Anytime algorithms for multi-agent decision making

Intelligent Autonomous Systems research group
Faculty of Science

UvA UNIVERSITEIT VAN AMSTERDAM

Reinoud Elhorst

Supervising professor: Nikos Vlassis

4 June 2004

Abstract

Multi-agent systems is an exciting new field with many theoretical and practical challenges. In this work we are interested in fully cooperative multi-agent systems where all agents share a common goal. A key aspect in such a system is the problem of coordination: how to ensure that the local (individual) decision making of each agent can produce globally good solutions for the team.

RoboCup is an effort to build a team of robotic players that will one day be able to defeat a team of human players in football. This has resulted in a competition nowadays in which robotic and simulation teams play each other. In this field (as in others), it is important that the agents coordinate their actions.

This need for coordination put down a number of problems in the field of multi-agent system. One of these is how to find the optimal combined action in a large action space, with many agents, effectively. Several solutions have been proposed. An obvious approach is to try all possible combinations of actions of all agents, and from that select the best one. In systems with a large number of agents, this task will consume too much time to be feasible.

Another approach, variable elimination, has been proposed in [Gue03]. This approach has a worst case time complexity equal to the aforementioned method of trying all possibilities. In many problems however where direct coordination is not needed between every two agents, variable elimination will perform considerably better.

Variable elimination is exact: it will always find the optimal combined action. However, many applications do not require the optimal action per se, and a sub-optimal action would suffice in many cases. Especially in cases where time is limited, one might choose to trade optimality of the action for running time. Variable elimination does not allow for this.

In this thesis we describe and test a new algorithm, coordinate ascent, which is an anytime algorithm. We will show that the algorithm returns satisfactory results within only a fraction of the time that variable elimination takes to find the solution to the same problem.

Another issue put down by the need for coordination is how to make sure that, once an optimal combined action has been found, all agents will indeed know which action to follow. It turns out that the problems herein are universal to all decision-making algorithms, but that deterministic algorithms have a clear a advantage. We find some problems here when workinh with anytime algorithms: even anytime algorithms that are traditionally considered to be deterministic might return different results for different running times. So, unless specific measures are taken to ensure running times are always equal, anytime algorithms seem to be less suited for this kind of coordination problems. We will present a system in which coordinate ascent can still be used in RoboCup.

Acknowledgements

Many people have been working with me and been supporting me in doing the research for, and the writing of, this thesis. First of all Nikos Vlassis, my supervisor, who was always available for good advise, and was able to point me in the right direction many times. Jelle Kok was a big help too, and his inside knowledge of the RoboCup domain and the UvA Trilearn team, as well as his knowledge on the broader field of multi-agent systems, saved me a lot of time.

Furthermore I need to thank my mother who supported me greatly during the writing of this thesis and forced me to continue even at times when things weren't going so well. And of course my friends who have listened to my stories of football playing robots over and over again, without actually understanding what they were about.

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

1.1 South-East England, Friday afternoon

The M1 motor way from London to Leeds is for a lot of people one of the least favourite places to be at 5 PM on an average Friday afternoon. Tens of thousands of motorists try to make their way home from a week of work for a relaxing weekend, only to end up in endless traffic jams. Standing at the M25 flyover an almost endless flow of traffic passes underneath, coming from 3 directions and trying to squeeze into the too few lanes northbound. Ironically enough, if we look at the cars as intelligent entities[1] with goals, and priorities attached to these goals, the overpass becomes an excellent place to explain the basics of multi-agent systems.

In a gross simplification we could assume that every car-entity has just one goal: get home fast as possible. It is pretty clear what actions the car should take: overtake as often as possible, squeeze into every little hole and never let some other car go first. Because every car-entity only cares about getting home itself, and all cars have to go through the same bottleneck, the result is competitive and can be modelled by a competitive multi-agent system.

Grossly simplifying the other way, we can say every car-entity works towards a solution whereby all the cars navigate the bottleneck as efficiently as possible. In that example every car-entity has the same goal with the same priority as all others, so everyone works towards that common goal. It should be noted however that the decision making process is still distributed: every car-entity has to decide on his own actions, based on the situation around them and the information he has on the intentions of the other entities (flashing way-lights, hand-signals). This is basically how a cooperative multi-agent system works.

[1]Intelligent, in this context, means that they observe, process, and (re)act.

1.2 Multi-agent systems

In the multi-agent domain we call our 'entities' or 'actors': 'agents'. An agent is 'something' that observes and acts, this might be either a human, an animal or a computer. Note that agents usually need to be able to 'choose', so a stone that 'observes' it is one foot above the ground and 'decides' to fall is not considered an agent, whereas a bird who is one feet above the ground and decides to stop flying is an agent. This example also shows that it doesn't matter so much whether something actually has a choice in the sociological sense of the word, but whether it has the physical ability to choose.[2]

In order to decide on which action to take the agent needs to have some preference over the actions in a certain situation; this 'preference' is called 'payoff' in the multi-agent domain. In the easiest situation this payoff is only based on the current situation and the agent's own actions, however the payoff is often dependent on other agents' actions as well. (e.g. In our traffic example, an agent might observe a free space in front and decide to move there. She[3] would do good however first to check whether she is not going to hit another agent who was going to move there as well.) So, in general, we might say that every agent has, for each situation and for each of the other agents' combination of actions, a payoff for each of her own actions.

A group of agents (note that this doesn't necessarily have to be all the agents in the system) working together towards a common goal are called a cooperative multi-agent system. Instead of each agent trying to maximise her own payoff, all agents are trying to maximise the collective payoff, the sum of all individual payoffs. As we discussed before, the individual payoffs are dependent on both the own agent's action and the other agents' actions (in a given situation), so the agents need to coordinate their actions in some way.

In the example of Figure 1.1 we have two lovers trying to meet for a night together. They have two choices, either meet in the park or at the harbour, which, basically, means that each of them has a choice to either go to the park or the harbour, and they need to coordinate their actions in such a way as to maximise payoff. First they have to select (on basis of the figure) what the best action for both (this is called the optimal combined action) would be. It is pretty clear that the best action in this case is: either both go to the park or both go to the harbour. However in larger problems, with more agents and more choices, the tables become larger very fast and other

[2]Clearly this definition does pose a philosophical problem: A computer running a program (and arguably even the human mind) is nothing more than a large system of physical processes, extremely more complex, but in basics not unlike, the stone falling to the ground.

[3]In this thesis all agents are considered to be female.

lover2 lover1	park	harbour
park	1	0
harbour	0	1

Figure 1.1: The need to coordinate

methods of deciding on the best action have to be employed. For the time being we skip the subject how they share the decision on where to meet.

1.3 A technical look at multi-agent systems

When we look at multi-agent systems from a technical point of view, we see a quite remarkable feature. Often cooperative multi-agent systems work in such a way that the same job could have been done by a single computer (or single 'maxi'-agent). This would eliminate many problems that have to be tackled when working with multiple agents. However, the multi-agent approach has a number of clear advantages [Vla03]. In this section we only look at cooperative multi-agent systems, since in a competitive multi-agent systems the agents typically do not have the same goals, and could therefore not be controlled by a single process.

Robustness Because a multi-agent system contains many agents, all being different processes, an error, crash or deadlock in one of the agents will usually result in only a slight degradation of the total effectiveness of the system, whereas the same sort of problem in a centralised system might cause the whole system to fail.

Scalability In a centralised system, adding a task, or expanding the tasks of a certain part of the system, might involve a full overhaul of the code. In multi-agent systems it is often enough to introduce a new agent for the task, and make sure that a few of the other agents know how to work together with the newly introduced entity. This makes maintaining and expanding the system way easier.

Parallelism In a multi-agent system each agent is a different entity that can very well be (and usually is) running as a separate process. This means that we can very easily carry some of the agents to one or more other systems, hereby sharing the resources of the system. Since agents interact through communication only, only the communication layer would have to be adjusted to make this possible. This also means that we can extend the system indefinitely by adding new agents and having these run on other systems. A centralised program on the other hand usually requires special reprogramming to enable it to run in parallel on several computers.

Efficiency It is often easier to design a lot of small, simple processes than design one large, complex one. Therefore, during the design stage, but also the maintenance and enhancement stages, one might save a great deal of time and effort by using a multi-agent system.

In addition to the reasons mentioned above, some cooperative multi-agent systems simply cannot be replaced by a single centralised system. This is especially true in environments where observing and acting is done remotely (e.g. by robots in a room) and the communication possibilities do not allow all data to be transferred forth and back between the observers/actors and the centralised system.

One of most exciting applications of cooperative multi-agent can be found in the RoboCup leagues. Here, robots and computer processes play (a sort of) football, and obviously need some way to coordinate their actions with their teammates. A closer look to the RoboCup domain is given in chapter 2.

1.4 Direction of the thesis

Within the domain of multi-agent systems, many different research directions exist. In this thesis we focus on the problem of decision making in cooperative systems: how do the agents know what combined action is the best in a complicated cooperative multi-agent domain. We look at the requirements the different methods have for the domain (particular in the area of communication) and introduce a new anytime algorithm for decision making, that outperforms traditional algorithms in speed, with small loss in accuracy.

We will also look at the practical implementation of the new algorithm in RoboCup. The Universiteit van Amsterdam has a team (Trilearn) in the Simulation League which has performed really well, and was (one of) the first to implement coordination algorithms in its code. We will see whether, and how, the new algorithm, and the other findings from this thesis, can be implemented into this team.

When writing this thesis, the idea was that most parts of it should be understandable to people without a background in computer science. Therefore in many cases informal and easily understandable explanatory styles have been chosen to describe certain problems or situations. By doing so it sometimes is possible that the explanation in this thesis does not cover the full formal problem setting. It is important to note that the explanatory descriptions do not mean that the conclusions in this thesis are not applicable in a larger and formal environment.

This thesis is part of a graduation project and as, such will, contain a lot of information very specific to the field of computer science. Some parts of the thesis will therefore be harder to read for people without the proper background. However the aim is that the main idea behind the thesis can still be understood if these parts are skipped.

Several of the ideas presented in this thesis will also appear in a separate paper [VEK04], which has been accepted into the IEEE SMC conference 2004[4].

[4]International Conference on Systems, Man and Cybernetics; http://www.ieeesmc2004.tudelft.nl/

Chapter 2

Practical application of a cooperative multi-agent system: RoboCup

2.1 Introduction

> "By the year 2050, develop a team of fully autonomous humanoid robots that can win against the human world soccer champion team."

