Application of Machine Learning Methods in a Multi-Agent System

Michael Wurst
supervisors:
Dr. Michal Pechouček,
Dr. Michael Schanz,
Prof. Dr. Olga Stěpankova

February 23, 2001

Abstract

The aim of this project is to apply Machine Learning methods in order to improve the performance of ProPlanT. Machine Learning is applied to find patterns in the communication among the agents. These pattern are used to provide a human user of ProPlanT with useful information, enabling him to optimize the system. A framework of tools has been developed, that allows the application of different transformation and learning methods. This framework has been applied to the problem of finding regularities concerning the formation and development of bottlenecks in the system resources. This should help the user to decide which resources to add to the system.

Contents

I Theoretical Background 5

1 Agents and Multi-Agent Systems 5
- 1.1 Why should Multi-Agent Systems be used? 5
- 1.2 Classification of Multi-Agent Systems 5

2 Machine Learning 6
- 2.1 Definition of Machine Learning 6
- 2.2 Machine Learning paradigms and algorithms 8
- 2.3 Knowledge Discovery 10
- 2.4 General Machine Learning /Knowledge Discovery tasks . . 12
  - 2.4.1 Classification 12
  - 2.4.2 Numerical prediction 12
  - 2.4.3 Association rules 12

3 Multi-Agent Systems and Machine Learning 13
- 3.1 Why should Machine Learning be used in Multi-Agent Systems at all? 13
- 3.2 Basic dimensions of Machine Learning in Multi-Agent Systems 13
- 3.3 Overview on earlier research on this subject 14

4 Multi-Agent Systems and Knowledge Discovery 15
- 4.1 The basic idea of applying Knowledge Discovery to Multi-Agent Systems 15
- 4.2 Knowledge Discovery in Multi-Agent Systems vs. Distributed Learning and Adaption in Multi-Agent Systems 16
- 4.3 Knowledge Discovery in Multi-Agent Systems vs. mathematical analysis of Multi-Agent Systems 17

5 ProPlanT 18
- 5.1 Types and structure of agents 18
- 5.2 Planning in ProPlanT 19
- 5.3 Knowledge representation and maintenance 19
- 5.4 The Meta-Agent 20
- 5.5 Messages in ProPlanT 20
  - 5.5.1 Administrative Messages 21
  - 5.5.2 Knowledge-base Messages 21
  - 5.5.3 Planning Messages 21

6 Measuring the performance of a Multi-Agent System for planning 21
- 6.1 quality of the plan 22
- 6.2 the planning process 22

7 Possible applications of Knowledge Discovery in ProPlanT 23

II A Framework of tools to support Knowledge Discovery and Machine Learning in ProPlanT 24

8 Overview of the tools **24**
8.1 Which tools are needed? 24
8.2 Tools, which were developed or used to support Knowledge Discovery in ProPlanT 25
8.2.1 Overview of the actual developed or applied tools . . 25
8.2.2 How do these tools work together? 26

9 The Generator Agent **27**
9.1 Introduction 27
9.2 The user interface 27
9.3 The sequence of requests 28
9.3.1 Reading the sequence of requests from a file 28
9.3.2 Generating the sequence of requests randomly 29
9.4 Practical Usage of the Generator Agent 29

10 Generation of more complex sequences **29**

11 The LogWriter Agent **30**
11.1 Introduction 30
11.2 The conversion of KQML Messages to Prolog facts 31
11.2.1 Options for the conversion 31
11.2.2 Handling of messages that contain multiple entries . 32
11.2.3 Conversion of the individual messages 32
11.3 Practical Usage of LogWriter 33
11.4 Some examples 33
11.4.1 Example 1: Filtering 33
11.4.2 Example 2: Simple transformation 33

12 Performance Measure **34**

13 The Knowledge Discovery Library **34**

III A practical example **35**

14 Detection and prediction of bottlenecks **35**
14.1 Introduction 35
14.2 Exact definition of the problem 36
14.2.1 Scope of the problem and some simplifications . . . 37
14.2.2 Remarks on terminology used in this section 38
14.2.3 The definition of a bottleneck 38
14.3 The test community 39
14.4 Background Knowledge 41
14.5 Detecting bottlenecks 42
14.5.1 Step 1:A simple algorithm to simulate planning . . . 43
14.5.2 Step 2:A Knowledge Discovery system to predict a correction factor for step 1 43
14.5.3 Evaluation 46
14.6 Predicting the future development of bottlenecks 50
14.6.1 Step 1: Measure the strength of a bottleneck 51
14.6.2 Step 2: Calculating the average change 51
14.6.3 Step 3: Calculate the expected total change 52
14.6.4 Evaluation and an Example 52
14.7 Open problems and future work 53

Introduction

Multi-Agent Systems have gained an increasing importance in the last years. While they are successfully applied in many domains, it showed that the ability to learn and to adapt to new situations is a central point in the development of Multi-Agent Systems. This work investigates the possibilities to apply Machine Learning methods to communication between agents in a special Multi-Agent System for distributed planning, the ProPlanT system. The basic idea is to observe the communication between the agents in this system and to find pattern in this communication, which can be used further to optimize the performance of the system.

The documentation is split into three parts:

I. *Theoretical background.*
This part gives a brief overview of Multi-Agent Systems and ProPlanT, Machine Learning and some basic problems of Machine Learning in Multi-Agent Systems. After this, the idea to apply Machine Learning methods to the communication of Multi-Agent Systems is presented as well as the benefits and limitations of such an approach. At the end of this part, the basic possibilities to apply this methods to ProPlanT are analyzed.

II. *A Framework of tools*
In order to make it possible to perform different experiments, a framework of tools was developed. These tools are abstract and simple enough to support different kinds of applications, but on the other hand they can save the user a lot of effort and allow him to focus on his work, rather than on technical details of the ProPlanT System. The documentation contains an analysis, which tools are necessary and how they work together and then proceeds with a description of the individual programs and libraries.

III. *Detection and prediction of bottlenecks in ProPlanT*
In this section, one of the possibilities to use Machine Learning in ProPlanT is investigated further: the detection and prediction of bottlenecks. This should also demonstrate, how the different tools, which are described in the second part, can work together to support a complex analysis of the system's behavior. While the first and the second part do not presume special limitations for the setting, in which learning is applied, for the experiments in the third part, we assume a simplified setting. Mainly only communities, which contain only one top-level agent (in ProPlanT a Production Planning Agent), and which have a fixed structure of task-decompositions, are considered. Furthermore the decomposition-tree cannot contain any alternative decompositions (contains no OR-nodes). Finally only the time needed for the execution of the resulting plan is considered, not the cost. To keep the system as general as possible, the time between two requests to the system is not considered as well, which means, that we assume, that the system is driven in batch-mode.

Part I

Theoretical Background

1 Agents and Multi-Agent Systems

As the word says, Multi-Agent Systems are collections of agents. What is an agent? The are many different definitions for the term "agent". Some of the most common characteristics of agents are however that they have their own goals and beliefs, and are to some degree *autonomous*, *rational* and *social*. Autonomy does not only mean that every agent has some degree of freedom, but also that he is able to give himself rules and then to follow these rules. This is important, because if an agent were only free in some way, it would be impossible for the outside world to rely on this agent. Whereas an agent, that is autonomous, is able to agree to commitments with other agents and then to act according to these commitments. Rationality means that the agent tries to achieve his goal in an optimal way, that all of his actions are optimal to reach his goal. Social behavior means that agents are able to communicate with other agents, to co- operate with them, to negotiate, to share knowledge, etc. Multi-Agent Systems can be seen as a collection of entities, which have their own goals and beliefs, which are to some degree autonomous, rational and social and are able to co-operate with each other.

1.1 Why should Multi-Agent Systems be used?

Although most problems that can be solved with Multi-Agent Systems, could also be solved in a "classical" way, there are several benefits of designing a system as a Multi-Agent System:

- It is easier to cope with a complex system, which is split up in smaller units. This may be also true for modular programming, but Multi-Agent Systems have the additional advantage, that within the individual agents are encapsulated not only objects or functions, but also goals, knowledge, etc.
- It is often difficult to make use of parallelism in centralized systems. Multi-Agent Systems can make use of parallel computing in a natural way. This helps to increase the performance of the whole system.
- Multi-Agent Systems are often more tolerant against errors. If one unit (one agent) looses some capabilities, the agent can be replaced or the tasks of this agent can be performed by another agent. In a centralized system it is mostly very complicated to proceed after a unit of the system was damaged.
- Some systems are Multi-Agent Systems because of their structure, for example groups of robots.

1.2 Classification of Multi-Agent Systems

There are many possible ways to classify Multi-Agent Systems. In [4] the author distinguishes three different aspects of Multi-Agent Systems:

environment, *agent-agent relation* and the *agents* themselves.

1. *Environment*
   Some possible properties of the environment, that have been taken into consideration: (see [4] p. 4)
   - To what extend is the environment known to the agent?
   - Is the environment predictable for the agent?
   - In which way can the agent control its environment?
   - Is the environment historical? This means, do future states depend on the entire history?
   - Can the environment change, while the agent is deliberating?
2. *Agent-Agent relation*
   In most cases the agents in a Multi-Agent System have to communicate to reach their individual goals or at least to reach them more efficiently. There are many different possibilities of agent-agent interaction, these are some basic properties:
   (a) frequency of interaction
   (b) persistency of interaction
   (c) pattern of interaction
   (d) purpose of interaction

   In order to communicate, the agents have to have a common language. While this is easy if all of the agents in the Multi-Agent System are homogeneous it can be quite difficult if agents from different origins, and with different abilities have to communicate. Therefore the Knowledge Query and Manipulation Language (KQML) and the Knowledge Interchange Format (KIF) where developed. KQML is a speech-act-based language, which can be used for all kinds of coordination among agents. KIF is a language to represent knowledge in an agent independent way.
3. *The agents*
   Some of the fundamental properties of agents were described above, but to what extend an agent is autonomous, social or rational or how his beliefs and goals are represented can vary significantly. Some systems consist of few, very sophisticated agents, others of a large amount of very simple agents. The agents in a system can be all of the same structure or of different structure, homogeneous or heterogeneous. The role that an agents fulfills can be fixed or dynamic.The agents in a Multi-Agent System can all work on the same goal or their goals can be conflicting.

This section should only give a brief overview on the subject of Multi-Agent Systems. For a detailed representation of the basics of Multi-Agent Systems as well as many advanced issues see [2].

2 Machine Learning

2.1 Definition of Machine Learning

There are many different definitions for the term "Machine Learning". In the simplest case, learning means only storing data. But normally it is expected that the learner generalizes from its experience and finds a more abstract hypothesis. The underlying assumption is that the experience the learner makes is governed by some regularity and that the learner can find this regularity. Otherwise it would make no sense to try to generalize over the observed data, because every new experience would be completely random to the learner. It is also possible that there is already a theory that describes parts of the domain in which learning takes place. In this case it is possible for the learner to use this "background knowledge" in learning. In fact human learning seems to rely on the synthesis of experience and background knowledge. It is important to keep this in mind, because many Machine Learning methods do not provide the possibility to use background knowledge and it is much harder to learn only from experience without any knowledge about the domain, which is the source of this experience. As this work is not so much concerned with the theory of Machine Learning, but with its application, the remainder of the section gives only a brief overview on Machine Learning. The basic problem is, how to define a learning problem in such a way, that it can be "solved" by a program, which means that there exists an algorithm that solves it. I will refer to the following definition taken from the book "Machine Learning" by T. Mitchell:

> "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E"([1] S.10)

Using this definition, defining a learning problem, means finding proper definitions for T, E and P.

- *The class of tasks T*
  The definition for T has to be stated in some formal way, for example as a mathematical function. How T is defined depends completely on the learnig problem and the domain, but in general it must be defined such that it can be learned from experience E.
- *The performance measure P*
  While it is easy to find a performance measure in some settings, in other cases it could be less obvious. A program that learns to play chess would be said to perform better, if it wins more often, but what should be the performance measure for a program that should learn how to write poems? Furthermore in many settings we don't have a deterministic environment and so the performance measure must be defined in a way that reflects this. For example a program that learns to play Backgammon. It could be just luck that the program wins all the time, after it has been trained. In these cases statistical methods have to be used to decide whether it should be said that the program has learned. It's not possible just to compare the number of games won.
- *The experience E*
  E defines the experience that an agent should use for learning. In some cases this experience can be explicitly chosen by the designer of the system, but in most cases it will be limited by the environment, in which the program acts (e.g. a learning robot has only it's

sensual data). One important distinction should be made between the different ways the system is told what to learn:

- "supervised learning": a teacher provides the program directly with the desired behavior for a special case (e.g. a program that should learn to recognize hand-written letters, could be provided with the picture of a letter and the information, which letter it is). So in these settings the program is provided with a pair (given value, desired value).
- "reinforcement learning": the program is only provided with the utility of its behavior concerning the learning task. It has to find out how to change its behavior to increase this utility by itself.
- "unsupervised learning": the program has even to find out what's useful on it's own.

Another problem is the quality of the experience. This quality depends on the learning task. Experience, which is of high quality for one task, could be of bad quality for another task. While quality generally means how suitable the experience is, to learn the learning task, there are two points of special interest: errors in data and noise. "Error" means that according to a given theory about the domain, this data could not appear. For example a damaged thermometer could return temperatures that could not appear in reality. In this case the data returned by this thermometer contains errors. Noise is data contained in the experience, that is random with respect to a given domain theory (without a theory it does not make any sense to talk about randomness). In learning, this data can be misleading, and if the ratio between information and noise contained in the data is too bad, learning may even not be possible at all.

2.2 Machine Learning paradigms and algorithms

After the learning problem has been defined, an algorithm is needed to solve it. The choice depends mainly on the learning task and the available experience. Some algorithms can deal with errors and noise in the data, while others get confused by such data. So the quality of data is an important point. But also the quantity of data can be problem. Some algorithms can learn from few data, others need a huge amount of data to learn probably. Besides of this, there can be some other requirements to the learning method:

- Should the concept, learned by the agent, be understandable for a human user?

  This could be desirable in settings in which a human expert wants to check, what a program learned, before allowing the system to act on it's own.
- Should background knowledge be used in the learning process?
- How efficient has the learning algorithm to be?
- Should the learning take place while the system is operating (online)?

