Proceedings

AMAPS2012

Algolog Multi-Agent Programming Seminar 2012

29 November 2012

DTU Informatics

Technical University of Denmark

Editors

Jørgen Villadsen

Andreas Schmidt Jensen

Algolog refers to the Algorithms and Logic section at DTU Informatics.

The focus of the seminar is on tools and techniques for programming multi-agent systems.

Key topics are multi-agent programming competitions and genuine multi-agent challenges such as organization, communication and agreement as well as algorithms and logics for multi-agent systems.

Contents

Engineering Multi-Agent Systems

Jørgen Villadsen
DTU Informatics

The focus of the Algolog Multi-Agent Programming Seminar (AMAPS) is similar to the aim and scope of the following yearly events:

- International Workshop on Programming Multi-Agent Systems (ProMAS), previous held in Melbourne, New York, Utrecht, Hakodate, Honolulu, Estoril, Budapest, Toronto, Taipei and Valencia (2012).
- International Workshop on Computational Logic in Multi-Agent Systems (CLIMA), previously held in London, Paphos, Copenhagen, Fort Lauderdale, Lisbon, London, Hakodate, Porto, Dresden, Hamburg, Lisbon, Barcelona and Montpellier (2012).

Quote from the homepage:

> *"The purpose of the CLIMA workshops is to provide a forum for discussing techniques, based on computational logic, for representing, programming and reasoning about agents and multi-agent systems in a formal way.*
>
> *Multi-Agent Systems are communities of problem-solving entities that can perceive and act upon their environment to achieve their individual goals as well as joint goals. The work on such systems integrates many technologies and concepts in artificial intelligence and other areas of computing as well as other disciplines. Over recent years, the agent paradigm gained popularity, due to its applicability to a full spectrum of domains, from search engines to educational aids to electronic commerce and trade, e-procurement, recommendation systems, simulation and routing, to cite only some.*
>
> *Computational logic provides a well-defined, general, and rigorous framework for studying syntax, semantics and procedures for various tasks by individual agents, as well as interaction amongst agents in multi-agent systems, for implementations, environments, tools, and standards, and for linking together specification and verification of properties of individual agents and multi-agent systems."*

The following page shows a specialization in Multi-Agent Systems on the BSc in Software Technology and the MSc in Computer Science and Engineering programs at DTU. The study plan satisfies the requirements for the Efficient and Intelligent Software study line.

In August 2012 a Dagstuhl Seminar was held with the title: Engineering Multi-Agent Systems

Additional information: http://www.dagstuhl.de/12342

At Schloss Dagstuhl in Germany computer scientists meet to discuss current research topics.

BSc in Software Technology (180 ECTS) *Other BSc programs are possible too*

Software Engineering (45 ECTS)

02121	Introduction to Software Technology	10
02122	Software Technology Project	10
02141	Computer Science Modelling	10
02161	Software Engineering 1	5
02162	Software Engineering 2	10 *

Algorithms and Logic Core (15 ECTS)

02105	Algorithms and Data Structures 1	5
02110	Algorithms and Data Structures 2	5
02180	Introduction to Artificial Intelligence	5

Mandatory Units (60 ECTS)

Mathematics / Physics / Chemistry / Programming / Embedded Systems

Other Units (60 ECTS)

Theory of Science in Engineering / Bachelor Project / Study Abroad / Electives

MSc in Computer Science and Engineering (120 ECTS)

Prerequisites and Mandatory Units (30 ECTS)

02156	Logical Systems and Logic Programming	5
02157	Functional Programming	5
02158	Concurrent Programming	5
02257	Applied Functional Programming	5
42490	Technology, Economics, Management and Organization	10

Multi-Agent Systems (57.5 ECTS)

02220	Distributed Systems	7.5
02224	Real-Time Systems	5
02249	Computationally Hard Problems	7.5
02281	Data Logic	5 *
02282	Algorithms for Massive Data Sets	7.5 *
02284	Knowledge-Based Systems	5
02285	Artificial Intelligence and Multi-Agent Systems	7.5
02286	Logic in Computer Science, Artificial Intelligence and Multi-Agent Systems	7.5
02291	System Integration	5

Other Units (32.5 ECTS)

Thesis / Electives

* = *Optional*

Additional information: http://cse.imm.dtu.dk

On the Multi-Agent Programming Contest

Kenneth Balsiger Andersen and Andreas Frøsig

DTU Informatics

Abstract. The aim of the annual agent contest is to stimulate research in the area of multi-agent systems, to identify key problems and to collect suitable benchmarks.
We provide a brief description of the 2012 competition and the Python-DTU system. We analyse the contest especially between our own system and Federal University of Santa Catarina's (UFSC) system as we had some very intense battles. We have run a lot of tests between the two systems in order to figure out what we could have done better.

1 Introduction

This paper describes some of our work with the Python-DTU team which participated in the Multi-Agent Programming Contest 2012 [3], and in particular how our system works and the results of an analysis we have made against the winning team from Federal University of Santa Catarina (UFSC).

In 2012 the members of the Python-DTU team were associate professor Jørgen Villadsen (DTU Informatics), Andreas Schmidt Jensen, Mikko Berggren Ettienne, Steen Vester, Kenneth Balsiger Andersen and Andreas Frøsig. Given that the scenario is very similar to that of last year, we decided to look into ways of improving the system from last year [1]. We have explained some of the main strategies of our team in this paper but we refer to [2] for more details.

In the tournament we got a very close second place against UFSC's team. For this reason we have made an analysis between the systems to see whether it was coincidental or if their system was genuinely better than ours. We also implemented some logic to neutralize a specific strategy of theirs to see what would have happened if we had done this in the tournament.

The paper is organized as follows. In section 2 we provide an overview of the contest. In section 3 we analyse the contest and describe some of the most important algorithms and designs used in our multi-agent system. In section 4 we describe a number of tests of our system against the winners of the contest, UFSC. Finally, we conclude our work by discussing possible improvements of our system in section 5.

2 Multi-Agent Programming Contest 2012

In this section we will give the reader an overview of the contest. MAPC is a contest, that have been running for several years and in 2011 and 2012 it deals with the so called "Agents on Mars". If additional information is required we refer to the official descriptions in [3].

2.1 Agents on Mars

The scenario in the contest is that mankind has populated Mars in the year 2033. In this world the citizens of Mars developed some autonomous intelligent agents called All Terrain Planetary Vehicles (ATPV), which they used to search for water wells. To strengthen the search for these wells, the World Emperor has handed out various achievements for different missions, which has resulted in sabotage amongst the different groups of settlers. The task of this contest is to implement these autonomous agents such that they are intelligent enough to collect more achievements than the other groups.

The contest is structured as a turn-based game, where the teams have a time limit to decide what to do each turn. When the agents have decided which actions to take, they have to send the plans to the server, which then calculate whether the actions succeed or not.

The map is structured as a graph consisting of vertices and edges. where the vertices are water wells and the weight of the edges is the energy cost to go from one well to the another. We will use the terms vertex and edge for the rest of this article.

Agent roles and actions The ATPVs described above are assigned different sets of skills matching their roles, thus we are working with a heterogeneous system.

In table 1 we have given an overview of the attributes and skill-sets of the different roles.

	Attack	Buy	Goto	Inspect	Parry	Probe	Recharge	Skip	Survey	Energy	Health	Strength	Visibility range
Explorer		×	×			×	×	×	×	24	4	0	2
Repairer		×	×		×		×	×	×	16	6	0	1
Saboteur	×	×	×		×		×	×	×	14	3	3	1
Sentinel		×	×		×		×	×	×	20	1	0	3
Inspector		×	×	×			×	×	×	16	6	0	1

Table 1. Table showing the skillsets and attributes of the different roles

In addition to the specifications above, an agent can get disabled if its current health is equal to 0. A disabled agent is allowed to perform the following actions: `goto`, `repair`, `recharge` and `skip` iff. their role allows it. Additionally the `recharge` rate and effect of devices bought are lowered when an agent is disabled.

Achievements Each team will earn Achievement Points when they reach a milestone. We have listed some examples of the different milestones below:

- Having zones with fixed values, e.g. 10 or 20
- Fixed numbers of probed vertices, e.g. 5 or 10
- Fixed numbers of surveyed edges, e.g. 10 or 20
- Fixed numbers of inspected vehicles, e.g. 5 or 10
- Fixed numbers of successful attacks, e.g. 5 or 10
- Fixed numbers of successful parries, e.g. 5 or 10

In the competition each step of each achievement is exponentially harder to reach than the previous. The Achievement Points earned during the game can be used to buy devices to improve the agents attributes.

3 System Analysis and Design

In this section we will describe how we want the agents to behave throughout the simulations and a short analysis of the world and thus what is needed from our agents.

3.1 Agent behaviour

Our resulting system is a decentralized solution with a focus on time performance.

Instead of letting the agents find goals only based on their private knowledge they use the distributed knowledge of the entire team. This adds some communication which in some cases is unnecessary but in most cases the extra knowledge will produce better goals for the agents.

In each step each agent will find its preferred goals autonomously and assign each of them a *benefit* based on its own desires (i.e. the type of agent), how many steps are needed to reach the location and so on. In order to make sure that multiple agents will not commit to the same goal they communicate in order to find the most suitable agent for each goal. This is done using our auction-based agreement algorithm which will be discussed in more detail in section 3.5.

The agents in this contest are situated in an inaccessible environment which means that the world state can change without the agents noticing from step to step, e.g. if the opponent's agents move outside our agents' visibility range. Hence our agents should be very reactive to observable changes in the environment.

At times our agents are proactive. The most important one being the communication between a disabled agent and a repairer. They use their shared knowledge in order to decide which of the agents should take the last step and who

should stay, so that they eventually are standing on the same vertex instead of simply switching positions. This is implemented by considering the current energy for each agent.

Some of our agents also attempt to be proactive by for example parrying if an opponent saboteur is on the same vertex. Furthermore, repairers will repair wounded agents since they are likely to be attacked again.

3.2 Getting achievements

In the beginning of a simulation every agent will work towards achieving as many type specific goals as possible in a more or less disorganized fashion, e.g. the inspector will inspect every opponent it sees. We do this to achieve as many achievements as possible as fast as possible.

After a certain number of steps the achievements will be so hard to get, that the agents will proceed to the zone control part of our strategy where our agents will do all they can to maximize our zone score each step. For example the explorers keep probing our target area to make sure we control as many vertices as possible.

3.3 Zone control

The zone control part of our strategy uses a very simple, but surprisingly effective, greedy algorithm. The algorithm works by choosing the vertex with the highest potential value. The first vertex is the one with the highest value, the rest of the vertices chosen by the algorithm are chosen by considering the potential value calculated with part of the colourings algorithm which can be read in detail in [3].

This algorithm will to some extent choose the optimal area or several areas which are still fairly easy to maintain, even though our choices are limited by our (partial) knowledge of the map and the missing parts of the area colouring algorithm.

During the zone control part every type of agent has a specific job.

- *Repairers and saboteurs* do not directly participate in the zone control, instead they are trying to defend and maintain the zone.
- *Inspectors* keep inspecting from their given expand node, because the opponents might have bought something which we need to make a counter move against.
- *Explorers* will probe unprobed vertices within the target zone. When all vertices are probed they are assigned a vertex by the zone control strategy.
- The *sentinels* will stay on a vertex assigned by the zone control strategy and will parry if some of the opponent's saboteurs move to the sentinels position.

3.4 Buying strategy

Only our saboteurs buy devices, and they buy exactly enough extra health so that they will not get disabled by a single attack from an opponent saboteur that has not upgraded his strength. Furthermore we buy enough strength to disable any opponent saboteur in a single attack by buying strength for all our saboteurs every time we inspect the opponent saboteurs and find that it has more health than all other inspected saboteurs. This buying strategy is chosen in hope of dominating the map which will make it possible to gain control of the zone we want. The advantage is that we only try to out-buy in one specific field, thus we are unlikely to use all our achievement points. As this is a quite aggressive buying strategy we had to wait to step 150 to have enough achievement points to execute it.

3.5 Making decisions

The agents need a consistent way of figuring out what to do. We do this by letting every agent find the nearest goals according to their type. They do this by using a modified best-first search (BFS) which returns a set of goals. To make sure that every agent always has at least one goal the BFS returns as many goals as we have agents. This is a very agent-centered procedure meaning the agents simply commit to the goal with the highest benefit, instead of coordinating any bigger schemes. However, since the goals are more or less dependent on each other there is some implicit coordination. For example the repairers will often follow the saboteurs as these search for opponents and thus more often will share a vertex with an opponent saboteur and get disabled.

To decide which goal to pursue the agents use an auction algorithm. Every agent can bid on the goals they want to commit to and will eventually be assigned the one they are best suited for. This results in a good solution, which however might not be optimal. For further details we refer to the paper about our system from 2011 [1].

Even though our planner calculates a few turns ahead the agents recalculate every turn. We do this to adapt to newly discovered obstacles and facts, such as an opponent saboteur or the fact that the agent has been disabled. The agents will not end up walking back and forth as their previous goal will now be one step closer, thus the benefit of the goal has increased. If another goal becomes more valuable it means that it is a better goal than the one the agent was pursuing, thus changing the commitment makes sense, so we do not lose anything on recalculating each turn.

4 Results

In this section we want to test our system against the winners of this years competition namely the team UFSC from Brazil. The reason we want to test is that the matches in the tournament were extremely close and we realized

that they had built a counter strategy to our buying strategy. The idea behind the counter strategy was to make our team use as many achievement points as possible, without using that many themselves. They did this by buying a lot of health on a single saboteur so that all our saboteurs bought strength to be able to handle this, they lured our strategy in one of the test matches. By doing this we used four times as many achievement points and thus we got a lower step score even though our zone score was better.

The plots below have been made by scanning through the log files created by the server counting won battles in all simulations that were valid by following the qualification rule of at most 5% missing actions. We extracted the data using a Python script which plotted the data using Gnuplot.

4.1 Original

First we want to test the system we used in the tournament against theirs to see whether it is genuinely inferior or it was up to chance.