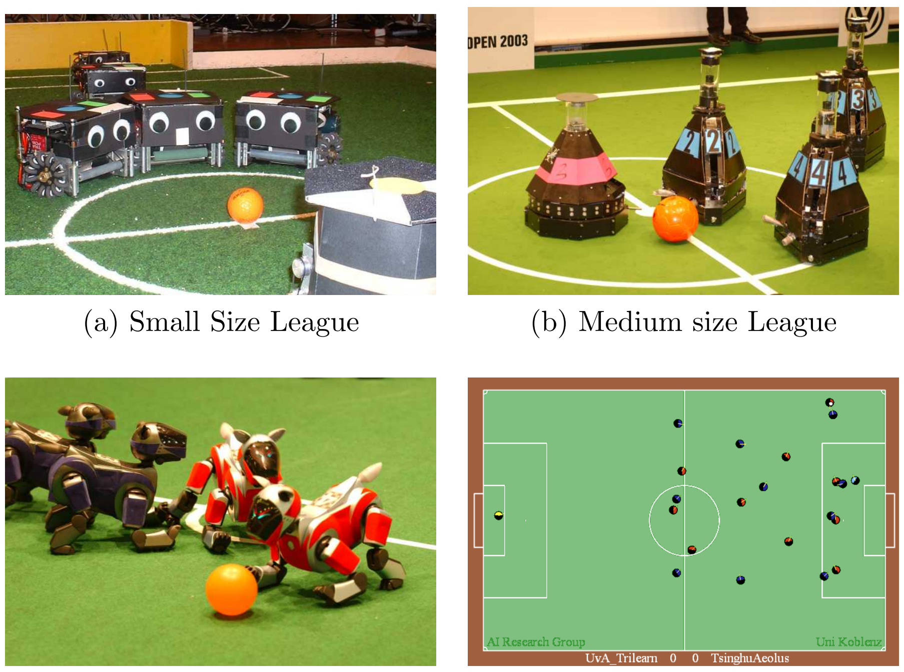
This quote proudly welcomes the visitor to the RoboCup web site[1]. A very ambitious project, that so far has resulted in separate leagues in which robots play other robots, (Figure 2.1 a, b and c) and one simulation league in which the robots are left out, and agents play a football game against each other on the computer (Figure 2.1 d).

2.2 RoboCup as standard AI problem

Since the beginning of AI, the field demonstrates its success largely by showing its success in standard problems. One of the most notable examples is of course Deep Blue's victory over Kasparov in chess, May 1997. Although chess is an extremely complex problem, it still is a clearly abstract one: it has a clear discrete domain, has turn based actions and the situation (i.e. the location of the pieces on the board) is always known to all players.

In this respect, RoboCup is a lot closer to real-life problems. It deals with a dynamic environment, in which real-time actions are required. The environment is not fully observable (i.e. an agent can only look in one direction at a time, and the data it receives contains a certain amount of noise).

[1]http://www.robocup.org



(a) Small Size League (b) Medium size League

(c) Four-legged League (d) Simulation League

Figure 2.1: Examples of different RoboCup leagues

Another very interesting aspect of RoboCup is the multi-agent nature of it: independent agents have to work together as a team, and only have limited communication available, so a central control is not feasible.[2]

One of the aims of the RoboCup project is to get RoboCup recognised as a standard problem in the AI. Whether this will succeed, we will only know in time, however some of the leagues where robotics are important are more and more being dominated by major companies, and universities are increasingly unable to allocate enough funds to compete on that level. On the other hand, ever more universities are including multi-agent systems in general (and sometimes specifically RoboCup) into their AI curriculum.

2.3 A closer look at the simulation league

2.3.1 Division into leagues

RoboCup is divided into a number of different leagues. This division allows for a division of the problem domain, so that not every league has to tackle all the problems contained in the final goal of realising a team that can take

[2]Note that not all these features are present in all leagues. See section 2.3.1 for a quick overview, or the RoboCup web site (http://www.robocup.org) for more elaborate description.

on the world champions of human football. The different leagues include a small size league, in which the robots are centrally managed, a four legged league (or AIBO league), in which the robots are commercially available products, a mid- size league, where both robot building and robot control are important, and a humanoid robot league, where the operation of bi-pedal robots is emphasised.

In addition a simulation league was included, where the robotic element was totally removed, and the focus is on the operation of the separate agents into a team. Most leagues use some sort of multi agent decision making, however the simulation league uses the most advanced implementation. This, and the fact that the UvA[3] has a team that is performing very well in the this league, made me choose look at this league in this thesis. The methods discussed in this thesis will be checked against practical applicability in this league.

2.3.2 The simulation league

In the simulation league, a game is played that somewhat resembles EA's FIFA- series of computer games. However, instead of a human controlling the players, each player is being controlled by a separate process—an agent—so a total of 22 agents (+ coaches) run to play the game. Obviously each group of 11 agents have to work together to win the game.

The simulation league is a very popular league, and the largest in RoboCup. This is partly because all that is needed to develop a team is a computer running Linux. Even to test one's team, all one has to do is to download another team, and run the program. It is also a very interesting league, since no robotics are needed to participate. Since all players on the field are equal[4], the physical properties of the players are known in advance. Reading the sensor data and steering the player are relatively simple operations and programmers can focus on higher level actions, such as tactics. As a result the tactical game play of the simulation league is more advanced than any other league.

2.3.3 Possibilities and constraints

To understand how to use coordination and the results of this research in the simulation league, it is important to understand what is and what is not possible in the simulation league. It is important to realise that every player is an agent. The time-domain of 10 minutes is divided into 6000 steps of 100 ms each. At the beginning of each timestep an agent gets her 'sensor-information', and then has to send out 'action information' before the end of

[3]Universiteit van Amsterdam, University of Amsterdam.

[4]There are different types of players, however every team may make use of the same types.

the time step. The agent doesn't receive all the information in the domain—she can only look in one direction, and even in that direction noise is added (the further away an object is, the more noise is added). Furthermore there is a limited form of communication: if an agent 'says' something, in the next timestep all agents within range will 'hear' that remark. It is only possible for each player to 'say' a limited number of things during a game.

It will be clear from the description above that the agents have to act independently to a large extend. As we will see in section 3.3, coordination without extensive communication possibilities puts strict requirements on the methods for coordination used.

2.4 RoboCup at the UvA: Trilearn

The Universiteit van Amsterdam (UvA) participates in RoboCup in two leagues. In the midsize league, the UvA, in cooperation with the University of Utrecht and Delft University of Technology, has developed the team 'Clockwork Orange'. The Universiteit van Amsterdam also has a team in the simulation league, 'UvA Trilearn'. This team was initially developed as a graduation project of Jelle Kok and Remco de Boer, and has since been maintained and improved by Jelle Kok as part of his PhD research. This team has a respectable record as winner of the German Open in the last three years, and winner of the RoboCup 2003.

One major factor in Trilearn's success has been that since two years multi-agent coordination is finding its way into the team. Because of the limited communication possibilities and the lack of common knowledge (each player receives different noise and therefore has a slightly different world picture), most teams have each agent decide her own best action, independent of the others. [KSV03] describes the methods that the UvA Trilearn team uses to circumvent these limitations.

As noted before, the results from this thesis will be checked against applicability in the Simulation League in general and the UvA Trilearn team in particular.

Chapter 3

Coordination games

3.1 The need to coordinate

When multiple agents work towards a common goal, they usually have to work together to reach that goal. In multi-agent systems we call this 'coordination'. This basically means that some agent will have to coordinate her actions to one or more other agents' actions. To give a impression on how this system works, we continue and elaborate on the example of section 1.2. Two agents are in love, and want to meet that night. In the simplest situation (Figure 3.1a the choice is just between the park on in the harbour, and the lovers goal is just to be together. Now we add the fact that both of them are fish-lovers, and the harbour has great fish-restaurants (Figure 3.1b). Note that they still have the same goal; if for instance one of them would like fish and the other one hated it, the agents would not be cooperative anymore and totally different methods of coordination / anticipation would apply. Furthermore it is important to realise that no prior arrangements have been made. In figure 3.1c we complicate things ever further, by offering the mall as an extra choice, which happens to be next to the park (so if one would go to the park and the other one to the mall, chances still are that they'll meet). Figure 3.1d adds the fact that the harbour is an extremely dangerous place to be by yourself. Finally, figure 3.1e recognises that the situation may change on basis of some external factor, called the 'context': Since you can sit outside in the park, that location is preferable in case of sunshine. In case of an overcast sky, the old system remains valid.

Obviously these are relatively easy examples; things can be complicated by adding extra agents, extra states or extra locations, though this would not add to the example. Important is to see the necessity of coordinating. In the example in figure 3.1d, for instance, the lovers could choose to go the harbour only if they were sure that the other one would do the same. When in doubt on the other's intentions, it would be safer for both to go to the park or the mall.

lover2 / lover1	park	harbour
park	1	0
harbour	0	1

(a)

lover2 / lover1	park	harbour
park	1	0
harbour	0	2

(b)

lover2 / lover1	park	harbour	mall
park	1	0	0.5
harbour	0	2	0
mall	0.5	0	1

(c)

lover2 / lover1	park	harbour	mall
park	1	–100	0.5
harbour	–100	2	–100
mall	0.5	–100	1

(d)

Sunny:

lover2 / lover1	park	harbour	mall
park	3	–100	0.5
harbour	–100	2	–100
mall	0.5	-100	1

Overcast:

lover2 / lover1	park	harbour	mall
park	1	–100	0.5
harbour	–100	2	–100
mall	0.5	–100	1

e)

Figure 3.1: The need to coordinate: two lovers want to meet, the rows are lover1's choice, the columns lover2's. The value in the table is the collective payoff. Five different scenarios are discribed by the figures *a*, *b*, *c*, *d* and *e*; see the text for more details

Obviously RoboCup has more than enough possibilities for coordinating. Giving the deep pass is only useful when someone is going to run in that direction. Likewise, in the defence, it should be decided which agent stays close to which striker, with obviously disastrous results when all defenders go after one striker, leaving the others alone.

3.2 Which decision is good: payoff

In section 3.1 we looked at some examples on coordination and why it is important to coordinate. Intuitively we recognise the action combination with the highest number with it as the best action, however in the computer world this obviously is formalised a little more. Typically each agent has a 'niceness parameter' for each action combination of all agents, which is called 'payoff'—this value notes how preferable a certain action combination is to this agent compared to other action-combinations. Typically an agent tries to choose such an action that her own payoff is maximised. In cooperative systems however, agents are not trying to maximise their own payoff, but the collective payoff, which is the sum of the payoff of all agents. Therefore we sometimes only note the collective payoff in cooperative systems, and ignore how this payoff is divided over the individual agents.[1]

Getting the values for the payoff is not a trivial matter. In the examples of section 3.1, the payoffs could be more or less deducted from the problem description, for more complicated problems this is not that easy though. It should be noted that the payoff only is valid for one 'timestep'[2], so high payoffs should be given to action combinations that will result in a better strategic position from where to achieve the final goal. In the RoboCup game for instance the only goal of the agents is to win the match. It is clear that scoring is a step in the right direction towards that goal, and thus it gets a high payoff. To score we need the ball, so intercepting a pass from the opponents gets a high payoff. As we get more into the details it becomes less and less clear what a good action would be. Several methods are around to deduct the payoff for each action combination [Vla03] and [SB98]. In this thesis we will not look into these and assume that appropriate payoffs are defined beforehand.

3.3 Methods of coordination

We have shown that it is necessary for cooperating agents to coordinate actions. How this coordination takes place depends on what constraints the specific problem has. In general we can recognise three ways of coordinating:

[1]This is what we did in figure 3.1.

[2]Technically the time-domain doesn't have to be discrete, however it greatly simplifies things to assume that it is. In the simulation league the time domain is discrete indeed.