The number of learning methods is principally unbounded and so it is impossible to give an overview over all of these algorithms. But there are some common paradigms, which are described in the remainder of this section:

- *Artificial Neural Networks (ANN)*
  ANN consist of a number of entities called neurons, which are connected to each other. There are special neurons for input and output to this network. By adjusting weights associated to the connections in the network, the ANN can be trained to show a particular behavior on the output neurons depending on the stimulation of the input neurons. By this ANN are suitable for learning concepts from examples. They are robust against errors in the training data. A major drawback of ANN is, that the concept learned by the net consists of weights and is hardly readable for a human user.

- *Learning sets of rules*
  The approaches that can be summarized as learning sets of rules, learn logical rules from the given training data. These rules can be simple IF-THEN rules, without variables, or more complicated rules expressed in first-order logic. Approaches that learn expressions in first-order-logic are often referred to as Inductive Logic Programming (ILP). Some benefits of these approaches are that they can make use of background knowledge in a natural way, that the concepts learned by the system are readable for a human user and that (at least first-order-logic) offers a rich language for forming concepts. The major drawbacks are that some of these methods tend to be quite inefficient and may not be able to deal with errors in the training data or in the background knowledge.

- *Instance Based Learning(IBL)*
  While most learning algorithms form an explicit hypothesis describing the observed data, methods belonging to IBL just store the observed data. When they have to apply their "knowledge" to a new problem they simply search for the instance that is most similar to this problem and return the answer stored together with this instance. Some approaches use a simple representation of the data, as an Euclidean space. More sophisticated approaches represent the instances using a symbolic language. The latter approach is called Case Based Reasoning. The benefits of IBL are that it is quite simple and new experience can simply be added to the database, allowing efficient on-line learning. While "learning" is very fast in these systems, answering a query can be quite inefficient, because all the training data has to be compared to the new instance. Another drawback is, that there is no explicit hypothesis computed and the system does not generalize from the training data. Furthermore the similarity of two instances can be quite misleading in some domains.

- *Probabilistic Baysian methods*
  The basic assumption made by these approaches is, that the quantities, that should be learned are governed by a probabilistic distribution. In the learning process these distributions are adjusted according to the observed data. A special approach of this kind are Baysian Belief Networks that represent a network of statistically dependent

quantities together with the aposteriory distributions depending on their direct predecessors.

- *Genetic Algorithms*
  These approaches interpret the learning problem as an optimization problem. The task is to find the hypothesis that explains the observed data best. In analogy to evolution every hypothesis explaining the data can be seen as the genetical code of an individual. These genetical codes are then mutated and recombined over a number of generations. In every generation, only the hypotheses survive, that explain the data best. These approaches are often very efficient. A problem is that they can get stuck on a local maximum and so only providing a sub-optimal hypothesis.
- *Analytical Learning*
  The approaches, that are referred to as Analytical Learning, assume, that there exists a correct (and complete) domain theory for the domain in which the learning problem is defined. It could seem that it makes no sense to learn in such a setting, because everything that could be learned could also be deducted using the domain theory. But as these deductions can be quite complicated, it can make sense to learn from observed data. One approach, called Explanation Based Learning (EBL), explains the observed data using the domain theory. These explanations are used to find the proper generalizations that lead form the data to a general hypothesis describing the data. An illustrative example (taken from [1]): A program should learn to recognize situations in the chess game, in which a player will loose it's queen within two moves, assuming the opponent plays an optimal strategy. The domain theory in this case is the set of chess rules. The problem could be solved, by simply applying the domain theory to a given position. But this can lead to a costly computation. Another possibility would be to use an inductive learning method without any domain theory, just providing it with situations and the information, whether a given situation is in the set of situations to be recognized or not. But as the chess game is very complicated, a huge number of examples would be needed, because there are many different generalizations. EBL will on the contrary only need few examples to learn the proper concept, because the proper generalizations are provided by the explanations. So Analytical Learning can be used in many domains, where deduction is too expensive and purely inductive methods require too many training data.

2.3 Knowledge Discovery

Strongly related to the area of Machine Learning is the research in the area of Knowledge Discovery. The basic idea of Knowledge Discovery is the following: there is some data (scientific observations, economical data, etc), which itself is not understandable for a human user. The task of Knowledge Discovery is to find regularities in this data, which are understandable and useful for the user. These regularities can been seen as knowledge. In[16] Knowledge Discovery is defined as:

"The non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data"([16] p.5)

To find regularities or patterns in data, the same algorithms can be used as for Machine Learning. In this case the "experience" of the learner is the data, which should be analyzed. The learning task is to find some kind of regularities (see below). The Performance Measure is usually obtained by applying the regularities to new data (see below). An example: A bank has a data base containing the personal data of people, which have borrowed money from this bank, together with the information, whether they paid the money back in time or not. A Knowledge Discovery system can now learn to divide the people, who are applying for a credit into two classes: good and bad debtors. The "knowledge", who are those people paying their credits back, and which don't can be used to optimize credit assignment in this company.

In addition to the Machine Learning methods, which are applied in Knowledge Discovery, there are several additional methods, which were developed only for Knowledge Discovery. The problems, which are addressed by these methods, are roughly the following:

- the data contains noise and errors
- the data is too complicated (e.g. the data sets have too many attributes)
- the output of the Machine Learning Method is not understandable for a human user

These problems led to the conclusion that we should not look at Knowledge Discovery as single method, but as iterative process.

In the description of the Knowledge Discovery process, which can be found in[16], the following steps are described :

1. Select the data, which should be analyzed and create the target data.
2. Clean the data: remove noise, apply methods to handle missing data fields or data fields, which contain errors. Usually this can be done only, if there is some background knowledge, which defines what should be considered as noise/error in a given domain.
3. Find an optimal representation of the data, which includes methods as dimensionality reduction and aggregation of quantities. Again, this can mostly be done only by applying knowledge about a given domain.
4. Choose a Knowledge Discovery task, the kind of regularity or pattern, which should be found.
5. Choose an algorithm for this task.
6. Apply the algorithm.
7. Interpret the results.
8. Evaluate the result and possibly return to one of the previous steps (e.g. to use another algorithm or transformation method).

2.4 General Machine Learning /Knowledge Discovery tasks

Although learning problems depend on the setting and on the goal, which should be achieved, there are some standard learning tasks and many special learning problems can be reduced to one of them. Three of these tasks are of special importance: *classification* and *numerical prediction* and *association rules*. There are several different algorithms for each of these problems, usually corresponding to the different paradigms for Machine Learning stated above (e.g. classification can be done with ILP, Neural Networks, Baysian methods, etc.)

2.4.1 Classification

In the example above, customers of a bank were classified into two classes (good and bad debtors), according to their personal data. This kind of learning task is called classification. Mathematically, the task is to find a function f, which maps a set of parameters to a finit set of classes:

$f : A_1 \times A_2 \times ... \rightarrow \{K_1, K_2, ..., K_n\}$

A representation of such a function can be looked at as knowledge (e.g. for scientific purposes) or this function itself can be directly applied to new data (e.g. the personal data of new applicants for a credit).

2.4.2 Numerical prediction

The intention of numerical prediction is to map a set of attributes to a numerical quantity. For example: The management of a power plant could have data, how much energy is needed, depending on the time of day, etc. Then numerical prediction can be used to predict the amount of energy, which will be needed the next day, and can use this knowledge to provide the optimal amount of energy. So mathematically the task is the following:

$f : A_1 \times A_2 \times ... \rightarrow R$

2.4.3 Association rules

Association rules have the following structure:

$IF\, a_1 = v_1 \,\wedge\, a_2 = v_2 \,...\wedge\, a_m = v_m\, THEN\, a_{m+1} = v_{m+1} \wedge ...a_n = v_n$

The task is to find out all the rules, which hold in a given collection of data (in this case the data consists of data sets, which contain the attributes a1..an). These rules are used for example in supermarkets to find out, which products are often bough together. The management of a supermarket can find out, that if someone buys eggs and milk, then this customer usually buys flour too. This knowledge can be used to arrange the products in the supermarket respectively (e.g. place the flour in between eggs and milk)

3 Multi-Agent Systems and Machine Learning

The relationship between Multi-Agent Systems and Machine Learning can be seen at least from two perspectives. First a system that should perform learning can be designed as a Multi-Agent System such that multiple agents try to achieve a common learning goal. Secondly Machine Learning can be used to adapt and optimize a Multi-Agent System. These two perspectives overlap and have many topics in common, but as this work is concerned with the application of Machine Learning in Multi-Agent Systems, only the second aspect will be discussed.

3.1 Why should Machine Learning be used in Multi-Agent Systems at all?

The first reason is, that it is mostly not possible to forecast the behavior of a Multi-Agent System in every detail, when developing it. Even if the developer knows how every agent acts, it is mostly not possible to predict the behavior of the system as a whole, as the mathematical methods that exist at the moment are not sufficient for such a complex calculation. This gets even more complicated, if the Multi-Agent System is confronted with an environment that is unpredictable and changes over time. Agents, that for example trade on a digital marketplace do not know the strategy of the opponent in advance. By "learning" the opponent's strategy, the agent can optimize its behavior and can more successfully negotiate with the opponent. Another example could be an agent that is sent to a foreign planet to explore it. The scientists may not know, which environment the agents will have to face, in advance. In all these cases, in which the developers can't even foresee all possible situation the Multi-Agent System has to deal with, learning agents can modify their behavior on their own to act optimal even under new or unknown circumstances.

A second reason is that it can be sometimes easier to develop the system to optimize itself, than to implement this optimization. Especially if the structure of the system is often modified, e.g. to scale it up, the administrators of this system would have to optimize it by hand, each time something is changed.

Lastly, the image we have of autonomous agents suggests that they also should be able to modify their knowledge and behavior over time. To apply Machine Learning in Multi-Agent Systems is by this a natural implication of the Multi-Agent System paradigm itself.

3.2 Basic dimensions of Machine Learning in Multi-Agent Systems

There are many different possibilities, how Machine Learning can be applied in Multi-Agent Systems. The following list of basic distinctions structures this space of possibilities in different dimensions:

- How many learning agents are in the system? Having a single agent that learns is similar to classical learning, applied in the Multi-Agent

System environment. The obvious drawback of such a system is that's it's hardly scaleable, because the learning agent becomes a bottleneck. Having more than one agent that learns can result in problems too. They will be discussed in detail in the next section.

- If there are several learning agents, can they learn on their own or have they to cooperate with others in order to learn? It's also possible, that they can learn on their own, but learning is optimized if they cooperate with others. There are several ways, how this co-operation can look like. One possibility is simple observation of other agents. More sophisticated approaches may allow the agents to give feedback to each other or to exchange knowledge in a specific language.
- Is the learning goal to optimize the behavior of every single agent or the system as a whole? If the goal is to optimize the behavior of the individual agents, are the individual learning goals conflicting?
- How much extra interaction is required for learning?

3.3 Overview on earlier research on this subject

Many interesting papers on the subject of Multi-Agent Systems and Machine Learning can be found in the proceedings from two conferences ([4] and [3]) and in the refering chapter of the book "Multi-Agent Systems" ([2]). The following summary should only give a very brief overview of what has been done in this area.

A number of approaches has been developed on the subject of single agents learning the strategies and skills of their opponents and co-operators. There have been several approaches to enable agents to learn the strategy of their opponents in negotiations (see [5],[6]). This knowledge can be used to negotiate with them more successfully. Another important problem is to learn the opponent's skills.[9] investigates this question in the robot soccer domain. In many settings it can be important even to learn the skills of the co-operating agents in order to work together with them more efficiently (see [7]).

Much work has been spent on investigating the possibilities to share knowledge between agents. This knowledge sharing can be seen as learning from others. An important problem is, to find a common language to exchange knowledge. KQML is a speech-act based language to provide a common language for co-operation. KIF is a language to express knowledge in a general way, making it understandable even for heterogeneous agents. Together they make it possible to share knowledge among heterogeneous agents (for more detailed information on KQML and KIF see [15]). A problem arising from this approach is that the knowledge of different agents can be inconsistent. How to deal with this problem when sharing knowledge is described in [10].

A general issue that has been addressed is the problem of learning how to co-operate efficiently. [8] describes the questions arising from a setting in which a group of agents learns from group-feedback and describes a possibility to learn "real team-solution" instead of having a number of agents that only optimize their own behavior. Furthermore there are some approaches in which the agents should learn to co-operate without explicit communication (see [12]).

Lastly there has been some work on agents that learn to communicate with each other. These agents have to develop a common language and learn the meaning of new terms appearing in the communication (see [11]).

Not mentioned here are all approaches in which single agents learn to do something.

4 Multi-Agent Systems and Knowledge Discovery

4.1 The basic idea of applying Knowledge Discovery to Multi-Agent Systems

Following the definition of Knowledge Discovery (see above), the basic idea is, to find usefull patterns in the behavior of the agents in a Multi-Agent System. Agents show their behavior through their actions and communication with others. As in the given Multi-Agent System (ProPlanT) every action is a message, the task of applying Knowledge Discovery to a Multi-Agent System can be stated more precisely as the task to discover useful pattern in the communication of a group of agents, that is in the messages that are exchanged in this group of agents. So the setting for this application of Machine Learning is the following. There is one central unit (possibly an agent too), that has the following properties:

- it has access to all messages, which are exchanged in the community
- it has knowledge about the current community of agents
- it employes Machine Learning methods to extract patterns from the messages, possibly by additionally using its knowledge about the community
- it interacts with the user (the administrator or developer of the Multi-Agent System) to provide useful information to him/her

From the point of view of the possible aspects of learning in Multi-Agent Systems, it could be said that there is a single learning agent in the system, that helps to optimize the performance of the whole system, and the only extra communication is to provide this learning agent with a copy of every message. But for two reasons the term "Knowledge Discovery in Multi-Agent Systems" fits better than the term "Learning in Multi-Agent Systems". Firstly the emphasis should be more on the passive analysis of the system than on autonomous adaption by the agents itself. And secondly much of the work on the area of Knowledge Discovery can be used for the given approach. Mainly the Knowledge Discovery process (see above) contains many points that are important for the given setting, as filtering and transformation of the data, interpretation of the results and interaction with the user.

