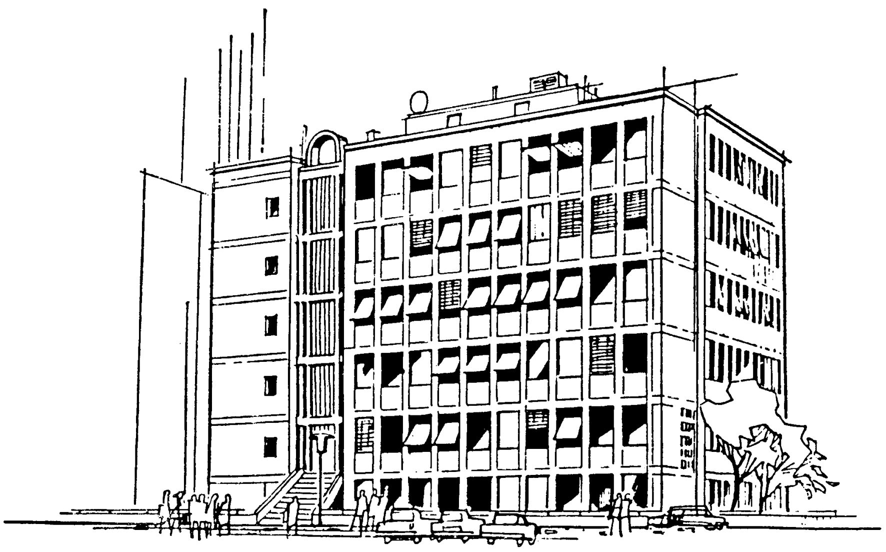
DIPLOMARBEIT

Shervin Nouyan

Implementation of an Insect Agent Based Model for Distributed Factory Coordination



Lehrstuhl für Nachrichtentechnik
Technische Universität München
Prof. Dr.-Ing. Joachim Hagenauer

Acknowledgements

This thesis could not have been written without the help of many people. First of all, I want to thank my two supervisors, Professor Gert Hauske, who accepted my supervision in Munich, and Professor Marco Dorigo, who gave me the possibility to work in his lab in Brussels. I was quite desperate before coming to Brussels, as it was surprisingly difficult to find someone to accept my supervision in Munich. For this reason, I am very thankful to Professor Hauske. He took away many stones from my way and enabled me to go to Brussels. Without his help, I would probably still be searching for a supervisor in Munich.

Professor Dorigo has introduced me to his lab, IRIDIA, and helped me a lot throughout the time I've spent here. Thanks to his advice and criticism I learned much. I look forward to start a PhD under his supervision.

I want to thank all the people who helped me during the last weeksof writing this thesis with the hope that I forget noone. Thank you Isabelle, for your love and your care. Thank you Vito, for your permanent help and the cigarette breaks, of course. Thank you Josh, for the advice and help you gave to me. Thank you Carlotta, for sitting next to me and taking care about me. Thank you Erol, for losing against me in table-tennis when I really needed some motivation. Thank you Mauro, Halva, Christian, Rodi, Gianni, Bruno and Michael, for helping me and making IRIDIA such a nice place. Thank you Max, for being incredibly patient and answering all my questions, no matter how stupid they are. Thank you Emre, for always being the last one to stay with me in the lab at the night.

Last but not least, I wish to thank my family, who I love and miss very much. Thank you for offering me a warm place I always can go to.

Abstract

We investigate a multi-agent algorithm inspired by the task allocation behavior of social insects for the dynamic task allocation problem (DTA). The problem is a non-deterministic scheduling problem, consisting of identical machines and different types of tasks. A painting facility, consisting of trucks and paint booths, can be taken as example for the DTA. Trucks roll off an assembly line to get painted by paint booths. Crucial to this problem is the time required by paint booths to reconfigure, that is, to change the color in which they paint the trucks. Such reconfigurations are costly in time.

In the insect-based approach, agents are in charge of machines and autonomously bid for tasks. The agents' control is based on a threshold model proposed by Bonabeau *et al.* [5], which is inspired by the methodology of division of labour in social insects. Applying this threshold model to the agents results in a robust system with the ability to adapt to changing demands. The agents tend to specialize for one type of task, so that unnecessary reconfigurations are prevented.

Our work is based on previously introduced models described by Cicirello *et al.* [12] and by Campos *et al.* [9]. Several improvements are proposed. These proposals introduce new rules into the model in order to speed up the system's adaptation, and modify existing rules of the model to overcome some observed problems. In order to verify our proposals, we test them on a wide set of DTA instances. The results of these tests are presented. A detailed analysis shows that our proposals result in an improved overall performance of the approach.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this thesis, an insect-based approach to a dynamic scheduling problem is presented. In general, scheduling problems consist of resources and tasks, which have to be allocated to the resources. In our system, which is based on previously introduced approaches by Campos *et al.* [9] and by Cicirello *et al.* [12], insect-like agents are in charge of the resources and bid for tasks.

Social insect colonies use intelligent and distributed methods to collectively solve complex problems. Cooperation among individual insects is largely self-organized and does not require any supervision. The collective behavior that emerges from a group of social insects is referred to as *swarm intelligence*. The research area that deals with applying swarm intelligence to various problems has come under increasing attention in the research community in the last years [3]. One of the early studies of swarm intelligence investigated the foraging behavior of ants. Ants lay trails of pheromone, a chemical substance that attracts other ants. Deneubourg *et al.* [16] showed that this process of laying a pheromone trail that others can follow, is a good strategy for finding the shortest path between a nest and a food source. In experiments with an Argentine ant species, Deneubourg *et al.* constructed a bridge with two branches, one twice as long as the other, separating the nest from a food source. Within few minutes the ants mostly selected the shorter branch. Based on the idea of pheromone laying and following, Dorigo *et al.* [19] developed a way to solve the well known and NP-complete[1] traveling salesman problem (TSP). The TSP is the problem of finding the shortest route that goes through a given number of cities exactly once (see Figure 1.1). The algorithm, implemented by Dorigo *et al.*, has obtained near optimal solutions.

[1]NP stands for nondeterministic polynomial. The TSP is NP-complete, as the number of computational steps required for its solution, grows faster than the number of cities raised to any finite power.

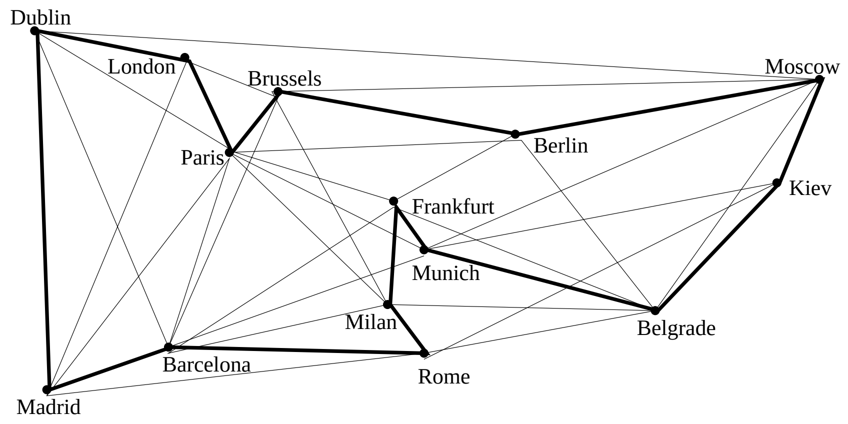


Figure 1.1: Traveling Salesman Problem: finding the shortest route that goes through a given number of cities exactly once. For a problem with fourteen cities, like the displayed one, there are approximately 10^{11} different routes. Ants use a methodology of laying and following trails of pheromone that can be used to find near-optimal solutions for the TSP very quickly.

The TSP is a classical and good example for applying swarm intelligence, but it is by far not the only one. Swarm intelligence has inspired researchers to create algorithms for a great variety of combinatorial optimization problems, as for instance, the quadratic assignment problem [29, 23], the job shop scheduling problem [14], the graph coloring problem [15] or the vehicle routing problem [8]. In the field of telecommunications networks AntNet [18] is a swarm intelligent approach to dynamic routing. A set of ants collectively solves the problem by indirectly exchanging information through the network nodes. They deposit information used to build probabilistic routing tables. At the time it was published it outperformed state-of-the-art algorithms for an extensive set of simulated situations. In robotics the recruitment of nestmates in some ant species inspired scientists to program robots in order to achieve an efficient teamwork behavior [4].

For our insect-based approach, inspiration is taken from the methodology of division of labor in social insects. In social insect societies, individuals tend to specialize for a certain task. This is believed to be more efficient than sequential task performance as it avoids unnecessary task switching, which costs time and energy. In order to explain this methodology, Bonabeau *et al.* [5] have developed a model of response thresholds, later applied by Campos *et al.* [9] and Cicirello *et al.* [12] for a dynamic scheduling problem, which we define as the dynamic task allocation problem (DTA). The environment for this problem can be compared to a painting facility, where trucks come out of an assembly line and get painted by paint booths. If necessary, paint booths can change their color, but doing so is related to a high cost in time, and eventually in money, as colors might get mixed and therefore cause mistakes. Thus, a specialization of the paint booths is desirable, so that fewer reconfigurations are required. The insect-based approach provides such a specialization. A set of agents, which represent the paint booths, autonomously bids for tasks. The way the solution is achieved is totally distributed, as no global information is used. One agent is not provided any information about the other agents

We have reimplemented the work of Campos *et al.* [9] and Cicirello *et al.* [12] and tested it on a wide set of DTA instances. The main contribution of this master thesis is the detailed analysis of the insect-based approach and improvement proposals in order to overcome some shortcomings of the original system. Mostly, the performance, achieved by the original system, was satisfying. However, especially for problems, where demands are dynamically changing, the system often took very much time to re-adapt or didn't succeed to re-adapt at all. In order to overcome this problem, we propose two additional rules to speed up the adaptation process. Moreover, we propose three modifications on the original approach.

The thesis is structured as follows. In the next chapter, we introduce scheduling problems. After describing a general model for deterministic scheduling, we specify the main objectives and performance measures, and show some common problem examples. Then, based on this information, the DTA is defined and discussed in more detail. Chapter 3 provides a brief overview of multi-agent systems (MAS). First, we define the terms agent and MAS and describe some areas where they are applied. Afterwards, market-based MAS are explained in more detail. Market-based MAS can be deployed for distributed scheduling problems. Therefore we present a simple market-based approach to the DTA that serves as comparison to the insect-based approach. The methodology of division of labor in social insects is described in Chapter 4. We start by giving an overview of the different forms of division of labor, and then describe the terms stigmergy – which plays a major role for the division of labor – and plasticity – which is one of the main qualities of division of labor. As mentioned before, in order to explain the methodology of division of labor, Bonabeau *et al.* [5] have developed a model of response thresholds. This threshold model is explained in detail, first for fixed thresholds, and then for dynamically changing thresholds. The insect-based approach to the DTA is introduced in Chapter 5. We give a formal description of the system and its rules. Chapter 6 is dedicated to our improvement proposals. The motivations are declared and examples of situations where we expect our proposals to be helpful are given. In Chapter 7 we briefly describe the examined problem instances and the way we determined the values for the set of parameters. The experimental results are provided in Chapter 8. First, we present the analysis of small problem instances, where a detailed study of single entities of the problem is possible. Then, we present large problem instances, that are meant to represent more realistic factory environments. In Chapter 9 we summarize the work and our results, and give a brief overview on future work.

Chapter 2

Scheduling Problems

The problem of scheduling applies to many important computing, business and everyday problems. In very general terms, scheduling problems assume a set of resources and a set of tasks which have to be allocated to the resources. Based on certain constraints and rules on the tasks and resources, the goal is to optimize or try to optimize a desired objective function.

In this chapter we will describe approaches to several forms of scheduling. First, deterministic scheduling will be examined, giving a formal description of the system's entities, introducing the most widely used performance measures and presenting some examples of scheduling problems. At the end of the chapter, we give a definition of the dynamic task allocation problem (DTA), which is the problem considered in this thesis.

2.1 Deterministic Scheduling Problems

Deterministic scheduling assumes that all information governing the schedule is known in advance. This includes for instance, task arrival and processing times or the availability of resources.

2.1.1 General Model

The general model of deterministic scheduling problems describes resources, task systems and performance measures dealt within deterministic scheduling.

We define a set of **resources** $\mathcal{R} = \{R_1, \ldots, R_m\}$. Resources generally provide a certain utility. For instance, a resource can be thought of as a computer, a machine in a factory, a server, or bandwidth in the internet. Depending on the problem, these resources can be identical, different either in functional capability or speed or different in both.

A general **task system** for a given set of resources can be defined as a five-tuple $\{\mathcal{T}, \prec, [\tau_{ij}], \{\mathcal{R}_j\}, [w_j]\}$ as follows:

1. $\mathcal{T} = \{T_1, \ldots, T_n\}$ is a set of tasks to be executed.
2. $\prec$ is a partial order relation defined on $\mathcal{T}$ which determines the constraints on the task succession. For instance $T_i \prec T_j$ signifies that T_i needs to be finished before T_j can begin.
3. $[\tau_{ij}]$ is an $m \times n$ matrix of execution times. The entry τ_{ij} is the time required by resource $R_i, 1 \leq i \leq m$, to execute task $T_j, 1 \leq j \leq n$. In the case of identical resources the vector $[\tau_j]$ sufficiently denotes the execution times of the tasks common to each resource.
4. $\{\mathcal{R}_j\} = [R_1(T_j), \ldots, R_m(T_j)], 1 \leq j \leq n$, assigns the value 1 to the ith component, in case resource R_i is required for execution of task T_j, and the value 0 otherwise.
5. The weights $w_j, 1 \leq j \leq n$ can be thought of as priorities or costs related to tasks. For instance, task T_j could be assigned a higher weight for having a higher priority.

This general model covers a reasonably large number of scheduling problems, and can be easily extended in order to cover others.

Figure 2.1 shows an example of a directed, acyclic graph to represent a simple task system with ten different tasks. In this example T_1, T_2, T_3 and T_{10} are so called *initial vertices* forming entry points and T_8, T_9 and T_{10} are *terminal vertices* forming terminal points of the task system. The *level* of a vertex T is the sum of execution times along a path from T to a terminal vertex such that this sum is maximal. A path is called a *critical path* if the path's first vertex is at the highest level. In the example of figure 2.1 the critical paths are T_2, T_5, T_8 and T_2, T_4, T_7, T_9 as these paths require most time in the system to be executed.

2.1.2 Performance Measures

The performance measure applied, usually depends on the aims of the considered scheduling problem. Two performance measures, the *makespan*[1] and the *mean weighted finishing* (or *flow*) time, are probably utilized more often than other, more specific measures. The makespan $f_{max}(S)$ is the time required by the schedule S to finish all tasks:

$$f_{max}(S) = \max_{1 \leq j \leq n} \{f_j(S)\},$$

[1]Also known as the *schedule-length* or the *maximum finishing* (or *flow*) time.

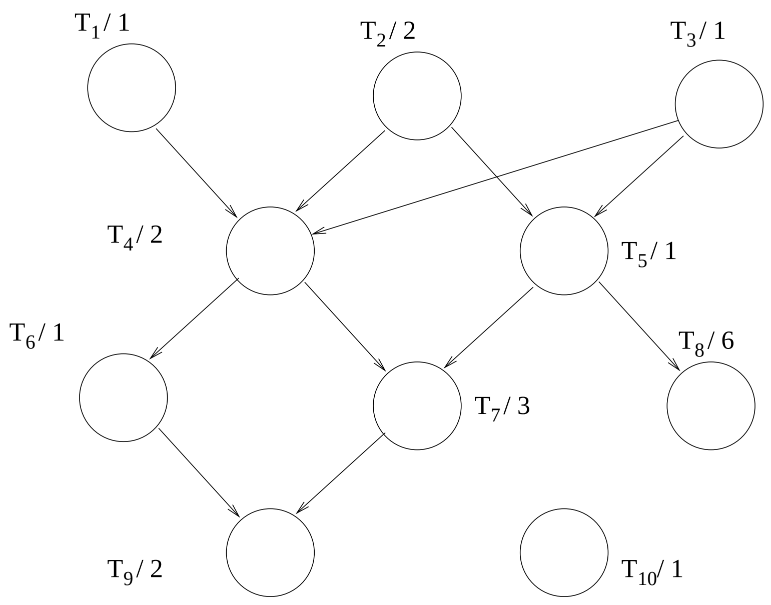


Figure 2.1: Graphical representation of a simple task system with ten different tasks and identical resources. The notation T_j/τ_j signifies the number of the task T_j and its execution time τ_j, respectively. The directed arcs display the partial order in which tasks must be executed. For instance, the arcs directed from the tasks T_1, T_2 and T_3 to task T_4 express that T_4 may only start execution after the execution of T_1, T_2 and T_3 is finished. Task T_{10} is not connected to any other task, because T_{10} does not need to follow any constraints on the task succession.

where $f_j(S)$ denotes the finishing time of T_j when the schedule S is applied. On the other side, the mean weighted flow time represents the average time spent by a task in the system:

$$\overline{f(S)} = \frac{1}{n}\sum_{j=1}^{n} w_j f_j(S),$$

where w_j denotes the weight. And if tasks are created during the schedule execution:

$$\overline{f(S)} = \frac{1}{n}\sum_{j=1}^{n} w_j (f_j(S) - a_j),$$

where a_j the arrival time of T_j.

Other performance measures of broad interest are the *mean number of tasks in the system* and the *maximum* and *mean tardiness*. The mean number of tasks $\overline{N}(S)$ in the system over an interval $[0, \omega(S)]$ is useful in computer scheduling to express expected inventory or storage requirements for tasks:

$$\overline{N}(S) = \frac{1}{i(S)} \int_0^{\omega(S)} N(t)dt,$$

with $N(t)$ being the number of uncompleted tasks in the system at time t.

In order to define the mean and maximum tardiness performance measures we first need to extend the general model of Section 2.1.1 by assigning to each task T_j a positive number d_j, called the *due date*. The due date expresses the time at which the task is desired to be finished with execution. If the due dates *must* be respected, they are also called *deadlines*. Given the definition of the due date, we can define the *tardiness* κ_j of task T_j as $\max\{0, f_j(S) - d_j\}$ if schedule S is applied:

$$\kappa_{max} = \max_{1 \leq j \leq n} \big\{ \max\{0, f_j(S) - d_j\} \big\},$$

and for the mean tardiness:

$$\overline{\kappa} = \frac{1}{n} \sum_{j=1}^{n} \big\{ \max\{0, f_j(S) - d_j\} \big\}.$$

In general, the optimization of more than one performance measure may be eligible in a scheduling problem. For instance, from the point of view of a client in the internet the minimization of the mean flow time might be desirable in order to have short waiting times. On the other side, from the server's point of view the minimization of the makespan may be preferable, desiring maximum throughput.