Centralised One agent might calculate what the optimal actions for all agents are, and then communicate these actions to the agents. For this it is needed that communication is available and reliable. In addition, it is necessary that the agent doing the calculation knows all payoff-functions for all agents, either by obtaining these though communication, or by knowing them beforehand. When we go back to the examples of section 3.1 and we would give both lovers a mobile phone, one could just call the other, ask her preferences, decide and tell the other one where to meet.

Distributed The agents could work together in a group in finding the optimal action in such a way that each agent only calculates its own action. The algorithm needed for this is basically a distributed version of the centralised algorithm, where, instead of the payoff-functions being transferred to one agent, each agent keeps its own functions and does that part of the calculation that need those functions by itself before sending the results to the next agent. Advantage is that each agent does approximately the same amount of work, and that some parts of the calculation could be done in parallel. In addition payoff-functions which are too large to communicate or are secret do not need to be transferred. On the downside, this method requires a lot of communication and communication moments.

Replicated Each agent runs the algorithm that the one agent ran in the centralised situation, to determine her own optimal action. Although the agent does not have to run the whole algorithm in all cases and can stop as soon as she has found her own optimal action, this action is usually only determined at the end of the algorithm, and the running time for each agent is about as long as the running time for the single agent in the centralised situation. As in the centralised situation, the agent doing the calculating (in the replicated case this means all agents) needs to know all payoff functions of all agents. If these are known beforehand (this is called common knowledge), this system can work in an environment without communication possibilities. One requisite in this situation is that the algorithm employed is deterministic.

Since each agent is calculating its own optimal combined action[3], and acting on that result, it is essential that all agents reach the same optimal combined action. This is also the case when multiple combined actions give the same maximum payoff. Hence we need a deterministic algorithm. Imagine the situation of figure 3.1a, where both the situation of both lovers in the park and the situation of both lovers in the

[3]As noted before an agent does not always have to find the complete optimal combined action, this does not affect the argument though.

harbour give a payoff of 1. Now if both lovers have this payoff-matrix, but they have no methods of communication, then one of them might decide that for both to meet in the park is a good idea, the other one might choose for both to meet in the harbour is smart, and they'll miss each other. A deterministic method might have included a rule saying that in case of a tie, the park (or the harbour) always has preference.

Throughout this paragraph we have been focussing on finding—and communicating—the optimal combined action. Later in this thesis we shall see that in some cases we will have to settle for a solution that is not the optimal one per se, but still a reasonably good one. The methods of coordinating and the constraints discussed in this paragraph apply in those situations as well.

3.4 Representing the problem: Matrix vs. Coordination Graphs

In the examples of section 3.1 we looked at some coordination problems and represented these as matrices. This matrix-notation has the great advantage that it is easy to spot the maximum—and so the optimal combined action. When the problem grows, the size of the matrix grows exponentially in the number of agents. Especially when not all agents have to coordinate with all others, the matrix will contain a lot of duplicate entries, and an easier representation is preferable.

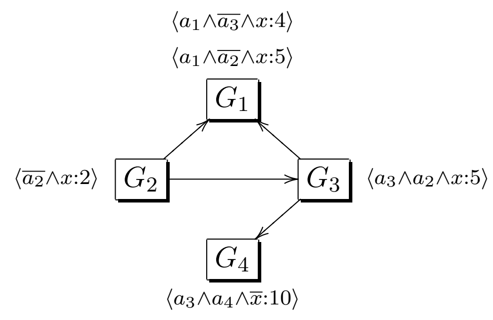
The framework of Coordination Graphs with value rules, as presented in [GVK02], represents coordination between a relatively large number of agents with limited coordination requirements. A value rule is a rule that says that in a certain situation a certain payoff will be received. A rule reading $\langle a_1 = 0 \wedge a_2 = 1 \wedge a_5 = 1 \wedge x = 1 \wedge z = 0 : 5\rangle$ means that a payoff of 5 'exists' when $action1 = 0$, $action2 = 1$ and $action5 = 1$, and the contexts $x = 1$ and $z = 0$ apply. The values for context y, and $4action3$ and $action4$ do not matter. In cases of binary actions and contexts, we sometimes abbreviate to $\langle \overline{a_1} \wedge a_2 \wedge a_5 \wedge x \wedge \overline{z} : 5\rangle$.

Figure 3.2a shows a random coordination problem with two possible states x and $\overline{x}$, four agents $i = 1, 2, 3, 4$ that each have 2 possible actions a_i or $\overline{a_i}$ in matrix notation; figure 3.2b shows a possible coordination graph for this problem. The nodes in the graph are the agents, and the edges represent which agents need to coordinate their actions directly; more precisely, the agent at the point of the arrow should coordinate her action to the action of the agent from where the arrow originated. Since our problems are of a cooperative nature, we did not show in our matrix which agent gets the payoff; this is being shown in the coordination graph, where the total payoff is distributed among the agents. Therefore from one matrix many different

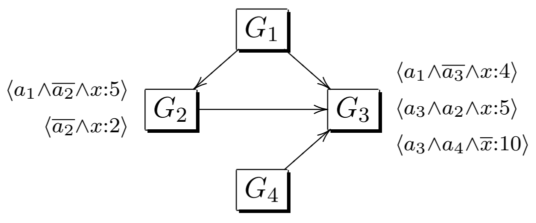
x		a_1		$\overline{a_1}$	
		a_2	$\overline{a_2}$	a_2	$\overline{a_2}$
a_3	a_4	5	7	5	2
	$\overline{a_4}$	5	7	5	2
$\overline{a_3}$	a_4	4	11	0	2
	$\overline{a_4}$	4	11	0	2

$\overline{x}$		a_1		$\overline{a_1}$	
		a_2	$\overline{a_2}$	a_2	$\overline{a_2}$
a_3	a_4	10	0	0	0
	$\overline{a_4}$	10	0	0	0
$\overline{a_3}$	a_4	10	0	0	0
	$\overline{a_4}$	10	0	0	0

(a)


$\langle a_1 \wedge \overline{a_3} \wedge x{:}4 \rangle$
$\langle a_1 \wedge \overline{a_2} \wedge x{:}5 \rangle$
G_1
$\langle \overline{a_2} \wedge x{:}2 \rangle$
G_2
G_3
$\langle a_3 \wedge a_2 \wedge x{:}5 \rangle$
G_4
$\langle a_3 \wedge a_4 \wedge \overline{x}{:}10 \rangle$

(b)


G_1
$\langle a_1 \wedge \overline{a_3} \wedge x{:}4 \rangle$
$\langle a_1 \wedge \overline{a_2} \wedge x{:}5 \rangle$
G_2
G_3
$\langle a_3 \wedge a_2 \wedge x{:}5 \rangle$
$\langle \overline{a_2} \wedge x{:}2 \rangle$
$\langle a_3 \wedge a_4 \wedge \overline{x}{:}10 \rangle$
G_4

(c)

Figure 3.2: A problem in matrix notation, and two possible coordination graphs for that same problem.

$$\begin{array}{c}
< a_1 = h \wedge a_2 \neq h : -50 > \\
< a_1 = m \wedge a_2 = m : 1 > \\
< a_1 = m \wedge a_2 = p : .5 > \\
< a_1 = p \wedge a_2 = p \wedge S : 3 >
\end{array}$$

$$\begin{tikzcd} \boxed{L_1} \arrow[r, leftrightarrow] & \boxed{L_2} \end{tikzcd}$$

$$\begin{array}{c}
< a_1 \neq h \wedge a_2 = h : -100 > \\
< a_1 = h \wedge a_2 \neq h : -50 > \\
< a_1 = h \wedge a_2 = h : 2 > \\
< a_1 = p \wedge a_2 = m : .5 > \\
< a_1 = p \wedge a_2 = p \wedge \overline{S} : 2 >
\end{array}$$

Figure 3.3: A coordination graph of the lovers' meeting; L_1 and L_2 are lover1 and lover2, a_1 and a_2 are the actions of lover1 and lover2 respectively; S represents a sunny day

possible coordination graphs can be made. Another possible coordination graph of the same problem is being given in figure 3.2c. As with matrices, we might also produce coordination graphs which do not keep track of which agents have which value rules. In these graphs the edges have no direction, and the value rules are noted on a separate list.

The example of the lovers trying to meet, represented by figure 3.1e, could be described by coordination graph 3.3. In that example, the coordination graph notation does not make the problem any easier to grasp, and here we see that in some situations the matrix notation is better. In general however, in larger systems in which not every agent has a payoff dependent on every other agent, coordination graphs *will* greatly improve things.

3.5 Decision making

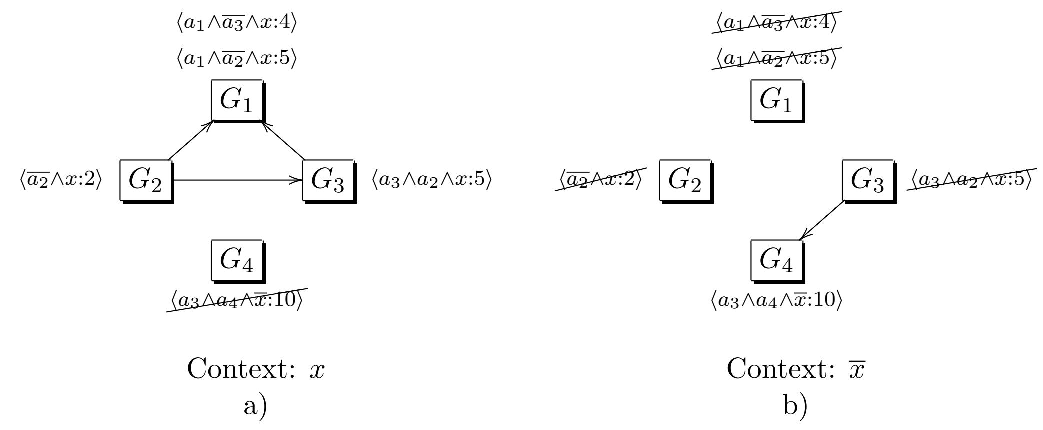
3.5.1 The optimal combined action

In section 3.3 we discussed methods for achieving coordination. We showed that one agent could find the (optimal) combined action and communicate this action to all others, the algorithm could be executed distributed, or each agent could run the same algorithm, thereby reaching the same solution for the optimal action.

In that we ignored the problem of how to find this optimal combined action. This problem is known as the problem of decision making. The agents need some algorithm to find this optimal combined action—some of the algorithms for doing this are discussed here.