To analyze the benefits and limitations of the applying Knowledge Discovery in Multi-Agent Systems, this approach will be compared with two other approaches: 1.Distributed learning and 2.Mathmatical analysis of Multi-Agent Systems.

4.2 Knowledge Discovery in Multi-Agent Systems vs. Distributed Learning and Adaption in Multi-Agent Systems

There are several advantages that Knowledge Discovery has, compared to distributed learning.

One problem of having several agents that learn is that they usually have only a limited view of the system as a whole. In these cases the agents can optimize their own behavior but this may not result in an optimization of the system. This problem can be addressed by giving these learning agents information about the whole system. But this would not only result in a huge amount of extra communication, but also make the computation within the individual agents much more complicated. This problem does not occure with centralized learning, because the central learner has automatically knowledge about the performance of the whole system. A second problem of distributed learning is, that some of the actions, which result from learning, are not in the scope of the learning agent. For example an agent, that learns to predict the lack of a needed resource, then he is normally not able to do something about this problem on its own, in case that this resource is something in real world. The only possibility is to contanct a human user. But the problem with this is, that the user would get requests from several agents for the same resource and that these requests could be conflicting, which would make it hard for the user to decide what to do. One possible solution would be to let the agents negotiate, what to request from the user. But this would again cost extra computation within the system. Using Knowledge Discovery this problem can be solved much easier. As there is only one learning agent, it can inform the user directly and suggest actions to him. Furthermore it is able to justify this suggestion, because all the reasoning is centralized. This is desirable, because often the action by the human user can be quite costly, e.g. providing a new resource and if the system justifies its suggestion it is much easier for the user to decide whether to follow it or not. If learning and reasoning is not centralized, it will be often impossible to give such justification (e.g. if reasoning is done by negotiation).

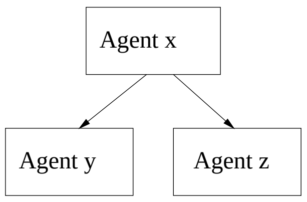
But there are also some limitations and drawbacks of Knowledge Discovery compared with distributed learning. Mostly these drawbacks result from the centralized character of Knowledge Discovery (as it is suggested here). One problem is, that the amount of data which is processed by one unit can be very high. This makes it hard to scale the system, because the central learner acts as a bottleneck. Furthermore the computation in the central learning unit can be very complicated, as it has to deal with data and knowledge of the whole system. In contrast to this, using distributed learning preserves all benefits of the Multi-Agent System paradigm. The individual agents have only to deal with a limited amount of data and knowledge, which make the computation they have to perform much easier. The learning system is modular and easily scalable.

As a conclusion it can be said, that both approaches have some benefits and drawbacks. It depends mostly on the application, which one to use, because none of both approaches can completely replace the other.

4.3 Knowledge Discovery in Multi-Agent Systems vs. mathematical analysis of Multi-Agent Systems

If a centralized unit is used to analyze the system in order to increase its performance, the second question is why to use Machine Learning and not classical mathematical analysis or pure logical reasoning.

Let's consider a simple example. Assume that there is one agent 'x', that sends requests for tasks to agents 'y' and 'z'.



Every task costs the same time in execution and agent 'x', waits until it's finished until it sends a new request. Now it is assumed further, that the algorithm agent 'x' uses to decide whom to send a request is very complicated and takes into account input data, like sensor data. The problem is, that the agents 'y' and 'z' have to be replaced once in a while, which takes the same time as the execution of one request. Unfortunately it is impossible to tell agent 'x' not to send any requests, while an agent is being replaced, so the only possibility is to predict, when an agent won't get a request and to exchange it exactly then. Now this can be done (at least) in two ways. The first possibility would be to analyze the algorithm, agent 'x' uses to distribute the tasks. This could lead to a complicated computation and may be it could be necessary to predict the sensor data as well. The second possibility is to take a look at the sequence of requests. This sequence could be for example this one: 'yzzzyzzz'. A close look shows, that there is always a request to agent 'y' followed by three requests to agent 'z'. This knowledge can be used to decide, when to replace which agent, if we assume, that this pattern will continue in the same way in the future: agent 'z' should be replaced at every (4*n)th request, agent 'y' any time else. The difference between the first and the second approach is, that in the first case the question is, why a certain behavior occurs, whereas in the second case only the fact, which behavior occurs is important. Knowledge discovery does not discover the reason, why the system behaves in a certain way, but only searches for regularities in this behavior.

Now it's possible to compare the mathematical analysis with the Knowledge Discovery approach.

Firstly a drawback of mathematical analysis is, that because of the complexity of most systems it is often not applicable. Especially the non-linear, discrete nature of the domain makes it hard to apply classical mathematical methods. This gets even more complicated if the agents interact with some environment. Then there is the need to analyze this environment mathematically too, which presumes that there is a sufficient theory about this environment, which is often not the case. The Knowledge Discovery approach has none of these problems. Firstly there are already very efficient and well-studied algorithms to find patterns in data, such that it is not necessary to develop new analysis methods. Secondly there is no need to have a theory about the environment, because if a certain pattern occurs, it is only important that it occurs and not why.

However there are some problems with this approach too. Firstly it is not suitable to proof anything. In some cases it could be important to proof, that a given system behaves, as it is expected. Because the Knowledge Discovery approach only observes the behavior and does not ask for the cause of this behavior, it is not possible to proof anything about the system's behavior in principle. A second problem is, that there is no garantue, that there are any regularities in the behavior at all, or that these regularities can be found by the applyed learning methods. In other words, there is no garantue that this approach works at all.

Summarizing it can be said, that as long as mathematical analysis is reasonably applicable, it is superior to Knowledge Discovery, because it can prove a certain behavior and is more precise. However there are many cases, in which mathematical analysis is not applicable, because of the complexity of the system, or because there is no sufficient theory about the Multi-Agent System or its environment. In these cases Knowledge Discovery is still applicable in principle, but the output of this procedure should only be seen as heuristic and there is no garantue that it works at all.

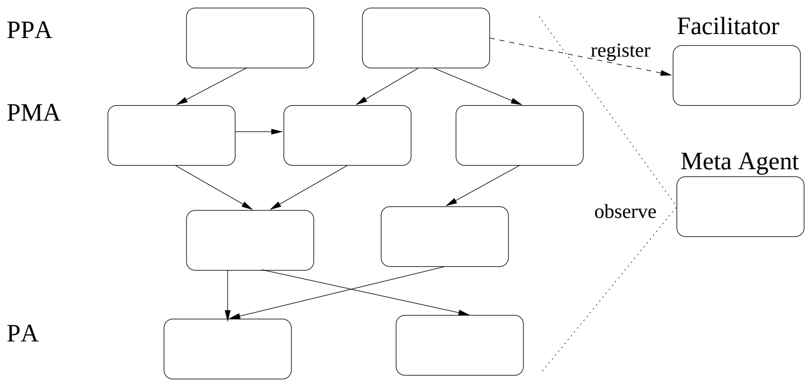
Additionally it should be said that both methods can be combined. On possibility is to use mathematical analysis as far as it is possible and then complete it by Knowledge Discovery Methods. An very simple example for such an approach can be found in the third part of this documentation. Another possibility is, to use Knowledge Discovery to "guide" mathematical analysis. Often it is much easier to prove something (e.g. some property of a system), if it is known what should be proved.

5 ProPlanT

ProPlanT is a Multi-Agent System for distributed planning. It differs from other systems for distributed planning in two major points. Firstly it uses a special knowledge representation and maintenance mechanism to provide the individual agents with information needed for efficient planning and secondly ProPlanT includes a special meta-agent that observes the community from a global viewpoint and can have influence on some aspects of the behavior of the system. (For more information on ProPlanT see [13])

5.1 Types and structure of agents

ProPlanT can contain several Production-Agents (PA), Production Management Agents (PMA) and Production Planning Agents (PPA). Furthermore it contains one Meta Agent (MA) and a Facilitator. The following figure shows, how these agents are structured.



The PPA agents are encapsulated programs, which determine the tasks to be planned, together with some constrains concerning these tasks. The PMA agents decompose the task to be planned and distribute the resulting sub-tasks further to other agents. The PA agents represent an entity in real world, for example a production unit. These agents consist mostly of encapsulated programs for scheduling, databases, etc. The MA observes the community (for more information see below). The Facilitator keeps an index of all agents in the community.

5.2 Planning in ProPlanT

If a PPA requests planning a task, it first sends this request, together with constrains concerning the execution cost and the execution time to a PMA, which is able to deal with this task. The PMA uses its knowledge to decompose this task and to distribute the resulting subtasks to PA or to other PMA. If he gets a positive reply from all contacted agents he returns itself a positive reply. If the planning of one of the subtasks fails, he tries to find other co-operators, or another decomposition of the task. Only if none of these attemps leads to a success, he himself returns a negative reply. This task decomposition can go through several levels. If a PA agent gets the request for a task, it does not delegate it further to other agents. It checks whether it is able to perform this task within the given constrains or not. According to this, it sends a positive or negative reply. In the first case, it also fixes the plan for the task, e.g. writes the task to the schedule of tasks to be performed on a production unit.

5.3 Knowledge representation and maintenance

The PMA contain a special structure for representing knowledge needed in planning. This structure is called tri-base. It consists of three different knowledge bases:

- *Co-operator base*: stores static information about other agents, as their IP-addresses.
- *Task-base*: stores information how to decompose tasks and keeps a list of prepared plans. A task is decomposed to one or more sub-tasks. Additionally there can be constrains, stating, that one of the

subtasks can be executed only after the execution of another sub-task has been finished (precedence rules).

The prepared plans indicate to which agents the subtasks should be delegated optimally.

- *State-base*: This knowledge-base stores information about the current load, operation-cost, etc. of other agents. This knowledge base is changed very often.

To keep the state-base up-to-date, ProPlanT uses a simple subscribe/ advertise mechanism. If an agent is interested in the information about another agent, it subscribes at this agent. It will then get in regular intervals, up-to-date information about this agent. The idea of using state-bases lies in the following problem. If an agent wants to decompose a task to delegate it further it has to find co-operators, which perform these subtasks. One common mechanism for this is the contract-net protocol. The agent sends a request to all agents that are able to perform a given task, then gets a number of bids (e.g. containing the cost of this task) and then chooses the optimal co-operator. The problem with this approach is that it can be too slow in some settings. For example in military settings the plan has to be fixed as quickly as possible. The knowledge representation in ProPlanT addresses this problem. The PMA do not have to contact the other agents to get bids for a task to be planned, but can simply use the information in their knowledge-base to find the best co-operator possible. This leads to a substantial saving of time while planning.

5.4 The Meta-Agent

One common problem in Multi-Agent Systems is that the individual agents are not aware of the whole system they are working in. In some cases the limited view of these agents can lead to problems, e.g. sub-optimal plans. The idea to deal with this problem is to create a global unit, which observes the whole community and can have an influence on it on a global level. The Meta-Agent in ProPlanT is such a unit. Its function can be divided in a passive and an active role. In the passive role the Meta-Agent works as a visualization and analyses tool, which provides the user with information on some aspects of the behavior of the system. In its active role the Meta-Agent can have influence on the community, e.g. by up-dating the knowledge bases of individual agents (not implemented yet).

5.5 Messages in ProPlanT

As learning in ProPlanT will mainly depend on the messages exchanged among agents, the remainder of this section contains an index of messages that appear in ProPlanT.

In general a ProPlanT KQML message contains the following entries:

1. perform: the action that should be performed
2. sender: the agent that has sent this message
3. receiver: the agent that receives this message

4. content: the content of the message, which depends on the performative
5. reply-with: an identifier, which the receiver should use in its reply to this message
6. in-reply-to: an identifier that states, to which message, this is the reply
7. language: the language in which the content is encoded

5.5.1 Administrative Messages

Register	Sent by agents to the Facilitator to register themselves
Kill	Sent to the agents by the Facilitator to terminate them

5.5.2 Knowledge-base Messages

Subscribe	With this message an agents subscribes himself at another agent for a number of tasks. The content specifies these tasks.
Advertise	This kind of message is used to send up-dates of the knowledge base to all subscribed agents. The content consists of a list of tasks together with the relevant information for each task, which is the execution start time, end time and cost for this tasks.

5.5.3 Planning Messages

Achieve	Requests the planning of a task. The content entry contains the name of the task to be planned, an intervall of time units to indicate in which period of time the the task has to be executed, a constraint stating the maximal cost of the plan and an option to choose whether to otimize for execution time or cost.
Evaluate	The same as Achieve but requests only information, whether it is possible to plan a given task and what the operation time and cost would be.
Reply	Respond to an Achieve or Evaluate message, containing the actual operation cost and time (start-time, end-time) for the requested task.
Sorry	Respond to an Achieve or Evaluate message in case that it was not possible to plan the requested task.
Unachieve	Request to delete the plan for a given task, which was requested and successfully planned. The content entry contains the name of this task.

6 Measuring the performance of a Multi-Agent System for planning

Following the definition of Machine Learning given in section 2.1, a measure for the performance of a Multi-Agent System has to be found, in order to decide whether its performance improved by learning. Although this measure strongly depends on the special learning problem, it is possible to give some general measures, describing the performance of the system as a whole. As ProPlanT is a distributed planning system, there are at least two different classes of performance measures: one concerning the quality of the resulting plan, another concerning the process of planning itself.

6.1 quality of the plan

The two main measures, which describe the quality of a plan, are the total cost of the suggested solution and the time needed to achieve the desired goal. Beside from these two essential quantities, there could be other interesting aspects. As the plans generated by the system are mainly used in real world, it is desirable to have robust plans, which generally means that a small error in the real system does not make the whole plan worthless. This could for example happen, if the plan assigns the whole work only to one machine, and this machine breaks. One way to achieve this is to explicitly model the reliability of resources and use this information while planning. A general way to make a plan more robust is to assign the work well balanced to the available resources. If one machine breaks down it could be easier to replace it by another one without changing the plan because it only is only in charge of a small portion of the whole plan. So robustness and well-balances utilization of resources are important points in the quality of the resulting plan, too. Another important quantity, which is strongly connected to the time needed to achieve the goal, is the total idle-time of a resource. The idle time is the time an entity is waiting for another entity in order to perform a task. Obviously this time should be reduced as much as possible. The criteria to measure the quality mentioned in this section are not independent in general. For example a reduction of the idle-time of agents will mostly lead to plans, which require less time in execution. On the other hand, a plan, which reduces the cost, could take longer in execution as a plan that is more expensive.