2.1.3 Problem Examples

Scheduling problems can be distinguished by a multiplicity of attributes. First, a distinction between *preemptive* and *nonpreemptive* scheduling can be made. Nonpreemptive scheduling makes the restriction that a task may not be interrupted once it has begun execution. In other words, the task must be allowed to run until its execution is finished. For instance, for factory applications of scheduling, this limitation is usually required. Preemptive scheduling on the other hand permits a task to be interrupted and that its execution can be continued at a later moment. Mostly the assumption is made that there is no loss of execution time when a task is interrupted. An example of preemptive scheduling is the execution of a program on a processor. Figure 2.2 shows a simple task system with six tasks to be executed on two identical resources and the optimal solutions for preemptive and nonpreemptive scheduling.

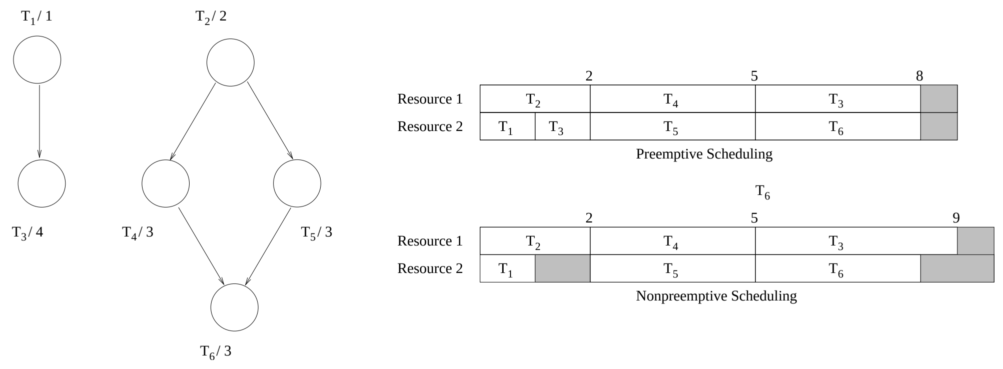


Figure 2.2: A simple task system with six tasks and two identical resources. The above optimal solution for preemptive scheduling interrupts T_3 at resource 2 and continues later at resource 1. For nonpreemptive scheduling execution interrupts are illegal. Grey shaded areas signify an idle resource, i.e., a resource not busy with any task.

One of the best known and most studied fields in deterministic scheduling problems is the area of Shop Scheduling problems, especially the Job Shop Scheduling problem (JSP) and the Open Shop Scheduling Problem (OSP) forming sub-instances of the more general Group Shop Scheduling problem (GSP) [1]. The terminology of GSP, OSP and JSP uses the expressions job instead of task and machine instead of resource, as they were inspired by factory-like problems. However, we keep using the terminology as in the precedent sections in order not to confuse the reader. To explain the GSP, the general model of Section 2.1.1 needs to be enhanced by three attributes. First, tasks are split into operations , i.e., for each task T_i there is a set of operations $\{o_{j1}, \ldots, o_{jh}\}$ that needs to be executed. Second, operations within a task T_i may form several groups $\{g_{j1}, \ldots, g_{jk}\}$, with $|g_{ji}| \geq 0$, $|g_{ji}|$ denoting the number of operations in the group g_{ji}. Between groups within the same task there is a strict partial order to be followed, whereas within a group the order of operation execution is arbitrary. So, if there are two groups g_{1i} and g_{1j} within the same task T_1 and the partial order $g_{1i} \prec g_{1j}$ is given, all operations contained in g_{1i} must be finished before execution of g_{1j} may begin. Operations in different tasks may share the same resource to be executed on. Operations sharing the same machine may not be executed in parallel, i.e. at the same time. Now that the GSP is defined it is easy to define the JSP and the OSP as they are special cases of the GSP. A JSP simply is a GSP with $|g_{ji}| = 1$ for all groups. This means that every operation forms a group. So, there are in fact no groups and a partial order must be defined for all operations within a task. A OSP on the other hand

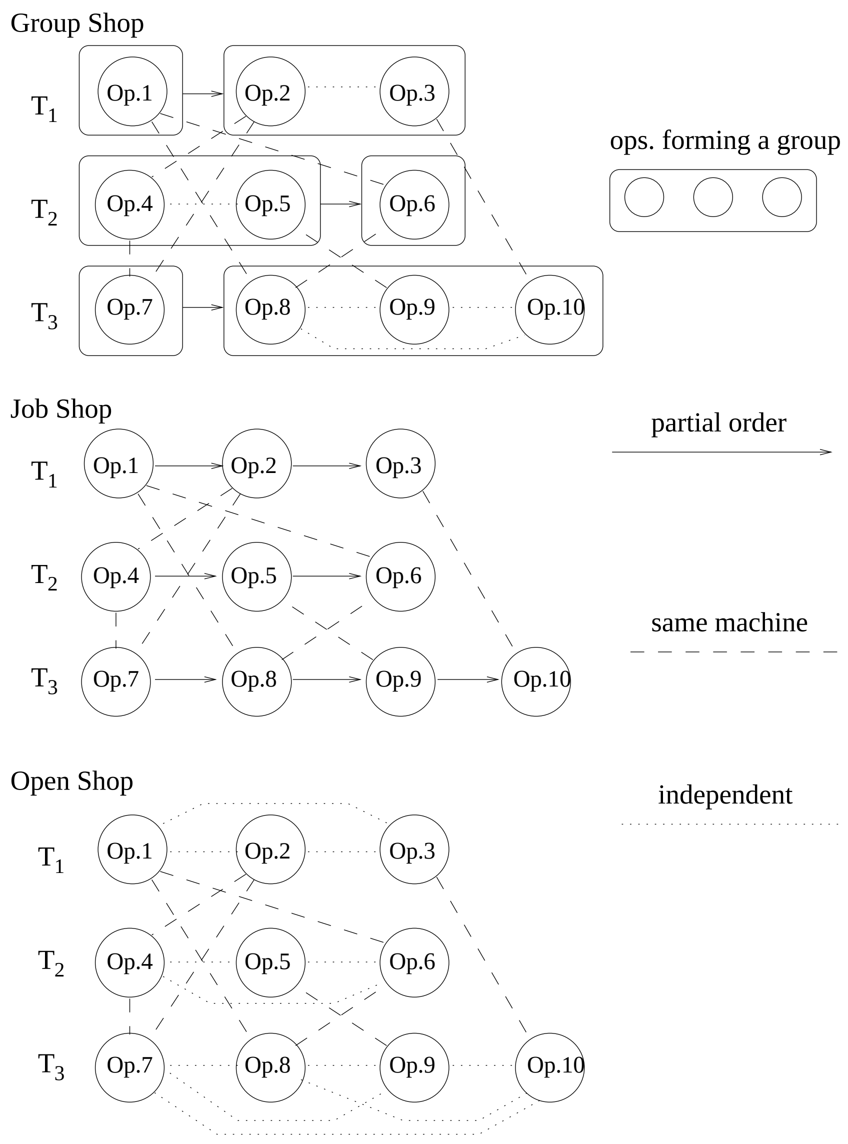


Figure 2.3: Graphical representation of a GSP, JSP and OSP problem with three tasks and ten operations. Execution times are omitted. Operations in the same box share the same group. "same machine" means that operations connected by the dashed line require the same machine. "independent" means that operations within one task connected with the dotted line can be executed without following any sequencing constraints. A similar example can be found in [2].

is a GSP with $|g_{ji}| = |T_i| \, \forall \, tasks$.[2] Therefore, a task is a group containing

[2]$|T_i|$ denotes the number of operations in T_i.

all the operations of the task. Thus, operation execution within a task does not need to follow any constraints on the order of task succession. Figure 2.3 shows a graphical representation of a simple problem example for the GSP, the OSP and the JSP with three tasks and ten operations.

2.2 The Dynamic Task Allocation Problem (DTA)

The dynamic task allocation problem (DTA) was previously introduced in a similar way by Campos *et al.* ([9]) and Cicirello *et al.* ([12]). The DTA task allocation problem is said to be *dynamic*, as during problem execution the environment may change.

Like all scheduling problems, the DTA consists of a set of resources and a set of tasks. The constraints and rules within the problem are chosen in order to simulate a factory-like environment. A real-world problem that expresses very well the characteristics of the DTA is given by the problem of paint booths painting trucks (see Figure2.4. This problem consists of a facility containing trucks and paint booths. Trucks roll off an assembly line at a certain rate in order to get painted by the paint booths. A truck's color is predetermined. Crucial to this problem is the time required by paint booths to reconfigure, that is, to change the color in which they paint the trucks. Such a reconfiguration is related to a high cost in time. Additionally, a reconfiguration can cause a failure and by that be related to a monetary cost as well. For instance a paint booth reconfiguring from black to white has the danger of mixing the two colors while reconfiguring and thus painting the next truck, that ought to be white, in grey. The number of required reconfigurations should be held as low as possible.

2.2.1 Definition of the DTA

Based on the preceding example and the general definition of scheduling problems from Section 2.1.1 we formally define the DTA as follows:

- $\mathcal{M} = \{M_1, \ldots, M_m\}$ is a set of resources or machines. Each machine has a queue which can be filled with tasks. Tasks must be processed in a first-in-first-out order. For a machine M_i a number $q_{max,i}$ gives the maximum number of tasks that may be held in its queue.

- $\mathcal{T} = \{T_1, \ldots, T_n\}$ is a set of tasks to be executed by the machines. Tasks are generated throughout a time-window $[0, \tau_{tot}]$ and released at discrete times. The set of tasks $\mathcal{T}$ has to be considered as a "prototype"

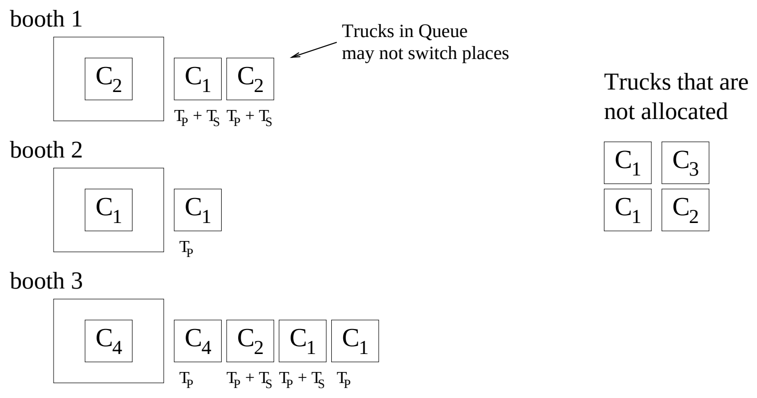


Figure 2.4: An example for a DTA problem with three paint booth painting trucks. Each paint booth has a queue of trucks. The trucks are represented by quadrangles with C_j signifying the color j they require. T_p and T_s are the processing and the setup times. Once a truck is allocated to a booth, the order is fixed. For instance, booth 1 will require a reconfiguration for both trucks in its queue.

set, i.e. a task $T_j, 0 \leq j \leq n$, within $\mathcal{T}$ is not necessarily unique. How many times a task of the same type is released to the environment depends on the problem instance. It might be more precise to use the term "task of type j" rather than "task T_j".

- An assignment of a task of type j to machine M_i implies a cost, consisting of the process time $\tau_{ij,def}$ and the setup time $\omega_{ij,s}$.

- $[\tau_{ij}]$ is an $m \times n$ matrix of average process times. The entry τ_{ij} is the average time required by machine $M_i, 1 \leq i \leq m$ to process a task of type $j, 1 \leq j \leq n$, in case machine M_i is able to process tasks of type j. Otherwise the entry τ_{ij} is 0. The definite process time $\tau_{ij,def}$ is equally distributed in $[\tau_{ij} - \Delta\tau, \tau_{ij} + \Delta\tau]$. The definite process time $\tau_{ij,def}$ does not necessarily equal the total execution time as a reconfiguration of a machine might require additional time.

- $[\omega_{ij,s}]$ is an $m \times n \times 2$ matrix of setup times. For the entry $\omega_{ij,s}$, the index s refers to the state of machine M_i at the time $t_{current}$ it is allocated a task of type j. Two states – depending on the type *preceding_type* of the last task in M_i's queue at time $t_{current}$ – are be distinguished:

  1. type $j =$ *preceding_type* $\Rightarrow$ no setup is required. The machine's state is $s = 0$ and the setup time is $\omega_{ij,s} = 0$, $(\Rightarrow \omega_{ij,0} = 0, ;\forall;i, 1 \leq i \leq m, \forall j, \leq j \leq n)$.

2. type $j \neq preceding\_type \Rightarrow$ a setup is required. The machine's state is $s = 1$ and the setup time is $\omega_{ij,s} > 0$.

2.2.2 Discussion of the DTA

The definition of the DTA covers a multiplicity of problem instances that exceed the range of problems approached by the insect-based system. We keep the problem definition as general as possible in order to hold a connection to other general scheduling models.

For instance we don't determine whether the DTA is a deterministic problem or not, i.e., whether all information governing the scheduling decision is known in advance. But for the experimentally considered problem instances (see Chapter 7) we assume that task arrival times are not known a priori. Thus, the examined instances of the DTA are non-deterministic problems.

Additionally, a task generation process is not specified. It may be that the distance between two task arrivals is exponentially distributed, or that each task has a certain probability to appear at each discrete time step[3] or even the extreme case of all tasks being created at time step 0. This last example would make the DTA instance a very simple scheduling problem requiring only a short algorithm to optimize a schedule. On the other hand, if task arrival times are exponentially distributed, a much more complex algorithm would be required, even if task arrival times were known in advance.

Something else we want to mention is our definition of the set of tasks $\mathcal{T}$ for the DTA, which we refer to as a "prototype" set. Often scheduling problems assume a set of tasks with each task being unique. But in real-time computing systems for instance, where scheduling plays a major role in predicting the system's ability to satisfy deadlines, a system's simulation is mostly assuming periodically appearing tasks [21]. In the definition of the DTA we do not assess whether a certain type of task appears one or more times during the time window $[0, \tau_{tot}]$. Though, for the DTA instances we examined, a type of task usually appears frequently.

A last thing to notify is the goal that we want to achieve, the performance measure which we want to use to describe the quality of a schedule. The main performance measure we make use of is the required makespan to finish execution of all tasks. But we also analyze a schedule's quality or more general, the behavior of our approach, with respect to a variety of other measures, as for example the amount of tasks finished during a given window of time (the total throughput), the number of required setups or the total

[3]We use the term *time step* to describe the discretized temporal environment. The generation of tasks begins at time step 0 and finishes at time step τ_{tot}.

time that machines stay idle during the schedule's length. All these measures will be explained in more detail in and in Chapter 8.

Chapter 3

Multi Agent Systems (MAS)

Our approach to the DTA may be viewed as a *multi agent systems* (MAS) inspired by the methodology of division of labor in social insects. Therefore, in this chapter we introduce MAS. The use of MAS in a variety of fields of computer science, engineering and artificial intelligence is increasing rapidly in the last five to ten years. The study of multi-agent systems focuses on systems in which many intelligent agents interact with each other. The agents are considered to be autonomous entities, such as software programs or robots. One reason for the success of MAS in recent years is the growing need for distributed intelligence. Distributed intelligence can be helpful for problems:

- ...that are physically distributed: for instance transportation networks or vehicle traffic management.
- ...that are functionally distributed: problems may be split into several subproblems. For instance, building a car requires several specialist groups for the design, the engine, the tyres and other specific tasks that are not functionally connected.
- ...that are too complex too keep a global point of view: If problems are too large to be analyzed as a whole, solutions based on a local viewpoint may allow the problem to be solved more quickly and more easily.

In the ongoing of this chapter, we first give a general definition of agents and MASs. Afterwards, different types of agents and areas of application are presented. At the end of the chapter, market-based MAS are introduced in more detail. We focus on market-based approaches to distributed scheduling problems, where tasks are represented by autonomous agents.

3.1 Agents

As agents are applied in a wide area, many different uses of the term agent can be found. Thus, it is difficult to give a commonly accepted notion of what it is that defines an agent or a MAS. Wooldridge and Jennings [49] describe agents by giving them the following characteristics:

- *autonomy* or the ability to function without intervention,
- *social ability* by which agents interact with other agents,
- *reactivity* allowing agents to perceive and respond to a changing environment and
- *pro-activeness* through which agents behave in a goal-directed way.

There are two major ways to differ between agents: The abilities of an individual agent, and an agent's behavior towards its environment [22].

In the first classification, we differentiate whether an agent is intelligent (or *cognitive*), i.e., capable of solving certain problems by itself, or whether an agent is very simple. A cognitive agent has enough knowledge to carry out its task(s) and handle interactions with other agents and with its environment. Cognitive agents have a representation of their world. Based on this representation they are generally capable of memorizing situations, analyzing them, foreseeing possible consequences of their actions, and thereby planning their behavior. In the other case, where individual agents are more simple, they are called *reactive*. Reactive agents do not possess a representation of their environment. Thus, they are incapable of foreseeing future events and planning their behavior. The basic idea behind the use of a reactive agents is that a MAS can demonstrate intelligent behavior even if agents are not individually intelligent [17]. A very good example for this is a social insect colony (see Section 4.1). Individual insects may be very simple and unable to fulfill complex tasks. Nonetheless, within their colony, insects are able to organize themselves and collectively find solutions for difficult problems like for instance finding the closest food source.

The second classification distinguishes the source of motivation that leads an agent. This may either be explicitly expressed within the agent – in this case the agent follows *teleonomic* behavior – or the agent only responds to stimuli from the environment and thus follows *reflex* behavior.

These two classifications can be combined to distinguish four types of agents as it is summarized in Table 3.1. Usually cognitive agents are intentional, i.e., they have explicit goals which motivate their actions, but the

	Cognitive agents	**Reactive agents**
Teleonomic behavior	Intentional agents	Drive-based agents
Reflex behavior	Module-based agents	Tropistic agents

Table 3.1: The table shows the different types of agents. A distinction is made on the individual abilities (horizontal) of an agent and its behavior towards its environment (vertical)

combination of a cognitive agent with reflex behavior is possible. For instance, this is applied for auxiliary agents who accomplish tasks which they are commanded to perform by other agents, without having explicit goals within themselves.

3.2 MAS

A MAS can simply be described as a loosely coupled network of agents, interacting to collectively solve problems which are beyond the individual capabilities of each agent [20]. Interactions among agents can be either cooperative or selfish. Cooperative agents share a common goal, as for instance in an ant colony, whereas selfish agents pursue their own interests, as for instance in the free market economy. Sycara [39] additionally points out, that individual agents within a MAS possess incomplete, local information only, and that there is no global control in a MAS.

MASs can be differed by their *organization*, that is, a framework for agent interactions that defines roles, behavior and authority relations among the agents. The roles describe the agents' functions, i.e., the position of the agents within an organization, and the set of activities that they are supposed to carry out to achieve the organization's objectives. For instance, an agent that manages execution requests and distributes them to competent agents is called *mediator*. A *planner* is an agent that determines the actions to be undertaken. A *supplier* performs a service for another agent who is called *customer*. And an agent whose main function is to execute, is called *executive*. Agents are not restricted to keep one fixed role in the system. Roles can evolve with the dynamics of the organization.

