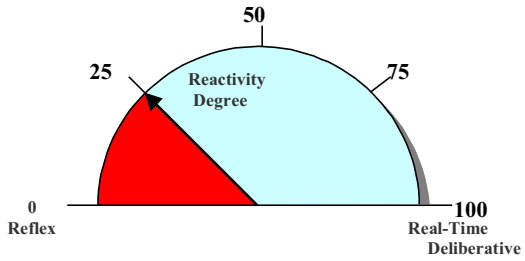


Figure 4. $\mathcal{AA}$'s Reactivity Degree

Moreover, if the situation is not so extreme, it couldn't be needed to wholly eliminate the slack, only just to reduce it. That is, the slack may be seen as a value in the interval [0, original slack]. Thus, there are two extreme execution modes (the one with 0 slack corresponding to the emergency mode or reflex mode, and the one with the original or maximum slack corresponding to the cognitive mode or real-time deliberative mode) with an indeterminate number of intermediate execution modes.

The use of this execution modes doesn´t violate the schedulability of the system, because it only changes the way of slack management.

Anyway, the proper management of the reactivity degree allows the $\mathcal{AA}$ to adapt to changes in the environment.

This management is one of the actions that may be carried out by the meta-reasoning done by the Control Module. This meta-reasoning process is able to adjust several parameters to change the AA reasoning process [4], as the reactivity degree, the second level scheduling policy, ...

4.2 Meta-Rules

Part of the meta-reasoning process is specified by the designer by means of a meta-rule language named *control language*. So, this language is used to establish the situations in front of which the agent has to change the reasoning process and how is this change carried out. For this reason, one meta-rule is triggered by an event (usually a modification of an agent's belief). Moreover, it has a condition that must be fulfilled to execute the actions specified in its right-hand side. This condition must check if the agent is facing a situation where it must change to face it.

The actions of these meta-rules establish the different ways the agent can adapt its way of behaving, allowing to change not only the reactivity of the agent, but also the usage of its process time.

Figure 5 presents an example of a meta-rule according to the syntax of the control language.

The Meta-Rules are translated to a data structure that is stored in a shared memory accessible from both parts of the Control Module. They are interpreted and executed at running time.

The decision of managing the Meta-Rules this way is to allow to meta-reason about the meta-reasoning process. This meta-meta-reasoning (that, nowadays, is one of the open issues in this work) will be able to learn and to forget meta-rules. The learning method used here will have to take into account its working in bounded time. Moreover, the agent will have to take into account not to fall in *digressing* while meta-meta-reasoning, that is, not to spend too much time in meta-reasoning and/or meta-meta-reasoning that there is not enough time left to execute nothing more.

```
(defMetaRule level 3
    ( MODIFICATION tank.tankA.level)
    ( tank.tankA.level >= 200 )
    ( tank.tankA.level < 400 )
    =>
    ( SetReactivityDegree ( 0.75 ) )
)
```

Figure 5. Example of meta-rule

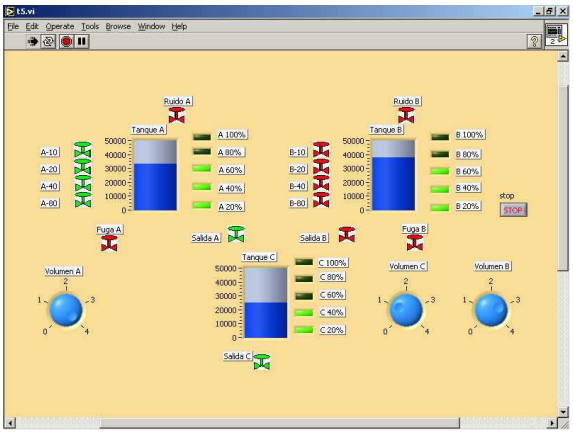


Figure 6. Simulated tanks in *LabView* 7

5 Example: Sewage Tanks

The main purpose of the following example is to show the possibilities of adaption using this new dynamic *Reactivity Degree* capability.

5.1 Problem Description

The objective of the agent of this example is to control some sewage tanks interconnected between them. The $\mathcal{AA}$ must control that the liquid level in all this tanks remain within an interval. This interval may be changed by the user even at running time.

It has to be taken into account that the system to control (the above mentioned tanks) will be simulated in a computer connected to the one of the $\mathcal{AA}$ by the serial port. So, though it is a simulated process, from the $\mathcal{AA}$'s point of view, it acts like a real physical process to control. The program used to make the simulation is *LabView* 7[3] (as shown in figure 6).

In fact, the $\mathcal{AA}$ has to control three water tanks: A and B, both with 5000 litres of capacity, and C with 10000 litres of capacity. There are five sensors at each tank to control their levels. These sensors are located every 20 % of the tank volume.

The water inputs to the system come from 4 taps with 10, 20, 40 and 80 litres per second of flow, respectively. Each one of these taps can just be open or closed but their sending flow can't be regulated (there are taps *all/nothing*).

[3] Program generator of virtual instrumentation developed for *National Instruments* allowing the simulation of physical processes and their real-time control

A and B tanks have one controlled output with a valve allowing them to empty their contents over tank C in a controlled way (by means of such valves). The flow of these outputs is 95 litres per second each one. On the other hand, there is valve to control the output flow of tank C, being of 180 litres per second. All these valves are also *all/nothing*.

Some additional inputs and outputs not automatically controlled have been added to introduce perturbations in the system. So, tanks A and B have one input tap, or noise, each one with 20 litres of flow and an output tap, or leak, of 10 litres per second of flow. The water proceeding of the leak doesn't go to tank C as in the automatic taps. It also has some controls to establish the interval of stored volume. These controls are represented by turning controls for each tank with the following meaning:

Position	Minimum Level	Maximum Level
0	0%	20%
1	20%	40%
2	40%	60%
3	60%	80%
4	80%	100%

5.2 $\mathcal{AA}$'s Design

The $\mathcal{AA}$ designed to control all the system is formed by three *in-agents*, one for each tank. This is the reason for *in-agents* in charge of A and B tanks to be identical.

Each *in-agent* is formed by three parts:

1. Sensorization: it is in charge of reading the sensors indicating the level of its respective tank. It is also in charge of reading the buttons of choosing the level (the ones in the control panel, and the emergency ones in the same simulator). This is the initial part of this critical *in-agent*.
2. Cognition: to calculate the actions to do (opening and closing of the corresponding tap). This part has three levels, the critical level 0 and two optional levels. The level 0 implements an algorithm to calculate a quick but low-quality answer:

   If the stored volume is under the asked one, open all the input taps and close the output valve of the corresponding tank.

   If such volume is above the asked one, close all the input taps and open the output valve.
3. Action: to carry out the actions calculated by the previous part, that is, to send the proper actions to the simulator to open or to close taps and valves.

The main time features of the critical *in-agents* of this $\mathcal{AA}$ are the following[4]:

in-agents	*Deadline*	Period	Optionals	*WCET**
In_Agent_A	2500000	2500000	T6–T9	200000
In_Agent_B	2700000	2700000	T10–T13	200000
In_Agent_C	2900000	2900000	T14–T17	200000

The column *WCET** indicates the worst-case execution time for the initial and final parts of the corresponding *in-agents* in this example, it is the same quantity for both cases).

[4] All the time features are in micro-seconds (because Linux works in this magnitude).

The purpose of this example is to check the improvement in the flexibility of the $\mathcal{AA}$ time management. For this reason, some Meta-Rules to change the $\mathcal{AA}$'s *Reactivity Degree* has been defined (for instance, to change it to 0 when some tank arrives to its maximum level, or the one in figure 5).

It has to be underlined that when the execution of a Meta-Rule changes the *Reactivity Degree* to 0, it will accomplish that **no** more optional parts will be executed and the execution of the tasks' final part will be advanced, that is, the $\mathcal{AA}$ will act as soon as possible to avoid the tanks overflow.

5.3 Execution Example

At this point, some chronograms are shown[5] to show how the above presented example works.

The $\mathcal{AA}$'s debugging toolkit uses to visualize chronograms *kiwi*[6]. When this toolkit is used, it stores in memory during the execution of the $\mathcal{AA}$ a set of trace events explaining this exection. After the $\mathcal{AA}$ has finished its execution, this events are translated into a file containing a *kiwi* chronogram. The chronograms included in this paper are captured from real executions of the $\mathcal{AA}$'s debugging toolkit. For this reason, the images include more information than the explained here. The necessary information to take into account in the following chronograms is:

The uppest row, labelled as *Kernel*, shows the execution intervals of the RS.
The next rows, and always in strictly decreasing priority order, show the critical tasks (corresponding to the user's model *in-agents*).
After that, it appears a row labelled as *Linux* representing Linux execution (during this time the $\mathcal{AA}$ is not executing nothing).
The next row corresponds to the DS.
The last part of the chronogram is composed by the rows corresponding to the optional tasks of the $\mathcal{AA}$, sorted also by priority.

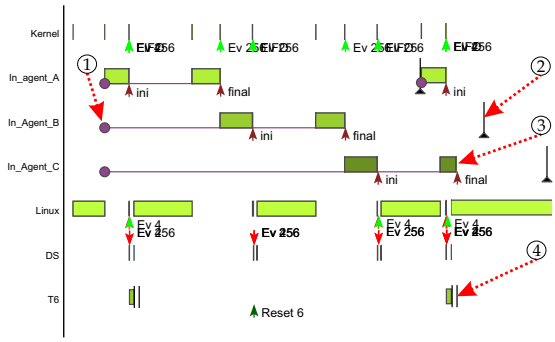


Figure 8. Kiwi's execution chronogram

To finish the explanation about the way tasks are represented in a chronogram, it is only necessary to indicate the highlighted parts of the figure 8:

[5] The $\mathcal{AA}$ has been executed over a Pentium III computer to 600 MHz with 128 Mb of RAM.