6.2 the planning process

As the user usually wants to get his plan as soon as possible, the most natural measure for the planning process is the respond time. The respond time is defined as the period of time between the user's request to the system and the respond from the system, either containing the plan, or a notice that the desired goal can not be achieved. Another aspect in measuring the planning process is the number of messages exchanged while planning. Mostly if communication is not reliable or expensive, this could be a point, which is equally important as the respond time. For example if mobile computers are used we would wish to reduce the time we have to be online as much as possible, even if the respond time increases by this.

While we would wish to optimize both, the planning process and the resulting plan, their goals will be mostly conflicting. If a simple algorithm is used in planning, the respond time will be shorter, but the resulting plan may not be optimal. On the other hand, using an elaborate algorithm for planning could lead to an optimal plan, but time needed to find this plan could be significantly higher. Similarly, a reduction of the number of messages exchanged, could have the result, that the information in the knowledge bases of the agents is not up-to-date, which could lead to worse plans. This section showed that there are many different aspects of performance in ProPlanT, which is not only important for the evaluation of the Machine Learning methods, which are applied, but should be already considered, while investigating the different possibilities Machine Learning in ProPlanT.

7 Possible applications of Knowledge Discovery in ProPlanT

As the approach of applying Knowledge Discovery in ProPlanT is still very abstract, it isn't possible to give a complete list of possible applications. But still there is an important difference between these applications: they can either be domain specific or independent. This distinction arises from the question, where the regularities are situated, which have to be presumed when applying Knowledge Discovery. This means which part of the system has to show a certain kind of regularity in its behavior in order that a certain application of Knowledge Discovery will work.

To discover this distinction it is important to keep in mind, that there is on the one hand a computational system, the Multi-Agent Systems, which consists of a number of agents (PA, PPA, PMA). On the other hand, at least PA normally represent some unit in the real world, as a machine for example. To represent this real world unit, the PA models some important aspects of it, in the case of a machine for example how long the execution of a certain tasks takes on this machine. If for some reason, one of the relevant quantities of the real world unit changes, then this change has to be reflected in the computational system as well. If a machine has an error, which causes that it is no longer able to perform a certain task, then the PA, which represents this machine, has to be changed respectively. Otherwise all future plans may be worthless (because they contain tasks for this machine, which it can not achieve).

This means, that there are two general possibilities, how a ProPlanT system can be influenced:

1. *internally*, which means that agents have influence on other agents. If an agent requests a task from a PA, then it will influence the schedule of this agent.
2. *externally*, this is the effect, which has been described above. Additional to changes in real world unit, which are represented by PA, the system is clearly influenced by the requests, which are sent to the system by the user. This sequence of requests can be governed by a probability distribution or by other regularities (as: after a is requested, always b is requested).

This distinction is important because the possible ways how ProPlanT is influenced show also the two possible areas in which regularities can occur: the computational system itself and real world. To learn to predict the next requests to the system, in order to optimize pre-planning, it would be important to know something about regularities in the sequence of requests sent to the system. As this sequence arises in real world, the regularities have to be presumed in real world as well. The important consequence of this is, that there is no possibility to presume these regularities for every possible ProPlanT community, but only for a given ProPlanT community, that is specialized to a certain domain. Regularities, which arise from internal behavior in the computational system of agents only, are domain independent, because they do not presume any regularities in real world.

Domain specific and independent regularities can overlap too, but it is important to think of which regularities a given application of Knowledge Discovery presumes, and whether these regularities can be presumed. If the regularities arise from the internal behavior, then this question can be answered, by only looking on the general ProPlanT system, without special domain. If the regularities are (partly) presumed in real world, then a domain has to be given, and the question can only be answered by looking at the domain.

The second part of this documentation introduces a framework of tools, which supports generally all kinds of application of Knowledge Discovery in ProPlanT, independently from the kind of regularities presumed. The third part presents an application of Knowledge Discovery in ProPlanT, which is domain independent.

Part II

A Framework of tools to support Knowledge Discovery and Machine Learning in ProPlanT

8 Overview of the tools

8.1 Which tools are needed?

Although every Knowledge Discovery problem has special requirements, it is possible to develop a set of general-purpose tools, which support all of these applications in ProPlanT. In general these tools should be easily modifiable and extendable, to allow the developer to perform different kinds of experiments with the system, as different learning methods, transformations rules, use of background knowledge, etc. Which tools are needed can be determined using the concept of the Knowledge Discovery process, which was presented in 2.3. Generally the following tasks have to be supported:

- *Generating data*
  To perform Knowledge Discovery there has to be data. For the application of Knowledge Discovery in Multi-Agent Systems this means,

that there has to be some communication or action of agents. In ProPlanT, as a distributed planning system, communication between the agents is initiated by a planning request from a PPA. So to generate data automatically there has to be a tool that generates these requests and sends them to the community of ProPlanT agents.

- *Transformation and Filtering*
  Before the Machine Learning methods are applied to the data, the data has to be transformed and generally additionally filtered. How this transformation and filtering looks like depends on the given Knowledge Discovery problem.
- *The machine learning methods*
  After the data has been filtered and transformed the Machine Learning methods themselves have to be applied. Often the learning task can be reduced to one of the standart learning tasks described in 2.4.1
- *Interpretation of the results and user interaction*
  How the interaction with the user and interpretation of the learning results look like, again depends strongly on the given Knowledge Discovery problem and it is not possible to give a general structure of this part of the Knowledge Discovery process.
- *Performance Measure*
  Lastly, to be able to evaluate the results there has to be some kind of performance measure. This performance measure can either measure general the performance of ProPlanT as suggested in 6. Or it can be specific for the given Knowledge Discovery problem.

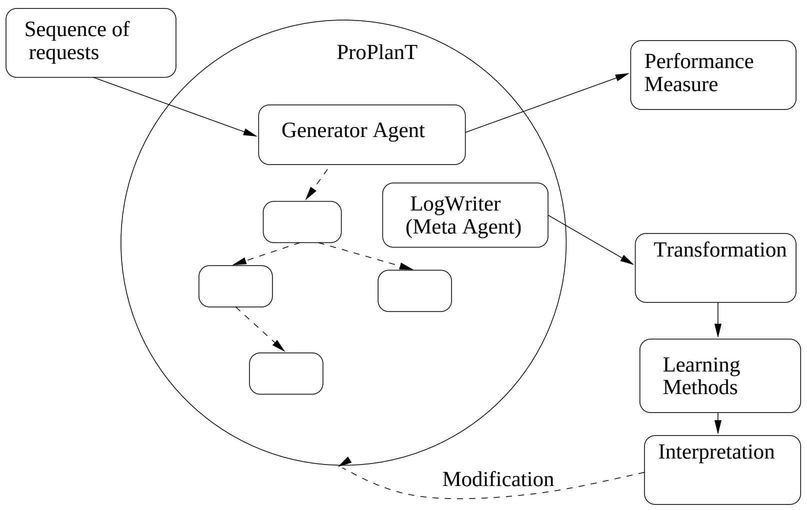
8.2 Tools, which were developed or used to support Knowledge Discovery in ProPlanT

8.2.1 Overview of the actual developed or applied tools

Based on this analyses a number of tools have been developed. The major problem of developing general purpose tools, that support all kinds of Knowledge Discovery, is that some parts of the Knowledge Discovery process are completely problem specific (as the transformation). The basic idea to deal with this problem is, to give the user the possibility to state, e.g. the rules for transformation, in a programming language. As programming language, Prolog was chosen. Roughly this framework makes it possible to save all messages (the data for Knowledge Discovery) in a format, that can be read by a Prolog system. The filtering and transformation is then performed in Prolog and after this, Prolog calls the Machine Learning methods from a library. After learning a second Prolog program helps the user to interpret the results of learning. The benefits of this solution are that the rules for transformation and filtering can be stated and modified intuitively (as Horn-clauses), that it is very easy to use background-knowledge and that an intelligent user-interface can be developed (e.g. an Expert System). Following the analysis arising from the Knowledge Discovery process and the idea to use Prolog for pre- and post-processing of the data, the following tools are needed to support Knowledge Discovery in ProPlanT:

- *The Generator Agent*
  The Generator Agent is designed to automatically perform test runs on a given ProPlanT community, such that it "simulates" the behavior of a user or costumer requesting tasks from the system. Roughly it is able to generate a sequence of tasks, send them as achieve messages to the concerned PMA and to write the results, contained within the reply or sorry message, to a file (together with the original task and some statistical data).
- *Sequence Definition*
  If the sequence of tasks, which is send to the community, should have some special structure, then this tool can be used to define this structure and to generate a sequence, which contains it. The generated sequence is written to a file, which can be read by the Generator Agent.
- *The LogWriter Agent*
  This agent gets a copy of every message, which is exchanged in a given ProPlanT community. It then converts these messages and writes them to a file, as Prolog facts. There are several options, which influence the exact format of these facts.
- *A Knowledge Discovery/Machine Learning library*
  For Machine Learning and Knowledge Discovery itself, a library called Weka is used, which is freely available and implements algorithms for classification and association rules, as well as filtering algorithms and meta-learning schemas, as attribute selection. To make it possible to call these Java methods directly from Prolog, a simple bridge from Prolog to Weka has been developed, making it possible to use these methods, without knowing anything about Java.
- *Performance Measure*
  For the performance measure a simple statistical tool is used, as for example the data analysis package in Excel.

8.2.2 How do these tools work together?



First the data, which is used in Knowledge Discovery has to be generated. So first the user has to apply the tool for sequence definition and generation to generate a sequence of requests. Then the Generator Agent is used to send these requests to the community. The LogWriter Agents writes all messages, which are exchanged to a file (in the Prolog output format). A Prolog program reads this file, filters the messages and transforms them according to the transformation rules stated by the user. Then Prolog invokes Machine Learning and presents the results to the user. With this knowledge the user can make modifications to the systems and perform a number of test-runs with this modified system as well as with the original system and write statistical data for both to a file. The performance measure can now be used to compare the performance of the original and the modified system. This can be used as feedback on the learning methods or transformation rules, or as evaluation for the final system.

9 The Generator Agent

9.1 Introduction

For a number of tasks, including Machine Learning, it is necessary to perform a huge number of test-runs with the ProPlanT system. Doing this by hand would cost a lot of effort, not only by starting the tasks and keeping track of the results, but often also by generating the desired sequence of achieve messages. The Generator Agent has been developed to perform this task automatically. Roughly it is able to generate a sequence of tasks, send them as achieve messages to the concerned PMA and to write the results, contained within the reply or sorry message, to a file (together with the original task and some statistical data). This file can be used for statistical analyses or if exactly the same sequence should be used once again (see below). The sequence of requested tasks can be defined in many different ways. Easiest is to tell the Generator Agent to read it simply from file. Another possibility is to use randomly generated sequences with specified distributions. The Generator Agent is based on the empty agent (see [14]) and is developed in Power++, a C++-dialect (see [25]).

9.2 The user interface

When the agent is started, the main window of the Generator Agent appears:

The main part of the interface is the list in which all requests are stored, together with the result for this query and some statistical data. One line has the following format:

<task>;<maxcost>;<starttime>;<maxtime>;<optimize>; <success|non-success>; <actual_cost>; <actual_starttime>;<actual endtime>;<answertime in ms>

<task> denotes the name of the task. <starttime> and <maxtime> are time units (natural numbers), denoting the time constraints, which were used in the request. <maxcost> denotes the maximal cost of the

plan in cost units (natural numbers) <optimize> denotes, whether in the requested plan should be optimized considering execution time or cost. A value of 0 means optimize for time, a value 1 means optimize for cost. <actual_cost>, <actual_starttime> and <actual_endtime> denote the actual parameters of the plan, as contained in the reply message, sent by the PMA. Again these values are stated in time units or cost units. In case of a non success, the fields with the actual values are left blank.

The buttons below have the following functionality:

Start	The Agent starts sending queries to the community, until the Stop button is pressed or the agent itself is stopped
Stop	The Agent stops sending queries to the community
Step	The Agent sends exactly one query to the community (and then stops)
Options	Opens the option dialog. At the moment the only option, that can be set, is the PMA to which the queries are sent.
WriteLog	Writes the content of the log list to a specified file.

9.3 The sequence of requests

As already stated in the introduction, there are two possibilities to state, how the sequence of requests (achieve messages), which is sent to the community by the Generator Agent, should look like: 1.Read this sequence from a file or 2.Generate it randomly. In the remainder of this section, these possibilities are presented in more detail.

9.3.1 Reading the sequence of requests from a file

This is the easier possibility of both, because it simply reads the sequence of requests from a file, which has to be provided by the user. Every line in this file has to contain one request.These lines must have the following format, where the domains of the individual parameters are the same as in the interface:

<task>;<maxcost>;<starttime>;<maxtime>;<optimize>

Any token that follows the last entry is ignored. If the Generator Agent reaches the end of the file, he uses the first entry of the file as request again and so on. Principally there are at least three possibilities to construct this file of requests:

1. write it by hand
2. use the file, which can be written with “WriteLog”, as the first five entries of each line have the same structure and the remaining tokens are ignored (see above)
3. generate this file using an external utility. This can be useful in cases, in which the sequence should have a more complex structure, than only random values at given distributions. An example for such a utility can be found in the next section.

9.3.2 Generating the sequence of requests randomly

The second possibility to construct the sequence of requests, which is sent to the community, is to generate it randomly. The basic idea is, that every quantity that is contained in a request (task, start time, maximal time, maximal cost and optimize), is governed by a distribution. By stating these distributions, the user can define the behavior of the Generator.

The first quantity contained in a request is the task to be planned. For this task the following probability distribution is available:

- *read from fil*e: the distribution is read from a file. Every line in this file represents one possible value, that is one possible task, that can be requested, together with the probability of this task, which should be used in generation. The lines in this file have the following format:

  name_of_task;probability
  In case that the sum of the probabilities is smaller than one, then the last entry has a probability, which fills up this 'gap' and differs from the stated one. If the sum of probabilities is higher than one, then the last entry or even the last entries have a smaller probability than the one, which has been stated. (E.g. if there are the following entries: ('a',0.6),('b', 0.3), ('c', 0.4), ('d',0.8) then 'c' will have the probabilities 0.1 and 'd' will have the probability 0.0).