Examples of organizations for MASs that have been explored are:

- **Hierarchy:** The MAS is split into several levels. Within each level, one agent or a small group of agents is given the authority for decision making and control. Interaction takes place through vertical communication in both directions. Superior agents are given the control over resources and decision making

- **Community of experts:** This organization is flat. Each agent is a specialist in a particular area. Agents interact by rules of order and behavior [28]. Agents coordinate their solutions through mutual adjustment in order to achieve overall coherence.

- **Market:** Control is distributed to the agents that compete for tasks or resources through a bidding mechanism. Agents interact through one variable, the price, which is used to value services [37]. Agents coordinate through mutual adjustment of prices. A detailed description of market-based MASs is given in Section 3.4.

3.3 Areas of Application

There are many areas of application for MASs. In this section we present two main categories: problem solving in the broadest sense, and modeling and simulations.

3.3.1 Problem Solving

Problem solving concerns all situations, where software agents accomplish tasks which are of some use. This category can be contrasted with the applications of robotics, as agents are purely computing agents without any real physical structure. We distinguish *distributed solving of problems*, where a complex task is carried out by an assembly of specialists possessing complementary skills, and *solving of distributed problems*, where the area in question is itself distributed.

Distributed Solving of Problems

In this case the solving method is distributed, while the area of operations is not. When a problem is so complex that one person cannot possess the expertise to solve it alone, it is necessary to split the task into less difficult subtasks.

For instance, when several specialists are required to build a racing car, one will have a particularly good knowledge about engines, another will handle the tyres, a third will concentrate on the interface for the driver and so on. These specialists cooperate with each other to solve the general problem.

One particular way of distributed solving of problems has been proposed by C. Iffenecker [27]. His Condor system is an aid to the designing of electromechanical products. Specialists in specifications, design, assembly, materials, planning, marketing and many other areas, are represented in the form of an assembly of cognitive agents in the Condor system. All these agents have their own expertise and intervene at different stages in the creation of the product.

In the area of paint shop systems, Morley [32] implemented a market-based MAS at the General Motors' Fort Wayne truck assembly plant. The problem was described in Section 2.2 when we introduced the DTA. In the system implemented by Morley, agents are in charge of humidifiers, burners, and the steam generators for the air supply system. In Section 3.4.4 we will give a simple model where an agent represents a paint booth, and in Chapter 8 we compare this approach to the insect-based approach.

Solving of Distributed Problems

The solving of distributed problems refers essentially to applications such as analysis, identification, fault finding and the control of physically distributed systems for which it is difficult to obtain a totally centralized overall view. For instance, if the task is to control a communications network, the domain which is represented by the network itself, constitutes a distributed system. This system needs to be monitored, and the monitoring tasks should be decentralized within the nodes of the network as much as possible. Distributed conception constitutes a good example of the solving of distributed problems.

The Ideal system [22], designed by Onera and Alcatel-Alsthom, is a characteristic example of a MAS monitoring and fault finding in a telecommunications network. It contains three sorts of agents: supervisors, with the task of locating failures and finding faults, follow-up agents, which have to maintain coherence between the real state of the network and the agents' view of it, and maintenance operators with the task of carrying out tests and repairing elements of the network.

3.3.2 Modeling and Simulation

Simulation consists of analyzing the properties of theoretical models of the surrounding world. Natural sciences as well as social sciences make particularly frequent use of simulations to try to explain or forecast natural phenomena. Therefore, researchers construct models of reality and then test their validity by running them on computers.

One of the most famous examples used in ecology, is the mathematical model of the dynamics of populations by Volterra *et al.* [42], which describes the growth rates for populations of prey animals and predator occupying the same territory:

$$\frac{dN_1}{dt} = r_1 N_1 - P N_1 N_2, \quad \frac{dN_2}{dt} = aPN_1N_2 - d_2N_2, \tag{3.1}$$

where P is the coefficient of predation, N_1 and N_2 are the numbers of prey animals and predators, a is the efficiency with which the predators convert food into offspring, r_1 determines the fertility of the prey animals and d_2 is the mortality rate of the predators.

Although, in the meanwhile numerous advances have been introduced to these models, they exhibit certain problems. For instance, it is not possible to link the size of a population to the decisions taken by individuals. Behaviors executed at the 'micro' level correspond with the global variables measured at the 'macro' level. In general, for such mathematical models, it is very difficult to take the actions of individuals into account. Another problem concerns the parameters. Considered in terms of usability and realism, the applied equations often contain a large number of parameters which are difficult to estimate. For example, in Equation 3.1 the coefficient a indicates the efficiency of converting food into offspring. This appears to be oversimplified as more complex behaviors like sexual strategies or use of territory are not taken into consideration.

MASs bring a new solution to the concept of modeling and simulating. They offer the possibility of directly representing individuals, their behavior and their interactions. For example, in a multi-agent population model, individuals will be directly represented by agents, and the number of individuals in a given species will be the result of the confrontations (cooperation, struggle, reproduction,...) of all the individuals represented in the system. The main qualities of multi-agent modeling are its capacity for integration and its flexibility. It is possible to integrate within the same model quantitative variables, differential equations and individual behavior. In addition, it is possible to add new types of agents with their own model of behavior.

The Simdelta simulator [7] is a good example for MAS modeling. It has been used to summarize the knowledge of several specialists who have spent many years studying the fisheries of the central Niger delta in Mali. This simulator makes it possible to simulate both, the dynamics of the fish population – taking into account numerous biological and topological factors which may affect its evolution – and the decision making of the fishermen. The technique employed utilizes three types of agents: the biotopes, which represent portions of the environment, the fish, which exhibit behavior that can be considered to be reactive, and the fishermen, who behave like cognitive agents.

Two series of experiments were carried out. The first is related to the study of the population dynamics of the fish in dependency to the continuous growth of fishing. The second series was intended to model the fishermen. This model demonstrated the importance of the decision-making mechanism in relation to the dynamics of the fish.

3.4 Market-Based MAS for Scheduling

In this section we give a more detailed introduction to market-based MASs, as they are applied for scheduling problems. In the approach that we present, autonomous agents are in charge of tasks and make *bids* for resources. In order to allocate resources to tasks, and thereby solve a scheduling problem, *bidding protocols* are executed and *prices* for resources are determined. Agents make decisions in form of evaluating the trade-offs of acquiring different required resources. These trade-offs are represented in terms of market prices. In general, within a market mechanism for scheduling, agents may also be in charge of resources instead of tasks. However, we limit ourselves to the approach in which agents represent tasks and not resources and present a theoretical framework for market-based mechanisms to distributed scheduling problems.

The idea of markets can provide several advantages to a MAS in general, and decentralized scheduling in particular. The necessary communication, which constitutes a typical bottleneck in decentralized scheduling, to determine a schedule applying a market-based MAS, can be limited to the exchange of bids and prices. Prior work applying market-inspired mechanisms to scheduling has produced promising results [44, 43].

The market-based approach to distributed scheduling problems is comparable to our insect-based approach. In [9], Campos *et al.* focus on the similarities of the two ideas for the dynamic task allocation problem. Both approaches are based on a MAS with bidding agents. In the market system, the agents are in charge of the tasks and make bids in order to receive resources, whereas in the insect-based approach, the agents represent the machines and bid for jobs.

3.4.1 General Model

Wellman *et al.* [45] propose a general resource allocation problem for markets as follows:

- $G = \{g_1, \dots, g_n\}$ is a set of n goods or resources.

- $A = \{a_1, \ldots, a_m\}$ is a set of m agents in charge of tasks. Additionally, there is the *seller* or *null agent* a_0. In general, a market may contain agents who are bidding for resources as well as agents who are selling resources. In this model there is only one agent who is selling goods, the null agent who is not in charge of tasks. All other agents bid for goods held by the null agent.

- A set of prices $p = \{p_1, \ldots, p_n\}$ for the respective goods.

Each good g_i has a *reserve price* q_i, representing the minimum price that the seller of the good is willing to accept. An agent a_j holds three variables characterizing the task he is in charge of. First, the task's *length* λ_j, which is similar to the execution time τ_j from the model of Section 2.1.1. If the condition $\lambda_j = 1, \forall j$ is satisfied, the scheduling problem is called *single unit*, otherwise it is called *multiple unit*. The second variable is the task's *deadline* d_j, having a similar definition to the one delivered in Section 2.1.2. And finally, the task's *value* $v_j(X(\lambda_j, d_j))$, expressing how high the set of goods $X(\lambda_j, d_j) \subset G$ is evaluated by a_j. $X(\lambda_j, d_j)$ is any set of goods that satisfy the requirements given by the length λ_j and the deadline d_j. Additionally, the value also denotes the maximum price that a_j is able[1] to pay for the required set of goods $X(\lambda_j, d_j)$. Reserve prices for goods as well as agents' values are usually counted in units of money (e.g., in \$).

An agent a_j is given a utility u_j:

$$u_j(X(\lambda_j, d_j)) = v_j(X(\lambda_j, d_j)) + M_j,$$

for holding the set of goods $X, X \subseteq G$, and M_j units of money. For the seller a_0, the utility is the sum of the money he received for allocated goods and the reserve prices for unallocated goods.

The *maximum surplus* $H_j(p)$:

$$H_j(p) = \max_{X(\lambda_j, d_j) \subset G} [v_j(X) - \sum_X p_i],$$

denotes the maximum amount of money agent a_j has left if a_j was allocated the set of goods $X(\lambda_j, d_j)$ for prices p. $H_j(p)$ also maximizes an agent's utility. It is worth noting, that for some prices the agent may maximize its surplus with the empty set, as for all valid nonempty sets the value of $H_j(p)$ would be negative. Such a negative value of $H_j(p)$ corresponds to the

[1]Whether the agent is willing to pay a given price for a good, as high as the corresponding value, depends on his strategy, the auction and of course other agents' bidding on the same good.

situation that for the given prices agent a_j cannot afford to buy any set of goods $X(\lambda_j, d_j)$ that satisfies the requirements of his task.

A schedule or solution f is a mapping, that determines which agent is allocated each good:

$$\begin{aligned} &f : G \to A \cup a_0, \\ &F_j = \{i | f(i) = a_j\}, \\ &F_0 = \{i | f(i) = a_0\}, \end{aligned}$$

where F_j represents the set of goods allocated to agent a_j and F_0 denotes the set of unallocated jobs. $f(i) = a_j$ expresses that good i is allocated to agent a_j.

The performance measure to evaluate the quality of a solution is given by its *global value* $v(f)$:

$$v(f) = \sum_{i \in F_0} q_i + \sum_{j=1}^{m} v_j(F_j),$$

where $v_j(F_j)$ is the value, agent a_j denotes to the set of goods F_j he was allocated. If F_j does not satisfy the requirements given by the tasks length λ_j and its deadline d_j, the a_j has no advantage holding F_j. Thus, the value denoted to the set of goods F_j is $v_j(F_j) = 0$. Otherwise, if the task's execution can be guaranteed by the set of goods F_j, then $v_j(F_j) = v_j(X(\lambda_j, d_j)$. The prices paid for the goods don't enter in the global value as it is the allocation of goods that has to be evaluated and not their prices.

Figure 3.1 visualizes the introduced expressions for a simple problem containing three agents in charge of one task each.

3.4.2 Price Equilibrium

A definition for a *price equilibrium* as given in [45] is: A solution f is in equilibrium at prices p iff

1. For all agents $a_j, v_j(F_j) - \sum_{i \in F_j} p_i = H_j(p)$.

2. For all $i, p_i \geq q_i$.

3. For all $i \in F_0, p_i = q_i$.

Basically a *price equilibrium* describes a state, where each agent maximizes its utility and surplus, given the current prices and allocation of goods. That means, for the given prices no agent would want to change his set of goods with any other set of goods as this would not improve the agent's situations with respect of the utility- and surplus-value

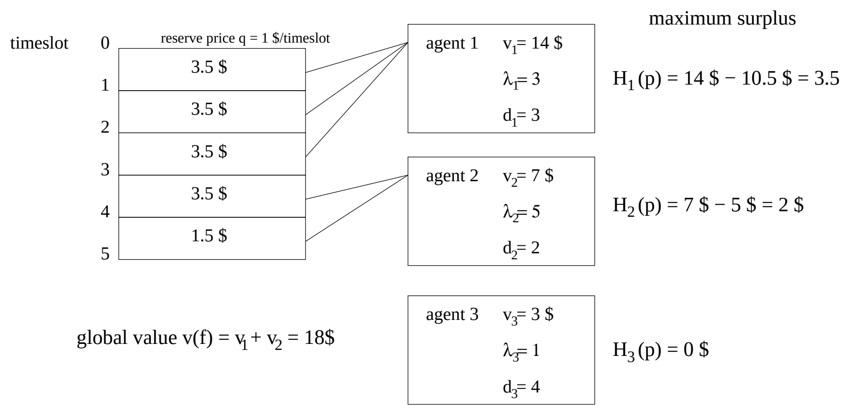

maximum surplus
timeslot
reserve price q = 1 $/timeslot
0
1
2
3
4
5
3.5 $
3.5 $
3.5 $
3.5 $
1.5 $
agent 1 v1= 14 $ λ1= 3 d1= 3
agent 2 v2= 7 $ λ2= 5 d2= 2
agent 3 v3= 3 $ λ3= 1 d3= 4
H1 (p) = 14 $ − 10.5 $ = 3.5
H2 (p) = 7 $ − 5 $ = 2 $
H3 (p) = 0 $
global value v(f) = v1 + v2 = 18$

Figure 3.1: A simple problem with three agents and five time slots as goods. The reserve price q_i is the same for every time slot. A connection between an agent and a time slot denotes that the time slot is allocated to the agent. The solution f optimizes its global value $v(f)$.

Equilibria do not necessarily exist. On the other side, an equilibrium is in general not unique. As for instance in the example of Figure 3.1, more than one equilibrium may exist for the same problem. The illustrated solution would also be in equilibrium with all prices being 25 Cents lower or higher. A solution in equilibrium is always optimal in terms of maximizing the solution's global value[2]. But obviously an optimal solution is not necessarily in equilibrium. Figure 3.2 illustrates a problem with no equilibrium at any price. If for this example the prices were in equilibrium, then $p_i \geq 3\$, \forall i$ would be necessary. Otherwise, if prices are lower than 3, agent a_2 would demand one of the time slots in order to maximize his utility. Agent a_1's value is not high enough to buy all time slots for more than 3$. Thus, the optimal solution which allocates all three time slots to a_1 is not in equilibrium. Here the nonexistence of equilibrium emerges from *complementarities* in agent preferences. This means, agent a_1 considers the two time slots complementary in the sense that it values one only if it has the other. A single complementarity is sufficient to prevent a price equilibrium. Such complementarities cannot appear in single-unit scheduling problems. For single-unit scheduling problems there always exists a price equilibrium that is unique[3]. Thus, a single-unit scheduling problem that is optimal, is supported by a price equilibrium.

[2]For a proof see [46].
[3]For a proof see [38]

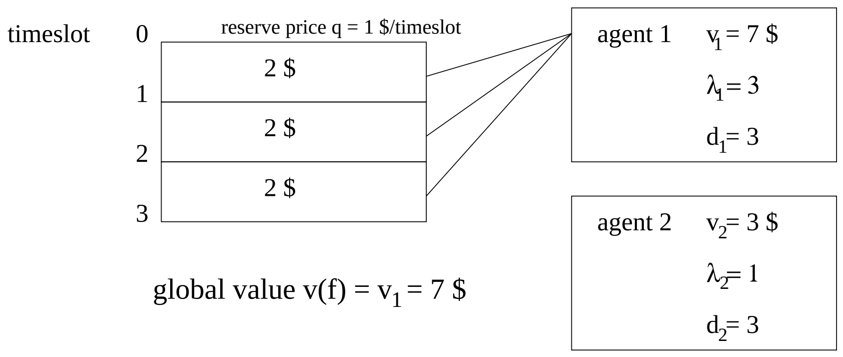


Figure 3.2: A problem with two agents and three time slots as example for a problem that cannot be in equilibrium at any prices. The illustrated solution maximizes the global value but does not deliver an equilibrium as agent a_2 would maximize its surplus by buying one of the three time slots. The reserve price q_i is the same for every time slot.

3.4.3 The Ascending Auction Protocol

Auctions provide efficient and distributed ways of allocating goods and tasks among agents. The term auction protocol refers to a mechanism containing a set of rules in order to determine a solution to the given problem, along with agent bidding strategies. In [30] the following definition for auctions is provided:

"An auction is a market institution with an explicit set of rules determining resource allocation and prices on the basis of bids from the market participants."

A general auction protocol can be split into three steps. First, agents send bids to the central instance of the mechanism to indicate their interest in exchanging goods. Then, the auction posts price quotes to inform the agent about the current state of the prices. These first two steps may be iterated until an allocation is determined by the auction. If an allocation was determined, the auction notifies the agents about which agents are allocated goods and the prices that have to be paid by these agents. These three steps may be performed once or repeated any number of times until a given condition – like for example a minimum global value for the solution – is satisfied.

Parameters to differ among auctions are for instance the price determination algorithm (prices may be fixed or variable with or without certain restrictions), the event timing (one or more auctions could be open at the same time), bid restrictions (agents may be restricted to make no more than one or a certain number of bids) or whether one or more goods are allocated through a single auction. In this section the ascending auction with a simple bidding strategy for the agents is described in order to give a general idea of how an allocation can be determined. The ascending auction is decentralized in the sense that agents calculate their own bidding strategy strictly based on local information only.

In the ascending auction, separate auctions determine prices for each good. Agents submit successively higher bids to the auctions, and the auctions report price quotes to all agents. When the bidding stops, the auctions respective goods are allocated to the highest bidder at the price the respective agent bid. If no bids were made the good is passed back to the seller.

The **bidding rules** determine the prices that are denoted to the goods within each auction. The current bid price β_i in the auction i is the highest bid in the auction so far if any bids were made. Otherwise β_i is undefined. An auctions ask price α_i is the minimum price that the auction i is ready to accept. If β_i is not defined, α_i is the reserve value q_i of the good. Otherwise the ask price is denoted $\alpha_i = \beta_i + \epsilon$ with ϵ being a fixed price increment. With these rules, prices do not decrease and the bidding processes termination is guaranteed.

Agents follow a simple **bidding strategy**. When an agent a_j enters a market, it bids the ask prices for the set of goods so as to maximize its surplus H_j based on current ask prices. If more than one set of tasks reaches the same surplus, one of these sets is chosen arbitrarily. In the ongoing of the auction, a_j may lose some of its bids as other agents continue to bid. If this happens, a_j again bids the ask price on the set of goods that maximize its surplus H_j, assuming that it can obtain the goods it currently winning at their bid prices. If no set of goods can provide a_j with a nonnegative surplus, it "drops out" of the auction.