[6] Toolkit to visualize chronograms developed in Tcl/Tk by Agustín Espinosa. It is freely available in *http://rtportal.upv.es/rtportal/apps/kiwi/*

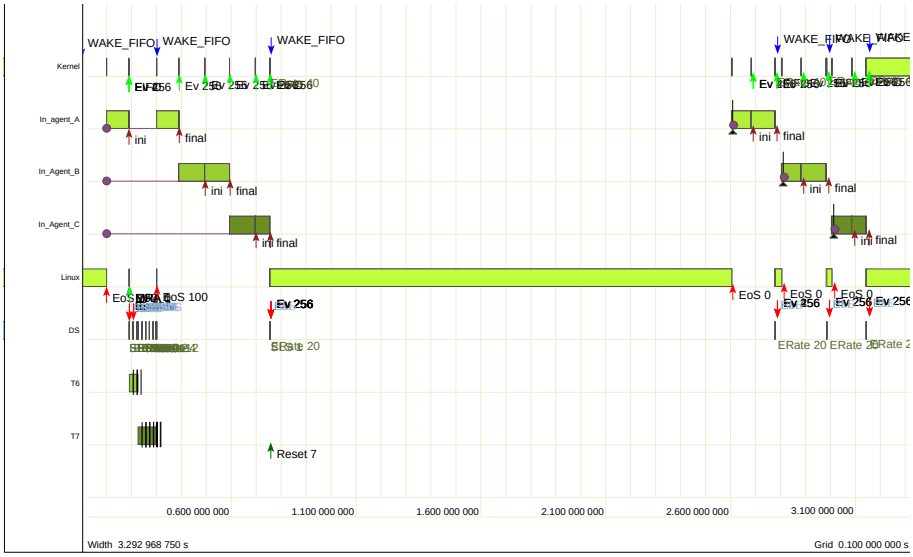


Figure 7. Execution 1 during 4 seconds

1. Activation of the task *In_Agent_B*. This task may be executed from this instant till its deadline.
2. Deadline of the task *In_Agent_B*.
3. Execution of the task *In_Agent_C*. These rectangles indicate the time when the different tasks are being executed.
4. The execution of the optional task *T8* is interrupted. Later, if the DS decides it so, its execution would be resumed (if there is available time before its deadline).

The most important aspect to notice in the chronogram of the figure 7 is a total change of the *Reactivity Degree* of the agent. Thus, during the execution of the optional parts of the first task, the agent detects an emergency situation (the level of one of the sewage tanks has surpassed the upper allowed limit), and then the *Reactivity Degree* is changed to 0. This change is made by means of the corresponding Meta-Rule that is activated by a modification in the tank level. The change makes the $\mathcal{AA}$ to react immediately (opening outgoing valves and closing implied taps). Like it can be observed, this makes that in the rest of the execution no more optional parts of the tasks are executed, since the finishing parts are executed immediately after the initial parts.

In the figure 9 it can be seen the same example as before, same duration also, but where the *Reactive Degree* goes to 50 % instead of 0, and then the variation of the execution of the other tasks can be observed, also the DS and the levels chosen by it for execution. This *Reactivity Degree* change is made by another Meta-Rule at which condition part checks that the level of the tank is in a range that recommends of not using all the available slack, but is enough to use 50 %.

This example also illustrates the empower that having a variable *Reactivity Degree* does to the $\mathcal{AA}$, allowing to adjust the time dedicated to deliberate about the current situation by using also the Meta-Rules. With these extensions, the $\mathcal{AA}$ manages to face new situations changing its behaviour very quickly, as seen in the emergency example changing to 0 the *Reactivity Degree*, making the agent answer as quick as possible. Moreover, it may adjust its behaviour not only to critical changes (emergency), but to any significant change chosen by the designer.

6 Conclusions

The present paper presents a new approach to the reactivity concept within the agent paradigm. In this approach, the reactivity is defined as a degree instead of a feature. This allows to define different reactivity degrees having different ways of reacting to the environment. This approach also allows to dynamically change the reactivity degree of an agent to adapt to significant changes in the environment.

This approach can be considered of greater importance when speaking of hard real-time agents. It has been implemented in a hard real-time agent architecture (ARTIS agent) and its increase in flexibility and adaptiveness has been checked.

Currently, the ARTIS agent architecture is being applied to other examples, including a mail-delivering robot in an office building.

References

[1] R. C. Arkin. *Behavior-Based Robotics*. The MIT Press, 1988.

[2] F. Barber, V. Botti, E. Onaindía, and A. Crespo. Temporal reasoning in reakt: An environment for real-time knowledge-based systems. *AICOMM*, 7(3):175–202, 1994.

[3] V. Botti, C. Carrascosa, V. Julián, and J. Soler. Modelling agents in hard real-time environments. In *MAAMAW'99 Proceedings*, volume 1647 of *LNAI*, pages 63–76. Springer-Verlag, 1999.

[4] C. Carrascosa, M. Rebollo, V. Julián, and V. Botti. Deliberative server for real-time agents. In *Multi-Agent Systems and Applications III: 3rd International Central and Eastern European Conference on Multi-Agent Systems, CEEMAS 2003*, volume 2691 of *LNAI*, pages 485–496. Springer, 2003.

[5] A. García-Fornes, A. Terrasa, V. Botti, and A. Crespo. Analyzing the schedulability of hard real-time artificial intelligence systems. *Engineering Applications of Artificial Intelligence*, pages 369–377, 1997.

[6] R. P. Goldman D. J. Musliner, and K. D. Krebsbach. Managing online self-adaptation in real-time environments. In *Proc. of Second International Workshop on Self Adaptive Software*, Balatonfured, Hungary, 2001.

[7] D. Musliner, E. Durfee, and K. Shin. Circa: a cooperative intelligent real-time control architecture. *IEEE Transactions on Systems, Man and Cybernetics*, 23(6), 1993.

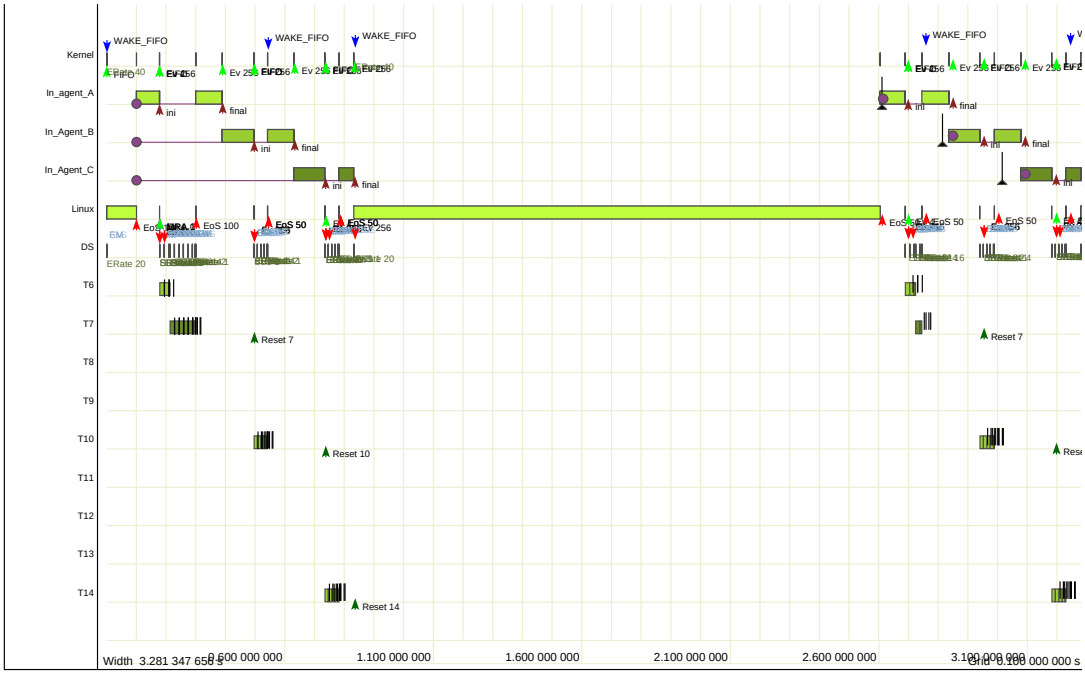


Figure 9. Execution 2 during 4 seconds

[8] D. J. Musliner. Safe learning in mission-critical domains: Time is of the essence. In *Working Notes of the AAAI Spring Symposium on Safe Learning Agents*, Stanford, California, 2002.

[9] P. Nii. Blackboard systems: The blackboard model of problem solving and the evolution of blackboard architectures. *The AI Magazine*, pages 38–53, Summer 1986.

[10] A. Raja and V. Lesser. Real-time meta-level control in multi agent systems. In *Proceedings of Multi-Agent Systems and Applications - ACAI 2001 and EASSS 2001 Student Sessions. Also Adaptability and Embodiment Using Multi-Agent Systems: AEMAS 2001 Workshop. Prague, Czech Republic*, 2001.

[11] J. V. Real. *Protocolos de Cambio de Modo Para Sistemas de Tiempo Real*. PhD thesis, Departamento de Informática de Sistemas y Computadores. Universidad Politécnica de Valencia, Enero 2000.

[12] S. Russell. Metareasoning. In The MIT Encyclopedia of the Cognitive Sciences, MIT Press, 1998.

[13] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall International Editions, 1995.

[14] J. Soler, V. Julián, C. Carrascosa, and V. Botti. Applying the artis agent architecture to mobile robot control. In *Proceedings of IBERAMIA'2000. Atibaia, Sao Paulo, Brasil*, volume I, pages 359– 368. Springer Verlag, 2000.

[15] J. Stankovic. Misconceptions about real-time computing. *IEEE Computer*, 12(10):10–19, 1988.

[16] A. Terrasa, A. García-Fornes, and V. Botti. Flexible real-time linux. *Real-Time Systems Journal*, 2:149–170, 2002.

[17] M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.

Extended Behavior Networks for Behavior Selection in Dynamic and Continuous Domains

Klaus Dorer[1]

Abstract. In this paper we present how behavior networks can be extended to model behavior selection of agents in dynamic and continuous domains. More precisely, the focus is on a mechanism for selection of concurrent behaviors by explicit representation of resources a behavior makes use of. Further it describes how the behavior selection process can be coupled with behavior execution in continuous domains. Behaviors may be influenced by the decidedness of the behavior selection as is the case in biological systems. Empirical results in the RoboCup domain show that both extensions improve the performance of soccer playing agents significantly.

1 INTRODUCTION

Behavior selection in dynamic domains is complicated by the fact that the deciding agent has limited amount of time for its decision before the situation has changed. This is usually addressed by improving the speed of the decision mechanism for dynamic domains. However, this does not take into account the possibility to improve the agent's performance by conducting multiple actions concurrently. Moreover, in some domains, concurrent actions are not simply a way of improving agents' performance, but they can become a necessary condition to perform tasks. For driving a car, for example, it is at least necessary to turn the steering wheel and accelerate or break concurrently. Most action selection mechanisms result in a single action to be performed. To overcome this limitation two possibilities exist: (1) either the decision mechanism is provided with a (usually huge) set of combined actions like 'turnLeft', 'turnLeftAndBreak', 'turnLeftAndAccelerate', etc., or (2) the decision mechanism decides on more complex behaviors like 'drive' that combine actions appropriately leaving the detailed decision to the execution module of the agent. The later is usually the preferred option accepting the disadvantage of more complex behaviors.