The remaining quantities range all over the set of natural numbers. For them, the following probability distributions are available:

- *constant*: this quantity has always the same defined value
- *uniform*: this quantity is governed by a uniform distribution. The range has to be defined by the user
- *read from file*: reads the probability distribution from a file. Every line contains a value (a natural number) together with the probability that this value appears. The lines have the following format:

  value;probability
  If the probabilties don't sum up to one, the effect is the same as for the distribution of the tasks.

9.4 Practical Usage of the Generator Agent

As the Generator Agent is re-implemented at the moment, beacuse it is written in an out-dated C++-dialect, the documentation, which describes how to use the Generator Agent can be found together with the most up-to-data version of the implementation, rather than in this documentation. However the basic concepts, which were presented in this section will remain the same even in future version.

10 Generation of more complex sequences

"sequences.lisp" is a small utility, written in the LISP language, which supports the generation of sequences, which are random, but contain regularities. These sequences can be written to a file, which can be read by the Generator Agent (see above). In general a sequence looks like this:

$a_1, a_2, ..., a_n$

Now the user can state rules of the form:

$IF\, a_i =$ 'a' $AND\, a_{i+1} =$ 'b' $THEN\, a_{i+2} =$ 'c'

In each step, the program checks, whether the condition of such a rule holds. If this is the case, then the next element in the sequence has simply the value defined in the right part of the rule. If no rule holds, then the next element in the sequence is generated randomly. If the conditions of more than one rule hold, then only the first of these is applied.

The program was developed under the Allegro Common Lisp environment (for more information on Allegro see [21]). For detailed information how to use this utility refer to file "sequences.lisp".

11 The LogWriter Agent

11.1 Introduction

In order to perform Knowledge Discovery on a collection of data, it is often necessary to transform this data, before applying Machine Learning methods to it. One reason for this is, that the original data is often too complex to be handled by the learning algorithm (for example contains too many attributes) or contains too much noise or errors. Another reason is, that most Machine Learning algorithms aren't able to use knowledge about the domain from which the data is obtained. So the process of transformation gives the user the opportunity to combine the facts, contained in the collection of data, with his background-knowledge concerning the domain, which can make the data to be analyzed more meaningful, even before the actual Machine Learning starts. How this transformation is actually performed depends strongly on what should be learned from the data, how the data does look like and which Machine Learning method is used. To reflect this problem, it is supposed, that the transformation is performed in the Prolog Language. Firstly this makes it possible to state the rules for transformation quite intuitively and makes them easy to modify, secondly this makes it possible to use knowledge about the current community in the transformation and thirdly Prolog makes it easy to develop a system which can co-operate with a human user in an intelligent way (for example an Expert System).

The ProPlanT System already offers the possibility to send a copy of every message to the LogWriter Agent. The task of the LogWriter Agent is simply to convert these messages to something, which can be read by a Prolog System and can be processed by a user defined program written in the Prolog language.

The LogWriter Agent uses the empty agent as basis (see [14]) and is implemented in Power++ (see [25])

The remainder of this section describes the conversion in detail.

11.2 The conversion of KQML Messages to Prolog facts

The basic idea of this conversion is, that every message is coverted to one or more facts in the Prolog language. The following is an example for such a fact (for a detailed description see below):

```
message(12,val('achieve'), val('PMA'), val('PA1'),
 query(val('a'), undef, val(10), val(30), undef),
 120, undef).
```

This means that at time 12 (the concept of time is discussed later), there was an achieve request from agent "PMA" to agent "PA1", requesting the planning of task "a" with an undefined start-time and end-time of 10 and a maximal cost of 30. The optimize parameter is left undefined. In-Reply-to is undefined too, Reply-with has the values 120.

In general the structure of the message fact is the same as the structure of a KQML message in ProPlanT (see 5.5) with the additional time-stamp. The conversion of the individual messages differs mainly in the structure of the content-entry and is described below.

11.2.1 Options for the conversion

There are three different options which influence, how the conversion from KQML messages to Prolog facts is performed:

- *time*
  the user can choose from three different concepts of time:
  - *real time*
    this is the time at which the message reached the LogWriter agent. This time is measured in msec since the start of the agent. Note that this is neither the time, at which this message was sent nor the time at which it arrived at its receiver. There is no guarantee that the logical order is preserved (e.g. that a reply message has a higher time stamp than the matching achieve message)
  - *enumeration*
    the messages are simply enumerated in the order in which they arrive at the LogWriter Agent. Again, there is no guarantee that the logical order is preserved.
  - *runs or requests*
    this kind of handling time works only in a specific setting. It is assumed that there is only one PPA in a given community and that this PPA always waits for the reply concerning a request, before he sends a new one. Every message can then be associated with the request, in which it appears. The LogWriter simply increases a counter by one, every time he gets a reply or sorry message.

- *simple or complex handling of missing data*
  As it's possible that a KQML-Message does not contain an entry for every field (because it's left out, undefined in a special case or due to an error) it's necessary to handle missing data. The LogWriter offers two ways to deal with this problem. The first one is, just to assign a special value to these quantities (as e.g. -1 for Integers and "_" for Strings). This makes the resulting messages simpler, but can lead to ambiguous entries. The more complicated variant treats these undefined values explicitly. If a value is undefined in the KQML message then the entry in the message fact is "undef". If it is defined then the value is encapsulated in special structure: "val(value)". This make it possible to distinguish clearly between defined and undefined value, but leads to a more complex processing.
- *ISO-Prolog or Visual Prolog*
  Visual Prolog is a very efficient and popular Prolog compiler. However it differs from the ISO-Prolog standart in some points. To support both Prolog implementations, this option gives the user the possibility to choose among them.
  (In fact the only two relevant differences for the conversion are, that Visual Prolog uses double-quotes instead of single-quoutes and that there is no dot at the end of a fact, because it is treated as an internal fact-base-file)

11.2.2 Handling of messages that contain multiple entries

Some messages contain multiple entries in the content field. For example an advertise message contains usually information on more than one task. Having a varying lenght and structure of messages would make the further processing in Prolog more complex and confusing. For this reason, the LogWriter splits these messages up into several message facts, each containing exactly one entry.

11.2.3 Conversion of the individual messages

Achieve

message(TIME,achieve,SENDER,RECEIVER,query(TASK,START-TIME,END-TIME,OPERATION-COST,OPTIMISE),REPLY-WITH,IN-REPLY-TO)

Reply
message(TIME,"reply",SENDER,RECEIVER, answer(TASK,START-TIME,END-TIME,OPERATION-COST,OPTIMISE),REPLY-WITH,IN-REPLY-TO)

Sorry
message(TIME,"sorry",SENDER,RECEIVER, query(TASK,START-TIME,END-TIME,OPERATION-COST,OPTIMISE),REPLY-WITH,IN-REPLY-TO)

Subscribe
message(TIME,"subscribe",SENDER,RECEIVER,task(TASK),REPLY-WITH,IN-REPLY-TO)

Advertise
message(TIME,"advertise",SENDER,RECEIVER,info(TASK,START-TIME,END-TIME,OPERATION-COST),REPLY-WITH,IN-REPLY-TO)

Unachieve
message(TIME,"unachieve",SENDER,RECEIVER,query(TASK,START-TIME,END-TIME,OPERATION-COST,OPTIMISE),REPLY-WITH,IN-REPLY-TO)

Evaluate
message(TIME,"evaluate",SENDER,RECEIVER,query(TASK,START-TIME,END-TIME,OPERATION-COST,OPTIMISE),REPLY-WITH,IN-REPLY-TO)

Register
message(TIME,"register",SENDER,RECEIVER,agent(AGENTNAME),REPLY-WITH,IN-REPLY-TO)

Others
Message(TIME,PERFORM,SENDER,RECEIVER,empty,REPLY-WITH,IN-REPLY-TO)

11.3 Practical Usage of LogWriter

As in the case of the Generator Agent, the LogWriter Agent is re-implemented at the moment, such that a final documentation on its usuage can not be given here. For this information refer to the most up-to-date version of the LogWriter.

11.4 Some examples

Although the transformation rules depend on the actual Knowledge Discovery problem, some example should show the possibilities of using the Prolog language to transform ProPlanT messages. Here it is assumed that Visual Prolog is used, but the code could be ported to any other Prolog very easily (part III contains an example in ISO-prolog)

11.4.1 Example 1: Filtering

The predicate *filter* deletes all messaged that have no sender and all messages with the KQML-perfomative "subscribe":

```
filter :-
retractall(message(_,val("subscribe"),_,_,_,_,_)),
retractall(message(_,_,undef,_,_,_,_)).
```

11.4.2 Example 2: Simple transformation

The simple predicate *advertise/3* just extracts some information from the original message:

```
advertise(T,X,C) :-
message(T,val("advertise"),_,_,info(val(X),_,_,C),_,_).
```

The predicate *no_ answer/2* combines the information from several message facts to find achieve messages which have not been answered. To "match" two messages together, the entries for "reply-with" and "in-reply-to" are used, which are assumed to be unambiguous.

```
no_answer(T,A) :-
message(T,val("achieve"),A,B,query(X,_,_,_,_),RW,_),
not(message(_,val("reply"),B,A,answer(X,_,_,_),_,RW)),
not(message(_,val("sorry"),B,A,query(X,_,_,_,_),_,RW)).
```

12 Performance Measure

The Performance Measure is used to compare a system, after it has been modified as a result of Knowledge Discovery, with the original system. This is important in order to evaluate learning goals and learning algorithms. How this performance measure looks like, depends on the Knowledge Discovery goal, that should be achieved. But since the general purpose of applying Machine Learning in the ProPlanT System is to improve performance, it is possible to give a general performance measure. Easiest is, just to compare by the means of some quantities describing aspects of the performance of the system. These quantities could be for example actual cost of a plan, actual time needed for a plan or respond time of the system. All of these quantities can be obtained using the Generator Agent with some random sequences. The only problem is, that as the sequences are random, the outcome is also random to some extend. So to be precise, the means of the quality of interest have to be compared with some method from statistical decision theory instead of comparing them directly. One very common method to compare the means of two sets of random samples, assuming they are distributed with a normal distribution is the unpaired t-test (Students' test). It is described for example in [17] (p. 1125). As this test is already implemented in nearly every statistical package, it has not been implemented again for this system.

13 The Knowledge Discovery Library

For the Knowledge Discovery tasks themselves a library called Weka is used (see [19]or [20]), which is written in the Java language (see [24]) and available under the GNU public-licence. Weka does not only implement a number of algorithms for classification, numerical prediction and association rules, but does also support filtering, meta schemes, and testing. Another benefit of using Weka is, that it can be used as library, as well as application, which makes it simple to experiment with the given data, before embedding the algorithms, which worked best into the final application.

As the transformation and the post-processing of the results should be performed in Prolog, the only problem to use Weka, is to find a possibility to call Java-methods from a Prolog program. The solution for this problem is B-Prolog (see [18]), a prolog implementation, which includes such a bi-directional interface with Java.

To make the learning methods available even to user, who do not know any-thing about Java and Weka, a small library of Prolog predicates (Weka.pl) was developed that provides basic learning facilities:

- Create a classifier from a data file. The data file has to be in the ARFF format, a file format, which does not only store the data but also information about the individual attributes. For more information about ARFF see [20].
- Create an empty classifier without any data, just setting the ranges of the attributes and the learning method.
- Add a data set to the existing data. This data set has to contain the class attribute for supervised learning.
- Classify a data set. This data set does not contain the class attribute. It is the task of the learning algorithm to find it.
- Save the whole classifier to a file.
- Load a classifier, which has been previously saved with the predicate save.

For a detailed description of these predicates as well as information how to set up B-Prolog and Weka, see “Weka.pl”.

Part III

A practical example

14 Detection and prediction of bottlenecks

14.1 Introduction

To present the basic idea of detecting bottlenecks, a small example. Figure 1 shows a very simple community of ProPlanT agents. There are two super-tasks 'x' and 'y'. 'x' is decomposed to 'a', 'b' and 'c'. 'y' is decomposed only to 'c'. The decomposition and assignment of sub-taks is done by the agent 'PMA'. Tasks 'a', 'b' and 'c' have the same execution time 5 and the same cost. Every PA is in charge of exactly one task (as shown in the figure).

Now the agent 'PMA' gets the following sequence of requests: 'x','y','y','y'. The figure shows, how the plans in the PA look like, after these tasks were requested. It is obvious, that if now task 'x' would be requested again, the sub-plans for 'a' and 'b' could be executed till execution time unit 10, while the sub-plan for task 'c' could be executed till 25. So the whole plan for task 'x' could be executed till time unit 25, which means that the whole plan for task 'x' is delayed only because of sub-task 'c'. So sub-task 'c' is a bottleneck for super-task 'x' at this moment.

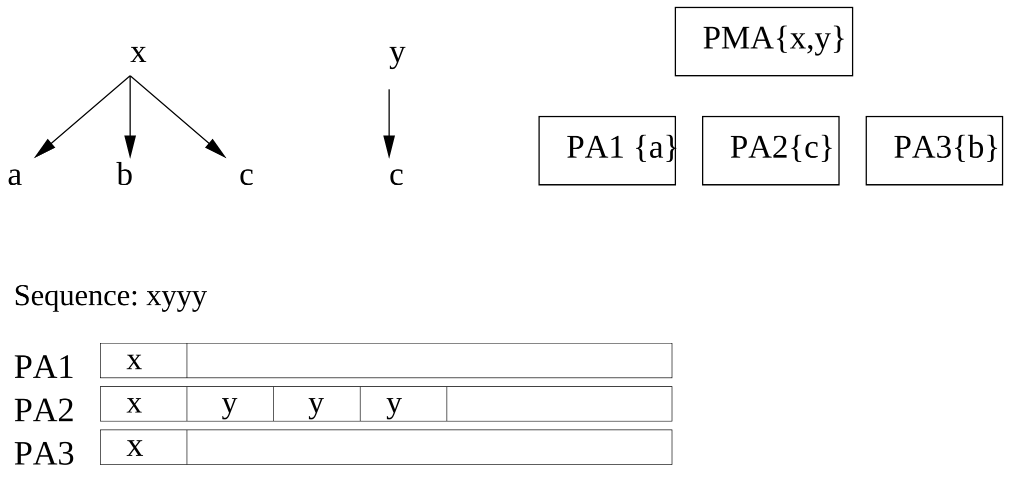


Figure 1: bottleneck example

These kinds of bottlenecks can be only temporal, which means that they vanish after a while, or they can be permanent, which means, that they will not vanish. This is an important question, because if a bottleneck is permanent then it could be a good idea to remove it, while a bottleneck, which is only temporal may not have to be removed. In the given example it is obvious, that the bottleneck will not vanish (because sub-task 'c' is required by both super-tasks).