3.5.2 The role of the context

Before we come to the actual decision making process, first a quick word on the context. In the examples in the beginning of this chapter, we saw that, in some cases, context plays a role in decision making. Context is a set of outside variables, which decide which payoffs are valid and which are not (as in the example of figure 3.1, where the choice for the park gets extra payoff in case the sun shines). Variables in the context cannot be influenced



$\langle a_1 \wedge \overline{a_3} \wedge x{:}4\rangle$
$\langle a_1 \wedge \overline{a_2} \wedge x{:}5\rangle$
G_1
$\langle \overline{a_2} \wedge x{:}2\rangle$
G_2
G_3
$\langle a_3 \wedge a_2 \wedge x{:}5\rangle$
G_4
~~$\langle a_3 \wedge a_4 \wedge \overline{x}{:}10\rangle$~~
Context: x
a)
~~$\langle a_1 \wedge \overline{a_3} \wedge x{:}4\rangle$~~
~~$\langle a_1 \wedge \overline{a_2} \wedge x{:}5\rangle$~~
G_1
~~$\langle \overline{a_2} \wedge x{:}2\rangle$~~
G_2
G_3
~~$\langle a_3 \wedge a_2 \wedge x{:}5\rangle$~~
G_4
$\langle a_3 \wedge a_4 \wedge \overline{x}{:}10\rangle$
Context: $\overline{x}$
b)

Figure 3.4: The coordination graph of figure 3.2b after the context has been established.

by the agents directly[4], and are therefore of no importance in the process of decision making. As soon as the actual context in a situation is known, all value rules which are invalid are being removed from the coordination graph (in some cases this includes removing some edges too). Stripping all these rules at a given timestep is a trivial operation that can be done in linear time; in figure 3.4, the coordination graph of figure 3.2b is being shown after the contexts x (a) and $\overline{x}$ (b) have been established. For this reason from this point forward we ignore the concept of context, and assume all value rules to be valid at the time.

3.5.3 Try all possibilities

When a human is asked to find the optimal solution for the problem of the lovers in figure 3.1e, he will quickly scan all entries in the matrix, pick the highest and recommend that as action for the lovers. Computers are way faster in comparing numbers and finding the highest value in a matrix, and scanning all possible action combinations to find the highest value is a simple and effective solution for small problems; even for small coordination graphs, which are quite easily converted to matrices.

When the problems grow this approach has a serious drawback. Finding the maximum value in a matrix is of linear complexity in time to the number of entries in the matrix. The number of entries in the matrix is $\prod |a_i|$, where $|a_i|$ is the number of possible actions of agent i. In the case that each agent

[4]Agents can influence the context of future timesteps. For instance, context variable x might be the player that currently holds the ball in a RoboCup game. By passing the ball to another player, x will have changed through an action of an agent. Whether this is a good thing to do or not, is coded in the payoff, and no direct look is given to the possible payoff in that other situation. As we noted in section 3.2, deciding on the correct payoffs is a complicated matter which is not covered in this thesis.

has n possible actions, k agents will result in a matrix with n^k entries. The time taken for trying every possibility is therefore exponential to the number of agents.

3.5.4 Variable Elimination

In section 3.4, we have seen that in problems where not all agents have to coordinate directly, the matrix notation contains many duplicate entries, and coordination graphs with value rules manage to note the problem simpler and shorter. The same goes for the process of decision making, where, using the coordination graph and variable elimination we will be able to solve problems faster.

Variable elimination is a process in which the agents are being eliminated from the value rules one by one, until only one agent is left. This agent then selects the best action for herself, and reports this to the penultimately eliminated agent. She then fills in the other agent's action, which results in value rules with only one agent left. This process continues until every agent has an action assigned. An elaborate explanation of the variable elimination algorithm can be found in appendix A.

Different implementations of variable elimination exist; which one is best depends on whether communication is available and whether agents know all payoff functions of all other agents beforehand. These implementations do not differ sufficiently enough to look at each one separately. In our tests we used a slightly modified version of the algorithm described in [Gue03], the modifications to the algorithm are explained in appendix A.

Variable elimination finds the optimal combined action (and the optimal payoff) in finite time. According to [Gue03], the algorithm has a time-complexity of at most exponential to the induced width of the coordination graph[5]. In the worst case, the case of a fully connected coordination graph, this method has the same time complexity as the method described in section 3.5.3. As we discussed before, many problems have a lower grade of connectivity and the solution will therefore be found way quicker than trying all possibilities.

The UvA Trilearn team uses variable elimination to coordinate the actions of some agents. How this is being done, and how the problems that arise from this were tackled, is described in [KSV03].

Especially large problems with heavily connected coordination graphs, will still not perform well with variable elimination. Furthermore, when time is a constraint, variable elimination is an all or nothing solution: either the time given to the algorithm was enough and the optimal combined action has been calculated, or the time was not enough and no result whatsoever is available. However, in several situations it would be preferable to get a

[5]The induced width is the maximum number of edges leaving a node at any time during the variable elimination.

‘good’, sub-optimal solution in case there is not enough time to calculate the optimal solution, rather than being stuck with nothing. Anytime algorithms have that possibility.

Chapter 4

Anytime algorithms

4.1 Introduction to anytime algorithms

As discussed before, in cases where we are not sure whether we have enough time to find the optimal combined action, we might be greatly helped if, after time runs out, we would have some result that is not necessarily the optimal result, but still a whole lot better than nothing (or: better than a random combined action). This kind of algorithm is called an ‘anytime algorithm’. An anytime algorithm can return a result at any time, and the longer the algorithm runs the better the result will be, until the optimal result has been reached. It is not necessary that the optimal result is reached in the same time as the fastest not-anytime algorithm, as long as the anytime algorithm will give a result close to the optimal one within a reasonable amount of time.

To what respect the preceding statement is true, and what is being meant by ‘a result close to the optimal one’ and ‘a reasonable amount of time’, largely depends on the problem that one is trying to solve. In some cases, only the optimal solution is worth mentioning, even if it takes a lot of time; in other cases, a ‘pretty good’ solution will do the trick. If we look back at the example of the lovers, they will have a good time, either in the park, the mall or at the harbour, although only one of these options is their optimal solution.

One anytime algorithm is the one we discussed in section 3.5.3: looping over all entries in the matrix to find the optimal solution. It is obvious that that method will be able to return the ‘best found so far’ at any time, while eventually reaching the optimal solution. We will introduce a more efficient algorithm, and see how it performs in relation to variable elimination, and whether it is usable in practice.

Algorithm 4.1 Coodinate ascent

```
define: scope($\rho$) is the set of all agents that define an action in $\rho$
define: $a_i$ the action of agent $i$
define: $A_i = \mathrm{Dom}(a_i)$
define: $a_{-i}$ the actions of all agents but agent $i$
define: $a^*$ the best combined action found so far
define: $\rho(a)$ is the payoff rule $\rho$ gives, in case of combined action $a$.

$P \leftarrow$ all valuerules
loop
  for $i \in$ agents do
    $a_i \leftarrow$ random $(A_i)$
  end for
  repeat
    $i \leftarrow \mathrm{next}(agent)$
    $\rho \leftarrow \{\rho_j \in P | i \in \mathrm{scope}(\rho_j)\}$
    $a_i^* \leftarrow \arg\max_{a_i \in A_i}(\sum \rho_j(a_{-i}^* \cup a_i))$
  until no agent has changed action since last time $i$ was checked
end loop
```

4.2 Coordinate ascent

Coordinate ascent is an anytime algorithm. The idea behind it is pretty straight-forward: One starts with a random choice for a combined action, then loops over all agents, and each agent will optimise her own action, while the actions of all other agents stay the same. This looping continues until no improvement can be made anymore; then a new random starting position is selected and the process is repeated. The highest value obtained so far will be returned when an answer is needed. The whole process is described by the pseudo-code in algorithm 4.1.

Basically, what the algorithm does is looking for Nash-equilibria (NE)[1]. When such a NE is found, there is no guarantee that it is Pareto-optimal, nor is there any way of finding this out. Therefore the code has to run again and again, because it might always find a better solution. When looking at the code, one notices quick enough that the algorithm only ends when it runs out of time. The algorithm has no guaranteed running time in which the optimal result will have been found.

To the basic algorithm some optimisations can be made. One suggestion would be to choose the starting values in some intelligent way instead of

[1]Technically Nash-equilibria need every agent to have her own set of payoff rules. Since in our situation all value rules are shared, one might think of Nash equilibria as local maxima of the combined payoff function, whereas the Pareto-optimal Nash equilibrium is the global maximum.

randomly. What an intelligent start is will largely depend on the type of problem. Although with random problems little can be gained by this, in some specific situations one might try an expected result as starting position.

Another suggestion would be to only optimise over all the actions of an agent, if at least one of the agents it has to coordinate with directly (so one of her neighbours in the coordination graph) has changed her action since this agent's last optimisation round.

In section 3.3 we discussed methods for coordinating, and we reached the conclusion that in case communication possibilities are limited, deterministic algorithms are needed. In its original form the coordinate ascent algorithm has four points at which it is non-deterministic. Most obvious the starting position is currently chosen randomly for every run; using a fixed random seed would easily solve this problem. Secondly, a fixed order between the agents must exist, so that the 'next' operator loops over the agents always in the same order. Furthermore, the 'argmax' function needs return always the same result, even if two actions result in the same payoff; having a preference over the actions in case of a tie would solve that problem. Finally, and most hidden, an anytime algorithm runs until an answer is needed. Depending on when this request is done, the algorithm will return different results. This issue is pretty complicated and will be discussed in section 4.3.

4.3 Deterministic vs. stochastic algorithms

In section 3.3 we showed the need for either a form of communication, or a deterministic algorithm. We also noted in section 4.2 that the coordinate ascent algorithm is not fully deterministic, because it uses an external trigger to decide when to return its results, and this trigger does not necessarily always come at the same moment. In truth, this is inherent to the nature of an anytime algorithm: because of its nature it will give different results at different times, and without some strict constraints on the timer there is no guarantee that the results will always be the same.

In a communicationless environment, the only way of making sure that the algorithm is deterministic (and so that the coordination will always succeed) is by making sure that the algorithm always stops after a fixed amount of runs. This might be done by hard coding this amount in the code, or defining the timer in CPU time rather than wall-clock time[2]. One of the great advantages of anytime algorithms is, though, that we can run them right until the moment that we need the result—this advantage is lost when the running time is decided upon beforehand[3].

[2]Obviously this only works when the algorithm is always run in the same environment; if one instruction takes one CPU-cycle in one instance, and two cycles in another environment, the count gets lost.

[3]This is not quite true; if the agents were to know exactly how much CPU-time they

In situations were the aforementioned constraints on the timer are not possible, one would have to look at the consequences of having a non-deterministic algorithm playing a part in coordination. If we would assume that all agents will have *approximately* the same amount of CPU-time available for the algorithm (which is the case in many practical applications), the coordinate ascent algorithm might or might not produce nice results. If two agents had about the same time for the algorithm, chances are good that the results they return—albeit different—are part of one route towards the same local maximum. Depending on the problem this might result in a not too bad result (imagine a situation where there is a gradual climb to the local maximum), or the result might be disastrous (imagine a situation where close cooperation is essential and if only one agent is not doing exactly what the others expect problems arise). In addition there is still the risk that one of the agents is already looking at another local maximum, and the results of the agents will be totally independent. Once again it depends on the problem whether the risks are acceptable—one can imagine that a failure to coordinate between the two lovers of figure 3.1c is an acceptable risk (they can always meet again next week), whereas for the lovers of figure 3.1d it would be unacceptable, since one of them might end up in the harbour by herself.

would get before an action was needed, and this was common knowledge, they could use exactly that amount and thereby having the algorithm run as long as possible. Also a very limited communication possibility would enable the agents to coordinate how many cycles the slowest of them could spend on the problem.