In the overall test we can see that we are slightly behind with a score of 143 against theirs 148, so it was some pretty close battles. By watching the battles it seemed as if we often got a zone separated from the main battle, meaning that we could keep an undisturbed zone while constantly disturbing theirs. This meant that we neutralized the score advantage they got from the achievement points. For obvious reasons this does not work as well in small maps as in large ones, so we want to compare the different simulations to each other.

To see why we are losing we look at the tests for each of the simulations which can be seen in figure 1, 3 and 5.

According to these tests we win a little over half of the battles in the first two simulations, but loose a lot in the last one. This could indicate that our hypothesis according to the map size are accurate. It makes sense that with a map of a certain size it is no longer possible for our agents to create an isolated zone and thus we cannot even the total step score.

4.2 Optimized version

In an attempt to even the score for the small maps we have implemented some extra logic in our agents such that they will only buy strength up to the second best of the opponent's saboteurs. We do this in order to minimize the difference between the achievement points while still having at least as good saboteurs as most of the opponent team's saboteurs, namely their second best. It should be noted that the tests below are not with the same version of the team that was used in the contest.

With this improvement we are winning $\sim$75% of the battles with a score of 172 against their 58, compared to before this is a great improvement which shows that their counter strategy was working very well against our team.

Now we want to see if the change made us good enough to win on all the simulations, so we compare the three different simulations on figure 2, 4 and 6.

As can be seen we are winning not only in the two first simulations but also in the third, where they had the upper hand before. We can still see that we win more in the two larger maps than the small one of simulation three. This strengthens our belief that the extra zone we get in the two big maps helps quite a bit.

5 Conclusion

All in all our system wins a little more than 50% of the simulations on the two larger maps. This means that these two simulations were pretty much up to chance at the competition whereas the third one had a much higher chance of going to UFSC.

Our greatest weakness was that our uncompromising attempt to have the strongest saboteurs could be countered by buying enough health on a single saboteur to make us use most of our achievement points for improving all of our saboteurs. Even though our saboteurs were better than the opponents and we had a much more stable zone, the difference in achievement points evened out the total step score.

The tests show that we win most simulations in the two larger maps and it seems like the reason is that we often get a second area which neutralizes the extra score UFSC gets from the difference in achievement points. On the small map of simulation 3 this is not possible for our system and we lose quite significantly in this case. After implementing some extra logic which neutralizes their counter strategy we have seen that we actually win ∼75% of the battles and more significantly we won on the small map as well which we did not before. With this new and improved system they do win more on the small map than on the larger ones, which is probably because we have an extra area in the two larger ones.

References

1. Mikko Berggren Ettienne, Steen Vester, and Jørgen Villadsen. Implementing a Multi-Agent System in Python with an Auction-Based Agreement Approach. In Louise A. Dennis, Olivier Boissier, and Rafael H. Bordini (Eds.): ProMAS 2011, LNCS 7217, 185-196, Springer 2012.
2. Jørgen Villadsen, Andreas Schmidt Jensen, Mikko Berggren Ettienne, Steen Vester, Kenneth Balsiger Andersen, and Andreas Frøsig. Reimplementing a Multi-Agent System in Python. In Mehdi Dastani, Brian Logan, Jomi F. Hübner (Eds.): ProMAS 2012, LNCS, Springer, to appear.
3. Tristan Behrens, Michael Köster, Federico Schlesinger, Jürgen Dix, and Jomi Hübner. Multi-Agent Programming Contest — Scenario Description — 2012 Edition. Available online: `http://www.multiagentcontest.org/`, 2012.