So generally there are two problems: 1.Detecting bottlenecks and 2.Decide whether the bottleneck is permanent or not. The first problem can be reduced mainly to the problem of simulating the planning of a super-task, as then it is obvious how the execution times of the several subtasks are related to another and a simple function can dicide, whether there is a bottleneck or not (according to a precise definition). So the problem is how to simulate the planning of a super-task efficiently. In this section an approach will be presented, that combines reasoning about background knowledge with Knowledge Discovery methods to solve exactly this problem. The second problem is to predict the future development of bottlenecks. For this problem a simple approach will be represented, that observes which influence the individual super-tasks have on a given bottleneck, and combines this observation with the relative frequency of occurrence of the individual super-tasks to calculate an expected value for the future development.

14.2 Exact definition of the problem

The following list contains all issues, which define the scope of the problem as well as additional restriction and simplifications. These restrictions and simplifications are introduced in order to start with a simple system, which then can be generalized by removing the restrictions one by one.

14.2.1 Scope of the problem and some simplifications

- The term "bottleneck" reflects, that there is an imbalance in the use of the underlying resources, in opposite to the term "overload of the system", which stands for an insufficiency of the whole system to deal with incoming requests. Overloads of the whole system depend on the period of real time between two requests to the system. This time is a continuous quantity. So the task to predict and find overloads of the whole system is more a task for statistical queuing theory than for Machine Learning. For this reason, we assume for our experiment, that the period of real time between two requests to the system is always zero. In other words this means, that the system is used in batch-mode; there is a queue of requests, which is never empty.
- What is considered as resources are in fact pseudo-resources. The reason that it is not possible to use the real resources is that they are encapsulated in the PA and not visible for the Meta Agent. Therefore, for this experiment, a resource is a task, which can be achieved by a PA, because they are visible for the Meta Agent. To go further, and even to consider tasks which can only be achieved by PMA as resources would be possible as well. But since the aim of Knowledge Discovery in this case is to support the decision, whether to provide the system with new resource or not, it would not be clear in these cases, which resources to provide, because the PMA itself, does not represent a resource. (In the remainder of the document, *Resources* will denote this set of pseudo-resources).
- Principally it would be possible to consider both, bottlenecks concerning execution time and execution cost. However, as it is normally not possible to lower the cost in a system, there is usually no possibility to remove such a bottleneck. Therefore, it is not reasonable to find or to predict them, because this information would have no effect.
- As stated above we focus on imbalances in the use of resources, so it would be possible not only to find bottlenecks, but also inverse bottlenecks. This could be used to support the decision whether to remove a given resource from the system. As we want to keep things simple, the focus will be on bottlenecks rather than on inverse bottlenecks.
- Finally, we have to introduce a real limitation of the first version of the analysis system. In the second part of this documentation were presented the different possibilities the LogWriter offers to attach time-stamps to messages: real-time, enumeration of messages and runs. For this experiment, the third possibility is used. This means that all messages, which belong to one request to the system, have the same time-stamp. This concept only works under the condition, that the community contains only one PPA and that this PPA waits until the system responds to its request before sending a new request to the community (every time the PPA gets a reply or sorry message, the time counter within the LogWriter is incremented by one).
- Additionally some other simplifications are made. Firstly the cost of resources will not be considered. In this experiment every resource costs the same. Secondly, there will not be any time or cost limits in the requests. (This is not really a restriction, because the analysis

system could filter out runs, which are not successful). Thirdly there are no alternative task decompositions, every task is decomposed to exactly one set of subtasks, which all have to be achieved, in order to achieve the super-task.

14.2.2 Remarks on terminology used in this section

To make the following a little bit less confusing, here are some remarks on the terminology, which will be used in this part:

- *Time*
  We distinguish between real time, this is time in real world, execution time, this is the time contained in the plans and planning time (or simply time), which is for this experiment the number of the considered request. So if we say, that at time 5 there is a bottleneck, this means, that after the first five requests have been sent to the community and before the sixth request is send to the system, there is a bottleneck
- *Tasks, super-tasks and resources*
  We distinguish super-tasks (or simply tasks) from resources. In ProPlanT both are considered as tasks. We use the term task only for requests, which are send by the PPA. Requests, which are sent to PA, are resources. The remaining requests are not interesting for this experiment.
- *Appearance and formation of bottlenecks*
  As we will see, a bottleneck can exist at a given time, but can be invisible. So we say a bottleneck appears, if it is visible. On the other hand we use the term formation of a bottleneck if it comes into existence (but is not necessarily visible).
- *Variables and Values*
  Specific values are enclosed in single quotes, to distinguish them from variables. So 'a' stands for the specific task named 'a', while e.g. t stands for the variable time, g for the variable super-task and r for the variable resource and so on.
- *Bottlenecks*
  At some points bottlenecks will be denoted as <super-task><resource>, e.g. t1r5. This means that 'r5' is a bottleneck for 't1'.

14.2.3 The definition of a bottleneck

Now it is possible to formulate the idea of a bottleneck: To achieve a task g at top-level at time t, normally several resources are needed. The plan is completely executed, at the moment, when the sub-plans for all resources are executed. If the execution of a single resource r takes much longer, compared to the execution of the other resources, then the execution of the whole plan is delayed because of this single resource. In such cases, it is said that the resource r is a bottleneck for task g at time t. So in general a bottleneck is a relation with arity three, among a top-level task g, a resource r and a point in planning time t, we call bottleneck(t,g,r). In case that r is not needed to achieve g, bottleneck(t,g,r) is false. The obvious problem with this predicate is, that it can not be observed directly, because at a given time there is only one request g. So bottleneck(t,g,r) would only be defined for values t and g, such that g is really requested at time t. In order to expand the definition of bottlenecks to be applicable to any parameters, we have to distinguish between the existence and the appearance of a bottleneck. A bottleneck(t,g,r) appears, if there is a request for task g at time t and r behaves as defined above. To find bottlenecks, which do not appear, we have to estimate, what would have happened, if task g would have been requested at time t. So, as already stated above, the problem of detecting bottlenecks is reduced to the problem of simulating the planning of a super-task.

To obtain a mathematical definition of bottleneck two other predicates are needed: endtime and subtask.

endtime(t,g,r,et) is true, if the execution of all sub-plans for resource r within a plan for task g would be finished at execution time et, if g would be requested at planning time t.

subtask(st,t) is true, if st is a subtask of t. This relation is transitive.

Now the predicate bottleneck can be defined:

$bottleneck(t, g, r) \equiv$

$|\{l \in Resources \parallel subtask(l, g) \wedge endtime(t, g, r, f_1) \wedge endtime(t, g, l, f_2) \wedge \alpha(f_1 f_2)\}| > \beta * |\{l \in Resources \parallel subtask(l, r)\}|$

In words it could be said, that a resource r is a bottleneck for a task g at time t, if the execution of the sub-plans for resource r is finished later than the plans for β*100% of the other resources needed to achieve g at time t. The parameter β makes it possible to have more than one bottleneck. For example with $\beta = 0.7$ if 10 different resources are needed for a task g and two of them take much longer than the rest, then there are two bottlenecks. α is a predicate, which makes it possible to compare the endtimes in different ways. For the remainder of this document, $\alpha(x, y) \equiv x > y + c$ is assumed.

Applied to the small example from the introduction, the following values can be calculated:

endtime(4,'x','a',10)
endtime(4,'x','b',10)
endtime(4,'x','c',25)

with $\alpha(x, y) \equiv x > y + 10$ and $\beta = 0.6$ for bottleneck(4,'x','c') this means, that

bottleneck(4,'x','c') $\equiv$ |{'a','b'}| $>$ 0.6 $*$ |{'a','b','c'}| $\equiv$ *true*

So bottleneck(4,'x','c') holds, which means that at planning time 4, resource 'c' is a bottleneck for task 'x'.

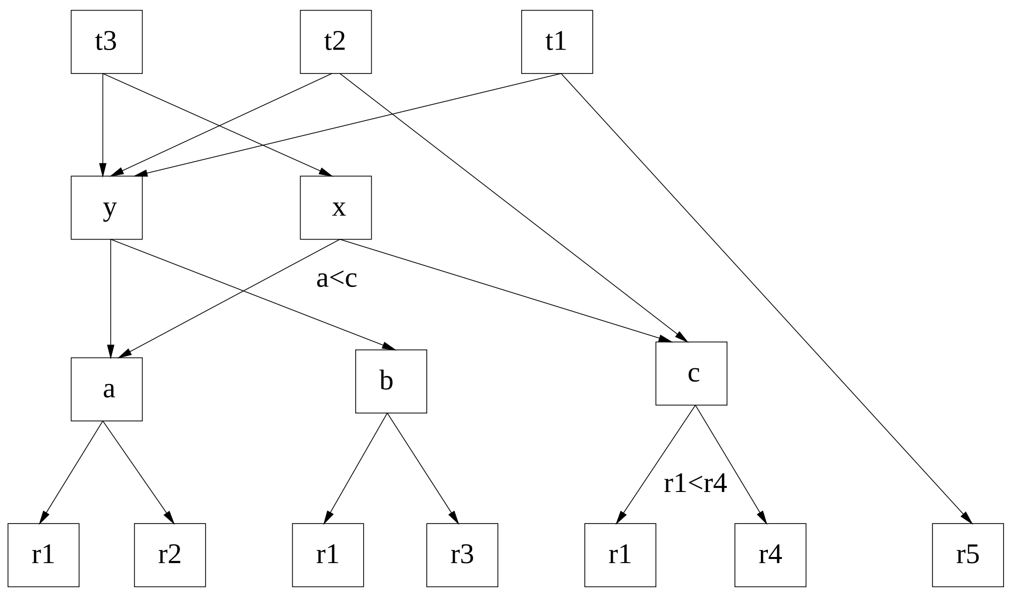
14.3 The test community

In order to obtain empirical results it is important to have a test community of ProPlanT agents. The following are the requirements of such a test community:

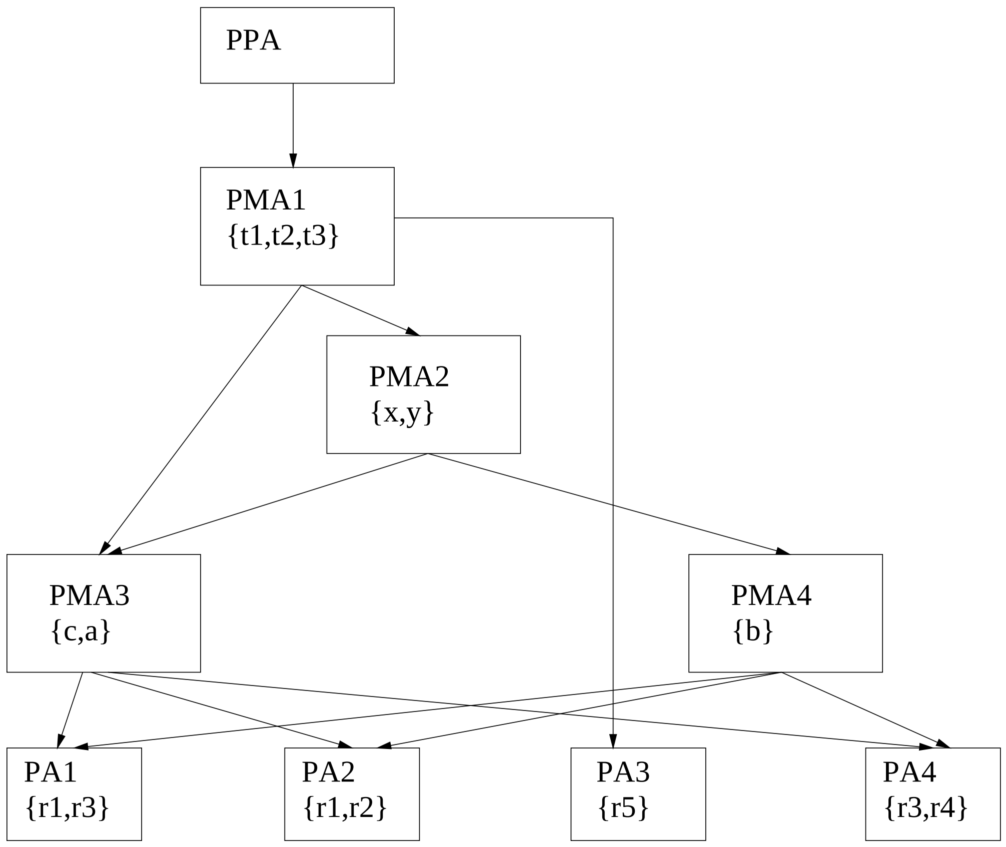
- it should be simple enough to be clear and easy to handle
- it should be complex enough, such that it would be hard for a human user to analyze it directly
- it should be representative, which means it should contain everything a real system could contain
- it should contain some bottlenecks

The following community contains only eight agents and for this reason it is not complex enough to get reliable results. But it already contains some features of more complicated communities as precedence rules, several levels of task decompositions and parallel as well as sequential execution within PAs.