This situation is even complicated in continuous domains, where actions may be performed with variable strength, degree, duration. For example, the 'turnLeft' action of the above example would have to be split into 'turnLeft5Degrees', 'turnLeft10Degrees', ... Again it is usually preferred to put the decision of the degree with which an action is performed into the low level behavior execution module. Behavior selection and behavior execution is usually strictly separated.

Extended behavior networks [2, 3] (EBNs) are a means to carry out behavior selection in dynamic and continuous domains. They extend original behavior networks [5, 6, 7] by explicit representation of goals with dynamic, i.e. situation-dependent, utility function and by the introduction of continuous state-propositions to represent attributes of continuous domains. In this paper we describe how EBNs are able to select multiple behaviors in a single decision cycle to be performed concurrently. We also show how the decidedness of behavior selection can be used to control the intensity with which behaviors are performed as is the case in biological systems [4].

The remainder of this paper is organized as follows: Section 2 describes the basic concept of behavior selection using extended behavior networks. In section 3 this concept is extended by introducing concurrent behavior selection. Section 4 explains how behaviors can be parametrized by the decidedness of the behavior selection. In section 5 we summarize empirical results gained in the RoboCup simulated soccer domain. Finally, in section 6 we discuss possible future work directions before concluding.

2 EXTENDED BEHAVIOR NETWORKS

Extended behavior networks [5, 2, 3] have been introduced to combine reactive and goal-directed behavior selection in dynamic and continuous domains. This section gives a short overview on the structure of extended behavior networks and the behavior selection mechanism using activation spreading. The next two sections will then describe two further extensions of EBNs, selection of concurrent behaviors and behavior parametrization, to improve action selection in dynamic and continuous domains.

2.1 Network Definition

Extended behavior networks consist of goals and so called competence modules that are linked into a network.

Definition 1 *A* goal *consists of a tuple (*GCon, ι, RCon*) with*

- GCon *the* goal condition *(conjunction of propositions, i.e. possibly negated atoms), the situation in which the goal is satisfied,*
- $\iota \in [0..1]$ *the (static)* importance *of the goal,*
- RCon *the* relevance condition *(conjunction and disjunction of propositions), i.e. the situation-dependent (dynamic) importance of the goal.*

Definition 2 *A* competence module *consists of a tuple (*Pre, b, Post, a*) with*

- Pre *the* precondition *and* $e = \tau_P(Pre, s)$ *the* executability *of the competence module in situation* s *where* $\tau_P(Pre, s)$ *is the (fuzzy) truth value of the precondition in situation s;*
- b *the* behavior *that is performed once the module is selected for execution;*

[1] Living Systems GmbH, Humboldtstrasse 11, D-78168 Donaueschingen Germany, email: kdorer@living-systems.com

- Post *a set of tuples (Eff, ex), where Eff is an expected effect (a proposition) and $ex = P(Eff|Pre)$ is the* probability *of Eff getting true after execution of behavior b,*
- a *the* activation $\in \mathbb{R}$*, representing a notion of the expected utility of the behavior (see below).*

Definition 3 *An* extended behavior network *(EBN) consists of a tuple $(\mathcal{G}, \mathcal{M}, \Pi)$, where $\mathcal{G}$ is a set of goals, $\mathcal{M}$ is a set of competence modules and Π is a set of* parameters *that control activation spreading (see below)*

- $\gamma \in [0..1[$ *controls the influence of activation of modules,*
- $\delta \in [0..1[$ *controls the influence of inhibition of modules,*
- $\beta \in [0..1[$ *the inertia of activation across activation cycles,*
- $\theta \in [0..\hat{a}]$ *the activation threshold that a module has to exceed to be selected for execution, with $\hat{a}$ the upper bound for a module's activation,*
- $\Delta\theta \in]0..\theta]$ *the threshold decay.*

2.2 Behavior Selection

The decision of which behavior to adopt should be based on the the expected utility out of executing such behavior. In EBNs, the expected utility of a behavior is approximated by a mechanism called *activation spreading*. The competence modules are connected to the goals and other competence modules of the network. Across those links activation is spread from the goals to the competence modules and among competence modules.

A competence module receives *activation* directly from a goal if the module has an effect that is equal to a proposition of the goal condition of that goal. The amount of activation depends on the probability ex of the effect to come true and the utility of the proposition in the goal condition. Activation from a goal represents the expected utility of the behavior to reach that goal. The utility of propositions that are part of a goal condition can be directly derived from the importance and relevance of the goal [2].

A competence module is *inhibited* by a goal if it has an effect proposition that is equal to a proposition of the goal condition and one of the two propositions is negated. Inhibition represents negative expected utility and is used to avoid the execution of behaviors that would lead to undesired effects.

A competence module x is linked to another competence module y if x has an effect that is equal to a proposition of the precondition of y. y is called a *successor* module of x. Module x gets activation from the successor the amount of which depends on the utility of the precondition and the probability of the effect to come true. The utility of propositions that are not part of a goal condition is not available directly. It can be determined indirectly using the activation of the containing module and the truth value of the proposition [2]. In this way, unsatisfied preconditions get implicit sub-goals of the network. Their utility directly depends on the utility of the competence module itself.

Finally a competence module x is linked to another competence module y if it has an effect that is equal to a proposition of the precondition of y and one of the two propositions is negated. y is called a *conflictor* of x, because it has an effect that destroys an already satisfied precondition of x. Again, a conflictor link from x to y is inhibiting (negative activation) to avoid undesired effects.

The activation of a module k at time t is then the sum of all incoming activation and the previous activation of the module decayed by β (defined in the set of parameters Π):

$$a_k^t = \beta a_k^{t-1} + \sum_i a_{kg_i}^t, \tag{1}$$

where $a_{kg_i}^t$ is the maximal activation module k receives at time t from goal g_i to which the module is linked directly or indirectly across incoming successor and conflictor links of other competence modules. For more details on activation spreading see [2, 3].

Behavior selection is done locally in each competence module in a cycle containing the following steps:

1. Calculate the activation a of the module.
2. Calculate the executability e of the module.
3. Calculate the execution-value $h(a, e)$, which is a monotonically increasing function of the activation and executability of a module (calculated e.g. by multiplication) [2].
4. If the highest value $h(a, e)$ of all competence modules lies above a threshold θ (defined in the set of parameters Π), execute the corresponding competence module's behavior b, reset θ to its original value in Π and go to 1.
5. Otherwise reduce θ by $\Delta\theta$ (also defined in Π) and go to 1.

In the first cycle of activation spreading, only competence modules that directly have links to goals get activation. Activation by successor and conflictor links is zero at that time, because no module has activation initially. So only behaviors that directly satisfy a goal will be taken into account for selection. In the second cycle also competence modules get activation that may reach the goal within two actions. They got activation through successor and conflictor links to modules that got activation in the first cycle. The more cycles activation is spread the longer is the (timely) horizon of action sequences taken into account that lead to goals. This cyclic approximation of expected utility of a behavior in EBNs is somewhat similar to a growing horizon when solving a Markov Decision Process (see e.g. [1]). For behavior selection a good trade-off is therefore necessary between running enough activation spreading cycles to look far enough into the future and acting fast enough.

3 CONCURRENT BEHAVIOR SELECTION

A shortcoming of the above described mechanism for behavior selection is that behavior selection results in a single behavior to be performed at any time. Humans on the other side are able to performed well trained behaviors concurrently if they do not use the same resources [10, 8]. A typist, for example, is able to type a text she is reading and speak aloud a text she is listening to at the same time [10]. Performing behaviors that use the same resources usually ends with no behavior performed successfully. For instance, when a human is undecided between the words 'close' and 'shut' it may end up pronouncing a non existing word 'clut' [9]. The common resource 'language processing' may not be used by multiple behaviors. It may, however, be influenced by multiple goals.

Sequential behavior selection of Maes networks [5] avoids the problem of resource conflicts. The disadvantage is on the one side a reduced performance in domains where multiple behaviors may be performed in parallel. On the other side it may prevent the completion of tasks completely for which concurrent behavior execution is essential (like car driving).

To perform multiple behaviors in parallel the agent needs knowledge about the resources used by the behaviors. The definition of competence modules and extended behavior networks has therefore to be extended with the notion of resources.

Let $\mathcal{R}$ be the set of all resources and $\tau_R : \mathcal{R} \times \mathcal{S} \rightarrow \mathbb{R}^+$ a function that assigns to each element of $\mathcal{R}$ an amount of available resources in the domain in state s. The function $\tau_U : \mathcal{M} \times \mathcal{R} \times \mathcal{S} \rightarrow \mathbb{R}^+$, with $\mathcal{M}$ the set of all competence modules, defines the expected amount of resource units used by the corresponding competence module in state s to reach its effects.

Definition 4 *A* resource node *is a tuple (res, g, θ_{Res}) with*

- $res \in \mathcal{R}$ *the* resource *represented by the node,*
- $g \in \mathbb{R}^+$ *the amount of* bound resource units, *i.e. units that are bound by a currently executing competence module and*
- $\theta_{Res} \in]0..\theta]$ *the resource specific* activation threshold *(where θ is the global activation threshold of the network).*

The definition of a competence module can then be extended to:

Definition 5 *A* competence module k *consists of a tuple (*Pre, b, Post, Res, a*) with* Pre, b, Post *and a as defined above and* Res *is a set of* resources $res \in \mathcal{R}$ *used by behavior b. $\tau_U(k, res, s)$ is the situation-dependent amount of resource units expected to be used by behavior b.*

Definition 6 *An* extended behavior network *EBN consists of a tuple ($\mathcal{G}, \mathcal{M}, \mathcal{U}, \Pi$), where $\mathcal{G}$ is a set of goals, $\mathcal{M}$ a set of competence modules, $\mathcal{U}$ a set of* resource nodes *and* Π *a set of* parameters *(see section 2).*

To coordinate concurrent behaviors the competence modules of $\mathcal{M}$ are connected with resource nodes in $\mathcal{U}$. A competence module has for each resource $res \in Res$ a link to the corresponding resource node. This link enables the competence module to check the availability of the resource. Concurrent behavior selection may therefore be calculated locally in each competence module. It is done in a cycle containing the following steps:

1. Calculate the execution-value h of the module as described above.
2. For each resource res used by competence module k, starting with the previously unavailable resource
   (a) Check if h exceeds the activation threshold θ_{Res_i} of the corresponding resource node.
   (b) Check if enough resource units are available in the current situation, i.e. check if $\tau_U \leq \tau_R(res, s)$. If so, bind the resource-units, i.e. increase the number of used resource-units of the resource node by the number of expected units the behavior will use.
3. If all tests in 2 succeeded
   (a) Execute the corresponding behavior.
   (b) Reset the activation thresholds of all resources used.
4. Release all bound resource-units, i.e. reduce the number of bound resource units of the resource node by the number of previously bound units.
5. Repeat from 1.