Chapter 5

Experiments and Results

5.1 The tests

We want to look at how the coordinate ascent algorithm behaves in relation to other algorithms, especially variable elimination. We do this in two stages. In stage one we look at the time-complexity of the variable elimination algorithm. In stage two we compare coordinate ascent to the variable elimination algorithm.

To assure the test results are reproducible, we need to specify the kind of problems we run the tests on. All problems are coordination graphs with value rules, but even in that domain many different types of problems exist, many of which might be encountered in practical applications. For instance large differences might exist in performance when applied to lightly or densely connected coordination graphs.

A description of the way we generated the problems, including the pseudocode for the generator, can be found in the appendix B. It shows that the generator uses four variables to generate its problems: nr_i is the number of agents in the problem, nr_a is the number of actions each agent can choose from, nr_ρ is the number of value rules that are generated per agent, and nr_s is a measure for the connectivity in the accompanying coordination graph. A value n for nr_s, means that the value rules of each agent only contain actions for that agent self (obviously) and actions for at most n other agents. Note that a node can still have a degree[1] which is higher than n, because the 'degree' measure (and the variable elimination algorithm) makes no distinction between incoming and outgoing edges. In this chapter we test with different values for the number of agents and the number of actions per agent. Furthermore we will distinguish three classes of connectivity: light, where $nr_s = 1$; medium where $nr_s = 3$ and dense, where $nr_s = nr_i - 1$.

It should be noted that since we are measuring (relative) times, the results greatly depend on the hardware, the developing environment, number

[1]The degree of a node is the number of edges connected to that node

of optimisations, etc. We do not claim to have used the fastest implementation possible for all algorithms, nor did we favour any particular implementation by optimising some code more than other. The pseudo code for the used variable elimination algorithm can be found in algorithm A.4 on page 43 of the appendix, the pseudo code for coordinate ascent is available in algorithm 4.1. We do feel that the results do give a fair image of the possibilities of the algorithms.

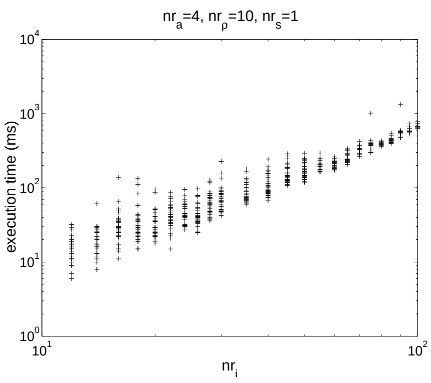
5.2 Performance of variable elimination

We argued before that coordinate ascent is an anytime algorithm that never really ends. As with all anytime algorithms it is unclear how to define the time performance of such algorithms. Obviously it will depend on whether one needs the global maximum payoff, or whether one is satisfied with a possibly less good (and less costly) result. In the next sections we will therefore show the performance of coordinate ascent compared to that of variable elimination. For this to give a clear result, it is important to first understand how well variable elimination performs.

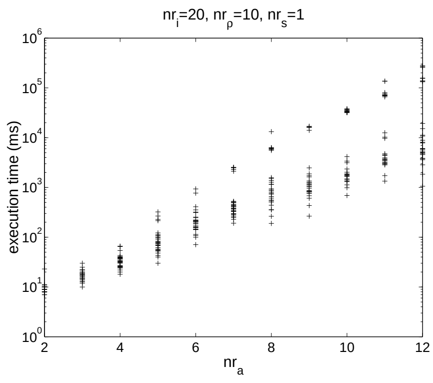
In figure 5.1, we see how well variable elimination performs in time. We measured the influence of two different variables on the time the algorithm took: the number of agents (nr_i) in plots a, c, d and *f*, and the number of possible actions per agent (nr_a) in plots b and e. We look at this for problems with light (plots a and b, $nr_s = 1$), medium (plots c, d and e, $nr_s = 3$) and dense (plot *f*, $nr_s = (nr_i - 1)$) connectivity in their accompanying coordination graphs. Each dot in the plots is the result of one run on a random problem generated with the parameters specified.

In plot 5.1a we see a nice polynomial relation between the number of agents, and the time the algorithm takes. If we look at the same data for the problem with medium connectivity (c and d), one sees that the algorithm needs more than polynomial time, but less than exponential time. Finally in the densely connected situation (plot *f*) we see a clear cut case of exponential time complexity.

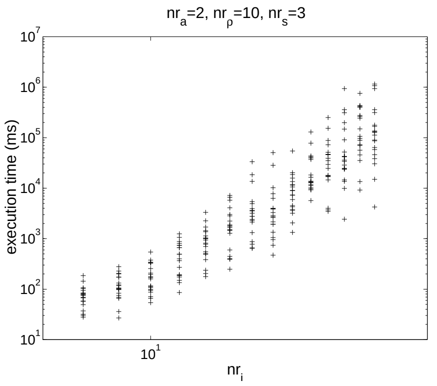
Looking at how well the problem scales to the number of actions, figure 5.1b shows something interesting. It looks like the time scales exponentially to the number of actions per agent, however we can distinguish (at least) two clearly different trends, one that climbs steeply, while the other, the lower one, might even be said not to be exponential at all. One could argue to see a similar division in figure 5.1e—which contains similar data for the problem with medium connectivity, however this is less clear. Looking for the exact causes of these two (or more) different lines falls outside the scope of this thesis.



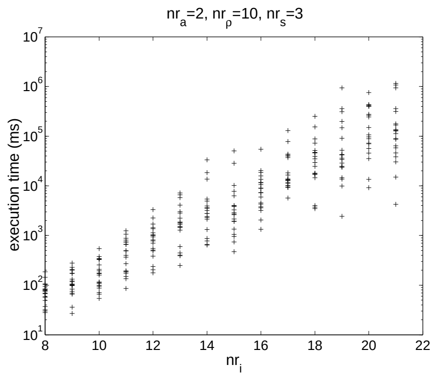
Lightly connected, scale agents;
both axes logarithmic
(a)



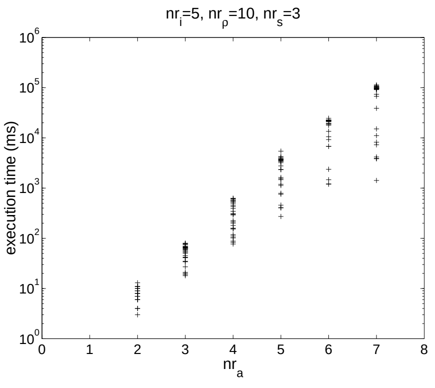
Lightly connected, scale actions;
y-axis logarithmic
(b)



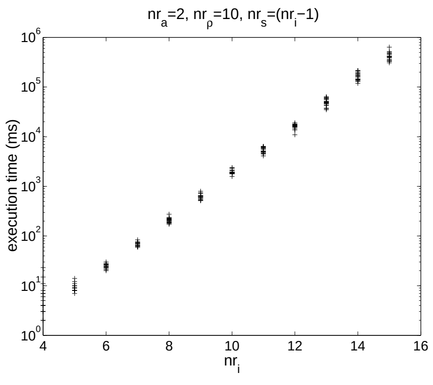
Medium connectivity, scale of agents;
both axes logarithmic
(c)



Medium connectivity, scale agents;
y-axis logarithmic
(d)



Medium connectivity, scale actions;
y-axis logarithmic
(e)



Densely connected, scale agents;
y-axis logarithmic
(f)

Figure 5.1: Performance of variable elimination

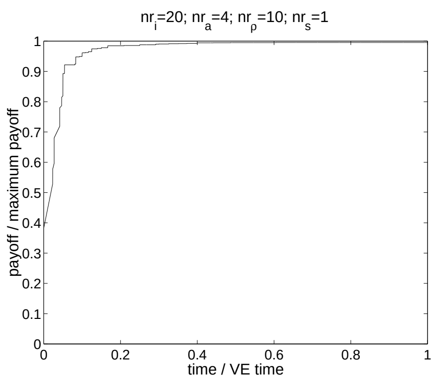
5.3 Comparing coordinate ascent and variable elimination

As we stated before, giving a good rating for how an anytime algorithm performs is hard. Some hypothetical algorithm might find a combined action that results in 50% of the maximum payoff in linear time, while it takes exponential time to find 90% of the payoff, and might not be guaranteed to reach 100% ever. This example shows that it clearly depends on the application how to judge the performance of such an algorithm.

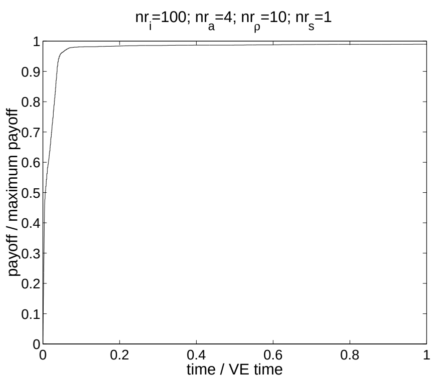
Instead of trying to find some manner on which to specify the performance of coordinate ascent directly, we compare it to variable elimination. Typically, the variable elimination algorithm runs for a while and then produces an answer. Since it cannot give any intermediate result, the payoff found is zero as long as the algorithm runs, and after it has finished the payoff jumps to maximum. In the plots comparing variable elimination and coordinate ascent, we use a scale so that the maximum payoff is 1. The time axis is scaled so that the time it took the variable elimination algorithm to complete is called 1. In this way the points in plots can be understood as the fraction of the payoff and execution time of the variable elimination algorithm. In other words, one can say that the trace of the variable elimination algorithm is a line from $(0, 0)$ to$(1, 0)$ to $(1, 1)$ per definition.

In figures 5.2 and 5.3 the comparison between coordinate ascent and variable elimination under different situations is shown. Coordinate ascent performs pretty well, and in quite some cases the climb of coordinate ascent was too steep to show on the normal scale, therefore note the scale of the x-axis in the plots. The plots were generated by solving 4 different randomly generated problems for each set of parameters. These problems were first solved by variable elimination once, and then by coordinate ascent 5 times, with different random seeds each time. Each coordinate ascent result was then scaled to its variable elimination companion, and the averages of the 20 runs were plotted. The jerky climb in the plots has two reasons. Firstly the coordinate ascent algorithm typically increases with jumps, and these jumps are still visible after the averages have been taken. Furthermore, in the tests with few agents or actions we ran into the limitations of our timing system. Time measurements had a 1 ms resolution, which proved insufficient in some cases.