This bidding strategy is quite simple, involving no anticipation of other agents strategies. For the single-unit problem this strategy is sufficient as the agents would not want to change their bids even if they observed the other agents actions. However, if a prediction of the other agents behavior is possible, the performance of another strategy using this information will in general be better.

The ascending auction performs well for single-unit problems. But, as shown by the example of Figure 3.3, to reach an equilibrium cannot be guaranteed, though there always exists one for single-unit problems. In the example, agent a_2 first choses to bid for good 2 as this maximizes his surplus as well as bidding for good 3. Agent a_1 can bid either for good 2 or for good 1. In the example it bids for good 1 what terminates the auction. An equilibrium is not reached as, $p_2 = 3\$ < p_1$, agent 1 would maximize its surplus by demanding good 2 at the final prices.

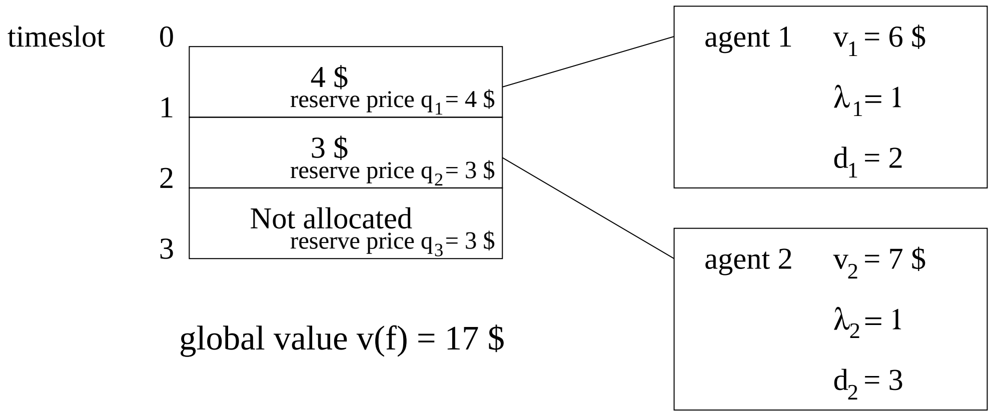


Figure 3.3: A single-unit problem example with three time slots and two agents, where the solution of the ascending auction does not reach an equilibrium.

An upper bound for the distance from the equilibrium price vector can be formulated by $\kappa\epsilon$, where $\kappa = \min(n, m)$ is the minimum of the number of goods and the number of agents bidding[4]. For the multiple-unit problem an upper bound like that cannot be formulated and the ascending auction can determine solutions which are arbitrarily far from optimal. Because of these limitations, combinatorial auction mechanisms were proposed for multiple-unit problems [35]. Combinatorial auctions accept bids of combination of goods. One of the problems of combinatorial auctions is the computational complexity as, given n goods there are 2^n different sets of goods.

3.4.4 Market-Based Approach to the DTA

The market-based based approach by Morley [32] is a good example for a very simple multi-agent system to outperform a centralized scheduling system in terms of increased throughput and lower costs for a real world problem. We refer to the problem that consists of paint booths painting trucks and that was already described in Section 2.2. Morley uses a simple bidding mechanism, where agents are in charge of the paint booths, and bid for trucks. They bid according to the booths' queue length and the required color of the last truck in the booths queue. This simple algorithm resulted in a 10% higher efficiency than the previously used centralized scheduler, when it was put into practice in a General Motors facility. The paint booths required half as many reconfigurations and a higher global throughput was reached.

[4]For a proof see [46].

Morley's algorithm is a manufacturing application and because of this, many of its details are kept secret. The described market-based approach is not the original version of the algorithm. Only the basic ideas are known and so we use a probably simplified version of the Morley-bidding-system as it is proposed by Cicirello *et al.* [12] and by Campos *et al.* [9]. Our version, which we refer to as **MORLEY1**, follows three simple rules:

1. If such a paint booth exists, allocate the truck to the paint booth with the smallest queue with space, that will not require a setup to paint the truck, i.e., the last truck in the paint booth's queue requires the same color as this truck.

2. If such a paint booth does not exist, allocate the truck to the paint booth with the smallest queue with space.

3. If no paint booth has any space in its queue, do not allocate the truck.

This simple algorithm makes only feasible allocation if a restriction about the maximum queue length is made. Otherwise, if queues may have arbitrarily long queues, a re-adaptation of the paint booth is impossible, as trucks are strictly allocated to paint booths that do not require a reconfiguration. Another problem arises, if all paint booths have a full queue. If this situation emerges, the first paint booth who finishes painting a truck, will be allocated the truck that is waiting for the longest time. This allocation does not take into consideration whether a reconfiguration is required or not. Therefore, we propose a slightly modified version of **MORLEY1**. **MORLEY1** executes the three rules whenever a truck is released. Our modified version, which we call **MORLEY2**, only allocates trucks to paint booths if at least one paint booth in the system has at most one truck in its queue. This means, trucks eventually have to wait for their allocation. Therefore we assume the possibility to "park" trucks if this is necessary.

The described algorithms, **MORLEY1** and **MORLEY2**, serve as a comparison to our approach for the experimentation described in Chapter 8.

Chapter 4

Division of Labor in Social Insects

In this chapter, we introduce the methodology of division of labor in social insects. This methodology has inspired Bonabeau *et al.* [5] to develop a threshold model, which lays the foundation for the agent based approach to the dynamic task allocation problem treated in this thesis. After presenting an overview over observations of division of labor in social insects, we give a formal description of the threshold model. First, we present the threshold model with fixed thresholds and afterwards we present the threshold model with dynamic thresholds. At the end of the chapter a further analysis of the model is presented, focusing on a discussion of the response function, which is an important element of the threshold model.

4.1 Division of Labor in Social Insects

In the complex system of social insect colonies, an important and widespread characteristic is division of labor, i.e., different activities are often performed simultaneously by specialized individuals. A lot of different tasks, such as foraging, care of the young or nest construction need to be distributed among the colony's workers. This methodology of division of labor is believed to be more efficient than sequential task performance by unspecialized workers, as it avoids unnecessary task switching, which costs time and energy. For example, if an ant worker continuously switches between foraging and taking care of the larvae, the worker will again and again have to relocate the food source as well as the larvae. A higher grade of effectivity can be achieved if the individuals avoid needless task switching.

4.1.1 Forms of Division of Labor

Division of labor can be exhibited in several forms. The most obvious one is the caste system which is applied by most social insects. In most ant species, for instance, three basic female castes are found: the worker, the soldier, and the queen. In general, division of labor is very often exhibited by the three following basic forms:

- *Temporal polyethism:* Individuals of the same age form an *age class* and tend to perform a same set of tasks. It is argued that age classes are rather determined by relative age within the colony than absolute age. This means, age classes performing same sets of tasks within different colonies may have different absolute ages, depending on the age distribution within the respective colony.
- *Worker polymorphism:* Workers can have different morphologies. Workers with different morphology are said to belong to different *morphological* or *physical castes* and tend to perform different tasks. An example for this is the soldier or major caste which is observed in several species of ants.
- *Individual variability:* Workers of the same age class or physical caste may still differ in the frequency and sequence of task performance. Individuals within age classes or physical castes performing same sets of tasks within a given period, are described by *behavioral castes.*

4.1.2 Stigmergy

Social insects exploit a particular form of indirect communication called *stigmergy*, in order to coordinate their activities. Stigmergy is usually based on modifications of the environment. These modifications lead to a positive or negative attraction, or stimulus, to other individuals. For instance ants lay a pheromone trail while foraging. The pheromone trail stimulates other ants to follow it. Another typical example can be observed in several termite species [47] initiating a large structure in an environment with some building material consisting of pellets of soil and excrements without any order. Different phases of building up a structure can be distinguished. At the beginning the workers pass through a state in which their work seems rather uncoordinated. A pellet placed at one position by a worker is often quickly picked up by another worker. After some time, seemingly by chance, some pellets get stuck on top of each other and thereby the behavior of the workers changes very fast. The little cluster of pellets is much more attractive to the termites than single pellets, so that they quickly begin to add more pellets to that cluster. In this way the environment is "cleaned" from single pellets and large structures are built up without any direct communication among the individuals, but only by modifications of the environment.

The term stigmergy was introduced by Grassé [25] (from the Greek *stigma*: sting, and *ergon*: work). Grassé describes the indirect communication that he observed in two species of termites: *Bellicositermes Natalensis* and *Cubitermes*. His original definition of stigmergy was: "Stimulation of workers by the performance they have achieved".

4.1.3 Plasticity

One of the most important characteristics within the methodology of division of labor in social insects is its *plasticity* in colony level, i.e. its ability to quickly adapt on changing conditions. A rigid specialization of the individual workers would lead to a stiff behavior at the colony level. Colonies with thousands of individuals need to adapt to changing conditions very quickly. The division of labor in social insects does not lead to rigidity in the colony with respect to tasks performed by individuals. The individual's specialization is flexible to several internal and external factors such as food availability, climatic conditions or phase of colony development. Individuals are able to adapt very well to changing demands as shown by Wilson [48]. This plasticity is exhibited for example, by ant species from the *Pheidole* genus.

In most species of this genus workers are physically divided into two fractions: The small minors, who fulfill most of the quotidian tasks, and the larger majors, who are responsible for seed milling, abdominal food storage, defense or a combination of these. Wilson [48] experimentally changed the proportion of majors to minors. By diminishing the fraction of minors, majors get engaged in the tasks usually performed by minors and replace them efficiently. Wilson [48] observed, that within one hour of the ratio change, majors adapt themselves to the new situation and take over the minors' work.

4.2 The Response Threshold Model

Inspired by the division of labor in social insects and in order to explain the behavior observed by Wilson [48], Bonabeau *et al.* [5] have developed a model of response thresholds. In [6] these thresholds remain fixed over time whereas in [40] the thresholds are dynamically updated with respect to the task being currently performed. The idea behind the model is very simple: if a stimulus, related to a task, exceeds the response threshold of a certain worker for the task, that worker will get engaged in performing the task. By new workers getting engaged in the task, the intensity of the stimulus is reduced and therefore the probability that other workers get engaged in the same task decreases as well.

4.2.1 Model with Fixed Thresholds

As already mentioned, Bonabeau *et al.* describe in [6] a model of response thresholds where the thresholds remain fixed over time. The model is mainly described by three variables:

- **Intensity of the stimulus** s_i: the stimulus s_i is referred to a particular task i. It can for instance be associated to a chemical concentration of pheromone, the density of pellets or anything else that can serve as an attraction for an individual worker. The higher the intensity of the stimulus s_i, the higher is the attraction towards workers to get engaged in performing the task i.

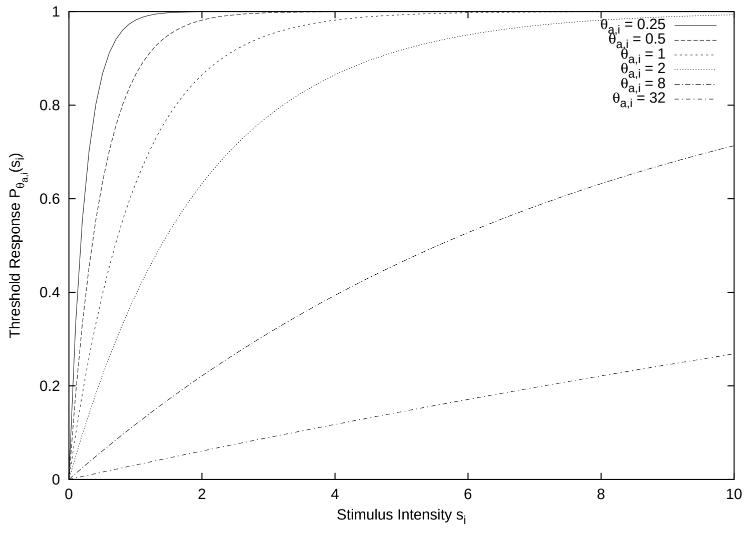
- **Response threshold** $\Theta_{a,i}$: the response threshold $\Theta_{a,i}$ is expressed in units of stimulus. The index a stands for the individual who is given the threshold value and the index i refers to the specific kind of task. The response threshold $\Theta_{a,i}$ is an internal value of the worker a that indicates the stimulus intensity above which the individual gets engaged in performing task i with high probability. Each worker a is given a set of response thresholds $\Theta_a = \{\Theta_{a,0}, \dots \Theta_{a,n}\}$ for all possible tasks $\{0, \dots, n\}$. One can also think of the response threshold to represent the level of specialization of the worker in a certain task. The higher the value of the response threshold, the lower is the level of specialization in the associated task.

- **Response function** $P_{\Theta_{a,i}}(s_i)$: the response function $P_{\Theta_{a,i}}(s)$ specifies the probability for the individual a to respond positively to the stimulus s_i emitted by task i. Two major conditions that need to be fulfilled by the response function are that the associated probability has to be close to 0 for $s_i \ll \Theta_{a,i}$ and close to 1 for $s_i \gg \Theta_{a,i}$.

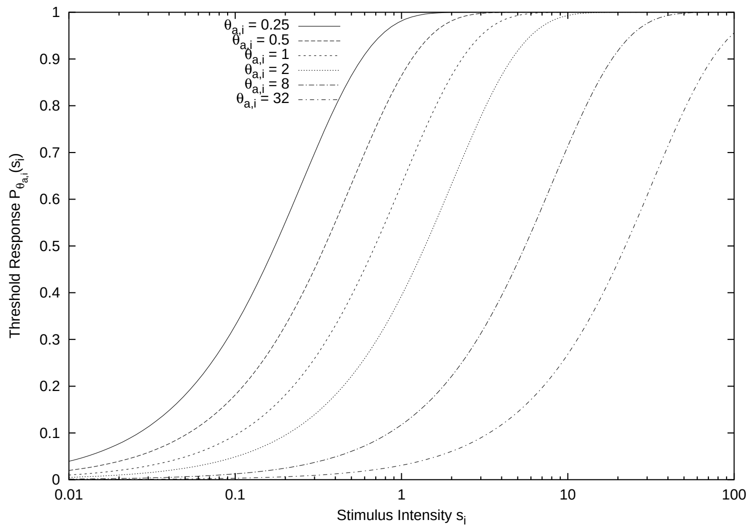
The requirements on $P_{\Theta_{a,i}}(s_i)$ can be satisfied by many different response functions. In [34] for instance, the threshold response function is given by:

$$P_{\Theta_{a,i}}(s_i) = 1 - e^{-s_i/\Theta_{a,i}}. \tag{4.1}$$

Figure 4.1(a) displays $P_{\Theta_{a,i}}(s_i)$ given by Equation 4.1 for different response thresholds $\Theta_{a,i}$. Figure 4.1(b) shows the same as a semi-logarithmic plot.



(a)



(b)

Figure 4.1: An example for a response function: $P_{\Theta_{a,i}}(s_i) = 1 - e^{-s_i/\Theta_{a,i}}$ is plotted with a linear scale for (a) and a semi-logarithmic scale for (b). The probability of a positive response is plotted for several values of $\Theta_{a,i}$. This response function is valid as the conditions: $s_i \ll \Theta_{a,i} \Rightarrow P_{\Theta_{a,i}}(s_i) \approx 0$ $s_i \gg \Theta_{a,i} \Rightarrow P_{\Theta_{a,i}}(s_i) \approx 1$, are satisfied.

Another family of response functions, utilized by Bonabeau *et al.* in [6], is:

$$P_{\Theta_{a,i}}(s_i) = \frac{s_i^n}{s_i^n + \Theta_{a,i}^n}, \tag{4.2}$$

where $n \geq 1$ determines how fast $P_{\Theta_{a,i}}(s_i)$ converges towards the value 1. This is shown in Figure 4.2 with several plots of Equation 4.2 with the same threshold $\Theta_{a,i}$ and different values for n.

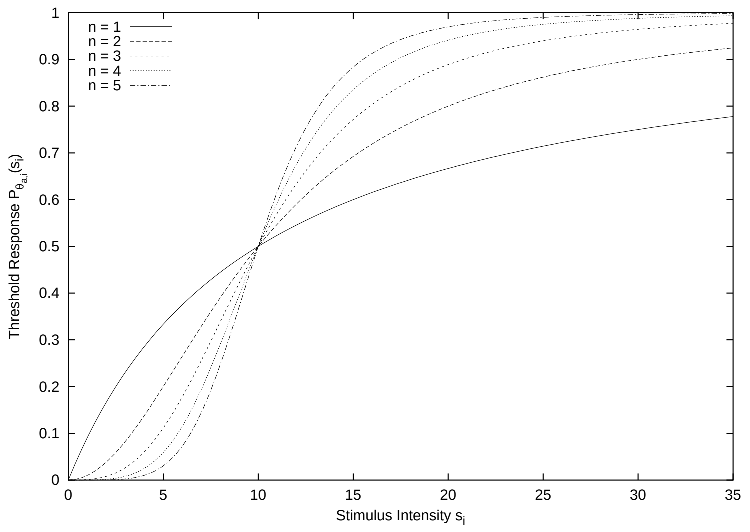
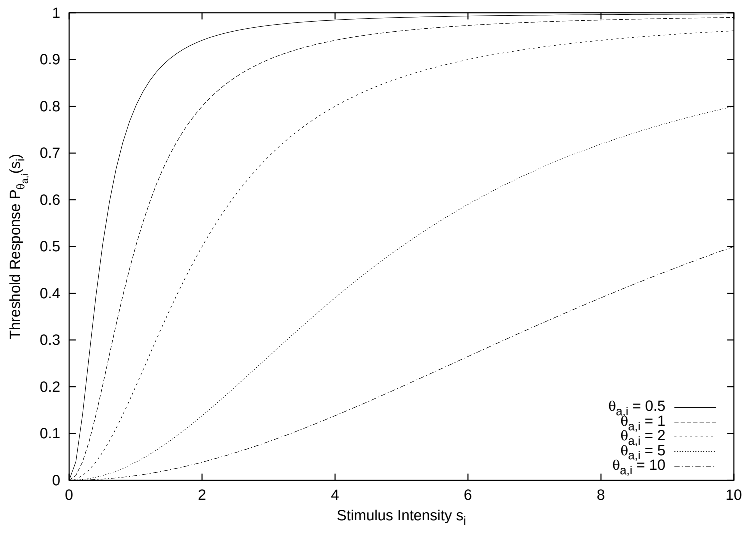


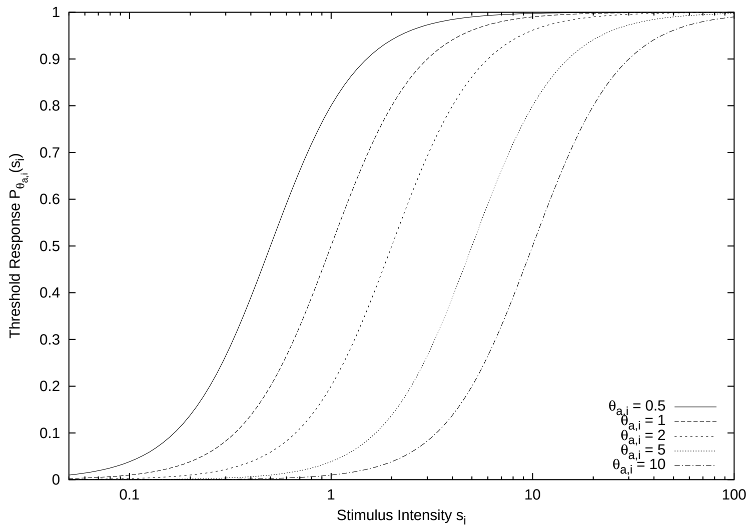
Figure 4.2: Another family of response functions:$P_{\Theta_{a,i}}(s_i) = \frac{s_i^n}{s_i^n+\Theta_{a,i}^n}$, for several values of n and $\Theta_{a,i} = 10$.