The super tasks are 't1','t2','t3'. The resources are 'r1', 'r2', 'r3', 'r4', 'r5'. The complete decomposition looks like this (an arrow means, that the second task is part of the decomposition of the first one. As already stated, this is an AND-tree, as alternative decompositions are not considered)



There are 4 PMA and 4 PA. The following diagram shows, how they work together (an arrow between two agents states, that the first agent can cooperate with the second; the tasks enclosed in brackets are the tasks the agent can deal with)



Finally the following table contains the execution times of the individual resources:

	PA1	PA2	PA3	PA4	PMA1	PMA2	PMA3	PMA4
r1	10	5	-	-	-	-	-	-
r2	-	7	-	-	-	-	-	-
r3	10	-	-	20	-	-	-	-
r4	-	-	-	3	-	-	-	-
r5	-	-	11	-	-	-	-	-
t1	-	-	-	-	+	-	-	-
t2	-	-	-	-	+	-	-	-
t3	-	-	-	-	+	-	-	-
x	-	-	-	-	-	+	-	-
y	-	-	-	-	-	+	-	-
a	-	-	-	-	-	-	+	-
b	-	-	-	-	-	-	-	+
c	-	-	-	-	-	-	+	-

'PA1' contains two machines, one for 'r1' and one for 'r3'. 'PA2' and 'PA4' contain only one machine and execute their tasks sequentially.

14.4 Background Knowledge

In order to detect and to predict bottlenecks, it is necessary to have some knowledge about the given group of agents. We will refer to this kind of knowledge as background knowledge. The following is a list of all predicates, which express this background knowledge:

ppa(A) is true, if A is a product-planning agent.

pma(A) is true, if A is a product management agent.

pa(A) is true, if A is a production agent.

resource(T) is true, if task T can be performed by a production agent (see above)

capable_of(A,T,D) is true, if agent A is a PA and is able to perform task T in time D (we assume here, that D is constant, see above)

subtask(ST,T) is true, if ST is a subtask of T.

The program generate_bgk.pl is able to generate parts of these facts automatically from the messages. However, especially the predicate subtask causes some problems:

1. If there is no request for a given super-task T in the message-file, then there is no possibility to generate the facts subtask(_,T).
2. If a PMA has to handle several requests in one run, it is not clear, how to relate the different super-tasks to the sub-tasks, so generally the tool will return wrong results in these cases.

This means, that the generated file 'bgk.pl' has to be checked for errors, before it is used. Its purpose is mainly to save the user some work. The second problem could be avoided by using a much more intelligent algorithm, but this would be out of the scope of this experiment.

14.5 Detecting bottlenecks

As stated above the problem of finding bottlenecks has been reduced to the problem of finding the values for the predicate endtime(t,g,r,et). This can be done in several ways. Firstly it would be possible simply to simulate the planning of super-task g at time t and to obtain the value et from the reply message. The problem with this approach is, that it would be quite costly to do this for every super-task. Another possibility would be, to use Numerical Prediction, as defined in 2.4.1 to obtain these values. The idea is to learn to predict the values for the predicate endtime. But it showed, that this approach does not lead to results, which are precise enough to be used to detect bottlenecks (see Evaluation). A third approach, which is presented here, combines the first two approaches. It first calculates a rough estimate for endtime using a simple planning algorithm (e.g not considering precedence rules) and then uses Numerical Prediction to predict a correction factor, which is added to the rough estimate. So in fact estimating the predicate endtime is split into two steps: 1. A simple algorithm to calculate a first, rough estimate and 2.A learning algorithm, which learns to adjust this first estimate to get a more precise one. These both steps are described in the following sub-sections.

14.5.1 Step 1:A simple algorithm to simulate planning

The algorithm which is presented in this section makes use of the fact, that the PA inform other agents about how long the execution of a task would take if it would be planned (see section 5). So it can be presumed that there is a predicate called aendtime(time,agent,resource,advertised_endtime), which can be obtained directly from the advertise messages. The algorithm uses some predicates from the background knowledge together with the predicate aendtime to calculate a rough estimate for endtime:

- construct a set, consisting of pairs (a,et), where a is the name of a PA, which is able to perform r and which advertised et as end-time for the execution of r at t. The resulting set is called L. e.g. L = {('PA1',120), ('PA2',140)}
- for i = 0 to number of occurrences of resource r in the plan for g - 1
- - find the minimal element (second component minimal) in L: (a,et)
  - find x, such that capable_of(a,r,x) holds (look up x)
  - remove this element (a,et) and add (a, et + x) to L
- return the minimal element of L (second component)

Example:

aendtime(3,'PA1','r1',10), aendtime(3,'PA2','r1',14)

capable_of('PA1','r1',10), capable_of('PA2','r1',5)

endtime(4,'t3','r1',X): X should be calculated:

1. Construct L: L= {('PA1',10),('PA2',14)}
2. As 'r1' is needed 4 times in every plan of 't3' the loop is executed three times:
   (a) L = {('PA1',10),('PA2',14)} remove ('PA1',10) add ('PA1',20)
   (b) L = {('PA1',20),('PA2',14)} remove ('PA2',14) add ('PA2',19)
   (c) L = {('PA1',20),('PA2',19)} remove ('PA2',19) add ('PA2',24)
3. L = {('PA1',10),('PA2',15)} the minimal element is ('PA1',20) .

The result is endtime(4,'t3','r1',20).

14.5.2 Step 2:A Knowledge Discovery system to predict a correction factor for step 1

As already stated, the correction factor should be obtained by learning. So first, the learning problem has to be defined more precisely. As described in the first part of this documentation, a common standard learning problem is numerical prediction. This is exactly the learning task needed for purpose of this experiment. A numerical value (correction factor) should be found from a set of attribute values. The underlying experience is the following. If task g is requested at time t, then we get the correct values for endtime at time t. So in these cases, we can additionally calculate the estimated endtime using the algorithm presented in step one (14.5) and the difference between this estimated value and the real value. This difference is exactly the correction factor and can be used to train the learner. After this training, the learner can be applied to new instances.

In the terminology of the Knowledge Discovery process, the following is done:

- *filtering*:
  every run is checked, according to some basic rules (e.g. the number of achieve messages is the same as the number of reply and sorry messages). If a run does not meet theses basic constraints it contains errors and has to be filtered out completly.
- *transformation and data reduction*:
  First the data, which is not needed is filtered out too (e.g. subscribe messages), then specific predicates are generated, as aendtime (see above). After this the data for learning can be generated, which means that aendtimes and the background knowledge are combined to calculate the estimated value for the last entry in the predicate endtime. After this the difference between this estimated and the real value, contained in the reply message is calculated and written to an ARFF file, together with some attributes (see below).
- *post-processing*:
  In this case the learned concept is applied directly to new data, which means that it is used for the estimation of the predicate endtime, if the correct value is not available (if the super-task wasn't requested at the given time).
- *evaluation:*
  In the same way as for creating the ARFF file for learning a second ARFF file can be created. Now the learner can be applied to data which is new to him. Now the mean difference between the estimation of the learner and the exact values contained in the training set can be calculated, which is measure for the quality of the learned concept (see Evaluation).

There are two choices for learning:

1. the learning algorithm
2. the set of attributes, which is used

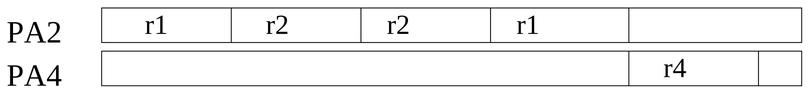
While it is easy to compare the performance of different algorithms, the choice of the attributes is much more complicated. To find a suitable set of attributes, we take a look at the algorithm in step one and try to find out, why it does not return the correct results:

1. *Unknown internal structure of the PA*
   An example: two PA can both achieve task 'a' and 'b', but 'PA1' represents two different machines, one for resource 'a' and one for resource 'b'. 'PA2' represents only one machine for both. The algorithm implicitly assumes, that a PA represents a separated machine for every resource (otherwise it would not be possible to calculate the

predicate endtime independently for several resources). The problem is, that if a super-task consists of both resource 'a' and resource 'b', then the algorithm will make a mistake with 'PA2'. If 'PA2' advertises, that it can achieve resource 'a' till execution end-time 5 and 'b' till 5, and resource 'a' and resource 'b' take 5 time-units each, then only one of the resources can really be achieved till the advertised execution time. Which one this is, depends on which request arrives at 'PA2' first. If the request for 'a' arrives first, then the execution endtime for 'a' will be 5, the execution endtime for 'b' 10, if the request for 'b' arrives first, it's the other way round. If we assume that this order (which request arrives usually first) does not change significantly, then the error, which occurs due to this problem will only depend on the super-task and the resource. This means that these quantities are sufficient as attributes for compensating this problem.

2. *Precedence rules*

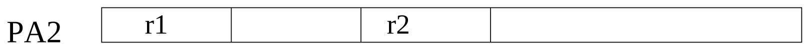
   As the algorithm does not at all consider precedence rules, this is clearly a source for errors. The following diagram shows this clearly:



   Although 'r4' could be executed at execution time 0, because of the precedence rule it has to be executed after 'r1'. The algorithm in step one would calculate 3 as estimated execution endtime for resource 'r4'. The real execution endtime is much higher and depends on the time at which 'r1' is executed. An extreme case would be, that 'PA4' advertises execution end-time 3 all the time, but actually 'r4' is never executed at this time. Depending how often this problem occurs, either we can hope that learning can compensate it (the attributes then are again super-task and resource) or we have to use another basis for learning, in this case for example the estimated execution endtime of 'r1'.

3. *Fragmentation of the plan of tasks*

   The following example shows the problem:



   'PA2' advertises execution endtime 10 for resource 'r1'. If we need resource 'r1' two times in a plan, then the algorithm assumes, that 'PA2' can execute 'r1' two times till 15. But this is not true, because the time between 10 and 15 is already occupied. The correct execution endtime for two executions of 'r1' is 20. The problem is, that the algorithm has only the values, which are advertised and does not know, how the plan of tasks of a PA exactly looks like. The only possibility to deal with this problem is to consider past events, because they have led to the fragmentation in the plan of tasks. As past events, for example requests to a given PA or to the whole system can be used.

4. *Sub-task assignment*

   Because of several factors, e.g. network speed, it is not possible to predict exactly, how sub-task assignment is performed. It could be possible, that an agent decides sub-optimally in some cases, or that the whole system optimizes for cost and not for time, etc. The learner should be able to deal with such problems. As choices for attributes, we use the differences of the advertised endtimes for the same resources form different agents. This is exactly the information, which the PMA use to perform sub-task assignment. For the given community this results in the following two attributes:

endtime(T,'PA1','r1',X), endtime(T,'PA2','r1',Y): ATT1 = X + Y
endtime(T,'PA1','r3',A), endtime(T,'PA4','r3',B): ATT2 = A + B

Summarizing the following set of attributes is used for learning in the given community:

- super-task
- resource
- last super-task
- difference for r1
- difference for r4

For learning algorithms, which can not deal with continuous attributes, the last two arguments are discretized. For 'r4' an extra learner is used (with the same attributes). As basis for the calculation of the execution endtime of 'r4' we do not use the algorithm in step one, but the estimated execution endtime of 'r1' (as there is a precedence rule between 'r1' and 'r4').

14.5.3 Evaluation

There are two major requirements, which this system should fulfill:

1. it should be efficient.
2. it should produce results, which do not differ too much from the exact results.

Whether the system is efficient can be decided by analyzing the computational complexity of the algorithms, which are applied. To find out, how good the results are, which the system produces they have to be compared empirically to the results which are produced by a simulation (using evaluate).

Computational complexity

For the algorithm used in step 1 the following formula can be found:

$$O(g(r) + (f(g,r) - 1) * log(g(r))))$$

where
f(g,r) is the number of times r is used in a plan for g and
g(r) is the number of agents, which are capable of r.

(Proof: The list L can be organized as heap. This heap can be build in linear time with the number of elements, which is here g(r). The inner part of the loop consistes of one 'downheap' operation: change the top element and let it sink down. Downheap can be performed in logarithmic time with the number of elements in the heap, so here in log(g(r))). The loop is passed f(g,r) times, according to the definition of the algorithm.)

As g(r) will be in any realistic case very small (because the agents are mostly specified to a small range of tasks), it can be considered as constant. Then the algorithm is in O(f(g,r)). For the detection of a bottleneck this means, that a given bottleneck(t,g,r) can be detected in a number of steps which increases linear with the number of resources needed for planning the task g. Such an algorithm is by far efficient enough for our needs.

The second step is even more efficient. An upper bound for the number of attributes is the number of PA times the number of resources plus 4 (if we use the above schema for attribute selection). If a Decision Tree is used for learning, this means that an instance can be classified in

$$log(\text{number\_of\_resources*number\_of\_agents}) =$$

$$log(\text{number\_of\_resources}) + log(\text{number\_of\_PA})$$

Normally the tree will be pruned, such that this value is only an unrealistic upper bound.

(Proof: A Decision Tree is a tree of highth smaller or equal than the number of attributes used for learning. See [1])

This shows that the system fulfills the first requirement of being efficient, the second question is, whether it produces results, which do not differ too much from the simulated ones.

Empirical Results

In general, two values are of special interest:

1. The absolute mean error, which is the mean difference between the results of simulation and the results of the short-cut algorithm. It is denoted in execution time units. If this value is much too high, it will not be possible to detect bottlenecks with the system, that has been presented.
2. The relative improvement, which results from learning, that is from the second step. A relative improvement of 50% means for example, that the error could be reduced to a half of its value without learning. This makes it possible to compare the performance of different learning algorithms and shows, whether learning makes sense in this setting or not. (resource r4 is not considered for this evaluation, because the algorithm in step one does not work for r4 (see above)).

To obtain empirical results, different learning methods were used:

1. Linear Regression, a well known algorithm for numerical prediction
2. Decision Tables

3. Decision Trees, which are normally used for classification rather than for numerical prediction. To use them for this purpose, the correction factor is split into several intervals. Each of these intervals is represented by a class. Now classification can be applied and the predicted value is the mean of all values in the interval, which is represented by the predicted class. This method is called Regression by Discretation. The algorithm used to build the Decision Tree is C4.5
4. Naïve Bayes. This is a Baysian Method for classification. The same transformation is used as for Decision Trees to apply it to numerical prediction.

Firstly a training set (135 instances) and a test set (44 instances) were generated with the same community of agents using the Generator agent. This led to the following results:

	Step1 only	Step 1+ 2	Rel. impr.	Step 1+2 (incl. R4)
C4.5	3.37	1.28	263.2 %	1.275
Naïve Bayes	3.37	3.08	109.4 %	3.05
Lin. Regression	3.37	3.31	101.7 %	3.45
Decision Table	3.37	1.34	251.2 %	1.625

This tabular shows that the absolute error is very low. So the estimation of the short cut differs not substantially from the simulated values, which shows that the system can be used to detect bottlenecks without any problems. The next interesting observation is, that learning had quite an influence on these results. C4.5 and Decision Table could reduce the error to more than a half. Although it shows too, that Linear Regression and Naïve Bayes are not suitable in this setting.