The activation threshold θ_{Res_i} ensures that the competence module with highest execution-value will be performed. θ_{Res_i} linearly decreases over time so that eventually a module exceeds the threshold and may be performed. If modules have equal execution-values in a range of $\Delta\theta$, the threshold reduction, the module that first binds the resource is performed. If the execution of the module with highest activation value is prevented by a missing resource, another module with less activation not using the missing resource may be performed. Modules with a disjunct set of resources Res may be performed concurrently.

Besides allowing concurrent behavior selection, this algorithm overcomes another limitation of original behavior networks. Behavior selection has previously been done by selecting the most active executable competence module for execution. Unfortunately, this information can not be calculated locally in a competence module. Therefore, the process of action selection could not be calculated distributively in each competence module. By introducing resource nodes, a competence module is now able to perform action selection locally. All information is available within the node or within linked nodes. The information a competence module gets across a link to a resource node is the current activation threshold and the number of bound resource units. Information a resource node gets from a competence module using the resource includes the number of resource units to bind and release and when to reset activation threshold.

4 BEHAVIOR PARAMETRIZATION

Most decision mechanisms for agents only have influence on the decision which behavior the agent should perform, but not on the behavior execution itself. In biological systems, however, the determinedness of a decision has influence on the execution of a behavior. “Intensity and endurance of an activity is determined by the volition strength of the goal intention”[4]. Of course different intensities (i.e. strength/degree of execution) of the same basic behavior could also be modeled by distinguishing these as different behaviors and let the decision mechanism decide between those. Obviously, at least in continuous domains, this would increase the number of behaviors considerably making the decision process much more complex. Therefore it would be desirable if the determinedness of the agent’s decision would directly influence the execution of the behavior itself. The behavior ‘run to ball’ of a soccer agent, for example, could be more or less intens depending on the determinedness of the agent to run. The higher the expected utility and the executability of the behavior the more effective it should be to spend resources (stamina) on this behavior. An adequate measure for determinedness in extended behavior networks is the execution-value h of a competence module (see section 2.2). It reflects the expected utility for reaching the goals of the agent as well as the executability of the behavior with respect to the situation.

The problem of using the execution-value is that its absolute value depends on the goals defined in the behavior network. This is because h is a function of the sum of all activation received by the goals it is contributing to directly or indirectly. In an extreme case all effects of a behavior might be defined as goals resulting in a high execution-value. In another network, none of the effects might be defined as goals and the module only receives activation indirectly through other modules. A parametrized behavior on the other side should be independent on the specific network architecture. It is therefore necessary to normalize the execution-value adequately. Following we describe three approaches to map execution-values to the codomain of $[0..1]$.

One obvious approach to normalize the execution-value is to divide it by the number of goals $|\mathcal{G}|$ of the behavior network. However, $|\mathcal{G}|$ is not available within a competence module. A competence module only knows the number of goals it (directly or indirectly) receives activity from. Normalization by using division by the number of goals violates the locality principle and is therefore inappropriate.

Another approach is to use the maximal ($\hat{h}$) and minimal ($\check{h}$) execution-value. It can be calculated locally within a competence module. The influence parameter p of a module can then be calculated as

$$p = \frac{h - \check{h}}{\hat{h} - \check{h}} \tag{2}$$

where h is the current execution-value of the competence module. This approach, however, is vulnerable to extremely high or low execution-values.

This does not matter if instead of extreme values the distribution of execution-values is taken into account. Assuming that execution-values are normally distributed it is enough to calculate mean and standard deviation of the execution-values. Mapping execution-values to an influence parameter p is then done by

$$p = \begin{cases} 0 & : \quad h < \mu - k \cdot s \\ \frac{h-(\mu - k \cdot s)}{2k \cdot s} & : \quad \mu - k \cdot s \leq h \leq \mu + k \cdot s \\ 1 & : \quad \mu + k \cdot s < h \end{cases} \tag{3}$$

where k defines the range of the normal distribution that is mapped to the interval $[0..1]$. The calculation of μ and s can be done incrementally:

$$\mu_{n+1} = \mu_n + \frac{h - \mu_n}{n+1} \quad and \tag{4}$$

$$s^2_{n+1} = (n+1) \cdot (\mu_{n+1} - \mu_n)^2 + \frac{(n-1) \cdot s^2}{n} \tag{5}$$

Section 5.2 presents empirical results of behavior parametrization gained in the RoboCup domain.

5 EMPIRICAL RESULTS

Empirical tests have been conducted in the RoboCup simulated soccer environment. In this domain agents represent soccer players. Two Teams of eleven soccer players each play against each other in a simulated dynamic and continuous soccer domain.

The domain is dynamic from the perspective of a single agent, because 21 other agents change the domain without this agent doing anything. Also the decision cycle within which an agent has to decide is quite short (100ms). Within one decision cycle an agent may decide for concurrent actions. Dashing, kicking or turning the agent's body may be done concurrently with turning the agent's head and talking to other agents. The RoboCup domain is therefore quite well suited for testing concurrent behavior selection.

The domain is continuous in most of the underlying attributes. Examples are the position and velocity of players and the ball and the view and body direction of the agents. Also most actions of the agents are continuous. Dashing is done with variable strength, turning with continuous momentum and kicking with continuous strength and direction. This makes the RoboCup domain an ideal testbed for behavior parametrization.

5.1 Concurrent Behavior Selection

Section 3 explained how extended behavior networks are able to decide on multiple concurrent behaviors. This enables the agent to reach a goal faster or to pursue multiple goals at once. This should lead to improved behavior control of the agent especially in dynamic domains where success also depends on the time an agent needs to decide and act.

Since version 5 of the RoboCup-soccerserver, commands can be executed concurrently, if they do not use the same resources. A `say`-command, for example, can be executed concurrently with a `kick`-, `dash`-, or `turn`-command and a `turn_neck`-command. The concurrent execution of such actions should improve the speed and reactivity of an agent.

This has been examined in a series of 30 soccer-games. Two identical teams of 11 agents played against each other. The only difference was that one team used concurrent behavior selection, while the other team used serial action selection. For the serial team only the action with highest execution-value within a cycle was executed. The concurrent team's agents were able to execute communication, head turning and running or kicking actions concurrently. An example for competence modules the behaviors of which may be performed concurrently is shown in figure 1. τ_R has been defined independent of the situation as $\tau_{legs} = 2$, $\tau_{neck} = 1$ and $\tau_{mouth} = 1$. Since no commands using legs may be performed concurently, τ_U was set to 2 for all behaviors using legs.

The soccer agents turned their head in direction of the ball in case the ball left the visible area of the agent (mindBall). This way the agent can run in an angle of up to 135° relative to the ball and keeping it in the visible area. Without turning the head this would only be 45°. This is especially useful for all positioning behaviors. An agent is only able to run forward and backward in body direction. If, for example, an offender positions itself in the middle of the field while the ball is on the wing it can run towards the goal while keeping the head turned to the ball. An agent that runs and turns the head in consecutive cycles is much slower than an agent that is dashing each cycle and turns its head concurrently. Separate turning of the head relative to the body was performed in about 8% of all cycles. This is not surprising since turning the head is only necessary once the ball is close to leave the visible area.

Also the agents communicated to each other their position and positions of some other players (sayPosition). The number of cycles an agent can communicate is restricted to 4% of all cycles by the soccerserver to restrict the bandwidth of communication. Only one agent is allowed to say something every second cycle in the server version 7 used for the experiments. The agents used a simple round robin scheduling that effectively allowed an agent to talk each 22 cycles. Again the agents of the concurrent team were able to talk while running or kicking. The agents of the serial team only talked if that behavior had higher activation as all other behaviors.

Since separate turning of the head was done in 8% and communication in 4% of the simulation cycles, concurrent behavior selection effectively only took place in 2% of the cycles. Despite this, the team using concurrent behavior selection scored significantly[2] more goals than the team using serial behavior selection (see table 1).

	serial	parallel	p ($n = 30$)
Mean no of goals	2.4	4.3	< 0.001

Table 1. Comparison of serial and parallel behavior selection of EBNs in the RoboCup domain.

[2] two samples t-test with $\alpha = 0.01$.

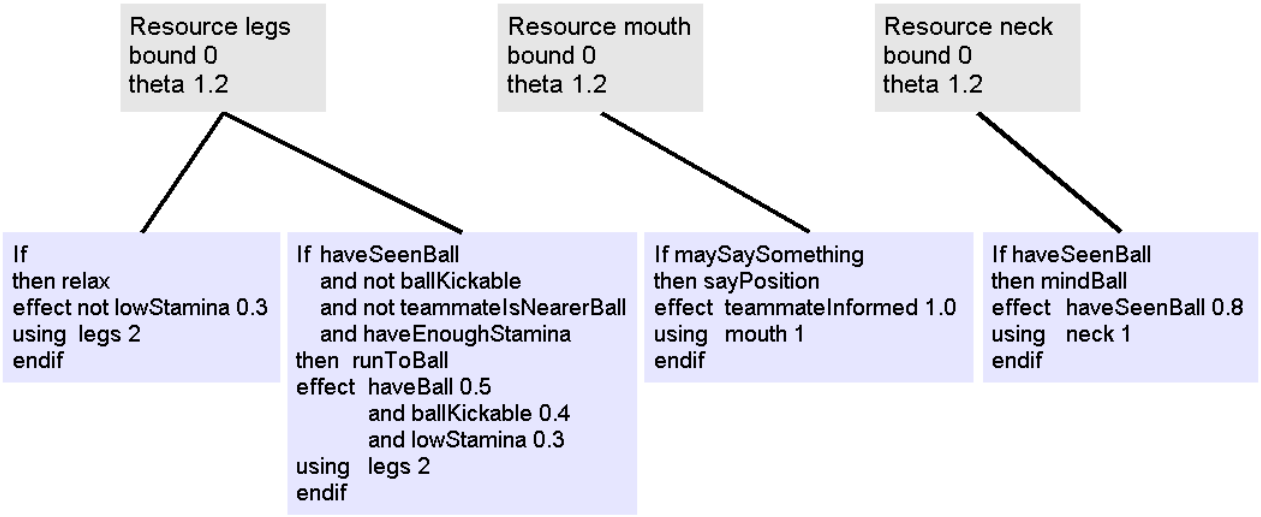


Figure 1. Parts of the network used for concurrent behavior selection in the RoboCup domain. Modules runToBall, sayPosition and mindBall may be performed concurrently. Modules relax and runToBall use the same resource legs and may not be performed concurrently.