For all $n \geq 1$, the response function is close to 0 for $s_i \ll \Theta_{a,i}$, and close to 1 for $s_i \gg \Theta_{a,i}$. Bonabeau *et al.* mainly make use of Equation 4.2 with $n = 2$. As the work in this master thesis is widely based on the work on response thresholds by Bonabeau *et al.*, for the rest of this chapter we will as well make use of Equation 4.2 with $n = 2$ and analyze it. The figures 4.3(a) and 4.3(b) plot Equation 4.2 with $n = 2$ for different values of the response threshold $\Theta_{a,i}$. The latter one uses a logarithmic scale for stimulus s_i.

This simple threshold model explains the different behavior of workers within a colony quite accurately. Consider the previously mentioned example from Section 4.1.3 about the ant species from the *Pheidole* genus. Workers are divided into minors who fulfill quotidian tasks and majors fulfilling tasks like seed milling or defense. If we assign threshold values for the various tasks to the two castes of workers, the minor workers would have smaller threshold values than the major workers for the quotidian tasks, as they are more likely to perform these tasks. In case the fraction of minors is diminished, as done by Wilson [48], the work usually performed by the minors will be left undone



(a)



(b)

Figure 4.3: The response function $P_{\Theta_{a,i}}(s_i) = \frac{s_i^2}{s_i^2+\Theta_{a,i}^2}$, for several values of $\Theta_{a,i}$. (a) shows a linear plot of this function and (b) shows a semi-logarithmic plot for this function.

and therefore the amount of the minors' work will grow. The growing of the amount of undone work determines a growing of the intensity of the corresponding stimulus. And after some time – in the experiment of Wilson [48] within one hour of the ratio change – the stimulus of the quotidian tasks will be high enough to overcome the comparably high thresholds of the major workers and they start performing the minors' work as well. A quantitative analysis using Monte Carlo simulations for the threshold model achieves comparable results to those observed by Wilson.

4.2.2 Model with Dynamic Thresholds

Even though the previously described threshold model is able to explain some phenomena in division of labor in social insects, there are still some limitations, as the assumption is made that the workers' thresholds remain fixed over time. Experiments on honey bees [36] have shown that task allocation also depends on aging, learning or both. And the temporal polyethism which says that individuals of the same age form an age class tending to perform identical sets of tasks, can also not be satisfied by the assumption that thresholds remain fixed over time. In order to overcome these problems, Théraulaz *et al.* [40] extended the fixed-threshold model by introducing additional rules that update the threshold values so that they adapt in time.

Théraulaz *et al.* [40] introduce two update rules depending on the task currently performed by the worker. The underlying idea is again very easy to understand. A worker busy with a certain task should be encouraged to continue performing that task rather than switching to another task, as this is more effective. More precisely, this means that if a worker a is performing a task i, the corresponding threshold $\Theta_{a,i}$ should be updated so as to encourage worker a to continue performing the same task. Thus, the threshold needs to be decreased over time. In parallel, worker a has to be discouraged to take tasks different from i, i.e., its threshold values $\Theta_{a,j}, j \neq i$, should be increased over time. Additionally, a threshold value should not exceed an upper and a lower bound. By defining Θ_{min} and Θ_{max} as the lower and the upper limit for the threshold values and δ_1 and δ_2 as the coefficients to determine the threshold values' decrease and increase respectively, the threshold update rules can be formulated as:

$$\Theta_{a,i}(t + \Delta t) = \Theta_{a,i}(t) - \delta_1 \Delta t, \qquad \Theta_{a,i}(t + \Delta t) \in [\Theta_{min}, \Theta_{max}], \tag{4.3}$$

if worker a is performing task i during the period $[t, t + \Delta t]$, and

$$\Theta_{a,i}(t + \Delta t) = \Theta_{a,i}(t) + \delta_2 \Delta t, \qquad \Theta_{a,i}(t + \Delta t) \in [\Theta_{min}, \Theta_{max}], \tag{4.4}$$

if worker a is not performing task i during the period $[t, t + \Delta t]$ and more generally:

$$\Theta_{a,i}(t + \Delta t) = \Theta_{a,i}(t) - (x\delta_1 - (1 - x)\delta_2)\Delta t \tag{4.5}$$

where $0 \leq x \leq 1$ is the fraction of the period $[t, t + \Delta t]$ during which worker a is performing task i.

4.2.3 Discussion of the Response Function

In order to analyze the response function $P_{\Theta_{a,i}}(s_i, t)$ we distinguish four different cases of dynamics of the threshold value $\Theta_{a,i}(t)$ and the stimulus intensity $s_i(t)$:

1. $\Theta_{a,i} = const., s_i = const.$
2. $\Theta_{a,i} = const., s_i = s_i(t) \neq const.$
3. $\Theta_{a,i} = \Theta_{a,i}(t) \neq const., s_i = const.$
4. $\Theta_{a,i} = \Theta_{a,i}(t) \neq const., s_i = s_i(t) \neq const.$

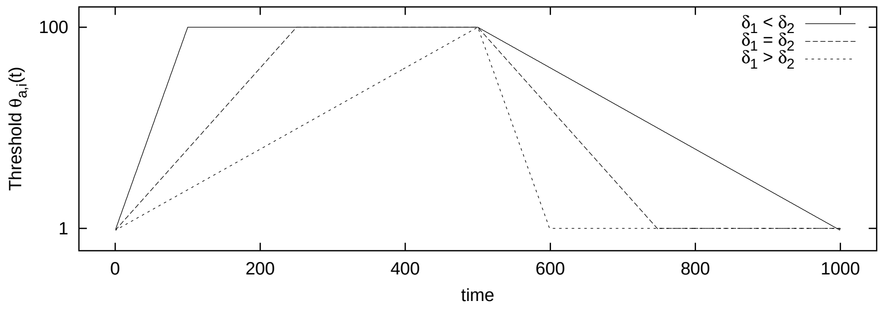
In the first case the threshold value, the stimulus intensity and the response function remain constant over time. For this case a further analysis is not required.

If the stimulus intensity changes over time as in the second case, the response function $P_{\Theta_{a,i}}(s_i, t)$ changes over time as well. The response function follows the behavior of the stimulus intensity, i.e., the response function grows if the stimulus intensity grows and vice versa, as shown by Figure 4.3. This correlation is not linear. The derivative of $P_{\Theta_{a,i}}(s_i, t)$ has its global maximum at $s_i = \Theta_{a,i}$. Thus, changes of the stimulus intensity s_i around Θ_{a_i} cause higher changes of the threshold function.

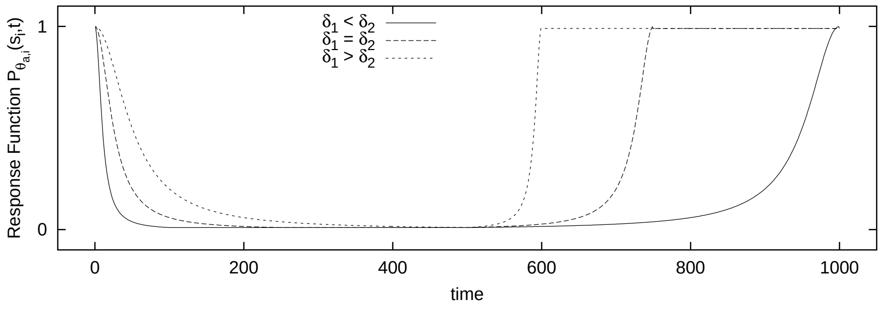
For the two latter cases, we assume that the threshold values can change over time. In the dynamic threshold model of Section 4.2.2 two threshold update rules are described by Equation 4.3 and Equation 4.4. If a worker a is currently performing task i, the respective threshold value $\Theta_{a,i}(t)$ is diminished, whereas all other threshold values $\Theta_{a,j}, i \neq j$, are increased. The rate of this adaptation process depends on two parameters, δ_1 and δ_2, of the update rules.

The parameter δ_1 determines how fast threshold values are increased and the parameter δ_2 determines how fast they are decreased. The values for these two parameters are crucial to the behavior of the response function as well as to the behavior of the threshold model in general. Consider a worker a who accepts a task T_i for which he has a high threshold value $\Theta_{a,j}(t)$. A high value for δ_1 leads to a rapid decrease of the respective threshold. Assuming that a threshold value $\Theta_{a,i}(t)$ represents the specialization of worker a in task T_i, we can say that high values of δ_1 result in a fast adaptation, and low values of δ_1 result in a slow adaptation. The parameter δ_2 has no influence on the adaptation rate. δ_2 determines the level of specialization, i.e., high values of δ_2 result in a low level of specialization, as a threshold value increases fast if the worker does not perform the respective task. Low values of δ_2 result in a high level of specialization, as a threshold value increases slowly if the worker does not perform the task.

Figure 4.4(a) shows the dynamics of the threshold value $\Theta_{a,i}(t)$ for three different sets of δ_1, δ_2: $\delta_1 < \delta_2, \delta_1 = \delta_2$ and $\delta_1 > \delta_2$. The worker is initially specialized for tasks of type i. But in the period [1,500] he performs a different type of task, so that the threshold value increases. Then, in the period [501,1000] the worker again performs tasks of type i and the threshold value decreases. Figure 4.4(b) shows the response function $P_{\Theta_{a,i}}(s_i, t)$ for the given progression of $\Theta_{a,i}(t)$ assuming that the stimulus intensity s_i remains constant.



(a)



(b)

Figure 4.4: (a): The dynamics of the threshold value $\Theta_{a,i}(t)$ for three different sets of update parameters δ_1, δ_2. (b): The corresponding response function $P_{\Theta_{a,i}}(s_i, t)$.

Two other parameters that have an influence on the response function are the lower and the upper limit for the threshold values, Θ_{min} and Θ_{max}, as:

$$P_{\Theta_{max}}(s_i, t) \leq P_{\Theta_{a,i}}(s_i, t) \leq P_{\Theta_{min}}(s_i, t),$$

these two values also determine the lower and the upper limit for the response function itself. Low values of Θ_{min} permit a high grade of specialization. On the other side, low values of Θ_{max} lead to a high minimum-probability of responding positively to a task.

In the last case of our differentiation of the response function dynamics we assume both, the threshold value and the stimulus intensity, to change over time. This case is a combination of the cases 2 and 3, which were examined in detail. Therefore, we don't make a further analysis for this case,

Chapter 5

The Insect-Based Approach to the DTA

In this section the insect-inspired approach to the DTA is explained. This approach is based on the dynamic threshold model and was previously introduced by Campos *et al.* [9] and by Cicirello *et al.*[12]. Campos *et al.* refer to the behavior of ants whereas Cicirello *et al.* speak of wasp-like agents.

One of the main objectives for an approach to the DTA is to minimize the amount of reconfigurations, or setups, as it is related to a high cost. The threshold model describes the specialization observed in social insects. In the approach, insect-like agents are in charge of machines. Applying the rules of the threshold model results in an adaptive system, where the agents specialize in types of tasks and hence prevent unnecessary reconfigurations.

Before we start describing the approach in detail, we want to clarify one attribute of the DTA. The DTA is a discrete problem in the sense that tasks are released to the environment in discrete times only. We refer to time steps and don't specify what exactly they represent in particular. So they may be considered as minutes, hours or just as time steps.

5.1 Agents, Thresholds and Stimuli

The approach to the DTA is an insect-based multi agent system. We define the set of agents:

$$\mathcal{A} = \{A_1, \ldots, A_m\},$$

who are in charge of the set of machines $\mathcal{M}$, $|\mathcal{M}| = m$. Each agent is responsible for one machine and autonomously determines its tendency to bid for a certain type of task. To do so, an agent A_i is provided a set of threshold values Θ_i for the set of tasks $\mathcal{T}$, $|\mathcal{T}| = n$:

$$\Theta_i = \{\Theta_{i,0}, \dots \Theta_{i,n}\} \tag{5.1}$$

$$\Theta_{min} \leq \Theta_{i,j} \leq \Theta_{max} \qquad \forall k \in \{0, \dots n\}, \tag{5.2}$$

as described in Section 4.2. $\Theta_{i,j}$ is the response threshold of agent A_i for tasks of type j. Θ_{min} and Θ_{max} are the values for the lower and the upper limit of the threshold. The threshold value $\Theta_{i,j}$ represents the level of specialization of agent A_i's machine in task T_j. Without loss of generality, we assume that each machine is able to process any kind of task. Of course, the model also works without this restriction but this was applied for all experiments and hence we apply it for the description of our model as well.

When a task T_j is created, it is assigned a stimulus S_0. The intensity of this stimulus grows until the task is allocated to a machine. Thereby, the longer a task stays unallocated, the higher is the respective stimulus intensity. The increase of the stimulus intensity is linear with respect to time and is calculated each time step by:

$$S_j = S_0 + \sigma \Delta t, \tag{5.3}$$

where S_j is the current intensity of the stimulus related to task T_j, S_0 is the initial intensity, σ is the rate of increase of the intensity, and Δt is the time the task stayed unallocated.

Given a task's stimulus intensity S_j and agent A_i's corresponding threshold value $\Theta_{i,j}$, agent A_i will bid for task T_j with probability:

$$P(\Theta_{i,j}, S_j) = \frac{S_j^2}{S_j^2 + \Theta_{i,j}^2}. \tag{5.4}$$

From this equation, we gather that in case the stimulus S_j and the threshold $\Theta_{i,j}$ are equal, the probability for agent A's machine to bid for a task is 50%. Squaring of the stimulus S_j and the threshold $\Theta_{i,j}$ lead to a nonlinear behavior of $P(\Theta_{i,j}, S_j)$ so that small changes of S_j around $\Theta_{i,j}$ cause high changes in $P(\Theta_{i,j}, S_j)$.

5.2 Threshold Update Rules

In the approach, an agent A_i's set of threshold values Θ_i is updated in a way comparable to the one in the dynamic threshold model of Section 4.2.2. We apply both update rules given by equations 4.3 and 4.4, and add a third update rule for the case that a machine is currently idle, i.e., it is not performing any task. The update rules are employed each time step.

The threshold value $\Theta_{i,j}$ is updated by:

$$\Theta_{i,j} = \Theta_{i,j} - \delta_1, \tag{5.5}$$

if agent A_i's machine is currently processing or setting up for a task of type j. This update rule results in the tendency to accept a task of type j again.

All the other threshold values $\Theta_{i,k}, k \neq j$, are updated by:

$$\Theta_{i,k} = \Theta_{i,k} + \delta_2, \tag{5.6}$$

in order to diminish the probability to bid for tasks of a type different from the one currently processed or set up. The combination of these two update rules leads to a specialization into the type of task that is currently being processed.

The third update rule is applied for the case that agent A_j's machine is currently not processing or setting up any task:

$$\Theta_{i,j} = \Theta_{i,j} - \delta_3^t, \tag{5.7}$$

where t refers to the amount of time, measured in time steps, agent A_i's machine is already idle. This update rule encourages the machine to take tasks of any kind with a probability increasing in time. By exponentiating the constant parameter δ_3 with t, the threshold value is decreased by higher steps the longer the machine stays idle. The thresholds should not immediately be set to a low value or be decreased too fast, as this would cause additional setups by pushing the machine to take a task of any type. It is advantageous that a machine stays idle for a short amount of time instead of taking any task, as a setup is related to a very high cost.

This last update rule for idle machines does not appear within the description of the dynamic threshold model of Théraulaz *et al.* [41] or in Campos *et al.* [9]. Cicirello *et al.* first proposed it in [13] without exponentiating δ_3 with t. The exponentiation appeared the first time in [11].

5.3 Dominance Contest

A task may only be processed by one machine at once. So far it is not stated yet what happens if two or more agents bid for the same task. A bid itself does not give the possibility to differ in such a situation, as a machine can only decide between either bidding or not. A value is not related to the bid. A simple possibility would be to decide arbitrarily between the bidding parties and allocate the task to one of them. This strategy does not take any information about the bidding machines into consideration and thus is not optimal. Consider the situation of two machines, one being idle and the other one having a long queue of tasks still to be executed. The situation could emerge that both bid for the same task and they both require a setup for it. It would be common sense to allocate the task to the idle machine, but the random strategy gives both the same probability to get the task. To overcome this problem Cicirello *et al.* proposed in [10] a method that they refer to as the *dominance contest.* The dominance contest compares the competing agents by a value that depends on their queues. This value is called the *force* value F. And for agent A_i we can calculate F_i as follows:

$$F_i = 1 + T_{p,i} + T_{s,i}, \tag{5.8}$$

where $T_{p,i}$ and $T_{s,a}$ are the sum of the processing times and the setup times respectively of the tasks in agent A_i's queue. A lower force value F corresponds to a shorter queue and leads to a higher probability to win in a dominance contest and vice versa[1].

If two agents A_1 and A_2 bid for a same task, the mechanism of the dominance contest announces probabilities to the competing agents, which express the chance of being allocated the task. Given their force values F_1 and F_2, the probability that A_1 is allocated the desired task can be calculated by:

$$P_1(F_1, F_2) = \frac{F_2^2}{F_1^2 + F_2^2}, \tag{5.9}$$

The probability for agent A_2 to be given the task is $P_2(F_1, F_2) = 1 - P_1(F_1, F_2)$.

In general, if the dominance contest consists of a set of competing agents $\mathcal{C} = \{A_1, \ldots, A_k\}, 2 \leq |\mathcal{C}| \leq n, \mathcal{C} \subset \mathcal{A}$, agent A_i will receive the task with probability:

$$P(F_1, \ldots, F_k)_i = \frac{\sum\limits_{\mathcal{C}\setminus A_i} F_l^2}{(n-1)\sum\limits_{\mathcal{C}} F_l^2}. \tag{5.10}$$

This equation was not given in [10], but we deduced it from Equation 5.9.

The presented mechanism is not deterministic. This means, machines are assigned probabilities for receiving the desired task. The dominance contest recommends agents with short queues by assigning them a high probability.