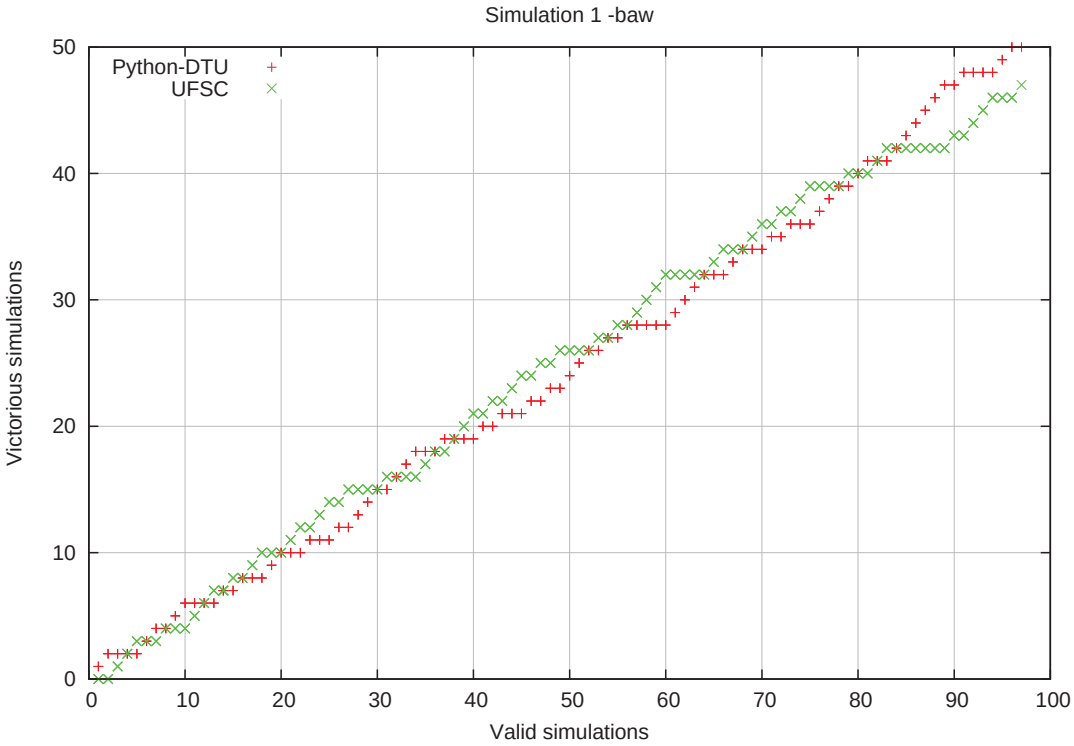


Fig. 1. Won matches on simulation 1 for our system with 300 vertices

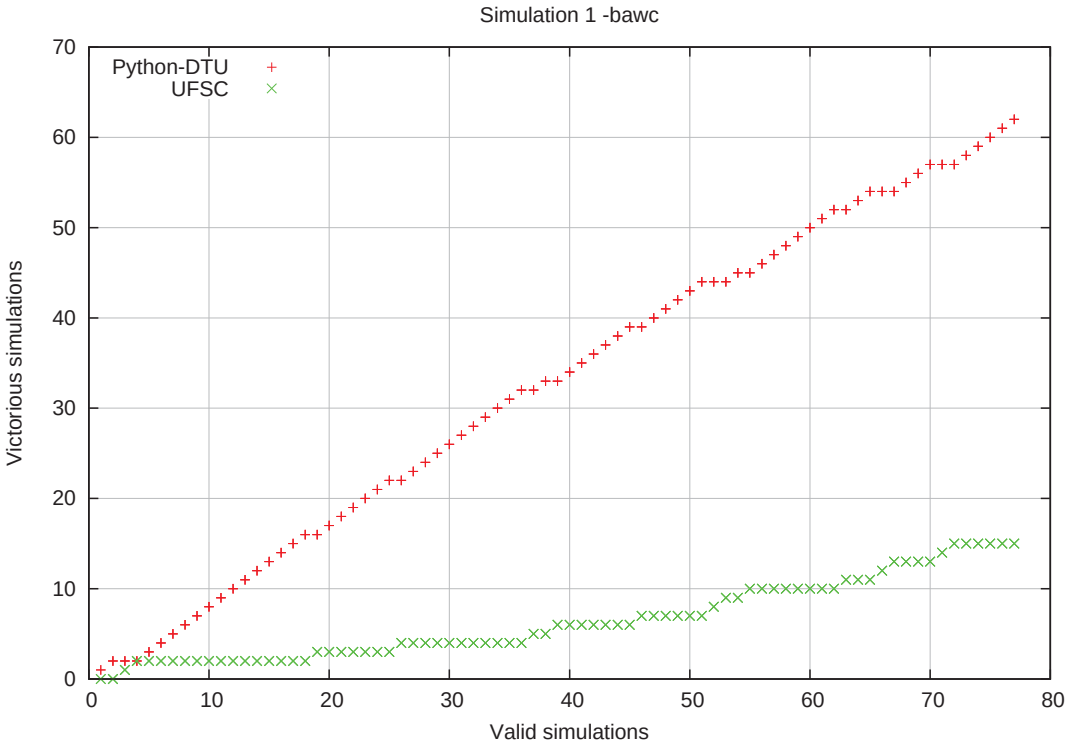


Fig. 2. Won matches on simulation 1 for our optimized system with 300 vertices

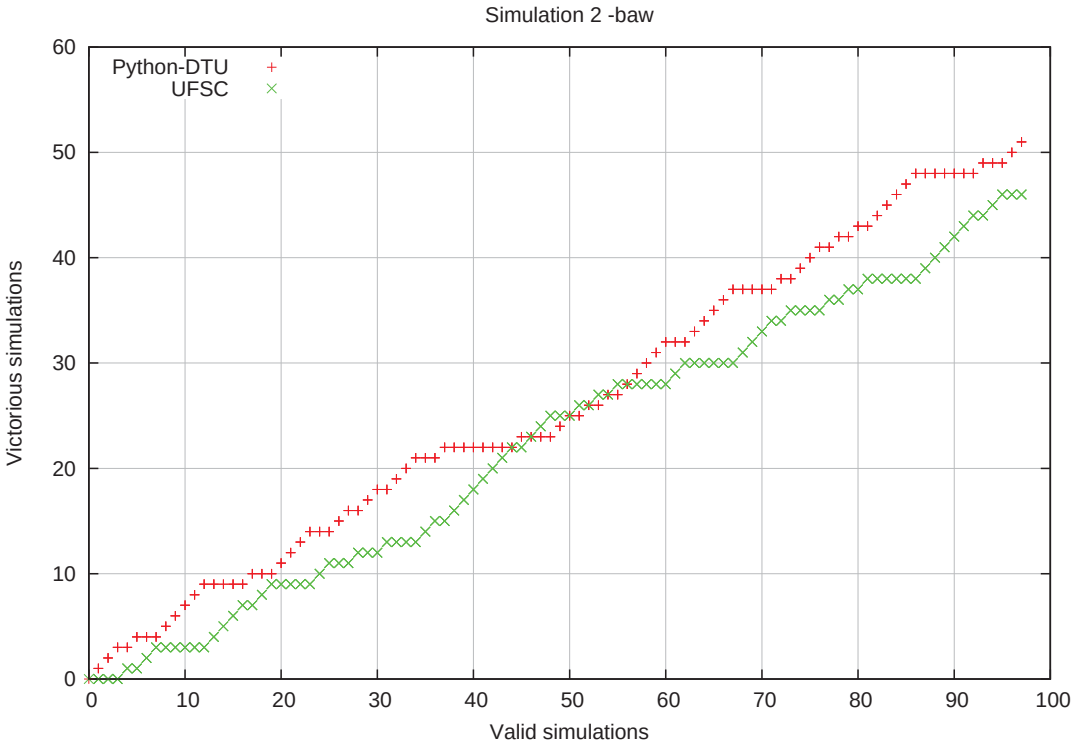


Fig. 3. Won matches on simulation 2 for our system with 240 vertices

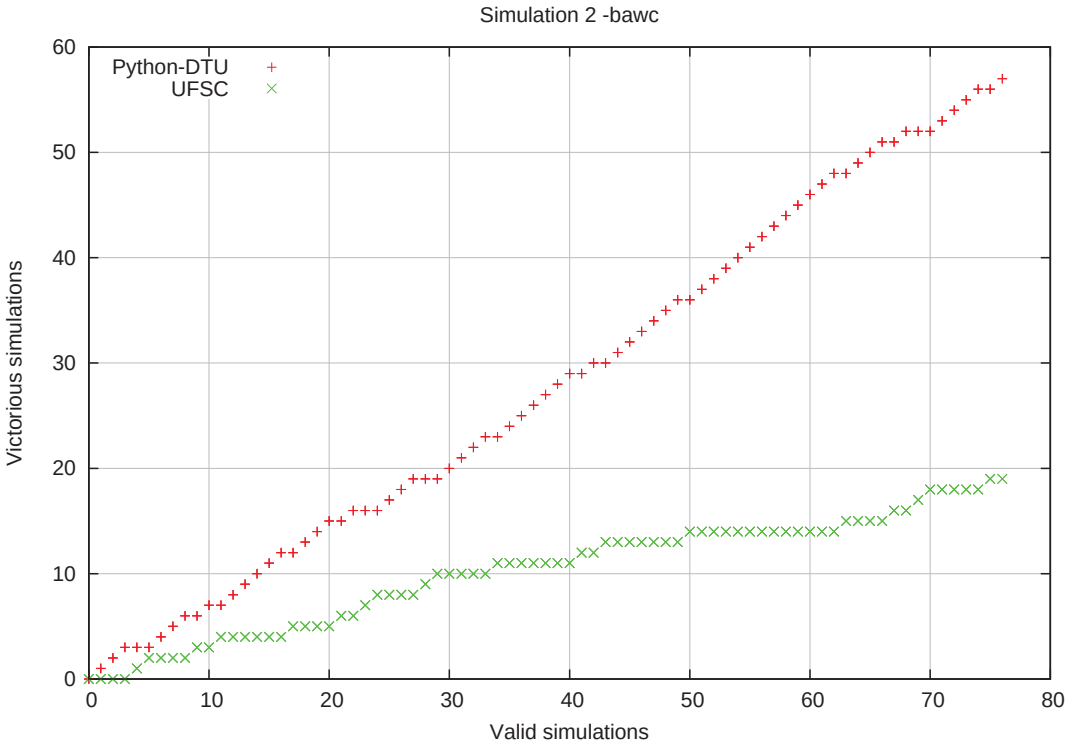


Fig. 4. Won matches on simulation 2 for our optimized system with 240 vertices

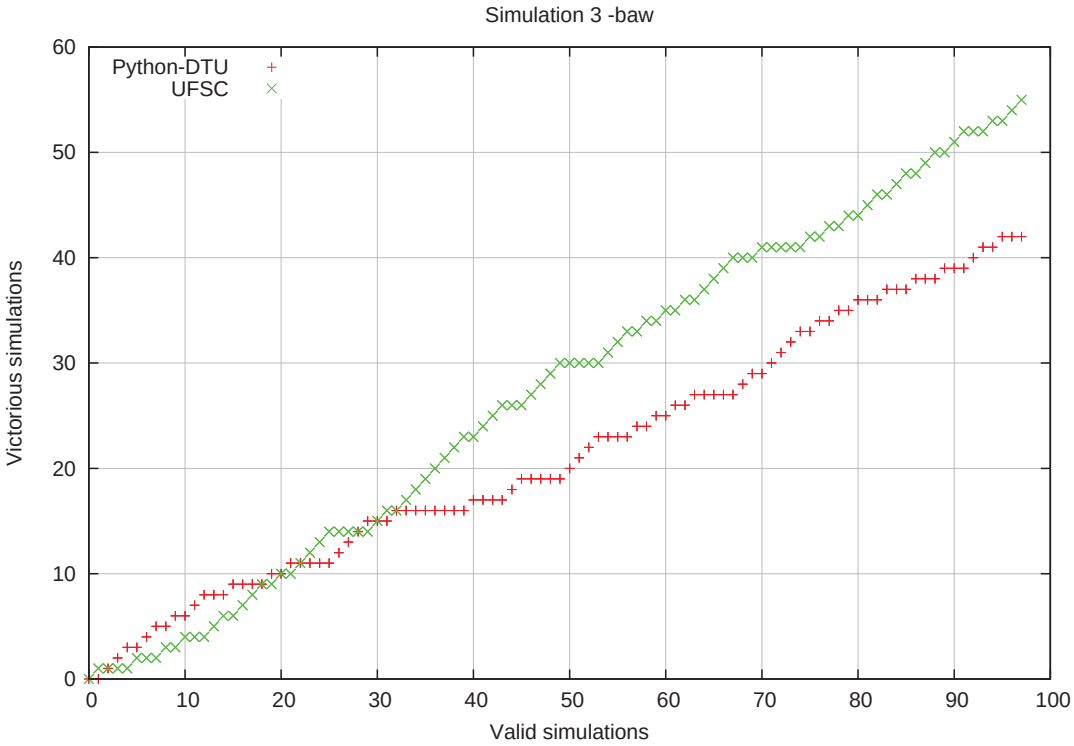


Fig. 5. Won matches on simulation 3 for our system with 200 vertices

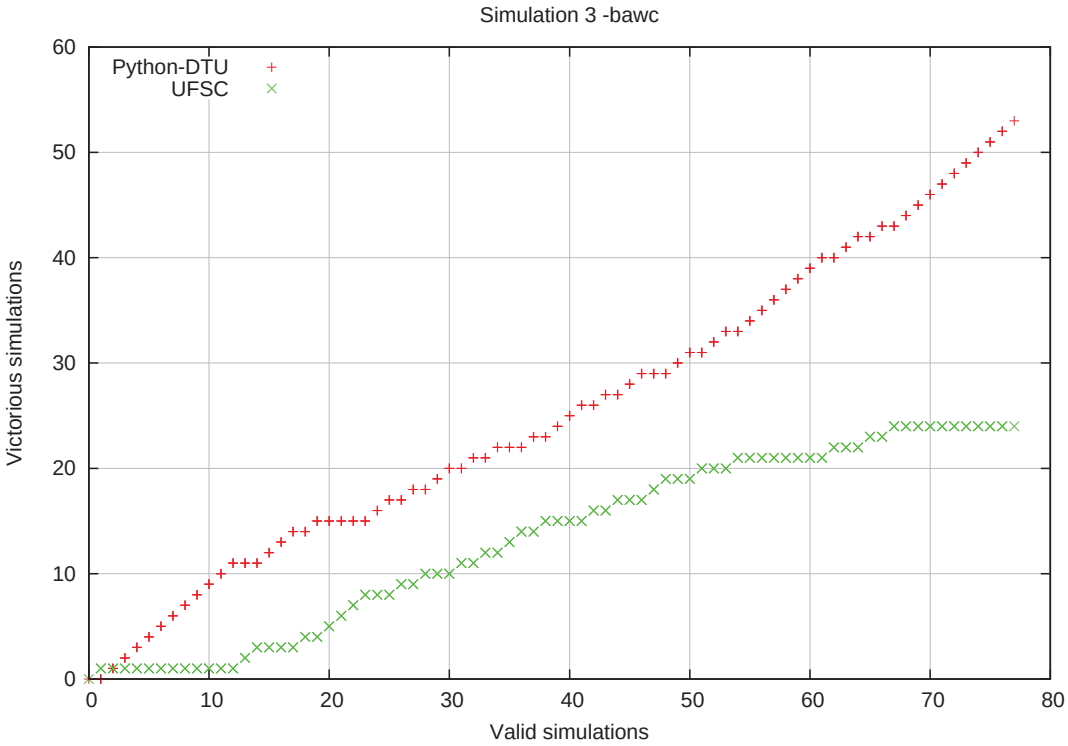


Fig. 6. Won matches on simulation 3 for our optimized system with 200 vertices

An Application of Game Theory in Program Synthesis

Steen Vester

DTU Informatics

Abstract. In this article we motivate the use of infinite concurrent games and the notion of Nash equilibria in program synthesis and present some of the results obtained in the area. Games have traditionally been used mostly in economics, but have recently received an increasing amount of attention in computer science with applications in several areas including logic, verification and multi-agent systems. Our motivation for studying Nash equilibria is to let the players in a game represent programmable devices and let strategies represent programs. Nash equilibria then correspond to systems where all devices are programmed optimally in some sense. Modelling the interaction in distributed systems by means of games with self-interested agents is in many cases more realistic than traditional analysis where opponents are assumed to act in the worst possible way instead of acting rationally and in their own interest.
Note that no new results are presented in this paper, it is merely a short introduction to Nash equilibria in concurrent games as well as an application of this topic in program synthesis.

1 Introduction

Concurrent games have been studied by a number of other authors, including [1,3,4,5,6,7,8,9,15,18]. A concurrent game in our context is played on a finite graph where the nodes represent the different states of the game and edges represent transitions between game states. The game is played an infinite number of rounds and in each round the players must concurrently and independently choose an action. The choices of actions of all the players uniquely determines the successor of the current game state. An example of a concurrent game can be seen in Figure 1. This is a mobile phone game (also used in [3,18]), where two mobile phonse fight for bandwidth in a network. They each have three transmitting power levels (0,1 and 2) and each turn a phone can choose to increase, decrease or keep its current power level. State t_{ij} corresponds to phone 1 using power level i and phone 2 using power level j. The goal of the phones is to obtain as much bandwidth as possible, but at the same time reduce energy consumption.

The infinite runs of games and interaction between players seems like a good framework for modelling the interaction of reactive systems which should (theoretically) be able to run forever. This includes applications such as web servers,

operating systems, mobile phones, etc. A typical use of games in verification is to design a two-player game, where one player corresponds to a programmable system and the other player corresponds to the environment [2,8,9,10,14,17]. In this case it often makes sense to search for strategies for the system player that obtains some objective no matter how the environment acts. However, the restriction of assuming that other players will act in the worst possible way does not necessarily make good sense in distributed systems or multi-agent systems where goals are not necessarily opposite. In this case it seems to make more sense to assume that other agents will act in their own interest. This is the main motivation for studying Nash equilibria instead of winning strategies, since Nash equilibria are strategy specifications where all players choose the optimal strategy with respect to their own objectives, given that all other players stick to the strategy specification.

2 Definitions

2.1 Concurrent Games

Definition 1 *A Concurrent Game is a tuple* $\mathcal{G} = (\text{States}, \text{Agt}, \text{Act}, \text{Mov}, \text{Tab})$ *where*

- States *is a finite set of states*
- Agt *is a finite set of agents*
- Act *is a finite set of actions*
- $\text{Mov}: \text{States} \times \text{Agt} \to 2^{\text{Act}} \setminus \{\emptyset\}$ *is the set of actions available to a given player in a given state*
- $\text{Tab}: \text{States} \times \text{Act}^{\text{Agt}} \to \text{States}$ *is a transition function that specifies the next state, given a state and an action of each player.*

A *move* $(m_A)_{A \in \text{Agt}}$ consists of an action for each player. It is said to be *legal* in the state s if $m_A \in \text{Mov}(s, A)$ for all $A \in \text{Agt}$. The legal plays $\text{Play}_{\mathcal{G}}$ of $\mathcal{G}$ are the infinite sequences $\rho = s_0 s_1 ... \in \text{States}^{\omega}$ such that for all i there is a legal move m_i such that $\text{Tab}(s_i, m_i) = s_{i+1}$. In the same way the set of *histories* $\text{Hist}_{\mathcal{G}}$ is the set of finite sequences of states that respects the Mov and Tab functions. We define the paths of $\mathcal{G}$ as $\text{Path}_{\mathcal{G}} = \text{Hist}_{\mathcal{G}} \cup \text{Play}_{\mathcal{G}}$. For a given path ρ we write ρ_i for the ith state of the path (the first state has index 0), $\rho_{\leq i}$ for the prefix containing the first i states of ρ and $|\rho|$ for the number of transitions in ρ when ρ is finite. When ρ is finite we also write $\text{last}(\rho)$ to denote the last state in ρ. For a path ρ the set of states occuring at least once

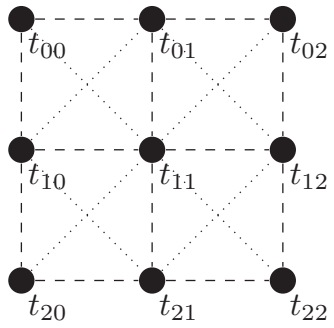


Fig. 1. Mobile game. Self-loops and actions are omitted.

and the set of states occuring infinitely many times is denoted $\mathrm{Occ}(\rho)$ and $\mathrm{Inf}(\rho)$ respectively.

We define a *strategy* $\sigma_A : \mathrm{Hist}_{\mathcal{G}} \to \mathrm{Act}$ of player A as a function specifying a legal move for each finite history. We call a subset $P \subseteq \mathrm{Agt}$ a *coalition* of players and define a strategy $\sigma_P = (\sigma_A)_{A \in P}$ of a coalition P as a tuple of strategies, one for each player in the coalition. A strategy for the coalition Agt is called a *strategy profile*. We denote the set of strategy profiles $\mathrm{Prof}_{\mathcal{G}}$. Given a strategy $(\sigma_A)_{P \in A}$ for a coalition P we say that a (finite or infinite) path $\pi = s_0 s_1 ...$ is *compatible* with the strategy if for all consecutive states s_i and s_{i+1} there exists a move $(m_A)_{A \in \mathrm{Agt}}$ such that $\mathrm{Tab}(s_i, (m_A)_{A \in \mathrm{Agt}}) = s_{i+1}$ and $m_A = \sigma_A(s_0 s_1 ... s_i)$ for all $A \in P$. The set of infinite plays compatible with $(\sigma_A)_{A \in P}$ from a state s is called the *outcomes* from s and is denoted $\mathrm{Out}_{\mathcal{G}}((\sigma_A)_{A \in P}, s)$. The set of finite histories compatible with $(\sigma_A)_{A \in P}$ from a state s is denoted $\mathrm{Out}^f_{\mathcal{G}}((\sigma_A)_{A \in P}, s)$. In particular, note that for a strategy profile, the set of outcomes from a given state is a singleton.

2.2 Objectives and Preferences

In our setting there are a number of *objectives* that can be accomplished in a concurrent game. An objective is simply a subset of all possible plays. We consider particular kinds of qualitative objectives which have been analyzed in the literature defined on sets T of states, namely reachability, safety and Büchi objectives defined as follows:

$$\Omega_{\mathrm{Reach}}(T) = \{\rho \in \mathrm{Play}_G | \mathrm{Occ}(\rho) \cap T \neq \emptyset\}$$
$$\Omega_{\mathrm{Safe}}(T) = \{\rho \in \mathrm{Play}_G | \mathrm{Occ}(\rho) \cap T = \emptyset\}$$
$$\Omega_{\mathrm{Büchi}}(T) = \{\rho \in \mathrm{Play}_G | \mathrm{Inf}(\rho) \cap T \neq \emptyset\}$$

which corresponds to reaching a state in T, keeping the play away from T and visiting a state in T infinitely many times respectively. We use the same framework as in [3] where players can have a number of different objectives and the payoff of a player given a play can depend on the combination of objectives that are accomplished in a quite general way. In this framework there are given a number of objectives $(\Omega_i)_{1 \leq i \leq n}$. The *payoff vector* of a given play ρ is defined as $v_\rho \in \{0,1\}^n$ such that $v_i = 1$ iff $\rho \in \Omega_i$. Thus, the payoff vector specifies which objectives are accomplished in a given play. We write $v = \mathbb{1}_T$ where $T \subseteq \{1, ..., n\}$ to denote $v_i = 1 \Leftrightarrow i \in T$. We simply denote $\mathbb{1}_{\{1,...,n\}}$ by $\mathbb{1}$. Each player A in a concurrent game is given a total preorder $\lesssim_A \subseteq \{0,1\}^n \times \{0,1\}^n$ which intuitively means that $v \lesssim_A w$ if player A prefers the objectives accomplished in w over the objectives accomplished in v. This preorder induces a preference relation $\precsim \subseteq \mathrm{Play}_{\mathcal{G}} \times \mathrm{Play}_{\mathcal{G}}$ over the possible plays defined by $\rho \precsim \rho' \Leftrightarrow v_\rho \lesssim v_{\rho'}$. Additionally we say that A strictly prefers $v_{\rho'}$ to v_ρ if $v_\rho \lesssim_A v_{\rho'}$ and $v_{\rho'} \not\lesssim_A v_\rho$. In this case we write $v_\rho <_A v_{\rho'}$ and $\rho \prec_A \rho'$.