Another interesting question is, whether only learning would also produce similar results. To obtain empirical values in this case, the setting has to be changed. Now not the algorithm from step one is used as first rough estimation of endtime, but the endtime for this resource from the last run (or the overall endtime of the last run, if the first is not defined). Then a learner is used in exactly the same way as above to learn a correction factor. The results are the following:

	Step 1 + 2	Step 2 only
C4.5	1.28	24.1
Naïve Bayes	3.08	20.0
Linear Regression	3.31	27.85
Decision Table	1.34	28.75

This clearly shows that learning alone does not lead to acceptable results. (Maybe these results could be improved to some extend by using better sets of attributes and a better basis, but it does not seem that this would change these results substantially).

One benefit of learning is, that the learner can assimilate even to new situations, which were not considered in advance. For the next experiment the setting is changed, such that the system contains an error. 'PA2' always replies to requests for 'r1' such that the execution endtime is three time-units higher than it should be (according to the advertise messages he sends). The question is, whether the system can cope with such an error. The following are the results:

	Step1 only	Step 1+ 2	Rel. improvement
C4.5	4.14	1.29	357.1 %
Naïve Bayes	4.14	2.8	147 %
Linear Regression	4.14	3.1	133.3 %
Decision Table	4.14	1.37	302.1 %

Obviously the total results were affected only little by the error in the system, although the results of the algorithm in step one are worse. This shows that learning can compensate minor errors in the system, which confuse the deterministic algorithm in step one.

Another interesting question is, what happens, if the concepts learned from one community are applied to a modification of this community. This is important, because there are often minor changes in a community and it would be unpleasant, if learning would have to be repeated after every small change. So some of the values in the original community are changed and the Generator Agent is used to create a second test set. The results are the following:

	Step1 only	Step 1+ 2	Rel. impr.	Step 1+2 (incl. R4)
C4.5	4.72	2.76	171.5 %	2.82
Naïve Bayes	4.72	4.47	105.6 %	4.9
Lin. Regression	4.72	4.73	99.0 %	4.8
Decision Table	4.72	3.13	151.0 %	3.4

Again the absolute error is quite low. C4.5 and Decision Table worked both well, but C4.5 work significantly better in this case, which can be seen as a clue, that C4.5 generalizes better over the details of a given community.

Summarizing it can be said, that at least for the given community the difference between estimated and simulated (real) results is surprisingly low. Furthermore it showed that only learning does not produce acceptable results. On the other hand the deterministic algorithm from step one can be improved by learning, especially if there are minor errors in the system, with which this algorithm alone can not deal. Important for the practical usage was, that it was possible to use a learned concept even in a modified community without having to repeat learning.

The comparison of different learning algorithms showed, that Linear Regression and Naïve Bayes were mostly not suitable for this learning task. C4.5 and Decision Tables both worked well, but C4.5 seems to work mostly better than Decision Tables. Especially in the last experiment, in which the learned concept was applied to a modified version of the original community, this was obvious.

14.6 Predicting the future development of bottlenecks

A detailed analyzes of the future development of bottlenecks would be rather complicated and is beyond the scope of this experiment. So our aim is only to find a simple heuristic that gives the user a hint, whether a given bottleneck is to going to vanish or not, whether it pays to do something about this bottleneck or not. For this reason, we want to find a method, which decides for a given bottleneck, whether it is likely to vanish or not. The general idea, how to achieve this is the following: We do not only consider, whether there is a bottleneck or not, but also calculate a quantity, which represents the strength of this bottleneck. Now we observe, how this value changes from request to request. We can then associate this change factor with the task, which was requested at this time. We now assume that every request appears with a fixed probability. Using this probabilities we can calculate the expected value for the total changes of this bottleneck. Is this expected value positive, then the bottleneck is not likely to vanish, is it negative it will probably vanish.

The following example should make this clearer (we use the simple community from above). The formula, which is used to calculate a value for the strength of a bottleneck is the following:

$$strength(t, g, r) =$$
$$avg\{f_1 - f_2 | endtime(t, g, r, f_1) \wedge endtime(t, g, l, f_2) \wedge$$
$$subtask(l, g) \wedge f_1 > f_2 + \alpha\}$$

Now the strenght of the bottleneck 'xc' can be calculated for every moment in planning time. The column 'request' denotes the task, which has been requested at the refering time. This diagram can be used to find out, which influence the individual requests at top-level have on the strength of the given bottleneck.

Time	Request	Strength of bottleneck('x','c')
1	'x'	-
2	'y'	-
3	'y'	5
4	'y'	10
5	'y'	15
6	'x'	20
7	-	20

We find the following result: A request for task 'x' leaves the strength of the bottleneck unchanged a request for task 'y' increases it. Without even looking at the probabilities we see, that this bottleneck is not very likely to vanish (in this case it will certainly not vanish).

The remainder of this section describes how this method works in general.

The prediction of future development of a bottleneck consists of three steps:

1. Measure the strength of the bottleneck at different times

2. Calculate a change factor for every bottleneck and every request to the system
3. Use the probabilities of the different requests to the system to calculate a mean change for a bottleneck.

14.6.1 Step 1: Measure the strength of a bottleneck

In general there are several possible measures for this problem. We will only consider two of them:
(note that the sets used here are multi-sets, which means that they can contain the same value several times)

1. *Average*

   $strength(t, g, r) =$
   $avg\{f_1 - f_2 | endtime(t, g, r, f_1) \wedge endtime(t, g, l, f_2) \wedge$
   $subtask(l, g) \wedge f_1 > f_2 + \alpha\}$

   This value has the benefit, that it represents every resource in the set. On the other hand, it has the disadvantage, that this value can change, only because the values in the set among themselves change somehow, which has no effect on the bottleneck.
2. *Maximal*

   $strength(t, g, r) =$
   $\max\{f_1 - f_2 | endtime(t, g, r, f_1) \wedge endtime(t, g, l, f_2) \wedge$
   $subtask(l, g) \wedge f_1 > f_2 + \alpha\}$

   This problem does not occur by using the maximal value, but the problem is, that this maximal value does not reflect all values in the set, but only one value from the set.

So both measures have advantages and disadvantages. We will compare them in the evaluation.

14.6.2 Step 2: Calculating the average change

For every bottleneck and every request to the system the following values is calculated:

$avg\_change(g, r, \text{req\_task}) =$

$avg\{strength(g, r, t_1) - strength(g, r, t_2) | request(t_1 - 1, \text{req\_task})\}$

where request(t,x) denotes that at planning time t, task x has been requested by the PPA.

In words: calculate for every bottleneck, how its strength on average changes by different requests to the system.

14.6.3 Step 3: Calculate the expected total change

We assume that every request to the system has a fixed probability. We know calculate the expected total change as follows:

$$\text{total\_change(g,r)} = \sum_{x \in Tasks} p(x) * avg\_change(g, r, x)$$

As this value will be normally very imprecise (at least with the measures above), it can be taken only as a hint, how the bottleneck probably will develop.

14.6.4 Evaluation and an Example

The test community presented in this section behaves as follows: the bottleneck in 'r5' for 't1' increases with 't1' and decreases with 't2' and 't3'. The bottlenecks in 'r3' for 't1' and 't2' increase with 't1' and 't2' and increase only little or not at all with 't3'.

We now send some tasks to the community and calculate the average change factors with both measures, to observe whether they find these regularities.

Average

	t1r5	t2r3	t3r3
t1	24	19	19
t2	-26	15	15
t3	-22	16	6

Maximum

	t1r5	t2r3	t3r3
t1	12	19	19
t2	-35	14	15
t3	-30	7	10

We see that both show the desired values. A problem is that the mean standard deviation in both cases is very high, for maximum 17.3 and for average 14.3. This shows that these values are not very reliable.

Using these values, we know can calculate the expected total change factors for the three bottlenecks (we assume that p('t1') = p('t2') = p('t3') = 1/3)

Average

t1r5 : 1/3 * 24 - 1/3*26 - 1/3 * 22 = -8

t2r3 : 1/3 * 19 + 1/3*15 + 1/3*16 = 16.7

t3r3 : 1/3* 19 + 1/3*15 + 1/3*6 = 13.3

Maximum

t1r5 : 1/3 * 12 - 1/3*35 - 1/3 * 30 = -17.7

t2r3 : 1/3 * 19 + 1/3*14 + 1/3*7 = 13.3

t3r3 : 1/3* 19 + 1/3*15 + 1/3*10 = 14.7

Although some of the values differ, both calculations come to the same results: bottleneck 't1r5' will be removed, bottlenecks 't2r3' and 't3r3' won't. We can check this prediction empirically. We first artificially produce these bottlenecks by sending ten times requests for 't1' to the system. After this we send random requests to the system, with probabilities p('t1') = p('t2') = p('t3') = 1/3. In a first try, the bottleneck 't1r5' vanished after 55 requests, after 100 requests it still wasn't there. Bottlenecks 't2r3' , 't3r3' did not vanish at all. After repeating this experiment several times 't2r3' and 't3r3' never vanished, and 't1r5' vanish on average after 53.7 requests.

14.7 Open problems and future work

Although the system already works fine within the defined scope, there are several open problems and future challenges:

- *Better Decision Support*
  At the moment the system doesn't directly support the decision, whether to add resources to the community, or not for the following reasons. Firstly, not every pair of task and resource, for which the predicate bottleneck is true at some time, is really a bottleneck in the more general sense. A bottleneck for 'r1', always leads to a bottleneck of 'r4', because 'r4' is bound with a precedence rule to 'r1'. Providing new resources for 'r4' would not solve this problem, but new resources for 'r1' would. Another problem is that resources, which can be executed on the same machine, are interdependent. This means for example that a bottleneck for 'r1' always leads to a bottleneck for 'r2', because 'r2' is executed on the same machine as 'r1'. In these cases, it's not clear, whether to provide a new resource for 'r1' or for 'r2'. A decision support system should also cover these problems.
- *Imprecise Prediction*
  Although the proposed method showed to be successful in giving the user a hint, whether a given bottleneck is likely to vanish or not, this prediction was not very precise. Mostly it would be very important for the user to know, when this bottleneck will vanish (after how many requests). To achieve such a precision requires far more elaborate methods, than the ones, which were proposed.
- *Meta Schema*
  At the moment, it's the task of the user to perform attribute selection for learning and to fix the parameters. This is not very contenting, because these tasks require some amount of knowledge about the system and about Knowledge Discovery in general. Rather the system itself should find the optimal values for these parameters. This could be done by a meta schema, which controls learning from a meta level.
- *Restrictions*
  To make the system applicable not only in communities, which fit into the stated restriction, but in general, these restrictions have to be removed. Mostly the problem that the community must not contain more than one PPA is a problem, because the concept of time would have to be changed.

- *User Interaction*
  In addition to use background knowledge concerning the technical details of a community, it would be desirable to involve the user directly in the analysis process in order to give him/her the possibility to bring in some knowledge, which concerns the environment in which the system is used (for example which resource can't be added for some reason, etc.). This could be done for example with an expert system.
- *Evaluation*
  To get more reliable and precise information about the performance of the system, a much more complicated community of agents would be necessary.

References

[1] T. Mitchell: Machine Learning, McGraw Hill, 1997

[2] G. Weiss (Editor): Multi-Agent Systems : A Modern Approach to Distributed Artificial Intelligence, MIT Press, 1999

[3] G. Weiss (Editor): Distributed Artificial Intelligence Meets Machine Learning : Learning in Multi-Agent Environments : Ecai'96 Workshop Ldais Budapest, Hungary, August 1, Springer Verlag, 1997

[4] G. Weiss, S. Sen (Editors): Adaption and Learning in Multi-Agent Systems : Ijcai '95 Workshop, Montreal, Canada, August 21, 1995, Proceedings (Lecture Notes in Computer Science), Springer Verlag 1996

[5] D. Carmel, S. Markovitch: Opponent Modelling in Multi-Agent Systems, in [4]

[6] Y. Mor, C.V. Goldman, J.S. Rosenschein: Learn Your Opponent's Strategy (in Polynomial Time)!, in [4]

[7] T. Ohko, K. Hiraki, Y. Anzai: Learning to Reduce Communication Cost on Task Negotiation Among Multiple Autonomous Mobile Robots, in [4]

[8] C. Versino, L. M. Gambardella: Learning Real Team Solutions, in [3]

[9] R. Nadella, S. Sen: Correlating Internal Parameters and External Performance: Learning Soccer Agents, in [3]

[10] A. F. Dragoni, P. Giorgini: Learning Agents' Reliability Through Baysian Conditioning: A Simulated Experiment, in: [3]

[11] H. Friedrich, M. Kaiser, O. Rogalla, R. Dillmann: Learning and Communication in Multi-Agent Systems, in: [3]

[12] A. L. C. Bazzan: Evolution of Coordination as a Metaphor for Learning in Mult-Agent Systems, in: [3]

[13] V. Marik, M. Pechoucek, O. Stepankova, J. Lazansky: ProPlanT: Multi-Agent System for Production Planning, International Journal of Applied Artificial Intelligence, 2000

[14] The ProPlanT technical definition

[15] http://www.csee.umbc.edu/kqml (KQML)

[16] U. Fayyad (Editor), G. Piatetsky-Shapiro (Editor), R. Uthurusamy (Editor): Advances in Knowledge Discovery and Data Mining, MIT Press, 1996

[17] E. Kreyszig: Advanced engenineering mathmatics, John Wiley & sons, inc. , 1999

[18] http://www.probp.com/ or http://www.sci.brooklyn.cuny.edu/zhou/bprolog.html (B-Prolog)

[19] http://www.cs.waikato.ac.nz/ml/weka/index.html (Weka)

[20] I. H. Witten, E. Frank: Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations, Morgan Kaufmann, 1999

[21] http://www.franz.com (Allegro Commen Lisp)

[22] http://www.pdc.dk/vip/ (Visual-Prolog)

[23] http://www.logic-programming.org/prolog_std.html (ISO Prolog)

[24] http://www.javasoft.com (Java)

[25] http://www.sybase.com/products/archivedproducts/power/ (Power++)