5.2 Behavior Parametrization

In section 4 we described how the execution of behaviors may be influenced by the decidedness of the action selection. This can ensure that the execution of a behavior is more appropriate to the current situation. The intensity of behavior execution can be adjusted to the importance of the current situation. The usage of resources is focused to these situations.

These effects can be shown by experiments in the RoboCup domain. Agents have limited stamina for running on the soccer field. They have to make pauses in order to recover from running. If an agent runs out of stamina it gets very slow. The faster an agent runs the more stamina is consumed. For the experiments the 'run to ball' behavior has been parametrized. A normalized execution-value of 0.0 was translated to 60% dash power a value of 1.0 to a dash power of 100% with linear interpolation. Relevance conditions in the goals (see [2]) ensure that the decidedness in important situations like being close to one of the goals is high. This should ensure that the agent consumes less stamina in less important situations and has more stamina available in important situations.

5.2.1 Normalization of the Execution-Value

Section 4 explained the need for normalization of the execution-value. Two approaches have been mentioned that can be used for normalization without violating the principle of locality. One possibility is to store the minimal and maximal execution-values of a competence module and map it to the interval $[0..1]$ (MinMax). Another possibility is to calculate the mean execution-value μ and its standard deviation s (incrementally). Then a range of values from $\mu - k \cdot s$ to $\mu + k \cdot s$ can be mapped to normalized execution-values in the interval $[0..1]$ (distribution).

Since MinMax normalization is vulnerable to extreme values one would expect to get worse results with this approach. This was empirically evaluated in 30 games of 2 Robocup soccer agent teams. One team played with MinMax normalization the other team played with distribution normalization. Besides that both teams have been exactly identical. For the distribution normalization we chose $k = 1$. As shown in table 2, the team with distribution normalization scored significantly more goals than the team with MinMax normalization.

	MinMax	distribution	p ($n = 30$)
mean number of goals	4.2	6.0	0.008

Table 2. Comparison of the MinMax normalization and normalization using the distribution of values.

5.2.2 Comparison of Parametrized and Static Behavior

As mentioned above, parametrized behavior execution should improve the utilization of resource 'stamina' in the Robocup domain. This should improve the overall performance of a team measured by the number of goals scored. This can be verified by experiments running Robocup games where one team uses parametrized behaviors and the other does not (static). Normalization of execution-values was done using the distribution method. The parameter for the execution-value of the static team was constant during one game. It was varied in the interval $[0..1]$, however, for different series of games. In this way parametrized behaviors can be compared with growing static parameter values. The hypothesis is that for low static values the disadvantage of being too slow (e.g. to reach a ball) outweighs the advantage of being less tired. For high static values the disadvantage of fast exhaustion should outweigh the advantage of being faster at the ball.

First it is interesting to look at the number of pauses an agent takes during a game. This is a measure for the consumption of stamina of the agent. As expected the number of pauses of the static team grows with increasing parameter values (Fig. 2).

It is interesting to compare the two teams at the intersection of both curves at value 0.7. Although both teams' agents have to make

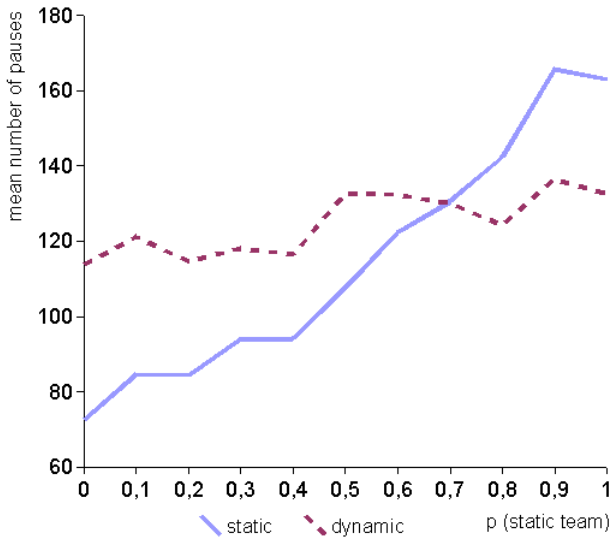


Figure 2. Mean number of pauses of the static and parametrized team

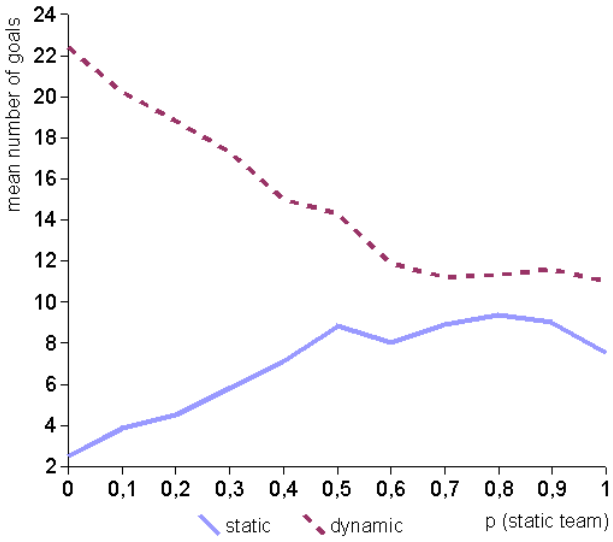


Figure 3. Mean number of goals of the static and parametrized team

the same number of pauses on average, the team with parametrized behaviors scored significantly more goals (Tab. 3). Although the average usage of resources of both teams is equal the team with parametrized behaviors makes more use out of it. It uses the resources in situations in which the goals of the agent are more relevant. In such situations the execution-values of behaviors directed towards such goals are higher.

$p_{statickteam} = 0.7$	static	parametrized	p ($n = 45$)
mean scored goals	8.9	11.2	0.003
mean number of pauses	130.6	130.2	0.950

Table 3. Comparison of the mean number of goals and pauses of players of static (parameter $p = 0.7$) and parametrized behavior execution.

The comparison of scored goals for the static and parametrized team shows significantly better results for the parametrized team for all parameter values used for the static team (Fig. 3).

6 CONCLUSION

In this paper, we describe a mechanism that can be used for an agent to select multiple actions to be performed concurrently using extended behavior networks. The concurrent action selection mechanism is calculated distributively in the competence modules (nodes) of the EBN. Conflicts between actions are moderated by resource nodes that are explicitly represented in the EBNs. In addition, we introduce a mechanism for EBNs to influence behavior execution using the execution-value of a competence module as a measure of the decidedness of the agent to perform the action. Both extensions improved the performance of agents in the RoboCup simulated soccer domain significantly.

Future work will mainly have to examine if these results generalize to other dynamic and continuous domains. Especially domains will be interesting, where the amount of available resources depends on the current situation. The stamina resource in the RoboCup domain that resembles how much 'energy' is left for dashing can not be used in this sense, because although enough stamina would be available for different behaviors the server does not allow concurrent dashing behaviors.

Also it would be interesting to examine the stability of the proposed concurrent behavior selection in cases where the estimated amount of resources used by a competence module's behavior may differ from the effectively used resources. The behavior selection itself should still work in such occasions, the performance of the agent, however, is expected to decrease.

References

[1] C. Boutilier, T. Dean, and S. Hanks, 'Decision–theoretic planning: Structural assumptions and computational leverage', *Journal of Artificial Intelligence Research*, **11**, 1–94, (1999).
[2] K. Dorer, 'Behavior networks for continuous domains using situation–dependent motivations', *Proceedings of the Sixteenth International Conference of Artificial Intelligence*, 1233–1238, (1999).
[3] K. Dorer, *Motivation, Handlungskontrolle und Zielmanagement in autonomen Agenten*, PhD thesis, Albert-Ludwigs University, Freiburg, 2000.
[4] H. Heckhausen, *Motivation und Handeln*, Springer, Berlin, 1989.
[5] P. Maes, 'The dynamics of action selection', *Proceedings of the International Joint Conference on Artificial Intelligence*, 991–997, (1989).
[6] P. Maes, 'How to do the right thing', *Connection Science Journal*, **1**(3), (1990).
[7] P. Maes, 'Situated agents can have goals', *Journal for Robotics and Autonomous Systems*, **6**(1), 49–70, (1990).
[8] D. Navon and D. Gopher, 'On the economy of the human processing system', *Psychological Review*, **86**(3), 214–255, (1979).
[9] D. A. Norman, 'Categorization of action slips', *Psychological Review*, **88**(1), 1–15, (1981).
[10] L. H. Shaffer, 'Multiple attention in continuous verbal tasks', in *Attention and Performance V*, eds., P. M. A. Rabbit and S. Dornic, Academic Press, New York, (1975).

Abstract World for Opportunistic Local Decisions in Multi-Agent Systems Using Bayesian Knowledge Bases[1]

Solomon Eyal Shimony and **Ami Berler** [2]

Abstract. Collaboration of multiple intelligent agents on a shared task is a complex research issue, which has numerous important applications, such as battle-field simulation, web-based agents, and AI in games. The problems addressed are particularly difficult when communication is limited or impossible. A common solution in multi-agent systems is to commit a team of collaborating agents to a joint plan. Since any deviation from the plan by an agent is hazardous, these solutions face up to potential unplanned "opportunistic" actions by ignoring them, or by ad-hoc rules determining whether to accept such opportunities.

Since neither of these solutions is desirable, we developed AWOL[3] (Abstract World for Opportunistic Local decisions), an abstract framework with a disciplined treatment of opportunistic action, in the context of an existing joint plan.

The idea is to model the (stochastic) tradeoff of opportunism vs. continued commitment to the joint plan, while abstracting away from the state of the world. The abstract model is evaluated using strict decision-theoretic criteria, with the goal of applying the optimal decision on whether to accept an opportunistic action in the original domain.

When this abstract domain is modeled as an Markov Decision Process (MDP) (and to an even greater extent, a Partially Observable MDP (POMDP), the complexity of finding an optimal decision, although many orders of magnitude lower than in a real or simulated domain, is still high.