We can briefly summarize the insect-based approach to the DTA as follows. Each time step tasks that have not been allocated yet, are offered to

[1]For this reason the term “force” might not be ideal as denotation for F. Commonly, force is a strength. But in this case a high force value is related to a high probability in losing the dominance contest.

the machines. Each agent, who is in charge of one machine, decides for every offered task whether he bids for the task or not. This decision is probabilistic and based on the stimulus intensity of the task and the respective threshold value of the worker. If more than one agent bids for the same task, a dominance contest is held to assign probabilities to the competing agents. These probabilities express the respective agent's chance of being allocated the task. If no agent bids for a certain task, it is offered again the next time step. Otherwise, if one or more agents bid for a certain task, this task is allocated to one of the agents and added to the respective machine's queue. At the end of each time step, the threshold values of the agents are updated and the stimulus of the tasks that remain unallocated are increased.

Chapter 6

Improvement Proposals

The insect-based model of Chapter 5 was analyzed in detail for a wide set of DTA instances. In general, it is able to adapt well and reaches satisfying performance. Still, for certain situations we observed problems, where the original system adapted very slow, or it even did not succeed to adapt at all. Therefore, we propose two new rules (Section 6.2) in order to speed up the adaptation of the threshold values. Additionally, we propose three modifications of existing rules (Section 6.1). All these proposals were previously introduced in [33].

6.1 Modifications of Existing Rules

The proposed modifications concern the update rules, the calculation of the force value, and the dominance contest. These three methods in general have a positive effect on the overall performance. We do not intend to change the general idea of their application. We rather want to modify them in order to overcome specific problems that were observed in the analysis of the system.

6.1.1 Update Rules (UR)

The update rules adapt the threshold values according to the task being currently processed. We recognized one problem about this. Machines may in general have a queue of tasks to be processed in the future. If an agent is allocated a task, a setup is required with respect to the last task in the respective machine's queue. Whether a setup is required or not does not depend on the task currently processed. It is the last task in a machine's queue only that determines whether a setup will be required or not. Figure 6.1 shows a machine with a queue of five tasks. The first four tasks among those five are tasks of type 1 and the last task in the queue is of a different type

2. Applying the update rules encourages the agent in charge of the machine to bid for tasks of type 1 and discourage it to bid for tasks of type 2. This behavior is not desirable. Adding any task from a different type than 2 to the queue would cause an additional setup. It would be advantageous in our eyes, to encourage the agent to bid for tasks of type 2 and discourage it to bid for tasks of a different type. We have chosen to modify the update rules, so that the last task in the machine's queue determines which threshold values are updated in order to reduce the number of necessary setups.

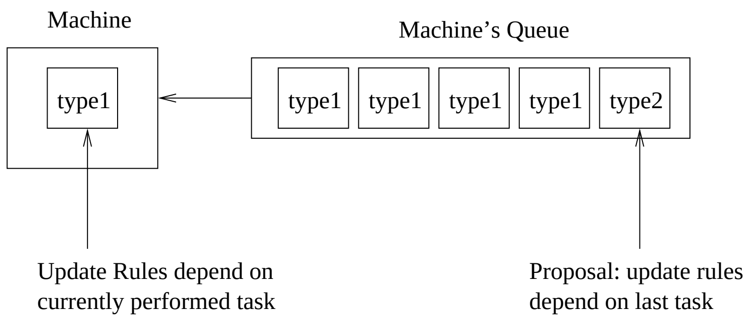

Machine
Machine's Queue
type1
type1
type1
type1
type1
type2
Update Rules depend on currently performed task
Proposal: update rules depend on last task

Figure 6.1: An example for a situation, where the original update might lead to additional setups. Our proposal **UR** modifies the update rule, so that they depend on the last task in a machine's queue, and not on the one that is currently processed.

6.1.2 Calculation of the Force Value (CFV)

We can construct a simple example to illustrate a shortcoming in the calculation of the force value. Figure 6.2 shows two agents A_1 and A_2, both with empty queues. Agent A_1's machine is set up for tasks of type 1 and agent A_2's machine is set up for tasks of type 2. Both bid for the same task, which is of type 2. Therefore they enter the dominance contest. As both queues are empty, the force values are equal as well. So, they are announced the same probability to receive the task. But, There is no reason to assign the task to agent A_2 as its machine would require a setup, what is not the case for A_1's machine. A distinction whether a setup would be required or not, does not enter in the calculation of the force value.

We try to solve the described problem by adding to the force value F_i of agent A_i the value $\omega_{ij,s}$, if A_j is bidding for a task of type j. $\omega_{ij,s}$ is described in Section 2.2.1 and takes into consideration whether a setup will be required or not, by taking the machines queue state s into account. $\omega_{ij,s}$ is the setup time for the corresponding task, in case a setup will be required, and 0 otherwise:

$$F_i = 1 + T_{p,i} + T_{s,i} + \omega_{ij,s} \tag{6.1}$$

$$\omega_{ij,s} = \begin{cases} T_{setup} & \text{if task requires a setup} \\ 0 & \text{if task does not require a setup,} \end{cases} \tag{6.2}$$

where $T_{p,i} + T_{s,i}$, like in Equation 5.8, is the sum of the process times and the setup times within agent A_i's queue.

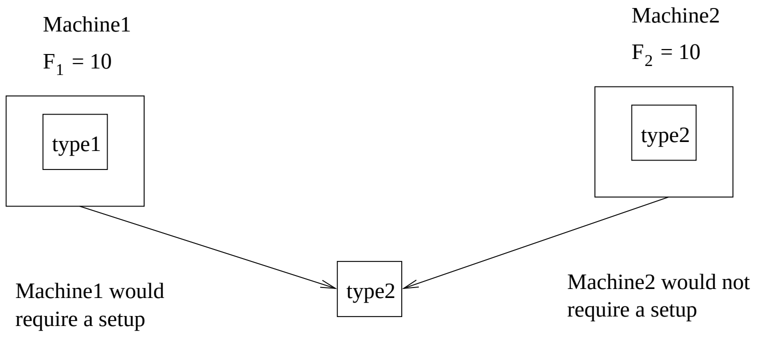

Machine1
F1 = 10
type1
Machine2
F2 = 10
type2
Machine1 would require a setup
type2
Machine2 would not require a setup

Figure 6.2: Two machines with empty queues bid for the same task. Whether a setup is required or not, is not considered in the original calculation of the force value. Therefore, we propose the modification **CFV**, which increases the force value in case a setup is required.

6.1.3 Dominance Contest (DC)

Another problem of the dominance contest is, that the more machines compete with each other in a dominance contest, the smaller are the differences between the probabilities to win. For example, if two machines compete and one has a force value of $F_1 = 1$ – what refers to an empty queue – and the other has a force value of $F_2 = 10$ the corresponding probabilities to win following Equation 5.9 are $P_1 = 0.99$ for the one machine and $P_2 = 0.01$ for the other. This big difference of the probabilities achieves the desired effect. For larger numbers of competing machines, the probabilities do not differ so much any more. Figure 6.3 shows a situation, where ten machines are competing for the same task. The first machine has a force values of $F_1 = 1$. All machines have a force value of $F_i = 10$. According to Equation 5.10, this results in the probabilities $P_1 = 0.111$, for the first machine, and $P_j = 0.099$ for all others. In general the probability for one competitor to win in a dominance contest of n competitors is never higher than $\frac{1}{n-1}$. To overcome this problem we use the following rule instead of Equation 5.10:

$$P(F_1, \dots F_n)_i = \frac{\frac{1}{F_i^2}}{\sum_{\mathcal{C}} \frac{1}{F_l^2}}, \tag{6.3}$$

where $P(F_1, \dots F_n)_i$ is the probability for agent i to be allocated the task, if the dominance contest consists of a set of competing agents $\mathcal{C} = \{A_1, \dots, A_k\}, 2 \leq |\mathcal{C}| \leq n, \mathcal{C} \subset \mathcal{A},$. In the example with ten machines and the forces $F_1 = 1$ and $F_i = 10, 2 \leq i \leq 10$, the respective probabilities become $P_1 = 0.917$ and $P_j = 0.0083$.

Additionally, as notational refinement, we define the *dominance value* D_i for an agent A_i being the inverse of the according force value F_i:

$$D_i = \frac{1}{F_i}, \tag{6.4}$$

as the use of the term force seems rather contradictory. A higher force leads to a lower probability to win a dominance contest. With this refinement we can replace Equation 6.3 with:

$$P(D_1, \dots D_n)_i = \frac{D_i^2}{\sum_{\mathcal{C}} D_l^2}. \tag{6.5}$$

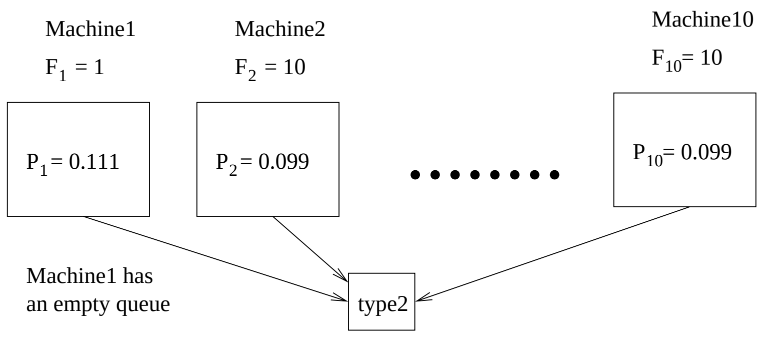


Figure 6.3: Ten machines enter the dominance contest. For such a high number of competing machines, the probabilities they are announced by the dominance contest do not differ very much. Our proposal **DC** modifies the calculation of the probabilities, so that even for a large number of competing machines, announced probabilities differ according to the difference of the force values.

6.2 Additional Rules to Optimize Threshold Values

The modifications on the existing rules can improve the performance for specific cases. But they don't have a direct effect on the adaptation speed. We observed difficulties of the system to adapt to dynamically changing situations. For instance, this occurs if probabilities for the creation of the tasks change during a simulation. The machines require a long time to adapt to the new situation or often do not adapt at all. Two additional rules have been defined in order to overcome this problem. The first one, **BCT**, concerns tasks that remain unallocated. And the second additional rule, **IMB**, concerns idle machines that refuse to bid.

6.2.1 No Bid for a Created Task (BCT)

Tasks are allocated to machines through a bidding mechanism. An agent, representing a machine, probabilistically decides whether it bids for a task or not. The situation may occur, where a task remains unallocated, as no agent bids for it. The stimulus, related to a task, increases over time. This process is very slow. This can cause tasks to have a very long waiting time and mainly affects tasks that are released rather seldom, so that no machine specializes for the respective type. This is especially a problem, in case probabilities of task appearances dynamically change during the simulation. Types of tasks, that previously appeared seldom may appear more frequently and vice versa. This is not immediately recognized by the machines, so that they probably remain idle, as they wait for the tasks that used to appear more frequently. The current threshold update rules provide the ability to adapt. But this adaptation is rather slowly. Therefore, we introduce an additional method in order to reduce the time that tasks stay unallocated:

$$\Theta_{i,j} = \Theta_{i,j} - \gamma_1 \qquad \forall \text{ agents } a. \tag{6.6}$$

For each agent A_i, the threshold value $\Theta_{i,j}$, referring to the task of type j, is decreased by the constant value γ_1, if the respective task was not assigned.

6.2.2 Idle Machine Does Not Bid (IMB)

Equation 5.7 offers an update rule for idle machines in order to encourage them to bid for tasks of any type. This update rule decreases all threshold values by a value that exponentially increases in time. The idea is, that a machine may stay idle for some time rather than being forced to take any task immediately. We observed, that this can cause machines to stay idle for very long and therefore has a negative effect on several performance measures. We propose an additional update rule, which is employed in case an idle machine does refuses to bid for a task it is offered:

$$\Theta_{i,j} = \Theta_{i,j} - \gamma_2. \tag{6.7}$$

In that case, the corresponding threshold value $\Theta_{i,j}$ of the refused task of type j is decreased by the fixed value γ_2. This rule is not a simple extension of the original update rule for idle machines, as it is only executed when an idle machine is offered a task and refuses to bid for it.

Chapter 7

Experimental Setup

The improvement proposals of the preceding chapter were compared to the original approach on a wide set of DTA instances. In this chapter, we describe these instances. Additionally, we explain how the various parameters of the approach were determined and optimized by a genetic algorithm.

7.1 Problem Instances

Mainly four parameters determine the problem instances: the number of machines, the number of different types of tasks, the task creation process and the maximum number of tasks a machine may store in its queue.

The experimental analysis is split up into the analysis of small problems, with either 2 or 4 machines, and large problems, with either 10 or 25 machines. For the small problems, we examined the behavior for either 2 or 4 different types of tasks. Otherwise, for large problem instances 10 different types of tasks are given. The small problems allow a clearer analysis, giving the possibility to analyze the behavior of individual machines towards certain types of tasks. This is rather complex for the large problems, which are meant to be closer to real-world problems and where we focus on the analysis of the whole system.

All problem instances have in common, that task arrival times are exponentially distributed. The distance between two task arrivals, $t_{n+1} - t_n$, is calculated by:

$$t_{n+1} - t_n = -\frac{\log r_{n+1}}{\lambda},$$

where r_{n+1} is a random value in $(0, 1]$, and λ is the expected number of tasks to arrive each time step. As indicated by Figure 7.1, the values of the times $\{\ldots, t_n, \ldots\}$ are rounded up in order to discretize them.

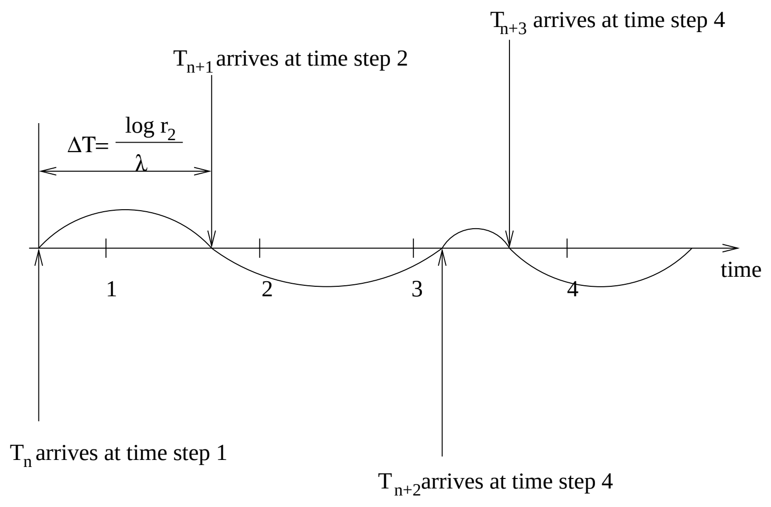

T_{n+3} arrives at time step 4
T_{n+1} arrives at time step 2
$\Delta T = \frac{\log r_2}{\lambda}$
1
2
3
4
time
T_n arrives at time step 1
T_{n+2} arrives at time step 4

Figure 7.1: Graphical representation of the exponential distribution of task arrival times. A task that arrives between the time steps τ and $\tau + 1$ is released to the environment at time step $\tau + 1$.

Whenever a task is released, a roulette wheel selection determines the type of the respective task. For each task, the roulette wheel selection takes into account a given probability to be selected. We analyze four different probability mixes:

1. Equal distribution (**equ**): When a task is released, the probability that the task is of type j is the same for all types of tasks.

2. Different distribution (**dif**): One half of the n types of tasks is released with a three times higher probability than the other half.

$$p(j) = 3p(k), 1 \leq j \leq \frac{n}{2} < k \leq n.$$

3. Abruptly changing distribution (**chg**): For the first half of the simulation, the probability mix is the same as in the different distribution. Then the probability mix changes abruptly, so that the previously less frequently appearing tasks are released more often, and the previously more frequently appearing tasks are released less often.

$$\begin{array}{rrrl} p(j) = & 3p(k), & 0 \leq & t < \frac{\tau_{tot}}{2}, \\ p(k) = & 3p(j), & \frac{\tau_{tot}}{2} \leq & t \leq \tau_{tot}, \\ 1 \leq & j \leq & \frac{n}{2} < & k \leq n. \end{array}$$

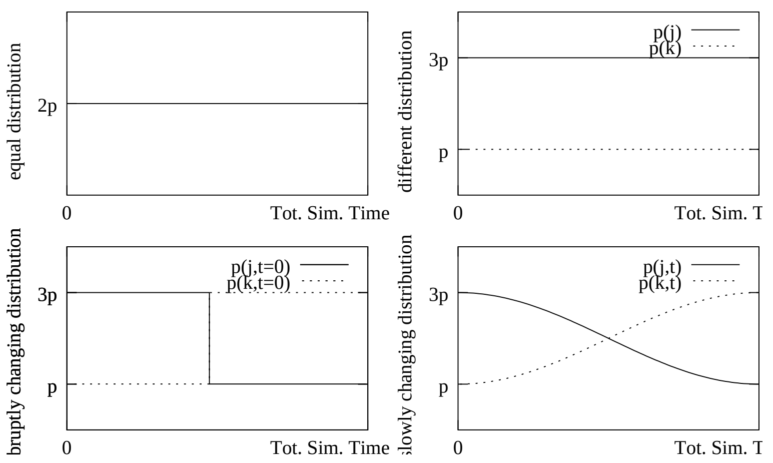


Figure 7.2: The four probability mixes over simulated time. Whenever a task is released, these probabilities are used for a roulette wheel selection to determine the type of the task. For the equal distribution all types appear with same probability. In the other cases, $p(j)$ and $p(k)$ each represent the probabilities of one half of the types of tasks. The value p equals $\frac{1}{2n}$, where n is the number of tasks.

4. Slowly changing distribution (**sin**): The initial probability mix is the same as in the different distribution. During the ongoing of the simulation the probabilities are continuously changing as follows:

$$\begin{gathered} p(j,t) = p(k,t=0) + \tfrac{p(j,t=0)-p(k,t=0)}{2}(1 + cos(\tfrac{t\pi}{\tau_{tot}}) \\ p(k,t) = p(k,t=0) + \tfrac{p(j,t=0)-p(k,t=0)}{2}(1 - cos(\tfrac{t\pi}{\tau_{tot}}) \\ 1 \leq j \leq \tfrac{n}{2} < k \leq n. \end{gathered}$$

Figure 7.2 visualizes the four probability mixes. The changing probability mixes were chosen in order to investigate the ability of an approach to adapt on a changing environment. In real factory environments, the arrival rates of different types of tasks might not change abruptly. Thus, the slowly changing distribution is probably more realistic than the abruptly changing distribution. Nevertheless, an abrupt change of task arrival rates represents the most difficult case, and therefore, it worths analyzing it.

The maximum number of tasks a machine may store in its queue is the last parameter to determine a problem instance. We consider the case, where a machine's queue may store at most 10 tasks. Additionally, we examine the case, in which there is no restriction related to the maximum queue length, i.e., a machine's queue may store arbitrarily many tasks.

Table 7.1 summarizes all problem instances that will be analyzed in the next chapter.