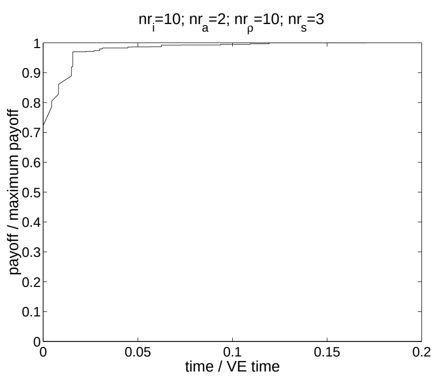
In all plots we see coordinate ascent perform rather well. In all situations, except the lightly connected one with 100 agents, coordinate ascent had reached a payoff higher than 99.9% of the maximum payoff after running the same time as the variable elimination algorithm. More impressively, the plots 5.2 *d* and *f*, and 5.3 *b* and *d* show that in situations with many agents or actions per agent, typically situations where variable elimination takes long to finish, coordinate ascent reaches near 100% payoff in times ranging



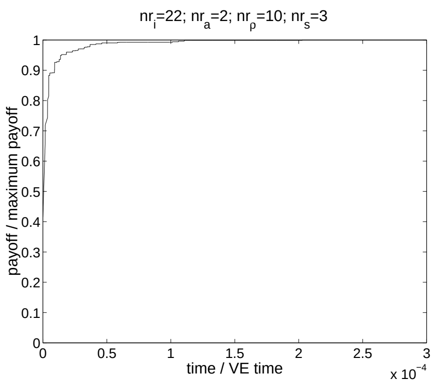
Lightly connected, few agents
(a)



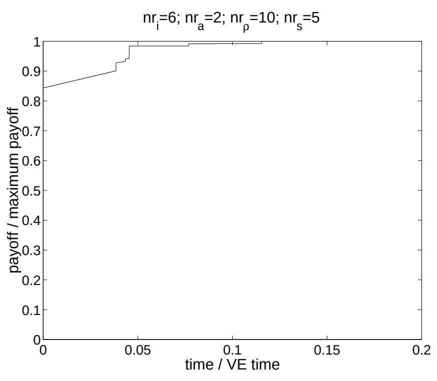
Lightly connected, many agents
(b)



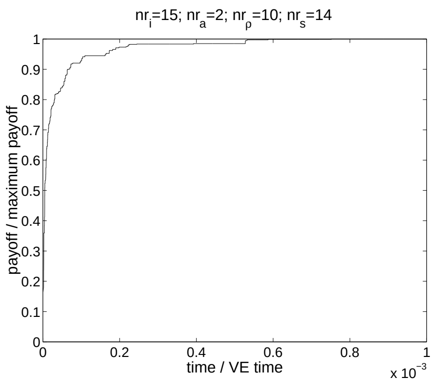
Medium connectivity, few agents
(c)



Medium connectivity, many agents
(d)

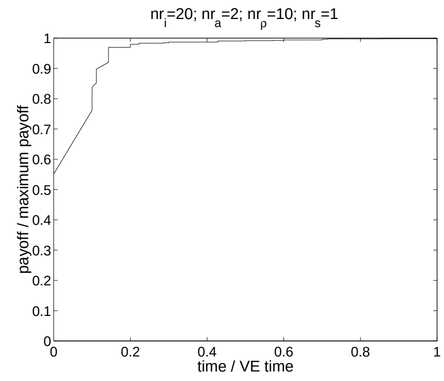


Densely connected, few agents
(e)

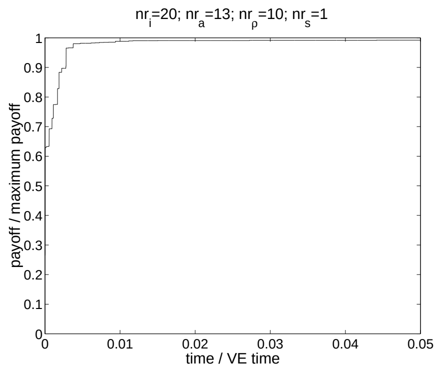


Densely connected, many agents
(f)

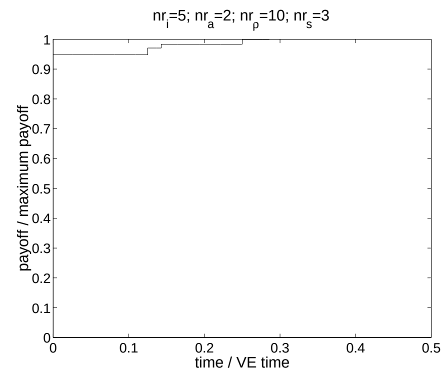
Figure 5.2: CA-VE comparison with varying numbers of agents.



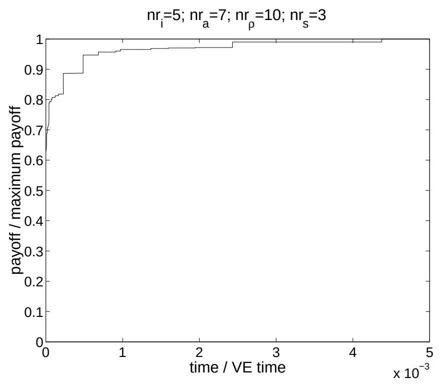
Lightly connected, few actions
(a)



Lightly connected, many actions
(b)



Medium connectivity, few actions
(c)



Medium connectivity, many actions
(d)

Figure 5.3: CA-VE comparison with varying numbers of actions.

between 0.015% and 2% of the time that variable elimination needs. With all these results it should still be noted that coordinate ascent does not know it has already reached the maximum payoff and will continue looking for better solutions.

Chapter 6

Conclusions and future work

6.1 Conclusions

From the results of chapter 5, it is not hard to spot that in many situations coordinate ascent produces extremely good results when compared to variable elimination. In section 5.2 it is argued that variable elimination does not scale well to the number of agents or the number of actions per agent in the situations where the accompanying coordination graph is complex (i.e. problems of medium and dense connectivity). We can see that especially in those situations (described by figures 5.2 *d* and *f*, and 5.3 *d*) coordinate ascent performs extremely well, suggesting that it scales considerably better than variable elimination. Even in the lightly connected case, where variable eliminations scales reasonable well, coordinate ascent produces almost similar results in the same amount of time, with the added benefactor of it being an anytime algorithm.

However, coordinate ascent can pose problems when used in distributed environments where communication possibilities are poor. Section 4.3 explains these problems, which basically come down to the fact that an anytime algorithm is very hard to make deterministic. For such an algorithm to be deterministic, the time that it runs should be deterministic as well, and this contradicts the idea that an anytime algorithm can run for any time, unless some very specific measures (with specific constraints) are taken. The problem with non-deterministic algorithms is that, if they are run in a replicated fashion, different agents might end up with different solutions, which might result in a non-coordinated combined action.

Furthermore, the coordinate ascent algorithm does not terminate until an answer is requested. It has no way of knowing when it has reached the optimal solution, nor does it guarantee convergence to this optimal solution[1].

[1]Theoretically speaking, one could argue that it does converge, and does terminate after each possible combined action has been tried as a starting position. However, we do not treat this as a practical option.

In conclusion we can say that in many cases coordinate ascent proves to be a very viable option for solving complicated coordination problems under time constraints, because it quite often reaches optimal (or close to optimal) solutions in a fraction of the time that variable elimination takes. An added bonus is that the algorithm returns intermediate results. However, many problems exist where either the non-determinability of the algorithm, or the fact that it does not guarantee convergence are sufficiently important to decide that coordinate ascent is unsuited. Provided that the problems are not too complicated, or in situations where time-constraints are of less importance, variable elimination will get the optimal result in finite time.

6.2 Applicability of the results in RoboCup

The RoboCup Simulation League is an excellent example of an environment which is poor in communication and requires the replicated system of coordination, and at the same time has very strict time constraints on its calculations. Currently the UvA Trilearn team uses variable elimination for its coordination efforts, and often time runs out for one of the agents before the calculation is complete. To avoid this, in the current implementation the number of used value rules is kept to a minimum. For coordinate ascent to work in this environment though, the algorithm should be deterministic.

We believe that the algorithm can be made sufficiently deterministic to work well in the RoboCup environment. Our tests show that especially in environments with few agents, a high level of connectivity and many different actions per agent, an environment like the one in which a RoboCup team needs to coordinate, coordinate ascent performs extremely well compared to variable elimination. In our tests in similar environments, we saw that the maximum payoff was always reached in just a fraction of the time needed for the variable elimination algorithm. When we assume that all agents have reached the globally maximum payoff before a result is needed, their running time is of no influence on the result any more, and therefore their running time is not a factor of non-determinability. This means that, assuming the coordinate ascent has enough time to reach the maximum payoff, its behaves as a deterministic algorithm. Because the running time available allows variable elimination to finish most of the time, the chances that coordinate ascent will not reach the maximum payoff are negligibly small.

In the future the coordination problems will become more complex, and the time available for them may decrease. Some simple tests on the problems should be enough in those cases to determine the chance that one or more or more of the agents do not find the maximum payoff in time. This way a risk analysis can decide whether coordinate ascent will be applicable in those domains.

6.3 Future work

In some of our (unpublished) tests we have seen that coordinate ascent seems to perform reasonable well with problems of larger complexity; we let the algorithm run on a densely connected, randomly generated problem of 400 agents and the characteristics of its results suggested that it was finding results close to the maximum payoff within seconds. We had no way of affirming this, since the problem is too large to run the variable elimination algorithm to determine the global maximum in payoff. It would be nice though to see how the coordinate ascent algorithm behaves in these (and larger) problems.

The plots comparing coordinate ascent with variable elimination all have a similar shape. It would suggest that a formula for the performance of coordinate ascent can be found, and might even be build on a theoretical analysis of the algorithm. Obviously such a general time-complexity function would assist greatly in determining runtime how close the algorithm is to finding the global maximum payoff.

Other anytime algorithms for coordination games have been proposed, some of which seem to guarantee convergence in limited time. It would be interesting to see how these algorithms behave compared to coordinate ascent. Furthermore, the suggestion has been made to port algorithms which work in Bayesian networks to coordination graphs [VEK04].

Appendix A

Variable elimination algorithm

A.1 Explanation of the algorithm

Variable elimination and coordination graphs are often used intertwined. It should be noted however that these are two different things: A coordination graph is a notation of a decision making problem, variable elimination is a way to solve this problem. It is true that variable elimination typically takes a coordination graph as its input, and explaining the variable elimination algorithm is easiest done by using a coordination graph. This does not undermine the fact that there are other ways of solving coordination graphs (coordinate ascent for instance).

The idea behind the algorithm is to take a coordination graph, and eliminate its nodes (or agents) one by one until only one is left. Obviously, deciding on the optimal action in a 'one agent coordination graph' is trivial. After this final agent has received its action, the elimination is reversed one step. We now have a coordination graph with two agents, of which one has already chosen her action; this one is being removed from the graph, so we have another 'one agent coordination graph' to solve. This process is continued until all agents have actions assigned, and this is the optimal combined action.

In the last paragraph we stepped over the question of how to eliminate agents. We already argued in section 3.4 that, since all agents are only interested in optimising collective payoff, it doesn't matter which agent has which value rule, and that we can move value rules around agents in a coordination graph. Furthermore we state that any agent not appearing in any value rules (i.e. without any value rules itself and without any edges in the coordination graph) is of no interest to the final result and can safely be removed from the graph.

First thing to do is decide which agent to eliminate first. The order in which the agents are eliminated has no effect on the final outcome, but does affect the speed of the algorithm. It is known that finding the optimal elimination order is an NP-hard problem, so trying to calculate that order first does not speed up things. A random elimination order will give acceptable results, however we like to improve on those a little bit by starting with the node with the least edges. This will be easiest to solve, and will result in less new edges produced than eliminating a heavily connected agent.