2.3 Nash Equilibria and Decision Problems

We have now defined the rules of the game and are ready to look at the solution concept of a Nash equilibrium, which is a strategy profile in which no player can improve by changing his strategy, given that all the other players keep their strategies fixed. The idea is that a Nash equilibrium corresponds to a stable state of the game since none of the players has interest in deviating unilaterally from their current strategy and therefore is in some sense rational. Formally, we define it as follows

Definition 2 *Let $\mathcal{G}$ be a concurrent game with preference relations $(\precsim_A)_{A\in\mathrm{Agt}}$ and let s be a state. Let $\sigma = (\sigma_A)_{A\in\mathrm{Agt}}$ be a strategy profile with $\mathrm{Out}(\sigma, s) = \{\pi\}$. Then σ is a Nash equilibrium from s if for every $B \in \mathrm{Agt}$ and every $\sigma'_B \in \mathrm{Strat}^B$ with $\mathrm{Out}(\sigma[\sigma_B \mapsto \sigma'_B], s) = \{\pi'\}$ we have $\pi \succsim_B \pi'$.*

The concept was first introduced in [11] in normal-form games and has been analyzed quite extensively since then [12,13,16]. It was proven in [11] that a Nash equilibrium always exists in mixed strategies (where players can choose actions probabilistically), however a Nash equilibrium in pure strategies does not always exist. Indeed consider the matching pennies game in Figure 2.

In this game player A wins if the two players choose the same action and player B wins if th two players choose different actions. This is modelled by a reachability game starting in t_0 where A has reachability objective $\{t_1\}$ and B has reachability objective $\{t_2\}$. There is no pure Nash equilibrium in this game, since in any strategy profile the losing player can deviate from his strategy and improve. We will only focus on pure Nash equilibria which will be called Nash equilibria in the sequel. Our study of Nash Equilibria is motivated by the idea of modelling distributed systems as games, where players correspond to programmable devices (possibly with different goals) and strategies correspond to programs. Then a Nash equilibrium is in some sense an optimal configuration of a system, since no device has any interest in deviating from the chosen program. A nice property of pure Nash equilibria compared to Nash equilibria in mixed strategies is that pure Nash equilibria will correspond to deterministic programs, whereas Nash equilibria in mixed strategies correspond to randomized programs.

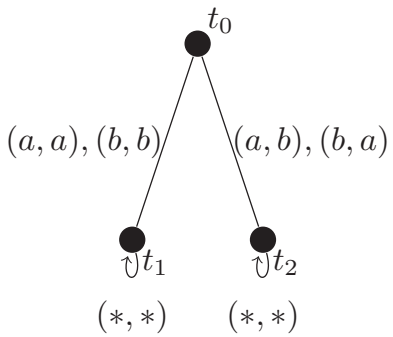


Fig. 2. Matching pennies game

The computational problem of deciding existence of pure strategy Nash equilibria in concurrent games are analyzed for different types of objectives in [3,4,5,6,7,18] with the same setting used here. We will focus our attention on the complexity results obtained on the following computational problems for different types of objectives.

Definition 3 (Existence Problem) *Given a game $\mathcal{G}$ and a state s does there exist a symmetric Nash equilibrium in $\mathcal{G}$ from s?*

Definition 4 (Constrained Existence Problem) *Given a game $\mathcal{G}$, a state s and two vectors u^A and w^A for each player A, does there exist a symmetric Nash equilibrium in $\mathcal{G}$ from s with some payoff $(v^A)_{A\in\mathrm{Agt}}$ such that $u^A \lesssim v^A \lesssim w^A$ for all $A \in \mathrm{Agt}$?*

Note that since there can be multiple equilibria with different payoffs it is also interesting to search for equilibria with particular payoffs or bounded payoffs. This is the motivation for considering the constrained existence problem as well as the existence problem. As we will see in the results section in many subclasses of concurrent games the complexities of the two problems are the same.

3 Results

As illustrated by the matching pennies game there does not necessarily exist a pure Nash equilibrium in a concurrent game. In finite normal-form games, there always exists a Nash equilibrium in mixed strategies and therefore a natural question is to ask whether this also holds for concurrent games. However, according to the following theorem this is not the case

Theorem 5 *There exists concurrent games with no Nash equilibria in mixed strategies.*

Proof. (sketch) Consider the two-player game in Figure 3 illustrating the snowball game [15,18].

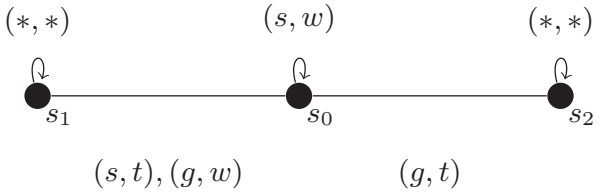


Fig. 3. Concurrent game with no Nash equilibria in mixed strategies

In this game, player A hides behind a hill and wishes to go to his house. At each time step he can either go (g) to the house or stay (s) in safety behind the hill. Player B is waiting with a snowball and at each time step he can either throw (t) or wait (w). If player A goes at the same time as player B throws, player A gets hit and loses. If he goes and player B does not throw, he does not get hit and wins. If player B throws and player A waits, then player A wins since

player B only has one snowball. Finally, if player A stays forever, then player B wins. This is modelled as a concurrent game with Büchi objectives where player A has the Büchi objective $\{s_1\}$ and player B has the Büchi objective $\{s_0, s_2\}$. It can be shown that this game has no Nash equilibrium, even in mixed strategies. It can be shown that player A can choose a strategy that ensures a win with probability $1 - \epsilon$ for any $0 < \epsilon \leq 1$, but not with probability 1. This is called limit-sure winning. Now, in any strategy profile where player A plays a strategy which is sure to win with probability at least $1 - \epsilon$ he wins with this probability or a higher probability (in case B uses a bad strategy). If he wins with a higher probability, then B can deviate and improve such that A wins with probability $1 - \epsilon$. Otherwise, player A can improve by choosing a strategy with a smaller ϵ. Thus, there cannot be a Nash equilibrium.

□

We now turn our attention to the (constrained) existence problem for pure Nash equilibria and present some of the main results that have been proved in the literature. Our first result is an undecidability result

Theorem 6 *The (constrained) existence problem is undecidable, even in the case of 2 players.*

Proof. See [7,18]. The proof is done by a reduction from the halting problem for Minsky machines.

□

This result is bad news, however for many types of objectives we can obtain a decidable (constrained) existence problem with a complexity that is not too bad. Results on complexity from [3,4,5,6] are shown in Figure 4.

4 Conclusion

We have motivated the use of applying the concept of Nash equilibrium in concurrent games for program synthesis in distributed systems where devices can have different objectives. In addition we have presented an undecidability result as well as a list of complexity results for various classes of objectives. A number of these cases have effective running times which makes it plausible that the technique can be used in practice.

There are still many possible extensions and open problems in the area. The game definitions can be altered to take into account thing like imperfect information, quantitative objectives and timing to make them able to model even more interesting situations. Trying to develop effective algorithms for finding Nash equilibria in games with these extra features is an interesting extension to the work presented in this paper.

Reachability Objectives

Preorder	Existence	Constr. Existence
Disjunction, Maximize	NP-c	NP-c
Parity	NP-h, in PSPACE	NP-h, in PSPACE
Subset	NP-c	NP-c
Conjunction, Counting	PSPACE-c	PSPACE-c
Lexicographic	PSPACE-c	PSPACE-c
(Monotonic) Boolean Circuit	PSPACE-c	PSPACE-c

Safety Objectives

Preorder	Existence	Constr. Existence
Conjunction	NP-c	NP-c
Parity, Maximize	PSPACE-c	PSPACE-c
Subset, Disjunction	PSPACE-c	PSPACE-c
Counting, Lexicographic	PSPACE-c	PSPACE-c
(Monotonic) Boolean Circuit	PSPACE-c	PSPACE-c

Büchi Objectives

Preorder	Existence	Constr. Existence
Maximize*, Disj., Subset	P-c	P-c
Conj., Lexicographic	P-h, in NP	NP-c
Counting	NP-c	NP-c
Monotonic Boolean Circuit	NP-c	NP-c
Parity	coNP-h, in $P_{||}^{NP}$	$P_{||}^{NP}$-c
Boolean Circuit	PSPACE-c	PSPACE-c

* Has been implemented in the tool Praline by Romain Brenguier.

Fig. 4. Complexity results of finding pure Nash equilibria in concurrent games with reachability, safety and Büchi objectives for different types of preorders

References

1. *R. Alur, T. A. Henzinger & O. Kupferman*, Alternating-Time Temporal Logic, Journal of the ACM, Vol. 49, No. 5, September 2002, pp. 672-713
2. *L. de Alfaro, T. A. Henzinger & R. Majumdar*, From Verification to Control: Dynamic Programs for Omega-Regular Objectives. LICS 2001: 279-290
3. *P. Bouyer, R. Brenguier, N. Markey & M. Ummels*, Concurrent Games with Ordered Objectives, Research report LSV-11-22, Version 2 - January 11, 2012
4. *P. Bouyer, R. Brenguier, N. Markey & M. Ummels*, Nash Equilibria in Concurrent Games with Büchi Objectives, in FSTTCS'11, LIPIcs, Mumbai, India, 2011. Leibniz-Zentrum für Informatik
5. *P. Bouyer, R. Brenguier & N. Markey*, Nash Equilibria for Reachability Objectives in Multi-Player Timed Games, Research report LSV-10-12 - June 2010
6. *P. Bouyer, R. Brenguier & N. Markey*, Nash equilibria in concurrent games, Part 1: Qualitative Objectives, Preliminary Version
7. *P. Bouyer, N. Markey & S. Vester*, Nash Equilibria in Symmetric Game Structures, To appear, 2012.
8. *K. Chatterjee, L. de Alfaro & Thomas A. Henzinger*, Qualitative concurrent parity games. ACM Trans. Comput. Log. 12(4): 28 (2011)
9. *V. Gripon & O. Serre*, Qualitative Concurrent Stochastic Games with Imperfect Information in Proceedings of 36th International Colloquium of Automata, Languages and Programming, Springer, Lecture Notes in Computer Science, Rhodes, Greece, pp. 200–211, July 2009.
10. *R. Mazala*, Infinite Games, E. Grädel et al. (Eds.): Automata, Logics, and Infinite Games, LNCS 2500, pp. 23-38. Springer 2002
11. *J. F. Nash Jr.*, Equilibrium points in n-person games. Proc. National Academy of Sciences of the USA, 36(1):48-49, 1950.
12. *M. J. Osborne & A. Rubinstein*, A Course in Game Theory, MIT Press, 1994
13. *C. Papadimitriou*, The complexity of finding Nash equilibria, Chapter 2 in Algorithmic Game Theory, edited by N. Nisam et al., Cambridge University Press, 2007
14. *J.-F. Raskin, K. Chatterjee, L. Doyen & T. A. Henzinger*, Algorithms for Omega-Regular Games with Imperfect Information. Logical Methods in Computer Science 3(3) (2007)
15. *O. Serre*, Game Theory Techniques in Computer Science - Lecture Notes, MPRI 2011-2012, January 4, 2012
16. *Y. Shoham & K. Leyton-Brown*, Multiagent Systems - Algorithmic, Game-Theoretic, and Logical Foundations, Cambridge University Press, 2009
17. *W. Thomas*, Infinite games and verification. In Proceedings of the International Conference on Computer Aided Verification CAV'02, volume 2404 of Lecture Notes in Computer Science, pages 58-64. Springer, 2002
18. *S. Vester*, Symmetric Nash Equilibria. Master thesis from Ecole Normale Superieure de Cachan, 2012

Belief Revision in the GOAL Agent Programming Language

Johannes S. Spurkeland

DTU Informatics

Abstract. In many cases agents in a multi-agent system may find themselves in a situation where inconsistencies arise. In order to properly deal with these, a good belief revision procedure is required. This paper attempts to illustrate the usefulness of such a procedure: a certain belief revision algorithm is considered in order to deal with inconsistencies and particularly the issue of inconsistencies and belief revision is examined in relation to the GOAL Agent Programming Language.

1 Introduction

When designing artificial intelligence, it is desirable to mimic the human way of reasoning as closely as possible to obtain a realistic intelligence albeit still artificial. This includes the ability to not only construct a plan for solving a given problem but also to be able to adapt the plan or discard it in favor of a new. In these situations the *environment* the *agents* (entities of artificial intelligence acting in the environment) act in should be considered *dynamic* and complicated as is the world we know. This will lead to situations where an agent's beliefs may be *inconsistent* and need to be revised. Therefore an important issue in the subject of modern artificial intelligence is that of *belief revision*.

This paper presents an algorithm for belief revision proposed in [9] and shows some examples of situations where belief revision is desired in order to avoid inconsistencies in an agent's knowledge base. The agent programming language GOAL will be introduced and belief revision will be discussed in this context.

2 Motivation

In many situations assumptions are made in order to optimize and simplify an artificially intelligent system. This often leads to solutions which are elegant and planning can be done without too many complications. However, such systems tend to be more difficult to realize in the real world – simply because the assumptions made are too restrictive to model the complex real world. E.g. human thoughts are themselves inconsistent as considered in [11]. It also considers an example of an expert system from [12] where the classical logical representation

of the experts' statements leads to inconsistency when attempting to reason with it. This is an example where it is not possible to uniquely define the cause and effect in the real world. Another example is where other entities actually have malicious intentions as e.g. when using multi-agent systems to model computer security; agents may represent hackers or other attackers on the system in question. One should also keep in mind that it might not always be possible or it might be too overwhelming to foresee all consequences and outcomes the environment provides – as in the real world which is constantly changing.