M	T	MODES	Max. Queue
2	2	**equ,dif,chg,sin**	0
2	4	**equ,dif,chg,sin**	0
4	2	**equ,dif,chg,sin**	0
10	10	**equ,dif,chg,sin**	0,10
25	10	**equ,dif,chg,sin**	0,10

Table 7.1: A summary of all examined DTA instances. In the headline, **M** and **T** indicate the number of machines and types of tasks.**MODES** signifies the different probability mixes, where **equ**, **dif**, **sin** and **chg** represent the equal, the different, the slowly and the abruptly changing distribution, in this order. **Max. Queue** signifies the maximum number of tasks that may be held in a machine's queue. The value 0 indicates, that there is no restriction related to the queue. For instance, problems with two machines and two different types of tasks are analyzed for all modes except the abruptly changing distribution and only without restrictions of the queue length.

Machines may cause failures. For instance, in the real-world example of a painting facility, this can occur if during a reconfiguration, colors get mixed. In order to simulate this, we implemented a 0.1% probability of each machine to cause a failure at each time step. This probability is increased to 1%, if the machine is currently performing a setup. If a failure is caused, the machine is inoperable for an amount of time, uniformly distributed between 0 and 50 time steps.

7.2 Parameters

In order to determine a good combination of values for various parameters, a *genetic algorithm* (GA) was used. The GA procedure we apply is very simple and probably far away from being optimal. GAs are inspired by nature's capability to evolve life well adapted to their environment. In GAs, a *population* of individuals form a *generation*. Each individual, also called *genotype*, represents a solution. A so called *fitness* function is used to compute the quality for each individual's solution. Additionally, mainly three operators, *tournament*, *crossover* and *mutation*, are used. The tournament operator selects the individuals with the highest fitness value as *parents* for the crossover. The crossover operator creates one or more *children* out of two or more parents, by combining their genotypes. The sum of all children forms the new generation. The mutation operator is used in order to enable

Parameter	**Range**
Update Rule Decrease Rate (Equation 5.5)	$0 \leq \delta_1 \leq 25.5$
Update Rule Increase Rate (Equation 5.6)	$0 \leq \delta_2 \leq 25.5$
Update Rule Idle Machines (Equation 5.7)	$1 \leq \delta_3 \leq 13.75$
Maximum Threshold	$0 \leq \Theta_{max} \leq 4080$

Table 7.2: The limiting ranges of the four parameters that were determined by the GA.

the population to improve. Therefore, each individual's genotype is changed in a random way. There are many different methods, how exactly the tournament, crossover and the mutation operators are executed. For a detailed description of various methods, we refer to Goldberg [24], Holland [26] and Mitchell [31].

Our GA determines four parameters of the original insect-based approach $(\delta_1, \delta_2, \delta_3, \Theta_{max})$. Ranges for each parameter were initially defined, in order to limit the space in which the GA searches. These ranges are given in Table 7.2.

Each parameter is represented as a binary string of eight bits. A genotype is a 32-bit string, describing the values of the four parameters. Fitness is defined as the reciprocal value of the makespan achieved by the respective individual. Our GA starts by creating a random population of 50 individuals. This population represents the first generation. The parameters, that are represented by the genotypes, are random values in the respectively given range. For each individual of this first generation, we run 30 simulation of a certain problem instance with the individual's parameter set and afterwards assign the resulting fitness as an average of the 30 runs to the individual. After all of the individuals are assigned their fitnesses, a tournament is executed to decide which individuals are copied into the next generation. Two individuals are chosen from the population randomly and their fitnesses are compared. The individual with the higher fitness value is copied with a 75% chance, otherwise the individual with the lower fitness value is copied. There are 50 tournaments, so that the population size remains constant. After the tournaments are finished, two individuals are picked up at random. There is a 75% probability that crossover occurs between them. If there is a crossover, a uniformly distributed random number in the range [0;32] is taken as the point of crossover. This means, the individuals' genotypes are exchanged from the beginning of their genotypes to that point. Individuals of a generation are given the possibility of crossover 50 times. Afterwards, the mutation operator is executed for every bit of every genotype. With a probability of 0.3%

Parameter	**Result**
Update Rule Decrease Rate (Equation 5.5)	$\delta_1 = 14.8$
Update Rule Increase Rate (Equation 5.6)	$\delta_2 = 12.7$
Update Rule Idle Machines (Equation 5.7)	$\delta_3 = 1.01$
Maximum Threshold	$\Theta_{max} = 2016$

Table 7.3: The results of the four parameters after the GA was run.

a bit is changed from 1 to 0 and vice versa. This procedure of tournament, crossover and mutation is repeated for every generation.

We have chosen the problem with four machines and two types of tasks as the problem to be run by the GA. Altogether, there are 100 generations. For the first 25 generations, we run the GA for the mentioned problem with equal distribution of task arrival rates. Then, for the for 25 generations each one, the GA is run with the different, the abruptly changing and for the last 25 generations with the slowly changing distribution. In the last generation, the individual with the highest fitness value is selected and its genotype determines the values of the parameters. The results are summarized in Table 7.3.

The other parameters of the system were not determined by the GA and are as follows:

• $\Theta_0 = 500$	Initial threshold value for all machines and all tasks
• $S_0 = 1$	Initial stimulus intensity when a task is created
• $\sigma = 1$	Rate of stimulus intensity increase
• $\gamma_1 = 50.0$	Additional rule **BCT**: No Bid for a Created Job
• $\gamma_2 = 50.0$	Additional rule **IMB**: Idle Machine Does Not Bid

Chapter 8

Experimental Results

This chapter describes the experiments on which the proposed improvements for the DTA problem were tested. The results obtained with the experiments are analyzed. First, we analyze small problem instances, that is, problems including two or four machines, giving a description of the temporal behavior of single machines. A comparison is made between the original system and the application of single applied improvements. Afterwards, problems with 10, 25 and 50 machines are examined.

The mainly applied performance measures are the makespan and the number of required setups per machine. Additionally, we use measures like the average queue size per machine, the idle time per machine, the time spent in the system by the tasks (cycle time), the time until tasks are allocated and other measures that will be described in detail during the analysis. The more general results on the makespan and the number of setups are provided for every problem instance and each task generation process. One task generation process per problem instance will be highlighted, giving a more detailed analysis with specific measures.

For an easier readability of the results, Table 8.1 gives a summary of most abbreviations used in this chapter. Nevertheless, abbreviations are generally explained in captions as well.

8.1 Small Problem Instances

In general, we can differ three cases of the problem's difficulty:

1. $|\mathbf{M}| > |\mathbf{T}|$: The number of machines is higher than the number of different types of tasks. An analysis was made for systems with four machines and two different types of tasks (**4M2T**). We consider this

Abbreviation	**Explanation**
4M2T	Describes the size of a problem instance. The first digit signifies the number of machines and the second digit signifies the number of different types of tasks.
equ	Equal distribution of task arrival times. Section 7.1 gives a detailed description of the four different distributions.
dif	Different distribution of task arrival times.
chg	Abruptly changing distribution of task arrival times.
sin	Slowly changing distribution of task arrival times.
4M2Tequ	Problem with four machines and two different types of tasks, applying the equal distribution of task arrival times.
ORIG	The original system, described in Chapter 5, is applied.
UR	The improvement proposal concerning the update rules is applied on its own. Explanations of the other improvement proposals can be found in Chapter 6.
UR+CFV+DC	The combination of the three improvement proposals **UR**, **CFV** and **DC** is applied.
ALL	The combination of all improvement proposals is applied.
MORLEY1, **MORLEY2**	The respective version of the market-based algorithms (Section 3.4.4) is applied.

Table 8.1: Explanation of the abbreviations used in this chapter.

case to be the easiest to reach a stable behavior. The expected number of tasks per time step is $\lambda = 0.16$.

2. $|\mathbf{M}| = |\mathbf{T}|$: The number of machines equals the number of tasks. In our experiments, the number of machines and types of tasks is two (**2M2T**). For the equal distribution, where all types of tasks appear with same probability, each machine should specialize on one type of task. Otherwise, if the different types of tasks appear with different probability, re-adaptation of the machines is required throughout the simulation. The expected number of tasks per time step is $\lambda = 0.12$.

3. $|\mathbf{M}| < |\mathbf{T}|$: The number of machines is smaller than the number of different types of tasks. In the studied cases, two machines are available to process four types of tasks (**2M4T**). This is the most difficult situation for the system. With respect to specialization, we expect that machines will generally not be able to reach a stable behavior, as reconfigurations will be required frequently in order to process all tasks. The expected number of tasks per time step is $\lambda = 0.08$.

In the following sections, these three cases will be analyzed in combination with the four task generation processes, **equ**, **dif**, **chg** and **sin** (Section 7.1).

8.1.1 Four Machines – Two Types of Tasks (4M2T)

Table 8.2(a) shows the experimental results of the makespan. The combination of all improvements (**ALL**) generally either requires the shortest makespan, or is very close to do so. This is mainly due to the proposed additional rules **BCT** and **IMB**. **ALL**, **BCT** and **IMB** always reach a better performance than the original system with respect to the makespan. The improvement proposals on existing rules, **UR**, **CFV** and **DC**, do not lead to a significant change in performance. **DC** is a modification on the dominance contest and can only have an effect, if many machines compete for the same task. For this problem instance, a dominance contest can never be held among more than four machines, **DC** does not lead to a real modification of the results. We expect to observe a stronger influence of **DC** in the larger problem instances in Section 8.2.

When considering the task generation processes where task appearance probabilities are not static over time, and in particular the abruptly changing distribution **chg**, we can observe a bigger difference in performance between **ALL**, **IMB** and **BCT** on one side, and the other approaches on the other side. **IMB** and **BCT** were proposed in order to speed up the adaptation process. This is especially helpful for the changing distributions. In the abruptly changing distribution, task appearance probabilities change at time step 1500 and machines need to adapt their thresholds to this new situation.

MS	4M2Tequ	4M2Tdif	4M2Tchg	4M2Tsin
ORIG	3382.8 ± 487.4	3245.7 ± 420.9	3698.7 ± 641.8	3479.6 ± 483.5
UR	3512.8 ± 496.8	3352.7 ± 773.5	3653.7 ± 629.3	3411.5 ± 457.6
CFV	3373.9 ± 480.2	3177.9 ± 208.3	3690.6 ± 641.7	3402.5 ± 441.8
DC	3400.1 ± 458.0	3380.0 ± 668.9	3732.9 ± 621.4	3445.6 ± 467.5
BCT	3171.2 ± 302.4	3114.0 ± 143.6	3096.8 ± 111.5	3200.9 ± 332.7
IMB	3084.9 ± 186.7	**3034.4 ± 21.6**	3030.9 ± 32.7	3064.1 ± 121.2
ALL	**3027.3 ± 16.3**	3037.6 ± 25.0	**3027.9 ± 19.1**	**3028.7 ± 18.3**

(a)

S	4M2Tequ	4M2Tdif	4M2Tchg	4M2Tsin
ORIG	1.37 ± 0.48	1.37 ± 0.59	1.35 ± 0.42	1.34 ± 0.53
UR	**1.24 ± 0.31**	**1.17 ± 0.24**	**1.24 ± 0.28**	**1.18 ± 0.28**
CFV	1.31 ± 0.45	1.27 ± 0.42	1.28 ± 0.33	1.27 ± 0.35
DC	1.28 ± 0.49	1.39 ± 0.55	1.32 ± 0.35	1.35 ± 0.62
BCT	1.41 ± 0.76	1.46 ± 0.62	1.78 ± 0.77	1.54 ± 0.78
IMB	1.28 ± 0.38	1.37 ± 0.56	2.06 ± 0.77	1.37 ± 0.34
ALL	1.36 ± 0.43	1.45 ± 0.46	2.06 ± 0.50	1.44 ± 0.36

(b)

Table 8.2: (a): The table shows the average makespan (**MS**) and its standard deviation for the **4M2T** problems. Smaller values are better. A comparison is made between the original system (**ORIG**), single applied modifications with the improvement proposals (**UR, CFV, DC, BCT, IMB**, for detailed description see Section 6), and the combination of all improvement proposals (**ALL**). The abbreviations **equ**, **dif**, **chg** and **sin** represent in this order the equal, the different, the abruptly changing, and the slowly changing distribution. In each column the approach with the best performance, i.e., shortest makespan, is highlighted. The values are averaged over 100 simulations. Tasks are generated during a window of time of 3000 time steps. Thus, the minimum reachable makespan is 3000.

The average number of setups per machine is shown in Table 8.2(b). As there are twice as many machines as types of tasks, very few setups are required in general. Thus, there is not a big difference between the approaches. The modification proposal concerning the update rules, **UR**, requires the fewest setups. But this may not only be considered as an advantage. On one side, unnecessary setups should be avoided, but on the other side, a system should not avoid a setup at any cost. In certain cases, especially for the

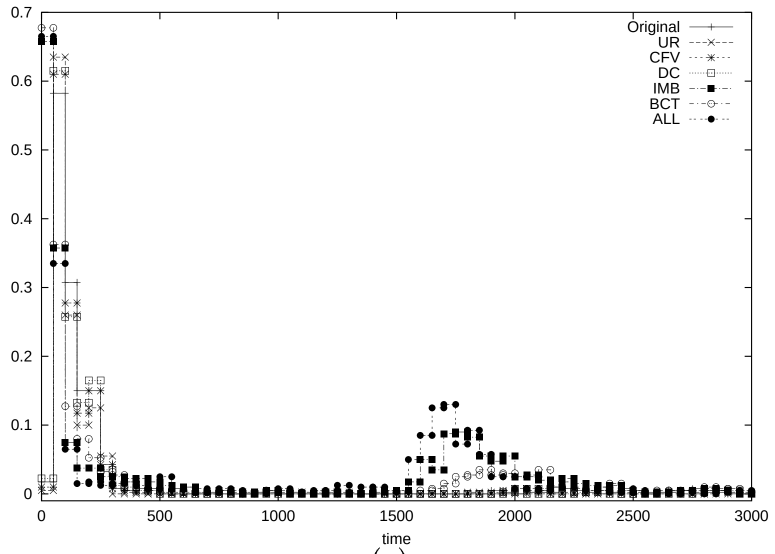
4M2Tchg	**Idle Time**	**Queue**	**Cycle Time**	**Alloc Time**
ORIG	1604.09 ± 593.26	6.94 ± 3.80	187.15 ± 126.60	1.40 ± 0.39
UR	1567.84 ± 577.11	6.05 ± 2.38	184.87 ± 124.33	1.36 ± 0.37
CFV	1606.06 ± 599.63	5.49 ± 3.94	185.07 ± 114.50	1.39 ± 0.33
DC	1649.23 ± 581.23	4.41 ± 3.17	196.49 ± 119.05	1.41 ± 0.42
BCT	1028.62 ± 122.76	3.05 ± 0.89	70.37 ± 27.82	0.42 ± 0.56
IMB	983.17 ± 101.13	1.77 ± 0.81	41.48 ± 14.39	0.58 ± 0.35
ALL	**964.61 ± 89.76**	**1.31 ± 0.51**	**32.89 ± 4.73**	**0.39 ± 0.04**

Table 8.3: The averages and standard deviation of the idle time, the queue size, the cycle time and the allocation time are shown for the **4M2T** problems with abruptly changing distribution. The values are averaged over 100 simulations.

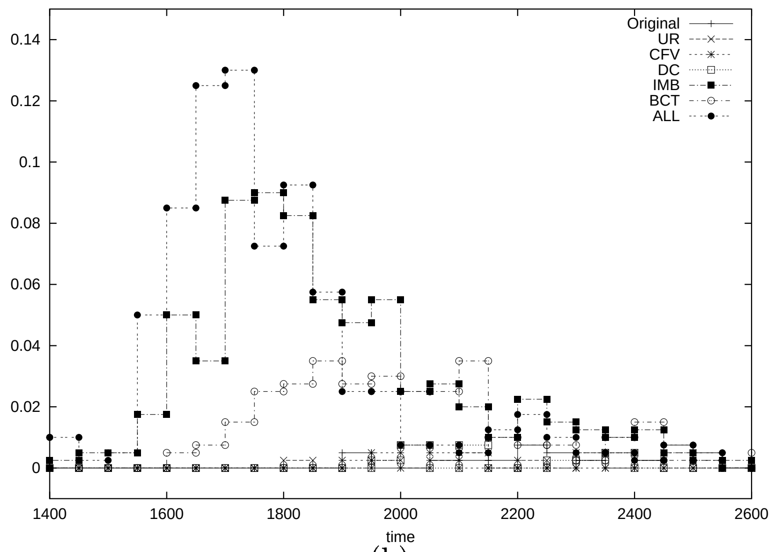
changing distributions, feasible adaptation requires setups. And adaptation can not be reached without a setup for another type of task. The used metrics confirm that a higher number of performed setups does not directly lead to a loss in performance. The reasons for this will be analyzed more in detail later on.

Table 8.3 shows averages of the idle time and queue size of the machines and the cycle and allocation time of tasks. The cycle time expresses average time a task spent in the system and the allocation time describes the time until a task is allocated to a machine. In all these measures the combination of all improvement proposals performs best, what is mainly due to the improvement proposal **IMB**, which performs only slightly worse. As shown by the measure of the cycle time, in the original system, tasks in average remain in the system about five times longer compared to the system where all improvement proposals are applied. This is a very important quality for real world applications, as for example a truck painting facility. In a physical world, tasks are related to the requirement of physical space. A longer cycle time is related to higher requirement of space. The same holds for the average queue size per machine, which is also significantly shorter for the combination of all improvement proposals. **ALL**, **IMB** and **BCT** have the shortest makespan what can be explained with the idle time. We suppose that the reduction of the idle time results from a faster adaptation. Given that the average number of setups is similar for all approaches, a difference in makespan results from a difference in idle time. Thus, **IMB** and **BCT** are effective improvements for the reduction of the cycle time and the idle time, directly influencing the makespan.