Once we have selected the node to eliminate, we collect all value rules from all agents that include this agent and transfer them to this agent (note that we only have to check this agent's direct neighbours). Now the agent starts removing herself from the rules. The agent knows the maximum payoff she can get given a set of actions from other agents, so she changes the value rules to reflect that. Then she sends the modified value rules to her neighbours, so that she does not have any rules left and does not appear in any value rule anymore. Now, she can remove herself. This process is repeated until we reach the final agent, after which the elimination is reversed to get all the actions.

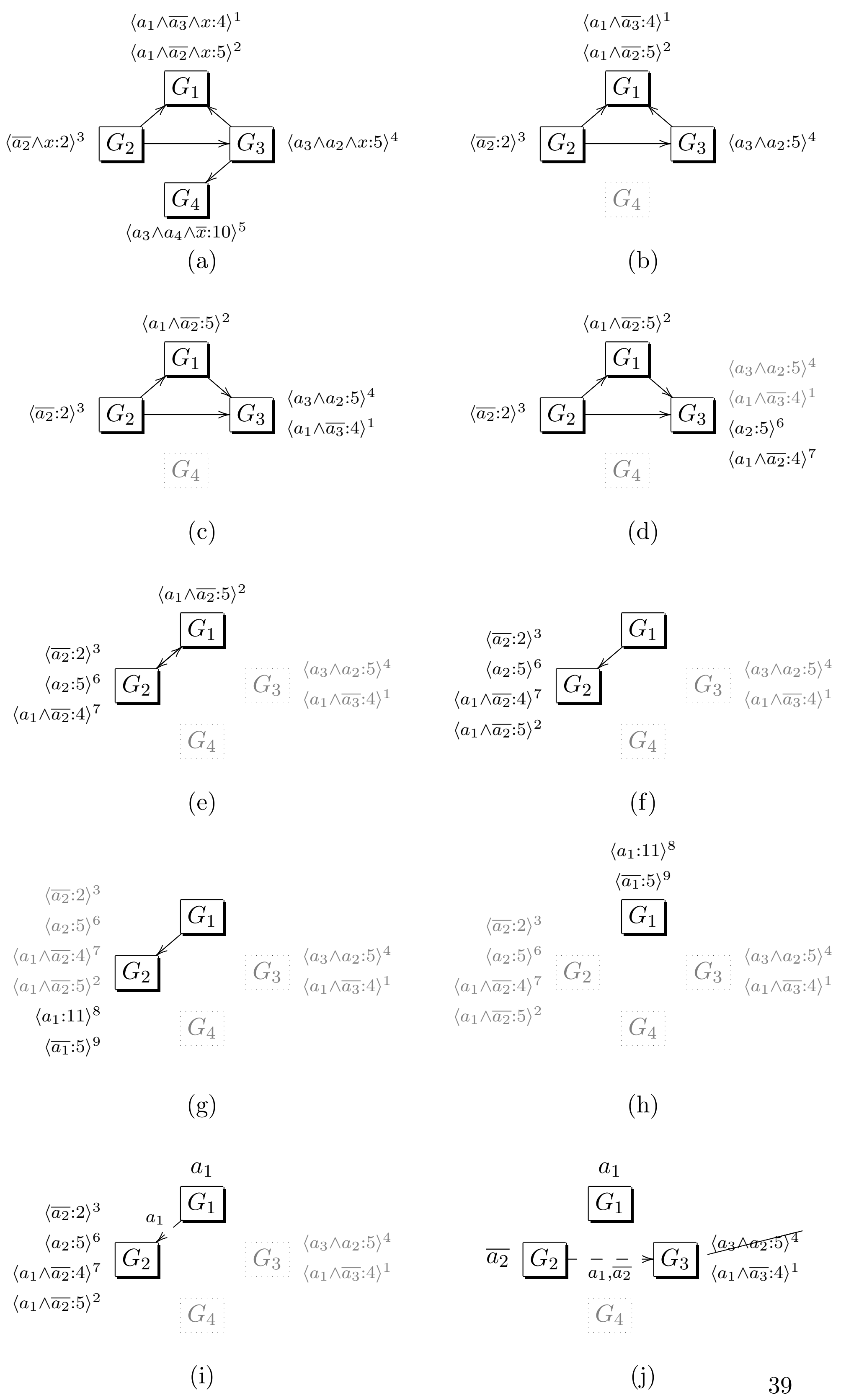
Note that this process can be done centralised, but alternatively each agent can only eliminate herself, so that the process is being done distributed (albeit not in parallel). For this to work, a lot of communication is necessary. As described in section A.3, Guestrin's implementation uses this distributed method, whereas our own implementation uses the centralised version of the algorithm.

A.2 Example

Perhaps the best way to get a feeling for variable elimination is to see an example of the algorithm in action. Figure A.1 shows step by step how we eliminate the agents in a coordination graph. The example does use the context (as opposed to what we suggested in section 3.5.2), but only to show what a minor step it is in the whole process. In this case we have numbered the value rules, to be able to refer to them in the text.

We start at a with a coordination graph. It is given that the context is x, so we remove all rules from the graph that are not consistent with context x and update the graph. b shows that G_4 does not have any rules anymore, and her action does not have an effect on the combined payoff, so she is eliminated. Next we want to eliminate G_3, to which end we collect all rules with a_3 in them at G_3, by moving rule 1 from G_1 to G_3: c. Note that by doing this the dependence relation between G_1 and G_3 reverses. Now G_3 eliminates herself from these rules.

When eliminating an agent from a set of value rules, the resulting rules must describe the amount of payoff this agent can get when the other agents play their actions. In this example, from rule 4 we can see that if G_2 selects



$\langle a_1 \wedge \overline{a_3} \wedge x{:}4\rangle^1$
$\langle a_1 \wedge \overline{a_2} \wedge x{:}5\rangle^2$
$\langle \overline{a_2} \wedge x{:}2\rangle^3$
$\langle a_3 \wedge a_2 \wedge x{:}5\rangle^4$
$\langle a_3 \wedge a_4 \wedge \overline{x}{:}10\rangle^5$
G_1
G_2
G_3
G_4
(a)
$\langle a_1 \wedge \overline{a_3}{:}4\rangle^1$
$\langle a_1 \wedge \overline{a_2}{:}5\rangle^2$
$\langle \overline{a_2}{:}2\rangle^3$
$\langle a_3 \wedge a_2{:}5\rangle^4$
(b)
$\langle a_1 \wedge \overline{a_2}{:}5\rangle^2$
$\langle \overline{a_2}{:}2\rangle^3$
$\langle a_3 \wedge a_2{:}5\rangle^4$
$\langle a_1 \wedge \overline{a_3}{:}4\rangle^1$
(c)
$\langle a_1 \wedge \overline{a_2}{:}5\rangle^2$
$\langle \overline{a_2}{:}2\rangle^3$
$\langle a_3 \wedge a_2{:}5\rangle^4$
$\langle a_1 \wedge \overline{a_3}{:}4\rangle^1$
$\langle a_2{:}5\rangle^6$
$\langle a_1 \wedge \overline{a_2}{:}4\rangle^7$
(d)
$\langle a_1 \wedge \overline{a_2}{:}5\rangle^2$
$\langle \overline{a_2}{:}2\rangle^3$
$\langle a_2{:}5\rangle^6$
$\langle a_1 \wedge \overline{a_2}{:}4\rangle^7$
$\langle a_3 \wedge a_2{:}5\rangle^4$
$\langle a_1 \wedge \overline{a_3}{:}4\rangle^1$
(e)
$\langle \overline{a_2}{:}2\rangle^3$
$\langle a_2{:}5\rangle^6$
$\langle a_1 \wedge \overline{a_2}{:}4\rangle^7$
$\langle a_1 \wedge \overline{a_2}{:}5\rangle^2$
$\langle a_3 \wedge a_2{:}5\rangle^4$
$\langle a_1 \wedge \overline{a_3}{:}4\rangle^1$
(f)
$\langle \overline{a_2}{:}2\rangle^3$
$\langle a_2{:}5\rangle^6$
$\langle a_1 \wedge \overline{a_2}{:}4\rangle^7$
$\langle a_1 \wedge \overline{a_2}{:}5\rangle^2$
$\langle a_1{:}11\rangle^8$
$\langle \overline{a_1}{:}5\rangle^9$
$\langle a_3 \wedge a_2{:}5\rangle^4$
$\langle a_1 \wedge \overline{a_3}{:}4\rangle^1$
(g)
$\langle a_1{:}11\rangle^8$
$\langle \overline{a_1}{:}5\rangle^9$
$\langle \overline{a_2}{:}2\rangle^3$
$\langle a_2{:}5\rangle^6$
$\langle a_1 \wedge \overline{a_2}{:}4\rangle^7$
$\langle a_1 \wedge \overline{a_2}{:}5\rangle^2$
$\langle a_3 \wedge a_2{:}5\rangle^4$
$\langle a_1 \wedge \overline{a_3}{:}4\rangle^1$
(h)
a_1
a_1
$\langle \overline{a_2}{:}2\rangle^3$
$\langle a_2{:}5\rangle^6$
$\langle a_1 \wedge \overline{a_2}{:}4\rangle^7$
$\langle a_1 \wedge \overline{a_2}{:}5\rangle^2$
$\langle a_3 \wedge a_2{:}5\rangle^4$
$\langle a_1 \wedge \overline{a_3}{:}4\rangle^1$
(i)
a_1
$\overline{a_2}$
$\overline{a_1},\overline{a_2}$
~~$\langle a_3 \wedge a_2{:}5\rangle^4$~~
$\langle a_1 \wedge \overline{a_3}{:}4\rangle^1$
(j)

Figure A.1: Example of variable elimination. Note that the parts of the figures noted in grey are the eliminated (yet cached) parts.

action a_2, G_3 can receive a payoff of 5 by doing a_3, hence rule 6 in figure *d*. If G_1 chooses action a_1, G_3 could receive a payoff of 4 by selecting $\overline{a_3}$ (see rule 1), however will only do that if G_2 has chosen a_2, since otherwise action a_3 would be selected for a payoff of 5, so we get rule 7.Now G_3 sends the resulting rules to G_2 and has eliminated herself from all rules, so from the coordination graph: *e*. Note that G_3 keeps the original rules in cache for later use.

Next, G_2 eliminates herself in a similar manner, by receiving the rules (f), creating the new rules 8 and 9 from them (g) and sending them to G_1 in figure *h*.

Now G_1 has two rules, which each depend only on her own action. Obviously she selects action a_1 for a total payoff of 11. She sends this decision back to G_2, who reactivates her cached rules 3, 6, 7 and 2 (i), and looks though them to decide that choosing action $\overline{a_2}$ will result in payoff 11, whereas selecting a_2 will only result in a payoff of 5, hence she chooses $\overline{a_2}$, and sends her choice, together with G_1's to G_3. G_3 looks though her cached rules, and decides that rule 4 will not work anymore (since that would have required the action a_2), so she has the choice of selecting $\overline{a_3}$ for a payoff of 4, or a_3 for a payoff of 0.