An example will now be proposed which is inspired by [3] and [4] where an agent-based transport information system is considered in order to improve the parking spot problem. In this paper cars are considered as autonomous entities which drive their passengers around the city. Each car thus poses as an agent. Such an agent may have the following trivial rules for how to behave in traffic lights:

$$green(X) \rightarrow go(X) \tag{1}$$
$$red(X) \rightarrow stop(X) \tag{2}$$

Furthermore assume that the agent can reason that if the brakes malfunction of a car it cannot stop:

$$failing(brakes, X) \rightarrow \neg stop(X) \tag{3}$$

Imagine now the situation where the light is *perceived* as green and a car in the crossing lane sends to the agent that its brakes fail. I.e. the agent now also believes the following:

$$green(me) \tag{4}$$
$$red(other) \tag{5}$$
$$failing(brakes, other) \tag{6}$$

This situation will now lead to inconsistencies when the agent attempts to reason with its beliefs. From (2) and (5) it is straight forward to deduce $stop(other)$ whereas from (3) and (6) $\neg stop(other)$ is deduced. Furthermore adding rules for the mutual exclusion of the *go* and *stop* predicates and having the rule $go(other) \rightarrow stop(me)$ saying to stop if the other does not, the agent will also be able to deduce that it both should go and should not go. The obvious choice for the agent here is to discard the thought of going onwards and stop to let the other car pass.

Assume now that the passenger of the car wants to go shopping and therefore the agent needs to find an available parking lot. To represent that the agent wants to get to the destination of the shop, it has the following:

$$dest(shop_1) \tag{7}$$

The agent currently does not know the whereabouts of the shop the passenger wants to go to so it broadcasts a request for such information. The agent receives a response:

$$dest(shop_1) \rightarrow goto(lot_1) \tag{8}$$

This basically tells the agent that if it wants to reach shop 1 it needs to get to parking lot 1. This is straight forward; however, shortly after the agent receives a second response from a third agent:

$$dest(shop_1) \rightarrow \neg goto(lot_1) \tag{9}$$
$$dest(shop_1) \rightarrow goto(lot_2) \tag{10}$$

The third agent has experienced that parking lot 1 is full which makes it send the first rule. It then also sends the second rule as a *plan* for getting parked desireably and thereby enabling the passenger to reach the destination shop.

Blissfully adding both responses to the belief base will, however, lead to inconsistencies again. Obviously (7) can be used with (8) and (9) to obtain $goto(lot_1)$ and $\neg goto(lot_1)$, respectively. Since the agent does not currently have any more information about the two, it does not know which of them to trust.

For more examples refer to e.g. [8]. They are of a more theoretical nature but may illustrate how one can deal with inconsistencies efficiently. The algorithm used there will be considered in more details in next section.

3 Belief Revision Algorithm

The algorithm considered in this paper is the one proposed by [9] (and [8]). The basic principle of the algorithm is to keep track of what beliefs the agent has and how the agent may *justify* these beliefs. The idea is basically the same as when humans reason – if two contradictory beliefs arise one of them is selected and the other discarded.

The agent is defined as having beliefs consisting of ground literals, l or $\neg l$, and rules. The rules take the form of universally quantified positive Horn clauses, $l_1 \wedge l_2 \wedge \ldots \wedge l_n \rightarrow l$ and the agent is required to reason with a weak logic, W, which only has generalised modus ponens as inference rule which formally is as follows where δ is a substitution function replacing all variables with constants:

$$\frac{\delta(l_1), \ldots, \delta(l_n) \qquad l_1 \wedge \ldots \wedge l_n \rightarrow l}{\delta(l)} \tag{GMP}$$

The approach is now to detect inconsistencies in the belief base. Notice this is a simple rule, $l \wedge \neg l \rightarrow \neg consistent(l)$. If an inconsistency is detected, the least preferred belief is removed. Furthermore the rules from which the belief can be derived are removed along with the beliefs which can only be derived from the removed belief. This process of removing beliefs is referred to as *contraction* by beliefs. This assumes two things. First of all that track is kept of how the beliefs relate to one another and second that there is a measure of how prefered a belief is.

To deal with the first problem, the notion of justifications is used. A justification is a pair, (A, s), of a belief A and a support list s. The support list contains the rule used to derive A and all the premises used for firing the rule. Thus the percepts or initial knowledge will have an empty support list. The

justifications for $green(me)$ and $\neg stop(other)$ from the example in section 2 are then as follows:

$(green(me), [])$
$(\neg stop(other), [failing(brakes, other), failing(brakes, X) \rightarrow \neg stop(X)])$

Notice that a belief may have several justifications. If the light had been green for the other and there was an exclusivity rule $go(X) \rightarrow \neg stop(X)$ then $\neg stop(other)$ would also have the justification:

$$(\neg stop(other), [go(other), go(X) \rightarrow \neg stop(X)])$$

Having this data structure, the beliefs and justifications can be regarded as a directed graph: incoming edges from the justifications of a belief and outgoing edges to justifications containing the belief in its support list. Figure 1 shows such a graph for the first part of the presented example. The elements at an odd depth are justifications whereas elements of an even depth are beliefs. One may also notice that the agent has the two contradictory nodes which mean either of the two subgraphs holding one of the two nodes must be contracted.

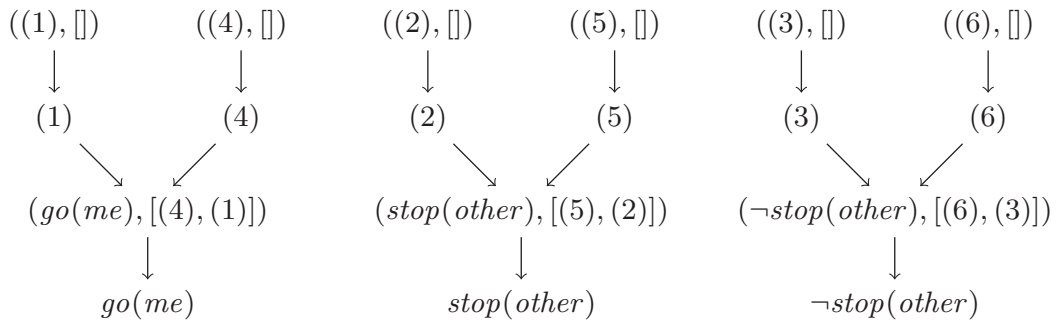


Fig. 1. Graph over the beliefs and justifications in the first part of the example from section 2.

In [7] the successors of a belief (justifications with the belief in the support list) are denoted dependencies. The algorithm is then simply considered to have a list of justifications and dependencies for each belief and recursively traverses the elements of these two lists to remove beliefs which are justifying or justified only by the belief selected for contraction. More specifically algorithm 1 provides pseudo code for the contraction algorithm similar to that from [7]. The contraction algorithm assumes that an inconsistency has been detected and the least preferred belief is given as input. The $w(x)$ function takes a support list as input and returns the least preferred belief of that list. It is also worth noting that the belief base needs to be in the quiescent setting in order for the algorithm to work properly. This is due to that if more beliefs can be inferred, the belief for contraction might be inferred again by other beliefs than the ones removed by the algorithm.

Algorithm 1 Contraction by belief A

for all $j = (B, s) \in \mathit{dependencies}(A)$ **do**
  $\mathit{remove}(j)$
end for
for all $j = (A, s) \in \mathit{justifications}(A)$ **do**
  if $s = []$ **then**
    $\mathit{remove}(j)$
  else
    $\mathit{contract}(w(s))$
  end if
end for
$\mathit{remove}(A)$

It may be desireable for the algorithm to also contract beliefs which no longer have justifications because they have been removed in the contraction process cf. [7]. Even though keeping the beliefs in the belief base might not currently lead to inconsistencies, something is intuitively wrong with keeping beliefs which are derived from what has been labeled wrong or untrustworthy information.

The problem of measuring how preferred a belief is does not have a trivial solution. In fact the problem is passed on to the designer of the multi-agent system in question. The requirement from the perspective of the algorithm is that there is an ordering relation such that for any two beliefs it is decidable which one is more preferable to the other.

Consider again the example from section 2. Intuitively percepts of the agent should be of highest preference since the agent ought to trust what it itself sees. However, in the example this would lead to the contraction of the belief $\neg \mathit{stop}(\mathit{other})$ which will mean the agent will decide to move forward and crash with the other car. Instead here the belief received from the other agent should actually have a higher preference than the percepts. This is, however, in general not desireable as other agents might not be trustworthy. This is even though the other agents might have good intentions. Consider e.g. the plan exchange between the agents in the example. Here the first agent sends a plan for getting to the closest parking lot not knowing it is full. If the agent chose to follow this plan and on the way perceived a sign showing the number of free spots in the parking lot, this percept should be of higher preference than following the plan through. This illustrates the care which needs to be taken in the process of deciding upon non-preferred beliefs.

4 GOAL

GOAL is a language for programming rational agents and as such is a target of interest in relation to belief revision. This section will examine GOAL and how it conforms with belief revision of inconsistent information. First the basic concepts of GOAL are explained and afterwards belief revision is considered in

GOAL. This paper will not explain how to set up and use GOAL – instead the reader is referred to [6].

4.1 The Basics

The basic structure of a GOAL multi-agent system consists of an environment and the agents which interact within this environment. For details how this works see [5].

Agents may consist of the following components:

1. Knowledge
2. Beliefs
3. Goals
4. Module sections
5. Action specification

Knowledge is what is known to be true. It should be considered axioms and as such should not be allowed to be inconsistent. Beliefs, however, represent what the agent thinks is true and may be both false and ambiguous – i.e. different rules may lead to inconsistent beliefs. The representation of knowledge, beliefs and goals are all in a Prolog based manner by standard (see [5] for more details). Below is shown the light signal rules of the example – knowledge can be declared using the `knowledge` keyword instead.

```
beliefs {
  % Green and red light rules.
  go(X) :- green(X).
  stop(X) :- red(X).
}
```

The action specification follows a STRIPS-style specification (see e.g. [10, ch. 10]) and actions are defined using a name, parameters, preconditions and postconditions. Imagine that the car agent has a drive action which takes a destination as parameter and requires that it is desired to get there along with that it can go onwards. The *me* predicate is a built-in predicate and the action specification may now look as follows:

```
actionspec {
  /* Action for the car agent to drive towards Dest provided it is desired and
       that it can go onwards. */
  drive(Dest) {
    pre { dest(Dest), me(Me), go(Me) }
    post { true }
  }
}
```

The post condition may seem to be somewhat odd – driving somewhere ought to change the agent's state. This is due to thinking of the drive action as a *durative* action. In [5] it is distinguished between two types of actions: instantaneous and durative. Durative actions take time to perform whereas instant actions will finish immediately and thereby affect the system immediately. Therefore the approach seems to be to let the outcome of durative actions be a consequence of their effect on the environment. That way durative actions can also be interrupted and their effect might not (entirely) come through.

The environment is specified by means of an Environment Interface Standard – for more information about this standard [5] refers to e.g. [2]. The agent has a percept base which contains the percepts of the current reasoning cycle. These can be accessed by using the predicate *percept*.

The module sections define the agent's behaviour. There are two modules by standard: the event module and the main module. The purpose of the main module is to define how the agent should decide what actions to perform whereas the purpose of the event module is to handle events such as percepts and incoming messages so that the belief base is always up-to-date when deciding upon actions cf. [5]. Therefore the event module is always the first to run in an agent's reasoning cycle. To add the logic for actually making the agent believe the light perceptions, the following event module may be used where *insert* and *delete* are two built-in functions for adding and deleting beliefs, respectively:

```
event module {
  program {
    % Green and red light percepts - on.
    forall bel(percept(green(X)), not(green(X))) do insert(green(X)).
    forall bel(percept(red(X)), not(red(X))) do insert(red(X)).

    % Green and red light percepts - off.
    forall bel(green(X), percept(not(green(X)))) do delete(green(X)).
    forall bel(red(X), percept(not(red(X)))) do delete(red(X)).
  }
}
```

The *bel* predicate is a built-in predicate to indicate that the agent believes the argument(s) of the predicate.

Assume that the *drive* action takes an extra argument which defines the mode the car should drive in. Assuming that when low on gas it drives economically and if busy it drives fast, the following main module may define the choice of actions:

```
main module {
  program {
    % Drive economically if low on gas.
    if bel(low(gas), dest(X)) then drive(dest(X), eco).

    % Drive fast if busy.
    if bel(busy, dest(X)) then drive(dest(X), fast).

    % Drive comfortably otherwise.
    if bel(dest(X)) then drive(dest(X), comfort).

    % If no driving options the car simply idles.
    if true then skip.
  }
}
```

Notice that the precondition that the *drive* action must have a destination is actually obsolete now since this is ensured when deciding upon action in the main module. The main module may evaluate actions in different kinds of orders. The default is linear which means that e.g. the economical option is chosen over the others if applicable. The *skip* action is not built-in but may be defined by simply having *true* as pre- and postconditions.

Messaging between agents works by means of a mailbox system using `send` and `received` predicates. It is similar to how percepts are handled with the

main difference that the mailbox is not emptied every reasoning cycle. GOAL supports three moods of messages: indicative, declarative and interrogative. These three moods basically represents a statement, a request and a question and are denoted using an operator in front of the message (see [5]). Handling the message that the other agent's brakes fail can be done by adding the following:

```
% Add the belief received from another agent that its brakes fail.
if bel(received(A, failing(brakes, A))) then insert(failing(brakes, A))
  + delete(received(A, failing(brakes, A))).
```

Messages are required to be a conjunction of positive literals which is a rather crucial point. This means that agents cannot share rules or plans. It is simply not possible to send or receive the rules (8), (9) and (10) from second part of the example. This also means that the only rules an agent will ever have are the ones it starts with coded into it. This means that the graph of algorithm 1 will be less complex during runtime.

Another lack of expressiveness in GOAL is that of representing inconsistencies. This will be examined further in the next section.