In order to reduce this complexity, we implement a compact, context-specific independence representation for the transition probabilities. Our representation uses rules, in a probabilistic model known as Bayesian Knowledge Bases (BKB). Since the latter are a compactly-represented generalization of Bayes Networks (BN), our scheme should have representation size and complexity advantages in representing distributions in the respective decision problem.

1 INTRODUCTION

Collaboration of several intelligent agents on a shared task is a complex research issue. Even when there is a global team utility, the environment may force decisions to be decentralized due to limited communication and uncertainty about the environment. Effective agent interactions in such domains raise various research challenges, in addition to the traditional single agent systems.

A common solution in multi-agent systems is to commit a team of collaborating agents to some form of prior joint commitment, such as Joint Plans [3, 4] and Shared Plans [9, 8, 7].

Stone and Veloso [17] presented "Set-Plays", as part of the locker room agreement - these are multi-step, multi-agent plans for execution in *specific* situations. However, an agent which acts in the context of a pre-plan, may encounter opportunities that have not been previously considered in the team plan or even contradict it. Although taking advantage of unexpected opportunities occurring during plan execution is possible, none of the existing mechanisms are intended to handle such beneficial occurrences. The practical solution in systems involving joint commitments are either to ignore opportunities, or to use ad-hoc rules for when to break a commitment.

Yet clearly neither solution is desirable, there should be a way to compute, or at least approximate, expected team utility for these actions, and act based on the result. Naturally, in evaluating the expected utility of an opportunistic action one should take into account undesirable possible consequences of opportunism, such as confusing the other team-mates, etc.

This paper is an initial attempt to introduce opportunism, in a disciplined manner, by aiming at a formal estimate of the expected utility of applying unplanned opportunistic actions, as suggested above. In order to focus specifically on plans and opportunism, we present the AWOL framework, that abstracts away from aspects of the environment that are irrelevant to the issue of opportunistic actions.

The framework is a stochastic model, consisting of a set of joint plans, where each plan is a sequence of joint steps. Each joint step assigns a role to each agent, and an agent action is selection of a plan.

Although we wish to abstract away from the environment, we still need to model unplanned opportunities that may appear during plan execution. This is done by introducing "opportunity variables" that changes state randomly - which we call "dummy agents" in our framework.

Currently, we are examining the impact of opportunism in the absence of communication (although communication can be handled in AWOL), except at the initial stages when the initial joint plan(s) is established. Quality of the resulting decisions and their sensitivity are measured for simulation runs as the following parameters are modified: probability of success of opportunistic plans, the way each agent models its team-mates' behavior, and distribution of roles within plans.

While the type of approximations we examine above are not necessarily new, they are novel, as far as we know, within the framework of opportunistic actions under joint plans. Obviously, computing expected utility in multi-agent systems is non-trivial, and in fact not always well defined [2]. Additionally, the environment in applications further complicates treatment of this issue. Although several

[1] We acknowledge the support of the Lynn and William Frankel Center for Computer Sciences and the partial support of the Paul Ivanier Center for Robotics and Production Management at BGU.

[2] Department of Computer Science Ben-Gurion University of the Negev email: shimony@cs.bgu.ac.il ; ami@cs.bgu.ac.il

[3] Unlike the military term AWOL (Absent WithOut Leave), here the agent defaults only in order to *increase* (expected) team utility.

strong commitments were made above about the state space, actions, etc. the domain is still too expansive to be able to define meaningful empirical evaluation, even when we make several further restrictions that allow us to specifically focus on the issue of opportunism in the context of plans. Another problem we encountered is that the computational complexity of the AWOL model itself is still high.

In our implementation of AWOL, we make use of a context-specific independence representation of the transition probabilities. We introduce the use of Bayesian Knowledge Bases (BKB) [14, 15], a ruled-based probabilistic model that extends the Bayes Networks model [11]. We believe that this representation should make solution of the decision problem more efficient. There are other models which exploit context-specific independence in probabilistic reasoning [12, 13]. Nevertheless, our use of BKB, to represent transition probabilities in decision models, is novel, as far as we know, within the scope of multi-agent systems.

The rest of the paper is organized as follows: Section 2 describes the AWOL framework, and the specific choices made within the framework. Section 3 introduces the BKB model and its implementation in our framework. Section 4 presents the design of several experiments on the model, and some preliminary results. We conclude with a discussion of related work, and future research.

2 THE AWOL FRAMEWORK

In most applications, the complexity of the domain, number of agents, a space state too large and the incomplete information about the team members result in a problem impossible to analyze formally. In order to support such analysis, we construct the AWOL (Abstract World for Opportunistic Local decision in multi-agent systems) abstract environment. In addition, we make provisions to perform experiments in the abstract domain. To that end, AWOL consists of two mechanisms:

1. A decentralized control problem, where each agent receives an observation and subsequently decides about the next action.
2. A problem generator and simulator that allows us to set up parameters for an instance of AWOL, apply various solution methods to the control problem, and evaluate the results.

The control problem is a Markov process, controlled by several agents, with partial observability and (in the basic AWOL model) no communication.

2.1 Definition of the control problem

In the general framework the domain is modeled as a tuple: $(\mathcal{G}, \mathcal{S}, \Psi, \mathcal{A}, \Gamma, \mathcal{O}, \mathcal{T}, \mathcal{R})$ where: $\mathcal{G}$ is the set of *N* agents that act in the domain, $\mathcal{S}$ is the abstract "plan-state" space, Ψ is the set of *M* plans in the domain, $\mathcal{A}$ is the set of abstract actions in the domain, Γ is a finite set of observations, $\mathcal{O}$ is a table of observation probabilities, $\mathcal{T}$ is a transition distribution, and $\mathcal{R}$ is a reward function.

The **State of the world** S is an N-tuple consisting of the agents states: $S = (s[1], ..., s[N])$ where the **State of an agent** g is: $S[g] = (i, k)$, with i an index into the set of plans Ψ, and k the step in the plan.

A plan is a function assigning to each agent a *role* at each step in the plan. Formally, if $\mathcal{F}$ is the set of roles, a plan $\psi \in \Psi$ is a function: $\psi : \mathcal{N} \times \mathcal{G} \longrightarrow \mathcal{F}$

Abstract agent **action** selects a plan $i \in \Psi$, and **joint action** A is a tuple $(A[1], ..., A[N])$ of agent actions.

Each agent receives information about its teammates by **observation**. In our model the observed variables are just the individual state of the agents, subject to noise. Since there are only two observed variables for each agent, the identifier plan number and the step, the observation function $\mathcal{O}$ in our abstract domain is relatively simple: $\mathcal{O} : \mathcal{S} \times \mathcal{S} \rightarrow [0,1]$.

The **Transition distribution** is defined in the general case as: $\mathcal{T} : \mathcal{S} \times \mathcal{A}^N \times \mathcal{S} \rightarrow [0,1]$. In our domain, the roles of the teammates in the plans are implicit in the state, and are critical in defining the actual transition probabilities, as shown below.

The **reward function** $\mathcal{R}$ in our framework depend only on the state of world, and within the state on the roles of the agents in the active plans.

2.2 Assumptions for the test-bed

In order to build a disciplined empirical test-bed, we need further restrictions. We also explain our choice for state variables made above.

For simplicity, we will be assuming that the agents are *synchronized*, i.e. if they are executing the same plan, they are also in the same step (how to achieve that in a real environment is not necessarily trivial, we are making this assumption as achieving step synchronization is beyond the scope of this paper). For each world state S_y, we denote by $\Psi_y \subseteq \Psi$ the set of plans active in this state, i.e. the set of plans executed by at least one agent.

Failure of any sort in executing a plan is represented by forcing all "failing" agents to execute an especially introduced *null* plan, which has only one step called $\emptyset$, and from which there is (usually) no way out. Usually, states with team members in the null plan have very low rewards.

In order to focus specifically on opportunism and further simplify the analysis, we actually limit each agent actions to two possibilities: attempting to remain in the same plan as in the current state, and the *opportunistic* action of attempting to select a different plan from the one the agent is executing in the current state. (In this paper, we actually do not allow the agent to select between different "opportunistic" actions - each agent g will have its own *single, predefined* "opportunistic" plan $o(g)$, which it selects whenever it attempts not to follow the current plan). Additionally, we assume that an agent already executing an opportunistic plan cannot move to any other plan (except for the null plan).

When the agent action is to attempt to stick with its current plan, there are two possible outcomes: one, the action succeeds (the agent moves to the next step in the plan), or two, it fails (the next state for the agent is the Null plan).

Likewise, when an agent action is opportunistic (select new plan i), the possible results are: one, the action succeeds (the agent's next state being the new plan), otherwise, the action fails: the agent next state is the Null plan. However, in this case the current plan does not necessarily fail (however, it usually has a higher probability of failing due to one or more of the agents defaulting on the joint commitment).

As we are ignoring the issue of synchronization, if there are other agents executing step k of plan i in the current state, the next agent state will be $(i, k + 1)$ (the same as the other agents executing plan i)

2.3 The transition distribution

With the above assumptions, we can now write down the form of the transition distribution. We will need to introduce some notation,

denoting specific sets of agents, as a function of the current state (denoted S_{old}) and next state (denoted S_{new}), when executing a joint action A. First, $\mathbf{G_r}$ denotes the agents that try to remain in their current plan i when the team is in state S and the joint action is A:

$$\mathbf{G_r}(i, S, A) \equiv \{g | S[g] = (i, k), A[g] = i\}$$

where $A[g]$ denotes the action by agent g in the joint action A. Likewise $\mathbf{G_0}$ denotes the set of agents executing plan i in state S and attempt an opportunistic action:

$$\mathbf{G_0}(i, S, A) \equiv \{g | S[g] = (i, k), A[g] \neq i\}$$

The set of agents that attempt to execute an opportunistic action and succeed is denoted:

$$\mathbf{G_{0_s}}(S_{new}, A, S_{old}) =$$

$$\{g | \exists i, c \; S_{old}[g] = (i, k) \wedge S_{new}[g] = (A[g], c)) \wedge A[g] \neq i\}$$

The set of agents that attempt to remain in their current plans is denoted:

$$\mathbf{G_{r_s}}(S_{new}, A, S_{old}) =$$

$$\{g | S_{old}[g] = (A[g], k) \wedge S_{new}[g] = (A[g], k + 1)\}$$

The set of agents that attempt to execute an opportunistic action and fail (and thus land in the Null plan) is denoted:

$$\mathbf{G_{0_f}}(S_{new}, A, S_{old}) =$$

$$\{g | \exists i \;\; S_{old}[g] = (i, k) \wedge S_{new}[g] = (Null, \emptyset) \wedge A[g] \neq i\}$$

The set of agents that attempt to continue in their current plan but fail (and thus land in the Null plan) is denoted:

$$\mathbf{G_{r_f}}(S_{new}, A, S_{old}) =$$

$$\{g | S_{old}[g] = (A[g], k) \wedge S_{new}[g] = (Null, \emptyset)\}$$

As a shorthand, we omit the arguments in the last four functions e.g. we use $\mathbf{G_{r_f}}$ to denote $\mathbf{G_{r_f}}(S_{new}, A, S_{old})$.