As we discussed before, the action of G_4 does not have any influence on the total result. Obviously a clean algorithm selects an action for G_4. Randomly selecting would unnecessarily make the algorithm non-deterministic, so usually the actions are ordered in some way, and in case of a tie always the one with lowest number in the ordering is chosen, say a_4.

So the optimal combined action is $a_1, \overline{a_2}, \overline{a_3}, a_4$. If we fill in these values in the original coordination graph of figure *a*, we would see that value rules 1, 2 and 3 are valid, giving a total payoff of 11, not coincidentally the same as the best choice for the last agent left in *h*, rule 8.

A.3 Code

Because we use the variable elimination algorithm as a basis on which to test the performance of the other algorithms, it is important to check exactly which algorithm we used. In addition it would be interesting to check the full implementation, however it would be too much to include the full listing of the program. It will suffice to say that the implementation of the algorithm was done in the same language (Java) as the implementation of coordinate ascent.

As we noted before, the original algorithm from [Gue03] uses a distributed (however not parallelisable) way to compute the optimal action. The pseudo-code for this algorithm can be seen in algorithm A.1, which makes use of the functions in algorithm A.2 and A.3.

Our implementation uses a slightly modified version of the algorithm,

Algorithm A.1 Guestrin's variable elimination

```
  **define:** scope($\rho$) is the set of all agents that define an action in $\rho$
  **define:** $a_i$ the action of agent $i$
  **define:** $a^*$ the optimal combined action
  **define:** $P_i$ all rules currently held by agent $i$
  **define:** $\rho(a)$ is the payoff rule $\rho$ gives, in case of action $a$.

  *each agent runs:*
  $i \leftarrow$ this agent
  $\epsilon^- \leftarrow$ the agent in the elimination order that comes before this one
  $\epsilon^+ \leftarrow$ the agent in the elimination order that comes after this one
  **if** $\epsilon^- \neq \emptyset$ **then**
    wait for a signal from $\epsilon^-$
  **end if**
  //collect all rules containing $i$ from neighbours
  $\rho \leftarrow \emptyset$
  **for** $j \in$ neighbours **do**
    $\rho \overset{\cup}{\leftarrow} \{r \in P_j | i \in \text{scope}(r)\}$
  **end for**
  $\rho\prime \leftarrow$ rulemaxout$(i, \rho)$
  **for** $r \in \rho\prime$ **do**
    $j \leftarrow$ one element from(scope($r$))
    send $r$ to $j$
    add edge from $k$ to $j$ for each $k \in$ scope$(r), k \neq j$
  **end for**
  delete $i$ and all edges to or from $i$ from coordination graph
  **if** $\epsilon^+ \neq \emptyset$ **then**
    send signal to $\epsilon^+$
    wait for signal from $\epsilon^+$
    receive $a^*$ from $\epsilon^+$
  **else**
    $a^* \leftarrow \emptyset$
  **end if**
  $a_i^* \leftarrow \arg\max_{a_i \in A_i}(\sum \rho(a^* \cup a_i))$
  $a^* \overset{\cup}{\leftarrow} a_i^*$
  **if** $\epsilon^- \neq \emptyset$ **then**
    send signal to $\epsilon^-$
    send $a^*$ to $\epsilon^-$
  **end if**
```

Algorithm A.2 The *ruleoutmax* function

```
**define:** $a_i$ the action of agent $i$
**define:** $A_i$ = Dom($a_i$)
**define:** $\rho = \langle c : v \rangle \Rightarrow c$ =actions($\rho$) and $v$ =payoff($\rho$)

function rulemaxout($i, \rho$)
$\rho\prime \leftarrow \emptyset$
//add completing rules
$\rho \overset{\cup}{\leftarrow} \{\langle a_i = k : 0\rangle | k \in A_i\}$
//Summing consistent rules
**while** there are two consistent rules $\rho_p$ and $\rho_q$ **do**
  **if** actions($\rho_p$) = actions($\rho_q$) **then**
    replace $\rho_p$ and $\rho_q$ by $\langle$actions($\rho_p$) : payoff($\rho_p$) + payoff($\rho_q$)$\rangle$
  **else**
    replace $\rho_p$ and $\rho_q$ by split($\rho_p\angle$actions($\rho_q$))$\cup$ split($\rho_q\angle$actions($\rho_p$))
  **end if**
**end while**
//Creating the rules without $a_i$
**repeat**
  **if** there are rules $\langle c \wedge a_i = m : v_k\rangle, \forall m \in A_i$ **then**
    remove these rules from $\rho$ and add rule $\langle c : max_k v_k\rangle$ to $\rho\prime$
  **else**
    select two rules $\rho_p = \langle c_1 \wedge a_i = m : v_1\rangle$ and $\rho_q = \langle c_2 \wedge a_i = n : v_2\rangle$
    so that $c_1$ is consistent with $c_2$, but $c_1 \neq c_2$ and replace them by
    split($\rho_p\angle c_2$) $\cup$ split($\rho_q\angle c_1$)
  **end if**
**until** $\rho = \emptyset$
return $\rho\prime$
```

Algorithm A.3 The *split* function

```
**define:** scope($\rho$) is the set of all agents that define an action in $\rho$
**define:** $\rho = \langle c : v \rangle \Rightarrow c$ =actions($\rho$) and $v$ =payoff($\rho$)

function split($\rho\angle b$)
$c \leftarrow$ action($\rho$)
**if** $c$ is not constistent with $b$ **then**
  $\{\rho\}$
**else if** scope($b$) $\subseteq$ scope($c$) **then**
  $\{\rho\}$
**else**
  {split($r\angle b | r \in$split($\rho\angle Y$)}, for some $Y \in$ (scope($b$) - scope($c$))
**end if**
```

Algorithm A.4 Our variable elimination

```
  **define:** scope($\rho$) is the set of all agents that define an action in $\rho$
  **define:** $a_i$ the action of agent $i$
  **define:** $\rho(a)$ is the payoff rule $\rho$ gives, in case of action $a$.

  $\epsilon \leftarrow$ elimination order, so that $\epsilon_1$ is first agent to be eliminated
  $P \leftarrow$ all valuerules
  $j = 1$
  **while** $j < |\epsilon|$ **do** {loop over all but one agents in the elimination order}
    $i \leftarrow \epsilon_{j++}$
    $\rho^i \leftarrow \{r \in P | i \in \text{scope}(r)\}$
    $P \leftarrow P - \rho^i$
    $\rho^i\prime \leftarrow \text{rulemaxout}(i, \rho^i)$
    $P \overset{\cup}{\leftarrow} \rho^i\prime$
  **end while**
  $a^* \leftarrow \emptyset$
  **while** $j > 0$ **do** {loop over all agents reversed order}
    $i \leftarrow \epsilon_{j--}$
    $a_i^* \leftarrow \arg\max_{a_i \in A_i}(\sum \rho^i(a^* \cup a_i))$
    $a^* \overset{\cup}{\leftarrow} a_i^*$
  **end while**
```

which is not distributed. Which approach is better fully depends on the possibilities and constraints of the environment in which the algorithm is executed. A situation where the agents will have to perform a lot of other tasks as well, or have partially hidden payoff functions, will be better served by the former method, whereas the latter method works better in a communication-poor environment. Our version of the algorithm is in algorithm A.4. Note that it uses the same 'split' and 'rulemaxout' functions as Guestrin's version.

Appendix B

Random problem generation

When comparing algorithms it is important to clearly note the kind of problems one tests the algorithm on. Some algorithms might perform really well on one sort of problem, while other might be better suited to tackle another sort of problems. Multi-agent systems is such a large field that there is no standard problem one can test against. For the tests run in this thesis, we used problems that were generated by our random problem generator. Algorithm B.1 contains the pseudo code for used problem generator. This pseudo code is kept easy-readable, to use it to generate highly connected problems some optimisations are suggested (and were used). These fall outside the scope of this document though.

The input to the random problem generator are values for the number of agents, the number of different actions per agent, maximum number of parents (incoming edges) an agent has and the average number of value rules that are being generated per agent. The output of the algorithm is a list of value rules for each agent. These lists were then fed into the decision making algorithms.

We did believe that these aspects of the problem define how complicated a problem is. Therefore we have tried to do scaling tests on each of these aspects. We have not tried whether scaling on other aspects of the problem (e.g. length of each value rule) has an effect on performance.

Algorithm B.1 The random problem generator

```
  **define:** $a_i$ the action of agent $i$
  **define:** random($a, b$) selects a random integerin the interval $[a, b]$
  **define:** randomvaluelength($n$) selects a random integer; each number $k$
  has a chance of
      $$\frac{\binom{n}{k}}{2^n}$$
  of being returned. This allows for a fair random selection between all
  possible value rules
  **define:** $\Leftarrow$ and $\Rightarrow$ denote in- and output respectively

  $nr_i \Leftarrow$ the number of agents
  $nr_a \Leftarrow$ the number of actions per agent
  $nr_s \Leftarrow$ maximum number of parents per agent
  $nr_\rho \Leftarrow$ number of value rules per agent
  $MINPAYOFF \leftarrow 1$
  $MAXPAYOFF \leftarrow 10$
  $I \leftarrow$ set of $nr_i$ agents
  $A_i \leftarrow$ set of $nr_a$ actions for each $i \in I$
  **for** $i \in I$ **do**
    $S \leftarrow$ set of $nr_s$ agents from $I - \{i\}$
    $\rho \leftarrow \emptyset$
    **for** $j = 1$ to $nr_\rho$ **do**
      $valuerulelength \leftarrow$randomvaluelength($nr_s$)
      $v \leftarrow$random($MINPAYOFF, MAXPAYOFF$)
      $S\prime \leftarrow$ random set of $valuerulelength$ agents from $S$
      $c \leftarrow \{a_i\} \in A_i \cup \bigcup_{s \in S\prime}(\{a_s\} \in A_s)$
      $\rho \overset{\cup}{\leftarrow} \langle c : v \rangle$
    **end for**
    $\rho \Rightarrow$ valuerules for agent $i$
  **end for**
```

Bibliography

[Gue03] C. E. Guestrin. *Planning under Uncertainty in Complex Structured Environments.* PhD thesis, Stanford University, Computer Science Department, Stanford University, August 2003. http://robotics.stanford.edu/˜guestrin/publications.html#thesis.

[GVK02] C. E. Guestrin, S. Venkataraman, and D. Koller. Context-specific multiagent coordination and planning with factored MDPs. In *Proc. 8th Nation. Conf. on Artificial Intelligence*, Edmonton, Canada, July 2002.

[KSV03] J. R. Kok, M. T. J. Spaan, and N. Vlassis. Multi-robot decision making using coordination graphs. In *Proc. 11th Int. Conf. on Advanced Robotics*, Coimbra, Portugal, June 2003.

[SB98] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, MA, 1998.

[VEK04] N. Vlassis, R. K. Elhorst, and J. R. Kok. Anytime algorithms for multiagent decision making using coordination graphs. In *Proc. Int. Conf. on Systems, Man and Cybernetics*, The Hague, The Netherlands, October 2004. (To appear).

[Vla03] N. Vlassis. A concise introduction to multiagent systems and distributed AI. Informatics Institute, University of Amsterdam, September 2003. http://www.science.uva.nl/˜vlassis/cimasdai.