4.2 Inconsistencies

While having the tools for implementing the algorithm dealing with inconsistencies, a rather crucial point is to be noted. Since the knowledge representation language of the belief base in GOAL is Prolog the rules will all take the form of positive Horn clauses with a positive consequent. This means only positive literals can be concluded using the rules in the belief base. Furthermore the action specifications ensure that if a negative literal is added then it is not actually added to the belief base. Instead if the positive literal is in the belief base it is removed and otherwise nothing happens. This is due to the closed world assumption. This is means that an agent will never be able to represent both the positive and the negative of a literal in its belief base. I.e. GOAL simply does not allow for representing inconsistencies.

In order to allow for the representation of inconsistencies, the notion of strong negation is introduced. In [1] this kind of negation does not rely on the closed world assumption. It explicitly says that the negation of a formula succeeds whereas negation as failure says that the formula does not succeed but also that it does not explicitly fail either. In other words, negation as failure can be read as "it is not currently believed that".

The basic principle is now to consider the strong negation, $\neg$, of a predicate as a positive literal. I.e. for literal, p, p' can be regarded as a positive literal with the meaning $\neg p$. In terms of Prolog this means querying p will succeed if p holds, fail if $\neg p$ holds and be inconsistent if both holds. In [1] it is furthermore considered possible to return the value unknown to such queries if neither p nor $\neg p$ can be proven in the current Prolog program. Another interesting observation they made is that the closed world assumption can be defined for any literal in the following way (where *not* denotes negation as failure):

$$\neg p(X_1, \ldots, X_n) \leftarrow not(p(X_1, \ldots, X_n))$$

In the terms of GOAL this means that it is fairly straight forward to introduce the notion of strong negation by using a `neg` predicate. There is no explicit problem in having both the belief and its negation in the belief base – it is required in the event module to check whether or not the belief base is still consistent by querying whether or not $neg(X), X$ follows from the belief base and act accordingly. It may be necessary to introduce a *pos* predicate denoting positive literals as GOAL does not allow for the query $bel(X)$.

4.3 Belief Revision

This section will consider how to implement the belief revision algorithm described in section 3 in GOAL.

Because the event module is the first thing which is executed in an agent's reasoning cycle and because it is desired to have an up-to-date belief base when performing belief revision, the belief revision algorithm should be implemented as the last procedure in the event module.

That GOAL relies on Prolog as representation language (in most cases) conforms well with the weak logic defined for use with the belief revision algorithm in [9] since prolog programs consists of Horn clauses and literals.

The question is now exactly how to associate and represent the justification and dependency lists. One idea is to simply let them be beliefs of the agent. This way one just has to make sure to add a justification when adding a belief. The rules for adding percepts to the agent from section 4.1 may be extended to also construct a justification:

```
% Add beliefs and justifications from red light percepts.
forall bel(percept(red(X)), not(red(X))) do insert(red(X))
  + insert(just(red(X),[],p)).
```

Similarly the justification can be deleted when the percept is no longer valid using the `delete` predicate. While the case is rather trivial when dealing with percepts (as they do not have any justifications) a similar approach may be taken for the actions and when adding beliefs from messages of other agents. Care needs to be taken though when considering actions since the motivation for taking actions are specified in the main module whereas the postconditions (i.e. effect) of the actions are specified in the action specification. It should be the conjunction of the motivation for taking an action and the preconditions of the action which support each of the added beliefs. There are several ways for obtaining this conjunction. A simple one is to simply let the action take them as parameters. Provided the action is instantaneous and adds the belief where the agent is at, the action specification could look like this:

```
actionspec {
% Action for the car agent to drive which also adds justifications.
  drive(Dest, Pre) {
    pre { dest(Dest), me(Me), go(Me), append(Pre, [dest(Dest), go(Me),
          action(drive)], S) }
    post { at(Dest), just(at(Dest), S) }
  }
}
```

The idea is to send the preconditions from the main module as a list in an argument to the action and then append that list to a list of preconditions of the action to obtain s. The *action* predicate then corresponds to the rule used for deriving the belief. In this case it is a plan which has been executed. Since actions and main modules cannot be altered dynamically, another predicate might be added as precondition say $not\ contracted(action(drive))$. If an action or particular instance of an action is then contracted using the belief revision algorithm, e.g. the belief $contracted(action(drive))$ may simply be added to invalidate any future run of that action. If the agent finds reason to believe in the action again, it may simply remove the belief again.

The case of durative actions is actually particularly simple and yet difficult to handle at the same time. It is simple in the sense that its effect comes in forms of percepts of changes in the environment and that is already handled. However, all percepts create a justification with an empty support list – i.e. there is no way of telling whether the percept is from a change in the environment due to an action the agent has performed or simply due to the environment itself (or even other agents interacting in the environment). In other words, durative actions cannot be contracted. This might seem fine in the sense that the agent will most likely not believe anything new immediately and as a direct consequence of the durative actions.

When having added the datatypes for the justifications the idea is to check for inconsistencies as the last thing in the event module. Then if two contradictory beliefs are found, the least preferred of the two is marked for contraction. The actual contraction may then be performed in several ways. One is to have three blocks. In the first all the positive beliefs marked for contraction are contracted and in the second all the negative. These two blocks follow the contraction algorithm 1 with the exception of the recursive call. This is what the third block is for. The recursive call is emulated by marking all the least preferred elements of the support lists to contract and in the third block the first of the recursive calls are then performed on these. Again the recursive call is emulated by marking beliefs for contraction. Now, however, the sequentiality of the program will make it have passed the three blocks for contraction. The idea is now to not let the agent perform any actions before there are no beliefs marked for contraction. Then the agent will do nothing and the next reasoning cycle will start. This time it will go directly to the third cycle and continue emulating recursive calls by doing this until no more beliefs are marked for contraction and the agent is allowed to do actions again. Below is code to illustrate this – the positive and negative blocks have been left out since they are similar to the recursive block.

```
% Detect inconsistencies
#define poscontract(X)  bel(p(neg(X),Pn), p(X,Pp), Pn > Pp).
#define negcontract(X)  bel(p(neg(X),Pn), p(X,Pp), Pn < Pp).

% Recursive contraction
listall C <- contract(X) do {
  forall just(Y, S), member(Z, C), member(Z,S) do delete(just(Y, S)).
  forall just(Y, []), member(Y, C) do delete(just(Y, _)).
  forall member(Y, C), just(Y, S) bel(w(just(Y,S), Z)) do insert(contract(Z)).
  forall member(Y, C) do delete(Y).
}
```

While it may work, this approach is not optimal. If the performance of an agent is important, it is not good to have it idling for several cycles because it is revising its thoughts. This is even though the number of recursive calls most likely is not that high because of the simplified graph due to static rules as mentioned in section 4.1. More optimally would be to have a Prolog procedure which can make use of recursion in its definition and then simply call this on the beliefs for contraction. One might e.g. import such a procedure in the knowledge section such that it will mark all the beliefs for contraction recursively and in the next cycle actually do the removal of them. This way only one extra cycle is wasted when having to contract beliefs.

Up until now the implementation of the preference relations has not been discussed. In the above code it is assumed that a preference of a belief is added as a predicate cf. justification. Furthermore it is assumed that when a support list of a justification being added is not empty, a predicate w is added having the justification and a minimum preferred belief. This simplifies the least preferred belief queries; however, one should keep in mind that these predicates all should be deleted when also deleting the corresponding belief.

Finally a remark on the quiescent setting requirement. In [7] the algorithm was considered with regards to implementation in Jason. There was a drawback that the quiescent setting could not be guaranteed. Here again an action may not be activated in a long time but it may still lead to inconsistencies. Therefore it cannot be completely guaranteed here either. However, when querying the belief base for inconsistencies (which is done every reasoning cycle) the Prolog engine will attempt to evaluate all the rules in the belief base in order to search for a proof. This means that the quiescent setting is for the belief base but not for the action rules.

5 Conclusion

It has been argued why belief revision is an important issue. A particular algorithm for belief revision has been considered. It has the advantages of being efficient and straight forward to implement; however, it has the disadvantages that it is only defined for a weak logic of the agent and that it requires the non-trivial question of a preference ordering. Issues of such an ordering has been pointed out. Furthermore issues with deleting information which albeit inconsistent may still be important in many cases has been pointed out.

GOAL has been examined in relation to belief revision. It has the strengths of using logic programming which means that it is very easy to learn for people with a background in logic programming and it provides elegant logical solutions. Furthermore the restrictions of logic programming conform well with the restrictions of the weak logic of the belief revision algorithm. However, it has been identified that at current state the GOAL language is actually more restrictive than required for the algorithm which results in that inconsistencies cannot be represented at all. The introduction of strong negation has been con-

sidered in order to mitigate this problem. Another less critical issue with GOAL is that it does not provide the means for plan sharing.

References

1. Chitta Baral and Michael Gelfond. Logic Programming and Knowledge Representation. The Journal of Logic Programming, 19: 73-148, Elsevier 1994.
2. Tristan M. Behrens, Koen V. Hindriks and Jürgen Dix. Towards an environment interface standard for agent platforms. Annals of Mathematics and Artificial Intelligence, 61: 261-295, Springer 2011.
3. Nesrine Bessghaier, Mahdi Zargayouna, and Flavien Balbo. An Agent-Based Community to Manage Urban Parking. In Yves Demazeau et al. (Eds.): Advances on PAAMS, AISC 155, 17-22, Springer 2012.
4. Nesrine Bessghaier, Mahdi Zargayouna, and Flavien Balbo. Management of Urban Parking: An Agent-Based Approach. In Allan Ramsay and Gennady Agre (Eds.): AIMSA 2012, LNCS 7557, 276-285, Springer 2012.
5. Koen V. Hindriks. Programming Rational Agents in GOAL, May 2011.
6. Koen V. Hindriks and Wouter Pasman. GOAL User Manual, July 2012.
7. Andreas S. Jensen and Jørgen Villadsen. Plan-Belief Revision in *Jason*. Technical University of Denmark, 2012.
8. Hai H. Nguyen. A Belief Revision System. B.Sc. thesis, University of Nottingham, 2009.
9. Hai H. Nguyen. Belief Revision in a Fact-Rule Agents Belief Base. In Anne Håkansson et al. (Eds.): KES-AMSTA 2009, LNAI 5559, 120-130, Springer 2009.
10. Stuart Russell and Peter Norvig. Artificial Intelligence – A Modern Approach. Pearson Education, third edition, 2010.
11. Johannes S. Spurkeland. Using Paraconsistent Logics in Knowledge-Based Systems. B.Sc. thesis, Technical University of Denmark, 2010.
12. Jørgen Villadsen. Paraconsistent Knowledge Bases and Many-Valued Logic. In Hele-Mai Haav and Ahto Kalja (Eds.): BalticDB&IS 2002, 2: 77-90, Institute of Cybernetics at Tallinn Technical University, 2002.

Organization-Oriented Programming in Multi-Agent Systems

Andreas Schmidt Jensen

DTU Informatics

Abstract. We discuss some of the limitations of traditional agent-centered multi-agent systems, and how these may be overcome by imposing a organizational structure upon the agents of the system. We discuss the issues that arise in such situation and focus on the conflicts that occur in the agent when it has to decide between committing to own desires and organizational obligations. We present a new approach based on qualitative decision theory which is able to resolve these conflicts.

1 Introduction

Agents taking part in a multi-agent system are usually seen as intelligent entities that autonomously are able to bring about (from their own perspectives) desirable states. In a fixed setting with a controlled number of agents and globally desirable states, the designer is most likely able to implement the agents such that their own desirable states coincide with the globally desirable states. Different programming tools exists for implementing autonomous agents in such setting, e.g. Jason [2] and GOAL [10].

In open societies agents often come from different sources and their desires cannot as such be assumed to coincide with the global desires. A suggestion is to impose an organization upon the agents which is able to influence the actions of the agent towards the desires of the organization. A number of formalisms and programming tools have been developed for this purpose [1,4,5,6,7,11,12].

When agents are constrained by an organization, their own goals may conflict with those of the organization. Previous work towards resolving such conflicts have often resulted in a solution in which desires and obligations are ordered a priori, where an agent either preferes desires over obligations or obligations over desires. This results in agents that are always selfish or always social. We argue in this paper that such distinction can be too hard; even a selfish agent would in some cases be required to prefer certain obligations over its desires. We consider an approach on how to resolve such conflicts which is based on work in the area of qualitative decision theory by Boutilier [3], where the expected consequences of bringing about a state is considered. We show that this results in agents that are not always either social or selfish, but are instead able to decide based on the consequences.

The paper is organized as follows: In section 2 we discuss organization-oriented multi-agent systems and the issues that arise when such systems and its agents are implemented. We discuss in section 2.1 the issues that arise when an agent enacts a role and has to choose between desires and obligations. In section 3 we present a new approach on how to solve such conflicts without having to put the agents into the categories "selfish" or "social". Finally we conclude our work and discuss future directions in section 4.

2 Multi-Agent Organizations

Within the area of multi-agent systems much work is done towards making the organization of such systems explicit [7,9,11,13]. In an agent-centered multi-agent system (ACMAS) the agent is in focus whereas in an organization-centered multi-agent system (OCMAS) the concern is the organization; i.e. the structure of the multi-agent system. Naturally all multi-agent systems have a structure, but it is most often implicitly defined by the agents and their relations. By explicitly defining the organization it is possible to focus on *what* the agents should do without at the same time deciding *how* they should do so. In other words, the organization makes it possible to create the structure of the system without specifying details about the implementation. In [9] the following assumption are made about ACMAS:

- Agents are free to communicate with any other agent.
- All of the agent's services are available to every other agent.
- The agent itself is responsible for constraining its accessibility from other agents.
- The agent should itself define its relation and contracts with other agents.
- Agents are supposed to be autonomous and no constraints are put on the way they interact.

These assumptions suggest that agents have a lot of freedom to follow their own desires and intentions. They are not necessarily selfish, but on the other hand no constraints are put upon them to ensure coherence between desires and intentions of different agents. This makes it very difficult to ensure that agents coming from different societies can cooperate in achieving common goals. It is therefore suggested to impose an organization upon the agents, which provides structure by partitioning the agents and a direction by specification of expectations.