We further assume that dependence exists only between the agents that are in the same $\mathbf{G_r}$ set, and that the transition distribution for the rest of the agents is independent of their teammates. The transition distribution under the above assumptions is the following:

$$p(S_{new} | A, S_{old}) = \prod_{i \in \Psi_{old}} \mathcal{P}_i \prod_{g \in \mathbf{G_0}(i,S,A)} \mathcal{P}_g \qquad (1)$$

where Ψ_{old} is the set of (non-null, non-opportunistic) plans being executed by some agent in state S_{old}, the product over a null set is 1 by convention, and:

$$\mathcal{P}_i = \begin{cases} p_{succ\_cont}(i, S_{old}, A) & \mathbf{G_r}(i, S_{old}, A) \subseteq \mathbf{G_{r_s}} \\ 1 - p_{succ\_cont}(i, S_{old}, A) & \mathbf{G_r}(i, S_{old}, A) \subseteq \mathbf{G_{r_f}} \\ 0 & Otherwise \end{cases}$$

$$\mathcal{P}_g = \begin{cases} p_{succ\_opp}(g, S_{old}, A) & g \in \mathbf{G_{0_s}} \\ 1 - p_{succ\_opp}(g, S_{old}, A) & g \in \mathbf{G_{0_f}} \\ 0 & Otherwise \end{cases}$$

where $p_{succ\_cont}(i, S, A)$ is a function that represents the the contribution of the set $\mathbf{G_r}$ of agents participating in plan i, to the probability that the plan will successfully move to the next step. Likewise $p_{succ\_opp}(g, S, A)$ represents the probability that an agent making an opportunistic will successfully begin to execute an opportunistic plan.

3 USING BAYESIAN KNOWLEDGE-BASES

In order to simplify decision problems, Dean and Wellman [5], use the well-known Bayes Networks (BN) to decompose the transition probability tables into much smaller conditional probability tables (CPT), using conditional independence assumptions.

Santos et al. [14, 15] presented a robust and flexible model for knowledge representation under uncertainty called *Bayesian Knowledge-Bases (BKB)*. BKBs are a ruled-based probabilistic model that extend BNs in a manner very similar to Poole's probabilities rules [12]. BKB represents objects/world states and their mutual relationships, using a directed graph. The graph consists of nodes which denote various random variable instantiations, while the edges represent conditional dependencies.

We argue that using BKBs can significantly reduce representation size and computation time in AWOL. Figure 1 describes the use of probabilistic networks, such as BNs and BKBs to model a Markov Decision Process (MDP).

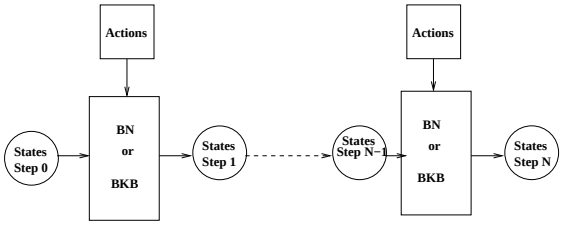


Figure 1. Overall description

The random variables in the BKB are the states of the agents in the team, dummy agents in each step in the plan, and their actions. As defined in our framework, the state of the agent can be: plan, opportunism, or null. A "dummy" agent can either be active or passive, in each step in the plan. Each agent can only execute two possible actions: remain, or opportunistic.

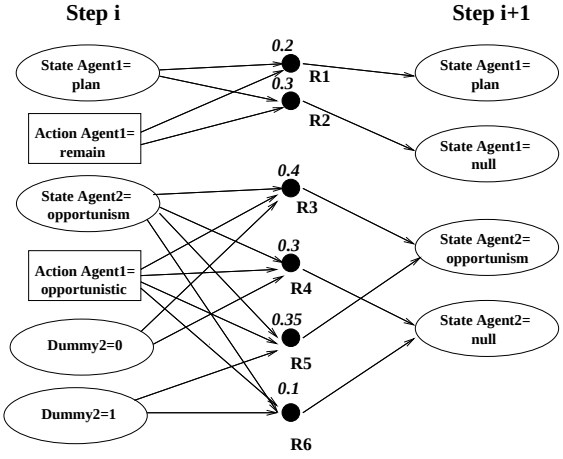


Figure 2. Fragment of BKB for two agents in the team

Figure 2 shows a fragment of a BKB for a team with two agents, where ovals represent states of agents (including dummy agents), and

rectangles represent actions. The state of the world is a tuple of agent states as defined in the section 2. Blackened circles represent rules in the BKB, where R_i is the name of the rule, and the number on each node is the conditional probability associated with the rule.

Figure 3 details one specific rule, which represents the conditional probability that Agent2 is in an opportunistic plan at Step i+1, depending on variables in the previous Step i (state of Agent2, its dummy agent value and action).

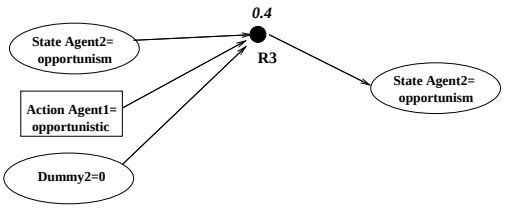


Figure 3. The rule represents the conditional probability P(Agent2 = opportunism | Agent2 = opportunism, Dummy2 = 0, Action2 = opportunistic) = 0.4

We observe that the number of rules necessary, in order to represent the transition probabilities, is smaller than the complete table. Thus, we can build a sparse BKB which includes only the relevant rules for the domain.

4 EMPIRICAL EVALUATION

Initial experimentation in AWOL is used to test whether the restricted model is sufficiently rich to represent interesting behavior resulting from opportunistic action.

4.1 Experimental setup

At this initial stage, we make the following additional assumptions in the experimental setup:

- In the initial state, all agents execute (the first step of) the same plan.
- Transition probability to an opportunistic plan is non-zero only if a specific "dummy agent" is in the "active" state. The dummy agents are all independent Markov processes. For each agent in an opportunistic plan, the team received an immediate "opportunity" reward. For the initial plan, the team is rewarded at the last step of the plan, depending on the agents still executing the plan at the last step.
- The domain is fully observable. Thus, one can compute an optimal global policy by solving an MDP (although in the future we will run experiments with partial observability).

As stated above, opportunities during plan execution are modeled by Markov process dummy agents. We add a "dummy agent" for each agent in the team. In addition, rather than invent different types of *roles*, we represent the roles using a **focus**, which is a real number within the range [0,1], to represent how important each agent is to a given plan.

We compare expected joint utility under various schemes where each agent makes assumptions about its team-members. As AWOL includes a centralized system that receives total information about the domain, it can compute a theoretically optimal global policy. However, AWOL can also model the individual agent, each of which can make only its own decisions and receive only partial information about the teammates through observation.

In the experiments, we use finite horizon (team utility is just sum of all rewards received). For control purposes, we compute a global optimal policy and its expected utility. We compare this control result to the expected utility received from the following "distributed" policy: each agent generates its own individual policy, and the individual policies are concatenated to create the distributed policy. Strictly speaking, since the environment is fully observable, each agent can compute a global optimal policy and then act according to its own role in this global policy. Instead, in order to simulate partial observability and lack of communication (in which case the above scheme may *not* result in execution of the optimal policy) we force each agent to model its team mates in other ways.

The current experiments are based on a Markov model, i.e. each agent assumes that the action its team-members executes depends probabilistically (in a simple way) on the current state. Specifically, the Markov model we use is - the team-member action depends on the state of its own "dummy agent" (i.e. attempt an opportunistic action with a certain given probability if the dummy agent is active).

4.1.1 *Computing expected utilities*

Computing expected utility for the global optimal policy is standard, as we simply use value iteration, and find the expected utility as the value function for the initial state. The case where each agent computes an individual policy is somewhat more complicated. First, each agent needs to represent its team-members as independent Markov Processes. What this means is that an agent g generates its own MDP, and solves it using value iteration, generating the local policies. Assuming each agent g acts according to its local policy π_g, we now have a global Markov process with rewards, and can evaluate an expected utility. The transition function $P_\pi(S_{new}|S_{old})$ for this Markov process is defined as $P(S_{new}|A_\pi(S_{old}), S_{old})$, where $A_\pi[g](S_{old}) = \pi_g(S_{old})$ for all agents g.

4.2 Preliminary results

We performed experiments on generating global and local policies, observing expected utilities. We used plans with 5 steps, and 3 agents. The following parameters were varied in our observations:

- Plan type: we used several plan types, e.g. flat (meaning all agents have equal focus at all steps of the plan), to time-varying sharply focused.
- The ratio of rewards for opportunistic plans vs. initial plan.
- Probability that dummy agents become active at each step (i.e. frequency of opportunistic triggers - denoted $Popp$).
- Probability that an opportunistic action succeeds, given that a trigger (dummy agent) is active in the current state - denoted $Psucc.opp.$.
- Probability that the initial plan does not fail at each step, given that all agents attempt to remain in the initial plan.
- Parameters of the Markov model used by each agents to represent its team-members. We used two such parameters: probability that an agent will attempt remain in the initial plan given that it observes a trigger (denoted $Prem|trig$), and given that it does not observe a trigger (denoted $Prem|\neg trig$).

Figure 4 depicts optimal global utility as the probability of success for opportunistic actions and frequency of opportunities is modified.

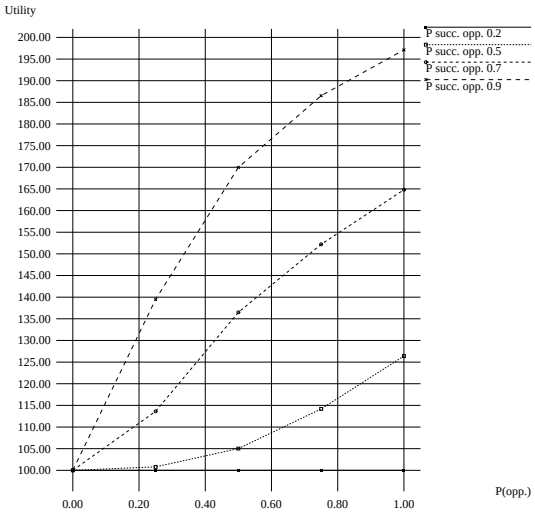


Figure 4. Optimal global expected utility

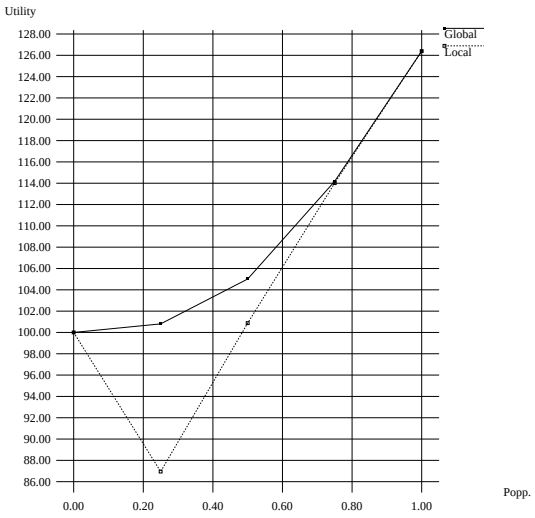


Figure 5. Global expected utility vs. distributed opportunity utility

Figure 5 plots expected utility for global and local utility vs. frequency of opportunistic trigger $Popp$. Reward for a successful completion of the initial plan is 100, and for a successful opportunistic action is 300.

As expected, global optimum is monotonically increasing with increased $Popp$. However, the local policies may actually achieve worse performance once opportunities are introduced, because agents incorrectly model other agents as taking opportunistic actions, which lead them to also attempt opportunistic action, even when inappropriate.

In the other side, Figure 6 shows how modifying the Markov model, representing how one agent believes the other agents will act, affect expected utility for the distributed policy. Although we did not introduce explicit observation errors (we did not implement POMDP solution, and in any case the distributed control problem is more complicated than a POMDP), the Markov model of team-mate behavior can be seen as if it introduced observation noise. Note that the Markov model parameter $Prem|trig$ tends to increase the expected utility when opportunistic actions are likely to fail (low $Psucc.opp.$), yet tends to decrease the expected utility when opportunistic actions are likely to succeed. This interesting effect occurs because when

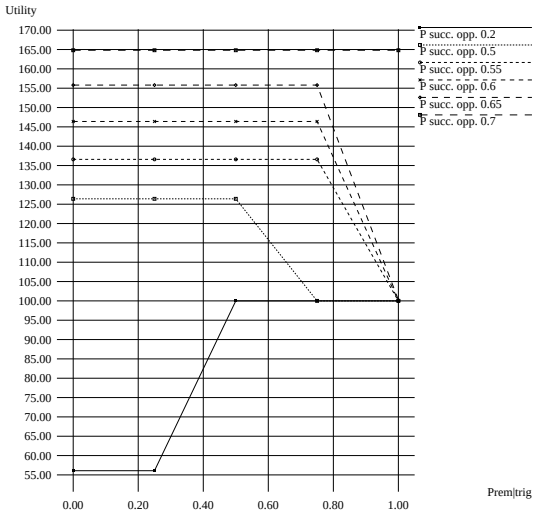


Figure 6. Expected Utility vs. Model Parameters

$Psucc.opp.$ is low, the optimal global policy is to remain in the initial plan. However, if $Prem|trig$ is low the agent believes its teammates will default, and thus believes it has no choice but to default as well, resulting in low expected utility. When $Prem|trig$ increases, the agents will estimate that their team-mates will not default, and thus choose to stick with the initial plan, resulting in better expected utility. The situation is reversed with high $Psucc.opp.$, where the optimal global policy is to try opportunistic actions.

5 DISCUSSION

5.1 Related Work

Many researchers developed different models for multi-agents environments, but none of then refer to opportunistic actions of the agents in the context of a pre-plan.

One of the papers inspiring our simulator for stochastic joint plan-state transitions was SPIRE, an experimental system presented by Sullivan, Glass et al. [18]. The authors used it in order to investigate the intention reconciliation in the context of a MAS. They considered the behavior of an agent A in the team when the agent A executes an action that contradicts the action A is expected to do by the team. Our treatment differs from SPIRE, in that our individual agents aim to optimize global, rather than self utility, and in that plan steps are sequentially dependent. Also, our scheme differs from the notion of replanning, in that typically opportunities appear even when the original joint plan can still be executed, as opposed to replanning, which usually occurs upon plan failure.

Durfee and Montgomery presented MICE [6], a semi-abstract model that implements a two-dimensional grid environment where

the agents simulate to act and interact but does not implement any specific reasoning method.

Lesser at al. [16] present a statistical model of the relationships between local cooperation, the environment, and the global utility, that has similar with our model. The authors present the notion of "self-interested", comparable with our "opportunistic actions". However, the model is based on negotiation between the agents, and the calculation of the global utility is different.

5.2 Conclusions and future work

As shown in this paper, the AWOL model shows sufficient structure to be interesting and may be useful in evaluating the role of opportunism in the context of joint plans. Clearly there is considerable further work to be done in the AWOL framework, starting with implementation and experimentation with true partial observability (rather than introducing partial observability through the back door as done in this paper). We envision three modes of operation where the AWOL framework can be useful. In the first mode, we attempt to understand how the model functions and through the model understand the tradeoffs between uncertainty, rewards, and opportunity, for different types of joint plans and other parameters, an investigation begun in this paper. In the second mode, an application environment, such as soccer simulation in RoboCup [10], or Unreal Tournament [1], would be "compiled down" into an AWOL model (i.e. the model parameters would be an abstraction of the application environment) and the decision on opportunistic action would be based on optimization in the resulting AWOL model. Finally, it may be possible to find a compact classifier (w.r.t. the parameters) for the decision on opportunistic action, in order to implement the optimal decision more efficiently.

Ongoing work aims at refinement of the AWOL model, using BKBs in order to solve different application-related instantiations of our framework. Currently, we are trying to compile-down cooperation problems in the Unreal Tournament computer game into AWOL. The abstract plans and actions of our model are implemented using script language supplied by the manufactures. Our agents act in a domain defined by the module Catch the Flag, that allows us to execute a series of experiments using different domain-specific plans.

Additionally, we need to complete the experiments on the impact of observation uncertainty, by having the agents optimize some forms of distributed POMDP. We are also interested in cases where the agent does not know all the distributions. We will need to handle the lack of knowledge by applying learning strategies.

Finally, as many of the application domains are adversarial, we intend to extend AWOL to handle the existence of an opposing team - in fact our reason for introducing "dummy agents" into AWOL is a handle introduced with this issue in mind.

REFERENCES

[1] Rogelio Adobbati, Andrew N. Marshall, Andrew Scholer, and Sheila Tejada. Gamebots: A 3d virtual world test-bed for multi-agent research.

[2] D.S. Bernstein, S. Zilberstein, and N. Immerman, 'The complexity of decentralized control of markov decision processes', *Mathematics of Operations Research.*, (2001).

[3] P. Cohen and H. Levesque, 'Teamwork', *Nous, Special Issue on Cognitive Science and AI*, **4**(25), 487–512, (1991).

[4] P.R. Cohen, H.J. Levesque, and I. Smith, 'On team formation', in *Contemporary Action Theory*, eds., J. Hintikka and R Tuomela, (1997).

[5] Thomas L. Dean and Michael P. Wellman, *Planning and control*, Morgan Kaufmann, 1991.

[6] E. Durfee and T. Montgomery, 'Mice: A flexible testbed for intelligent coordination experiments', in *In Proceedings of the Ninth Workshop on Distributed AI*, pp. 25–40, Rosario, Washington, (1989).

[7] Barbara Grosz and Sarit Kraus, 'The evolution of shared plans', in *Foundations and Theories of Rational Agents*, eds., A. Rao and M. Wooldridge, 227–262.

[8] Barbara Grosz and Sarit Kraus, 'Collaborative plans for complex group actions', *Artificial Intelligence*, **82**(2), 269–357, (1996).

[9] Karen E. Lochbaum, Barbara Grosz, and Candice L. Sidner, 'Models of plans to support communication: An initial report', in *Proceedings of the Eighth National Conference on Artificial Intelligence*, eds., Thomas Dietterich and William Swartout, pp. 485–490, Menlo Park, CA, (1990). AAAI Press.

[10] Itsuki Noda and Hitoshi Matsubara, 'Soccer server and researches on multi-agent system', in *Proceedings of the IROS-96 Workshop on Robcup*, Osaka, Japan, (November 1996).

[11] Judea Pearl, *Probabilistic Reasoning in Intelligent Systems, Networks of Pausible Inference*, Morgan Kaufmann Publishers, Palo Alto, CA, 1988.

[12] David Poole, 'Probabilistic partial evaluation: Exploiting rule structure in probabilistic inference', in *Proc. Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*, pp. 1284–1291, Nagoya, Japan, (August 1997).

[13] David Poole and Nevin Lianwen Zhang, 'Exploiting contextual independence in probabilistic inference', *Journal of Artificial Intelligence Research*, **18**, 263–313, (2003).

[14] T. Rosen, S.E. Shimony, and E. Santos Jr., 'Reasoning with bkbs- algorithms and complexity', *Annals of Mathematics and Artificial Intelligence*, (40), 403–425, (2004).

[15] Eugene Jr. Santos and Eugene S. Santos, 'A framework for building knowledge-bases under uncertainty', *Journal of Experimental and Theorical Artificial Intelligence*, (11), 265–286, (1999).

[16] Jiaying Shen, Xiaoqin Zhang, and Victor Lesser, 'Degree of local cooperation and its implication on global utility', in *In Proceedings of Third International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2004)*, (July 2004).

[17] Peter Stone and Manuela Veloso, 'Task descomposition, dyanmic role assignment, and low-bandwidth communicatin for real-time startegic teamwork', *Artificial Intelligence*, **2**(110), 241–273, (June 1999).

[18] D. G. Sullivan, A. Glass, B. J. Grosz, and S. Kraus, 'Intention reconciliation in the context of teamwork: An initial empirical investigation', *Lecture Notes in Computer Science*, **1652**, 149–162, (1999).

.