The main principles of OCMAS are as follows [9]:

Principle 1: The organizational level describes the "what" and not the "how".

Principle 2: The organization provides only descriptions of *expected* behavior. There are no agent description and therefore no mental issues at the organizational level.

Principle 3: An organization provides a way for partitioning a system, each partition constitutes a context of interaction for agents.

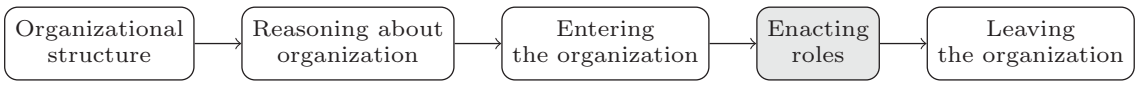


Fig. 1. The relevant stages in organization-oriented programming and the focus of this paper.

Using these principles it is possible to design an organizational model by which agents, if participating in the organization, will be influenced towards achieving the goals of the organization. An agent participates in an organization by choosing to enact one or more of the roles of the organization. The purpose of joining an organization is two-fold: the agent is able to fulfill some of the goals of the organization, and interaction and cooperation with other participants enables the agent to fulfill some of its own goals. In order for the agent to fulfill the purpose of a role, the expected behavior must be defined, i.e. constraints must be put on the role-enacting agent in order for the organization to ensure that progress is made towards fulfilling its goals.

In [13] three phases of organizational participation from the agent's point of view are considered; entering, enacting roles and leaving. The agent should be able to decide whether it benefits from entering the organization, which roles can it enact and finally whether it has fulfilled its purpose in the organization and should leave. When looking at a broader picture, a few more phases come to mind. First of all the organizational structure must be defined by formalizing the notion of roles, groups, interaction, obligations. Several formalisms have been proposed for this, e.g. *Moise and S-Moise*$^+$ [11,12] in which the organization is defined via a structural, functional and deontic specification. Second of all we should be able to reason about the characteristics of the organization. The *logic of agent organizations* [8] is a logical formalism for (1) defining the structural requirements and (2) being able to analyse the organization to verify that it is well-defined and efficient. The five overall phases and their dependencies are depicted in figure 1.

In the following we turn to the fourth phase, enacting roles, and focus on the conflicts that arise when an agent has to choose between its own desires and the obligations imposed by the organization via the role(s).

2.1 Conflicting influences

Whenever an agent has more than one desire, it needs to be able to decide which desire(s) it intends to commit to. This is even more the case when the agent enters an organization since its behavior is further constrained by obligations towards the organization. In this paper we call obligations and desires "decision influences". Having these should influence how the agent chooses to act. Should it commit to bringing about the state in the obligation or rather its own desires? Several approaches are proposed on how to solve this problem [1,4,5,6].

For instance, the BOID architecture [4] impose a strict ordering between beliefs, obligations, intentions and desires, and different agent types emerge from

such orderings, such as a social agent, which prefers obligations before desires, or a selfish agent which prefers desires before obligations.

We believe this ordering is too strong; if an agent is social it will always choose obligations over desires, and vice versa for selfish agents. This might not always be appropriate. For instance, a selfish agent might desire not to go to work, but if the consequence of not fulfilling the obligation of going to work is severe (i.e. getting fired), even a selfish agent should consider this consequence before deciding not to go to work. We consider a different approach which allows the agent to take into account the *consequences* of bringing about a state.

For instance, if it rains the agent can decide whether it should bring an umbrella. Not bringing an umbrella has the expected consequence that the agent will get wet. Choosing to bring an umbrella has the expected consequence that it will not get wet. Of course, the agent has certain beliefs, which also should have impact on the decision. If it believes the rain will stop very soon, it might choose not to bring an umbrella.

3 Modelling Influence and Consequence

We base our work on the Logic for Qualitative Decision Theory by Boutilier [3] by extending the notion of preference to allow multiple modalities in order to represent individual preferences. An agent has the ultimate desire of achieving the goals to which it is committed. This can be modelled by a possible worlds-model in which the agent has achieved its goal when it is in a world where those goals holds. The most preferred world in an ideal setting is the world in which all of the agents goals are achieved. However, such world is often unreachable, for a number of reasons: the agent could have contradicting goals, other agents could prevent the agent from achieving all of its goals, an organization could impose obligations which contradicts the agents goals and so on. We need to be able to order the worlds in a preference relation in order to choose the most preferred world in a sub-ideal situation.

The consequence of bringing about a state (achieving a goal) should however also be taken into account. If the consequence of achieving a personally desirable goal is to be fired from your workplace, even though the desire was more preferred than the obligations from work, it might not be reasonable to pursue.

We work with models of form:

$$M = \langle W, Ag, \leq_P^1, \ldots, \leq_P^n, \leq_N, \pi \rangle$$

where W is the non-empty set of worlds, Ag is the set of agents, $\leq_P^i$ is the transitive, connected preference ordering for each agent[1], $\leq_N$ is the transitive, connected normality ordering, and π is the valuation function. The normality ordering is used to model how likely each world is, e.g. it is normally cold when it is snowing.

[1] We adopt the notion by Boutilier and others that we prefer minimal models, so $v \leq_P^i w$ denotes that according to agent i, v is at least as preferred as w.

The semantics are given as follows:

$$
\begin{aligned}
M, w \models p &\iff p \in \pi(w) \\
M, w \models \neg\varphi &\iff M, w \not\models \varphi \\
M, w \models \varphi \wedge \psi &\iff M, w \models \varphi \wedge M, w \models \psi \\
M, w \models \Box^i_P \varphi &\iff \forall v \in W, v \leq^i_P w, M, v \models \varphi \\
M, w \models \overleftarrow{\Box}^i_P \varphi &\iff \forall v \in W, w <^i_P v, M, v \models \varphi \\
M, w \models \Box_N \varphi &\iff \forall v \in W, v \leq_N w, M, v \models \varphi \\
M, w \models \overleftarrow{\Box}_N \varphi &\iff \forall v \in W, w <_N v, M, v \models \varphi
\end{aligned}
$$

We can define the other operators ($\vee, \rightarrow, \Diamond, \overleftarrow{\Diamond}$) as usual. Finally we can talk about a formula being true in all worlds or some worlds: $\overleftrightarrow{\Box}^i_P \varphi \equiv \Box^i_P \varphi \wedge \overleftarrow{\Box}^i_P \varphi$ and $\overleftrightarrow{\Diamond}^i_P \varphi \equiv \Diamond^i_P \varphi \vee \overleftarrow{\Diamond}^i_P \varphi$, respectively (similarly for normality). The follow axiom captures their interaction

PN $\quad \overleftrightarrow{\Box}^i_P A \equiv \overleftrightarrow{\Box}_N A$ for any i,

A full axiomatization of QDT-logic can be found in [3], where the following abbreviations are also defined:

$$
\begin{aligned}
I(B \mid A) &\equiv \overleftrightarrow{\Box}^i_P \neg A \vee \overleftrightarrow{\Diamond}^i_P (A \wedge \Box^i_P (A \rightarrow B)) && \text{(Conditional preference)} \\
A \leq^i_P B &\equiv \overleftrightarrow{\Box}^i_P (B \rightarrow \Diamond^i_P A) && \text{(Relative preference)} \\
T(B \mid A) &\equiv \neg I(\neg B \mid A) && \text{(Conditional tolerance)} \\
A \Rightarrow B &\equiv \overleftrightarrow{\Box}_N \neg A \vee \overleftrightarrow{\Diamond}_N (A \wedge \Box_N (A \rightarrow B)) && \text{(Normative conditional)}
\end{aligned}
$$

The abbreviations state that B is ideally true if A holds, A is at least as preferred as B, B is tolerable given A and that B normally is the case when A is, respectively. We furthermore define the following abbreviations to be used in the process of making a decision:

$$
\begin{aligned}
P \not\leq^i_P Q &\equiv \neg(P \leq^i_P Q) && \text{(Not as preferred)} \\
P <^i_P Q &\equiv (P \leq^i_P Q \wedge Q \not\leq^i_P P) && \text{(Strictly preferred)} \\
P \approx^i_P Q &\equiv (P \leq^i_P Q \wedge Q \leq^i_P P) \\
&\qquad \vee (P \not\leq^i_P Q \wedge Q \not\leq^i_P P) && \text{(Equally preferred)} \\
A \leq^i_{T(C)} B &\equiv (T(A \mid C) \wedge \neg T(B \mid C)) \vee \\
&\qquad ((T(A \mid C) \leftrightarrow T(B \mid C)) \wedge \\
&\qquad (A \leq^i_P B \vee A \approx^i_P B)) && \text{(Relative tolerance)}
\end{aligned}
$$

Thus A is at least as tolerable as B w.r.t C when either A is tolerable given C and B is not, or both A and B are tolerable given C (or both are not), and A is at least as preferred as B, or they are equally preferable. This means that even if neither are tolerable, they are still comparable.

3.1 Making a decision

We now show how the extended QDT-logic can be used to decide between conflicting desires and obligations. We define a model for use in an organizational setting as follows:

$$\mathcal{M}_C = \langle M, D, O, C, P, B \rangle,$$

where

- M is an extended QDT-model as defined above,
- D is for each agent the set of desires,
- O is the set of obligations,
- C is for each agent the set of controllable propositions[2],
- B is the belief base for each agent.

We define the set of potential consequences $C'(i)$ for an agent i such that if $\varphi \in C(i)$ then $\varphi, \neg\varphi \in C'(i)$.

Definition 1 (Expected consequences). *Given an agent i, its belief base $B(i)$ as a conjunction of literals, the set of potential consequences $C'(i)$ and a literal A. The expected consequences of bringing about state A, denoted $EC_i(A)$, is given by:*

$$EC_i(A) = \bigwedge C_A \text{ for all } C_A \in \{C_A \mid (B(i) \wedge A \Rightarrow C_A) \text{ where } C_A \in C'(i)\}$$

i.e. the conjunction of all literals C_A that are normally consequences of bringing about A given the current belief base. If there are no expected consequences, then $EC_i(A) = \top$.

Consider an agent i, and a normality ordering in which we have that

$$A \wedge \alpha \Rightarrow B, \qquad A \wedge \neg\alpha \Rightarrow C, \quad D \wedge \neg\alpha \Rightarrow E,$$

and belief base $B(i) = \{\alpha\}$. Then we have that $EC_i(A) = \{B\}$ and $EC_i(D) = \emptyset$. If $B(i) = \{\neg\alpha\}$, then $EC_i(A) = \{C\}$ and $EC_i(D) = \{E\}$.

An agent can make a decision by selecting from the set of potentially conflicting influences, $D(i) \cup O$, the most preferred influences having the most tolerable consequences.

Definition 2 (Decision). *Given an agent i, the set of influences $\mathcal{I}(i) = D(i) \cup O$ and the expected consequences $EC_i(A)$ for all $A \in \mathcal{I}(i)$, we can get the set of best influences (the decision) the agent should choose from, denoted by the function $Dec : Ag \to 2^{\mathcal{I}(i)}$ as follows:*

$$\begin{aligned} Dec(i) = \{A \mid\ & A \in \mathcal{I}(i), \text{ and} \\ & \text{for all } B \in \mathcal{I}(i), B \neq A, \text{ either} \\ & A <^i_P B, \text{ or} \\ & A \approx^i_P B \text{ and } EC(A) \leq^i_{T(A \vee B)} EC(B)\} \end{aligned}$$

[2] A controllable proposition is, roughly, a proposition which the agent is able to influence, directly or indirectly, by an action. E.g. *snow* is not controllable, whereas *work* is.

(a) Alice's preferences (b) Expectation

Fig. 2. The preference and normality induced by world expectations and the agent's preferences. We abbreviate *snow* with S, and *work* with W. A negated proposition is denoted with an overline, e.g. $\overline{S}$.

Given a model $\mathcal{M}$, an agent i can then choose an arbitrary literal from $Dec(i)$, since all of these are equally preferred and with equally tolerable consequences.

If there are no expected consequences of bringing about a certain state, i.e. if $EC(\varphi) = \emptyset$, then we consider the state tolerable, since we do not expect any consequences. Therefore for all other consequences, γ, we have to consider $\top \leq^i_{T(C)} \gamma$ and $\gamma \leq^i_{T(C)} \top$. Note that $T(\top \mid \varphi)$ is true iff φ is true in any world[3]. Furthermore, $\top \leq^i_P \varphi$ is always true, and $\varphi \leq^i_P \top$ is true iff φ is true in all worlds. Thus it is possible to make a decision even if some obligations or desires have no known consequences.

3.2 Case study

We consider Alice, who normally goes to work, but during a snow storm, she normally stays at home. In her ideal world, it does not snow, but if it snows, she preferrably stays at home. This induces structures for preference and normality such as the ones shown in figure 2(a) and 2(b). The preference model satisfies $I(\neg snow \mid \top)$ and $I(\neg work \mid snow)$ and the normality model satisfies $\top \Rightarrow work$ and $snow \Rightarrow \neg work$.

Alice furthermore has an obligation to go to work and a desire to stay at home. We show how to decide what Alice should do in different situations.

Thus we have a model where the possible worlds, and their valuations and orderings are given in figure 2. Furthermore, we have that $D(a) = \{\neg work\}$ and $C = O = \{work\}$.

We consider the situations when it does not snow and when it snows. In both situations we need to consider the relative preference between all literals in $D(a) \cup O = \{\neg work, work\}$ and how relatively tolerable their consequences are.

1. $\boldsymbol{KB} = \{\neg snow\}$

   Since there are no known consequences, both situations are equally tolerable,

[3] Since $T(\top \mid \varphi) \equiv \hat{\Diamond}^i_P \varphi \wedge \hat{\Box}^i_P (\neg\varphi \vee \Diamond^i_P (\varphi \wedge \top)$.

so we only need to consider the relative preference. Clearly, the two situations are equally preferable (both $work$ and $\neg work$ are at the bottom of figure 2(a)), so we get $Dec(a) = \{\neg work, work\}$.

2. $\boldsymbol{KB} = \{snow\}$
   Again there are no known consequences, but since it snows, we only consider Alice's preferred snow-worlds. Then $\neg work$ is more preferred than $work$. Thus we get $Dec(a) = \{\neg work\}$.

The first situation is not that interesting, because it does not help the agent make a decision between obligations and desires. However, since there are no known consequences of bringing about either state, this is the best we can do.

We now change the model by specifying that if it does not snow and Alice does not come to work, she can expect to get fired, formally $\neg work \wedge \neg snow \Rightarrow$ *fired*. Furthermore, ideally Alice does not get fired, $I(\neg \textit{fired} \mid \top)$. This induces a different kind of partial structure for normality and preference, shown in figure 3. We can then return to the situation where it does not snow, and now show that the consequence of not going to work when it does not snow normally results in getting fired.

(a) Alice's preferences (b) Expectation

Fig. 3. The preference and normality induced by world expectations and the agent's preferences, when the consequence of getting fired is considered. We abbreviate *snow* and work as before, and *fired* with F.

3. $\boldsymbol{KB} = \{\neg snow\}$
   We have that $EC(work) = \top$ and $EC(\neg work) = \textit{fired}$. We can then calculate $\top \leq^i_{T(work \vee \neg work)} \textit{fired}$ and $\textit{fired} \leq^i_{T(\neg work \vee work)} \top$. From the model, we can see that $\neg T(\textit{fired} \mid work \vee \neg work)$. From this we can conclude that while $work$ and $\neg work$ are equally preferable, the consequence of working is more tolerable than that of not working. Therefore $Dec(a) = \{work\}$.

Furthermore if $\boldsymbol{KB} = \{snow\}$ the agent still chooses $\neg work$, since *fired* is not a consequence of this. Note that at this point we have not labeled the agent as

"social" or "selfish" – the agents preference and the expected consequences are taking into account, and this leads to the results above. It chooses an obligation over a desire when the expected consequence of doing otherwise is intolerable. If the agent has the desire of leaving early, and this does not have the expected consequence of getting fired, the agent might very well choose to leave early, thereby choosing a desire over an obligation.

4 Conclusion

We have discussed agent-centered and organization-centered multi-agent systems. Certain characteristics of agent-centered multi-agent systems make it hard to ensure coherence between an agent's own desires and the global desires of the system. We have discussed how to solve this issue via some of the main principles of organization-centered multi-agent systems, which suggests constraining the agents using roles, groups and obligations.

We have argued that conflicts are prone to arise when agents enact roles in an organization, since their own desires may conflict the obligations of the role(s) they are enacting. We have proposed a way of solving such conflicts that uses qualitative decision theory rather than labeling the agents "selfish" or "social" in advance. This solution works by taking the consequence of bringing about a state into consideration, thus letting the agent take its preferences into account, without choosing something that results in an intolerable state. We have argued that this indeed lets the agents reach a decision without strictly preferring desires over obligations or vice versa.

Future work

We are currently working on building a prototype in Prolog, which given a world-model, and a preference and normality ordering, is able to decide which influence(s) to commit to. The idea is to integrate this prototype into the GOAL agent programming language to be able to let agents resolve conflicts in more advanced scenarios.

References

1. Natasha Alechina, Mehdi Dastani, and Brian Logan. Programming Norm-Aware Agents. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '12, pages 1057–1064, Richland, SC, 2012. International Foundation for Autonomous Agents and Multiagent Systems.
2. Rafael H. Bordini, Michael Wooldridge, and Jomi Fred Hübner. *Programming Multi-Agent Systems in AgentSpeak using Jason (Wiley Series in Agent Technology)*. John Wiley & Sons, 2007.
3. Craig Boutilier. Toward a Logic for Qualitative Decision Theory. In *In Proceedings of the KR'94*, pages 75–86. Morgan Kaufmann, 1994.

4. Jan Broersen, Mehdi Dastani, Joris Hulstijn, Zisheng Huang, and Leendert van der Torre. The BOID Architecture – Conflicts Between Beliefs, Obligations, Intentions and Desires . In *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 9–16. ACM Press, 2001.
5. F. Dignum, D. Morley, E. A. Sonenberg, and L. Cavedon. Towards Socially Sophisticated BDI Agents. In *Proceedings of the Fourth International Conference on MultiAgent Systems (ICMAS-2000)*, ICMAS '00, pages 111–118, Washington, DC, USA, 2000. IEEE Computer Society.
6. Frank Dignum, David Kinny, and Liz Sonenberg. From Desires, Obligations and Norms to Goals. *COGNITIVE SCIENCE QUARTERLY*, 2:2002, 2002.
7. Virginia Dignum. *A model for organizational interaction: based on agents, founded in logic*. PhD thesis, Universiteit Utrecht, 2004.
8. Virginia Dignum and Frank Dignum. A logic of agent organizations. *Logic Journal of the IGPL 20*, 1:283–316, 2012.
9. Jacques Ferber, Olivier Gutknecht, and Fabien Michel. From Agents to Organizations: An Organizational View of Multi-agent Systems. In *LNCS 2935: Procs. of AOSE03*, pages 214–230. Springer Verlag, 2003.
10. Koen V. Hindriks. Programming Rational Agents in GOAL. *Multi-Agent Programming: Languages, Tools and Applications*, 2:119–157, 2009.
11. Jomi Fred Hübner, Jaime Simão Sichman, and Olivier Boissier. A Model for the Structural, Functional, and Deontic Specification of Organizations in Multiagent Systems. In *Proceedings of the 16th Brazilian Symposium on Artificial Intelligence: Advances in Artificial Intelligence*, SBIA '02, pages 118–128, London, UK, 2002. Springer-Verlag.
12. Jomi Fred Hübner, Jaime Simo Sichman, and Olivier Boissier. S-moise+: A middleware for developing organised multi-agent systems. In *COIN I, volume 3913 of LNAI*, pages 64–78. Springer, 2006.
13. M. Birna Riemsdijk, Koen Hindriks, and Catholijn Jonker. Programming Organization-Aware Agents. In *Proceedings of the 10th International Workshop on Engineering Societies in the Agents World X*, ESAW '09, pages 98–112, Berlin, Heidelberg, 2009. Springer-Verlag.

Intelligent Surveillance with Autonomous Underwater Vehicles

Thor Helms, John Bruntse Larsen & Jens Peter Träff
DTU Informatics

Reason for this project

Nowadays aquatic monitoring tasks are primarily carried out by large ships, sailing for long spans of time, and relying heavily on approximation for analysis. By using multiple Automated Underwater Vehicles (AUVs), the reliability of the observations can be increased significantly thus rendering more precise analysis possible providing an improved basis for making decisions.

One example could be the determination of fish-quota through scanning of ICES (International Council for the Exploration of the Sea) squares in the ocean. The methods used today are prone to error and requires a fully crewed vehicle at sea for several days/weeks. By making clever use of information from multiple AUVs, both the measurement-errors as well as the required manpower can be reduced.

Résumé of our project

We have analyzed some of the technical requirements for an AUV with respect to the software, and investigated the existing research in these areas. We have looked at various features within the Guidance-Navigation-Control (GNC) model, and our conclusion is to suggest using the GNC scheme in an event based system, such as Robot Operating System (ROS). Within each part of the GNC scheme, advanced features, such as automated learning, sensor fusion and artificial intelligence, can be used.

Future work

As AUVs can be lost at any point during their mission, multiple AUVs working together can have no leader deciding all the actions. In our opinion, this makes AUVs an interesting research platform for applied multi-agent systems and swarm-theory.

Future projects building on this would have the potential of entering the so-called Grøn Dyst conference at DTU, as we did in the spring of 2012.

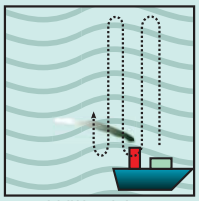
The following page shows the posters for the AUV project.

Additional information: http://www.groendyst.dtu.dk/English.aspx

DTU's Grøn Dyst (Green Battle) is a student conference which sets focus on sustainability, climate technology, and the environment.

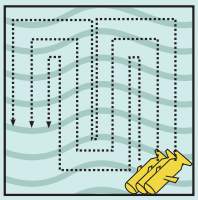
Automated Environmental Monitoring

Fish Stock



With ship

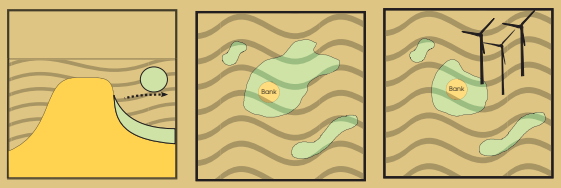
- Requires a lot of fuel.
- Leaks a lot of oil.
- Expensive in operation.
- The integration error is large (~10 km), giving inaccurate results.



With AUVs

- Low fuel for each AUV.
- Leaks very little oil.
- Unmanned and cheap in operation.
- Working AUVs in parallel decreases the integration error.

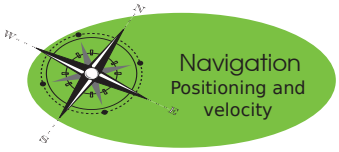
Dogger Bank



Large source of phytoplankton in the sea.
Disturbing the production and spread of the phytoplankton with windmills can be problematic to the sea life. This creates an interest in mapping the area affected by the Dogger Bank.

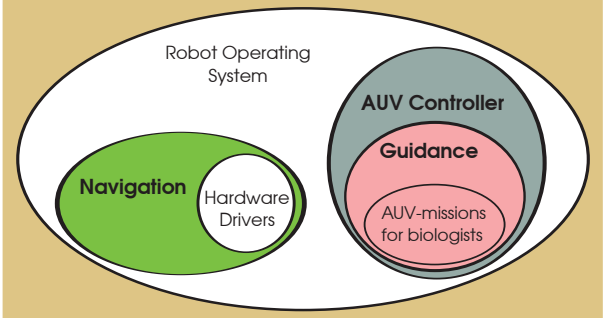
The cloud spreads slowly
Since the cloud of phytoplankton moves slowly, mapping the cloud must be done over a longer period. Currently this is done with ships, but it gives inaccurate results and is expensive.

AUV Path Finding



DTU Informatics **Students:** *Jens Peter Träff, John Larsen, Thor Helms* **Supervisor:** *Jørgen Villadsen* **Assisting guidance:** *Mikael Bliksted, Steen Silberg*

AUV programming

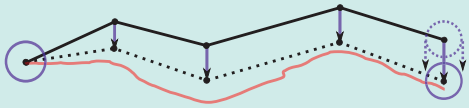


AUV Controller
- Carries out a given instruction such as moving to a specific location. Uses navigation for feedback.

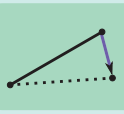
Guidance
- A sequence of controller instructions for the AUV.
- Contains behaviour for handling unexpected situations, such as encountering a fishing net.
- Specifies handling of communication.

AUV Missions for Biologists
Missions are specified with an intuitive interface and automatically translated to controller instructions.

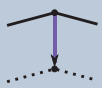
Navigation



Path Correction
- Takes a predicted path and measurements from the sensors as input.
- Uses a Kalman filter to get a corrected estimate of the path.
- Multiple sensors can be used to give a greater accuracy.
- Performs the correction in real time.



Model Correction
When a GPS signal is recieved the system learns from its performance and updates the model.



Data Correction
When a GPS signal is recieved the updated position is used to further improve the estimated path.

Predicted Path — Estimated Path — Measurements
Actual Path — Improved Path — GPS/Correction

DTU Informatics **Students:** *Jens Peter Träff, John Larsen, Thor Helms* **Supervisor:** *Jørgen Villadsen* **Assisting guidance:** *Mikael Bliksted, Steen Silberg*

Study at KAIST, South Korea, Dual Degree MSc Program in Computer Science

Jørgen Villadsen
DTU Informatics

The dual degree program between DTU and KAIST (Korean Advanced Institute of Science and Technology) supports the exchange of students and enables students to receive MSc degrees of both universities:

- The students will have a supervisor at DTU as well as at KAIST and must choose courses at the host university that only overlap slightly with courses taken at the home university.
- Two consecutive semesters must be taken at the host university, either the second and third semester (thesis written at the home university) or the third and fourth semester (thesis written at the host university).
- The host university will arrange for accommodation but the students are responsible for their own travel and living expenses during the exchange (the home as well as the host university will try to provide financial support).
- After successful completion the students receive the MSc in Computer Science of KAIST and the MSc in Computer Science and Engineering of DTU with a special designation of the dual degree program.

Students on the MSc in Computer Science and Engineering program must apply for the dual degree program not later than in their first semester!

"I considered the dual degree agreement between DTU Informatics and KAIST Computer Science as an opportunity to experience studying in Asia and getting a diploma from both DTU and KAIST. However I did not expect to become as involved in the research at KAIST as I eventually did.

At DTU I primarily studied logic and efficient algorithms, both theoretically and in applications. As a dual degree student at KAIST I decided to join the Semantic Web Research Center (SWRC) and participate in the research there. The focus in SWRC is web applications with PHP/SQL-languages and Java and I have been busy with co-authoring papers and implementing web applications, which is significantly different from what I did at DTU. On the other hand it has also given me ideas for how the research at SWRC could be relevant to my studies at DTU. Discussions with the other members have been inspiring and we help each other through the daily obstacles at SWRC. It is a small team and there are high ambitions for our research.

It has been a great challenge to juggle between research at SWRC and doing course work; something I still struggle with. It can feel pressuring to have such little free time but for me the dual degree program at KAIST has so far been a unique experience that I would not want to live without."

John Bruntse Larsen (KAIST-DTU student)

Additional information: http://www.imm.dtu.dk/English/Teaching/MSc/KAIST_DTU.aspx

Available Proceedings:

Algolog Multi-Agent Programming Seminar 2011

2 December 2011 – 40 Pages

Algolog Multi-Agent Programming Seminar 2012

29 November 2012 – 48 Pages

Algorithms and Logic Section
DTU Informatics

http://www.imm.dtu.dk/algolog