The level of adaptation of the system can be analyzed, observing the distribution of setups over time, shown in Figure 8.1(a), while figure Fig-



(a)



(b)

Figure 8.1: The average number of setups per machine per 50 time steps is displayed for the **4M2Tchg** problem. Values are summed up over 50 time steps and the result is taken as one point in the plot. The values are averaged over 100 simulations. (a) shows this over 3000 time steps, whereas (b) highlights the period of time, after probabilities of task appearances change abruptly at time step 1500.

ure 8.1(b) shows the same distribution focusing on the switch of task appearance probabilities at time step 1500. The improvement proposal **IMB**

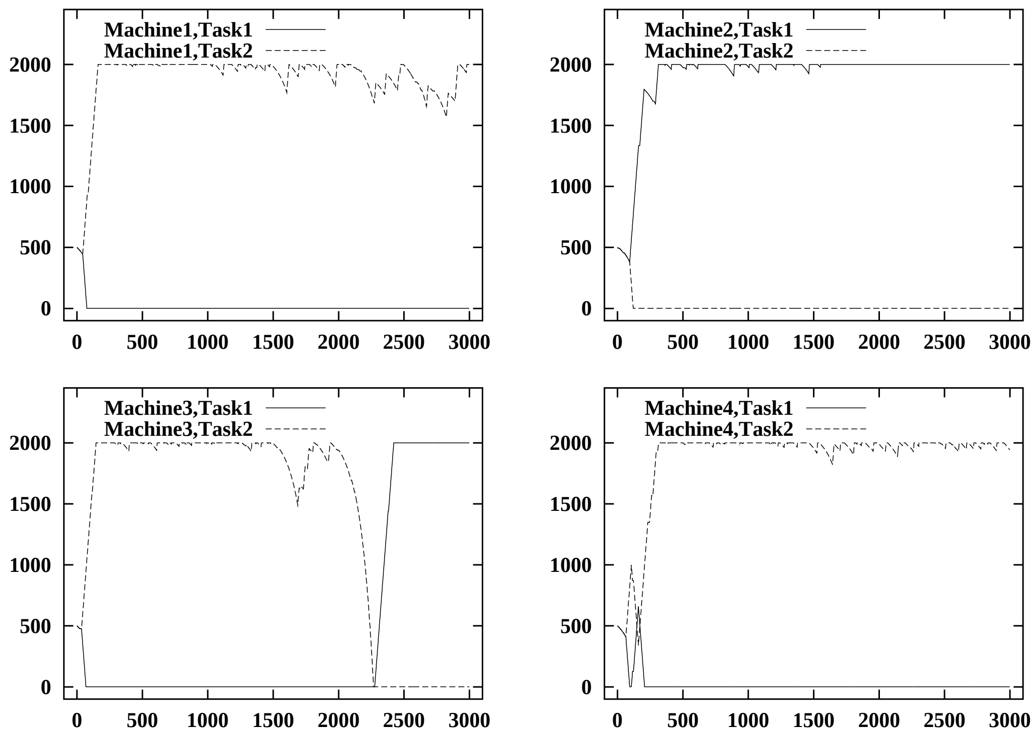


Figure 8.2: Example for the threshold values of the four machines for the **4M2Tchg** problem, when the original system is applied. A low threshold value refers to a high grade of specialization and vice versa. The lower and the upper threshold bound for the threshold values are $\Theta_{min} = 1, \Theta_{max} = 2000$. Probabilities of task appearances change abruptly at time step 1500.

and the combination of improvements, **ALL**, react comparably fast to the new situation. Setups are mainly performed in the period $[1550, 1900]$. **BCT** reacts more slowly, mainly performing tasks in the period $[1750, 2150]$. The other approaches do not seem to re-adapt at all or to adapt only very slowly. In fact, **IMB** and **BCT** both reduce the threshold values of machines, making it easier to switch from one task to the other and thereby speeding up the re-adaptation process.

The behavior of the threshold values over time for each machine, is shown in Figure 8.2 for the original system, in Figure 8.3 for **IMB** and in Figure 8.4 for **ALL**. These figures represent one typical iteration of a simulation, and not the average over several iteration. The temporal behavior of threshold values cannot be sensefully averaged over several iterations, as the result would not be a correct representation of the system. We analyzed several iterations and chose one for each approach considered to be representative. The temporal behavior of the threshold value offers a multiplicity of possibil-

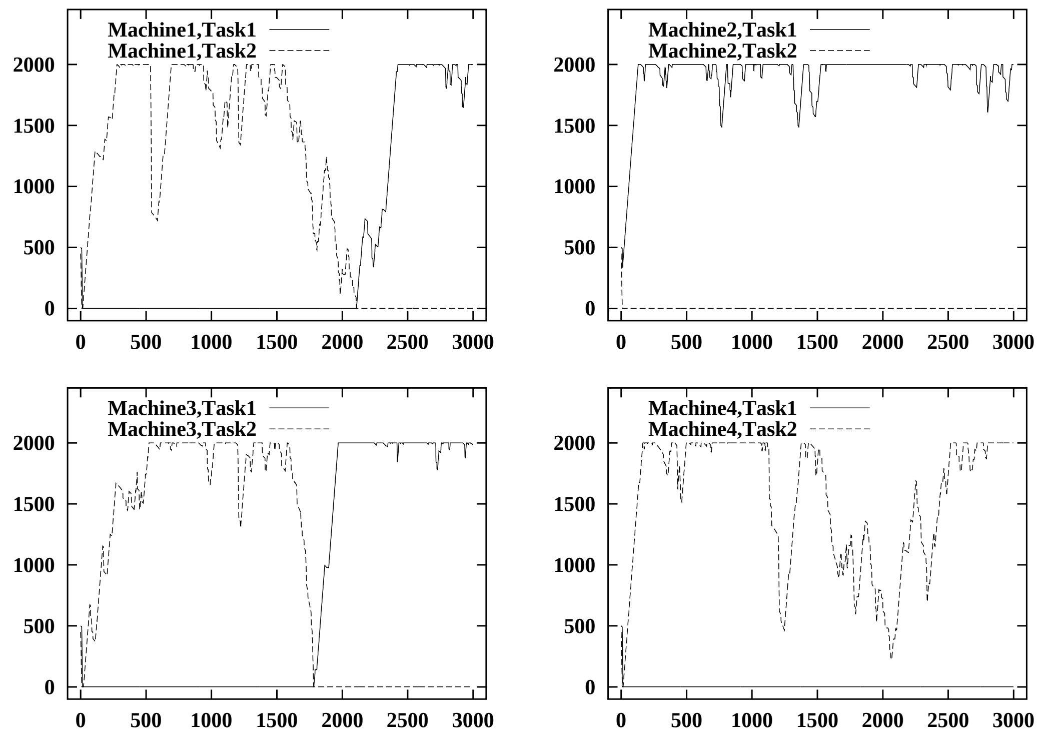


Figure 8.3: Example for the threshold values of the four machines for the **4M2Tchg** problem, when the improvement proposal **IMB** is applied. A low threshold value refers to a high grade of specialization and vice versa. The lower and the upper threshold bound for the threshold values are $\Theta_{min} = 1, \Theta_{max} = 2000$. Probabilities of task appearances change abruptly at time step 1500.

ities to analyze the system's behavior in detail. For instance, one can deduce the type of task a machine is specialized for simply by differing between the respective threshold values. A low threshold value refers to a high grade of specialization and vice versa. Additionally, the current state of a machine can be recognized, according to the three update rules (see Equation 5.7 in Section 5.2 for more details), observing whether threshold values diminish, increase or remain constant[1]. Otherwise, if at least one threshold value increases or remains constant at the upper bound Θ_{max}, the respective machine cannot be idle.

For the **4M2Tchg** problem, in the period $[0, 1500]$, tasks of type 1 are generated with a three times higher probability than tasks of type 2. Then, for the next 1500 time steps, in the period $[1500, 3000]$, task arrival probabil-

[1]If the update rule **BCT** is applied, the temporal behavior of all other machines needs to be considered as well.

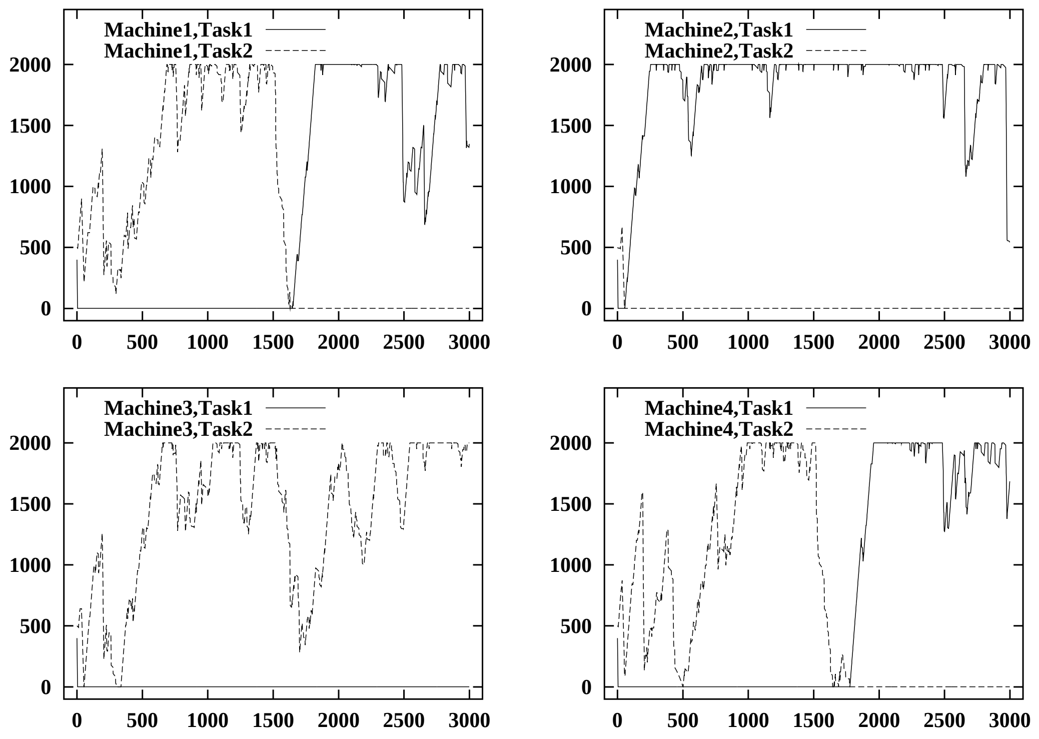


Figure 8.4: Example for the threshold values of the four machines for the **4M2Tchg** problem, when all improvements are applied. A low threshold value refers to a high grade of specialization and vice versa. The lower and the upper threshold bound for the threshold values are $\Theta_{min} = 1, \Theta_{max} = 2000$. Probabilities of task appearances change abruptly at time step 1500.

ities switch among the two types, so that tasks of type 2 are generated three times more often than tasks of type 1. Thus, as four machines are available, during the period $[0, 1500]$, three machines should specialize for tasks of type 1, and only one machine should process tasks of type 2. Afterwards, in the period $[1500, 3000]$, machines should re-adapt, so that only one machine stays specialized for tasks of type 1, and all other machines process tasks of type 2. Considering this, the original system performs rather poorly. It succeeds in initially adapting to task arrivals as it should. From Figure 8.2, we can deduce that three machines start performing tasks of type 1, and only machine 2 specializes in tasks of type 2. But, after time step 1500, only machine 3 re-adapts to the new situation, so that each type of task is performed by two machines. Additionally, we can point out that machine 3 takes very long to adapt itself. Shortly before time step 2000 machine 3 gets idle as there are no tasks of type 1 in the system any more. Then, its respective threshold value for tasks of type 2 diminishes slowly, following the update rule for idle machines. Only after about 300 time steps, the respective threshold value is low enough to let machine 3 bid for tasks of type 2.[2] This example of the original system has been chosen, as it represents the typical performance. Sometimes, the performance is even worse, as the initial adaptation of three machines processing tasks of type 1 is kept, even after time step 1500. Thus the system is not able to readapt at all. Very seldom, the system correctly re-adapts to the changed probability mix of task arrival rates, but even in these cases, adaptation is rather slow and not satisfying.

As discussed above, **IMB** leads to the biggest improvement, compared to the original system. Figure 8.3 shows the threshold values of the four machines over time. The machines initially adapt correctly to the probability mix of task arrival rates. While machine 2 specializes on the rarely appearing type of task 2, the three other machines perform tasks of type 1. Later, when task arrival rates change, the four machines succeed to re-adapt. First, machine 3 starts performing tasks of type 2 from approximately time step 1800 on. Later, from approximately time step 2100 on, machine 1 as well starts to perform tasks of type 2. Machine 3 re-adapts faster, as it remains idle and does not perform tasks of type 1 any more from approximately time step 1580 on. It remains idle and is offered tasks of type 2, which it doesn't bid for due to a high threshold value. The improvement proposal **IMB** leads to a fast re-adaptation. Machine 1 takes a little longer, as it still accepts eight tasks of type 1, before the threshold value for tasks of type 2 is low enough to bid for respective tasks. But in any case, even if here it takes comparably long, the machines succeed to adapt to the new probability mix of task arrival rates. The threshold value for tasks of type 2 of machine 4 are diminished in parallel to machine 1. In many other iterations with improvement proposal **IMB**, the adaptation process was much faster. In fact, the shown example performs even worse than average. Nevertheless, we have chosen this example, in order to show, that even in rather difficult situations, the system succeeds to adapt to the abruptly changing distribution.

The performance of the combination of all improvement proposals, **ALL**, is for all 4M2T problem instances is very similar to the one reached by **IMB**, but the adaptation process is even faster. Figure 8.4 shows the threshold values of the each machines over time. The shown example is very typical for the behavior of **ALL**. Due to the two proposed additional rules, **IMB** and **BCT**, the threshold values change very quickly and correctly adapt to changing demands.

[2]Given the value of the parameter δ_3, and assuming that a threshold value is only diminished by the update rule for idle machines, a threshold value is diminished from 2000 to 1 in 306 time steps.

8.1.2 Two Machines – Two Types of Tasks (2M2T)

The **2M2T** problems may be considered to be more difficult than the **4M2T**. For the **2M2Tequ** problem, where task arrival rates are equally distributed, each machine should specialize for one type of task. For all other **2M2T** problems, such a simple solution is not possible. For instance, the **2M2Tdif** problem generates two types of tasks with different task arrival rates. Thus, one machine should specialize on the more frequently arriving type of task, whereas the other machine should switch from one task to the other periodically.

Table 8.4 shows the experimental results of the average makespan and the average number of required setups per machine. As in the **4M2T** problems, with respect to the makespan, the combination of all improvement proposals either performs best, or second best behind the **UR** approach. In general, the difference in performance among the different approaches is small. Only for the **2M2Tdif** approach, **ALL** reaches a notably better performance than all other approaches. In all **2M2T** problems, except for the problem with different distribution of task arrival rates, **BCT** performs worst, with particularly bad performance for the assumed easy **2M2Tequ** problem, where both tasks are released to the environment with equal probability.

One indicator for the bad performance of **BCT** can be found in Table 8.4(b), where the required number of setups per machine are shown. For all **2M2T** problems, the **BCT** approach requires much more setups than most other approaches and has a particularly high standard deviation. For the **dif** and the **chg** distribution this also holds for the **IMB** approach. This instability is due to the fact, that machines often do have very low threshold values for both types of tasks, as will be shown later. The improvement proposal concerning the update rules (**UR**) performs very well for the **2M2T** problems. It requires very few setups in general, specifically for the **dif** and the **chg** distribution.

For the equal distribution of task arrival rates, Table 8.5 shows the idle time and the average queue size per machine, the average cycle and allocation time per machine. As **BCT** requires far more setups than all other approaches, **BCT** also spends much more time setting up for tasks. This results in the longer makespan.

Figure 8.5 shows the temporal distribution of setups per machine for the **2M2Tequ** problem. Except for **BCT**, all approaches succeed to adapt to the environment after approximately 1000 time steps. **BCT** on the other side, in average requires setups throughout the simulation.

BCT exhibits two different classes of behavior,as can be seen in Figure 8.6 where the threshold values are plotted over time. In Figures 8.6(a) and 8.6(b),

MS	2M2Tequ	2M2Tdif	2M2Tchg	2M2Tsin
ORIG	3245.7 ± 159.6	4494.5 ± 280.4	3758.3 ± 315.0	3273.0 ± 246.5
UR	**3198.7 ± 125.1**	4347.4 ± 240.6	**3675.9 ± 186.2**	3231.2 ± 140.1
CFV	3260.6 ± 165.0	4414.5 ± 292.0	3729.5 ± 264.6	3275.0 ± 249.0
DC	3271.0 ± 218.4	4485.9 ± 275.9	3743.5 ± 253.6	3267.6 ± 223.3
BCT	3719.8 ± 671.2	4258.8 ± 292.1	3911.1 ± 407.3	3580.1 ± 632.5
IMB	3205.8 ± 120.9	4305.9 ± 319.1	3795.1 ± 289.7	3221.8 ± 181.5
ALL	3203.9 ± 127.6	**4095.2 ± 239.1**	3714.8 ± 213.9	**3201.4 ± 114.9**

(a)

S	2M2Tequ	2M2Tdif	2M2Tchg	2M2Tsin
ORIG	2.40 ± 2.09	5.16 ± 8.29	6.25 ± 10.89	3.67 ± 6.56
UR	2.07 ± 1.67	**1.77 ± 1.28**	**1.88 ± 1.52**	2.19 ± 1.78
CFV	2.81 ± 3.08	3.23 ± 4.60	5.46 ± 9.64	3.70 ± 5.30
DC	3.02 ± 4.10	4.82 ± 7.74	4.99 ± 8.42	4.15 ± 5.57
BCT	19.25 ± 21.93	12.20 ± 12.70	11.03 ± 13.79	13.91 ± 18.61
IMB	2.20 ± 1.90	13.06 ± 12.89	8.56 ± 10.23	2.98 ± 4.72
ALL	**1.52 ± 0.65**	5.75 ± 1.65	4.04 ± 1.48	**1.74 ± 0.69**

(b)

Table 8.4: The average makespan (**MS**) and its standard deviation for the **2M2T** problems is shown in (a). Smaller values are better. (b) shows the average number of performed setups per machine. A comparison is made between the original system (**ORIG**), single applied modifications with the improvement proposals, and the combination of all improvement proposals (**ALL**). **equ**, **dif**, **chg** and **sin** represent the four task generation processes. In each column the approach with the best performance, i.e., shortest makespan, is highlighted. The values are averaged over 100 simulations. Tasks are generated during a window of time of 3000 time steps. Thus, the minimum reachable makespan is 3000. As machines are not initially set up for any type of task, at least one setup is required per machine.

both machines succeed to adapt specializing for one type of task. In the second case, as shown by Figures 8.6(c) and 8.6(d), the machines are not able to adapt and continuously switch among the types. The improvement proposal **BCT** diminishes the threshold values of both machines, in case both machines do not bid for a task. The motivation for this proposal is an increase of adaptation speed, particularly for changing environments, where task arrival rates change over time. For the **2M2T** problems, **BCT** does not lead to increased adaptation speed. Machines are seldom able to make a stable decision for one type of task and thereby specialize.

However, even though it contains the improvement proposal **BCT**, the combination of all improvements behaves stable and succeeds to adapt. In combination with the other improvement proposals, the instability of **BCT**

2M2Tequ	**Idle Time**	**Queue**	**Cycle Time**	**Alloc Time**
ORIG	240.09 ± 99.17	13.01 ± 3.36	161.59 ± 73.52	4.31 ± 4.02
UR	245.14 ± 98.11	7.31 ± 1.86	141.73 ± 68.88	3.77 ± 3.69
CFV	242.34 ± 91.70	10.10 ± 2.65	171.59 ± 91.99	4.47 ± 4.72
DC	241.73 ± 114.40	9.75 ± 2.53	182.93 ± 122.95	5.23 ±6.81
BCT	**184.71 ± 131.31**	17.19 ± 6.98	420.79 ± 354.89	1.56 ± 0.62
IMB	240.76 ± 111.07	9.21 ± 2.19	126.75 ± 52.00	2.18 ± 2.17
ALL	237.73 ± 103.64	**6.33 ± 1.40**	**116.76 ± 48.09**	**0.96 ± 0.09**

Table 8.5: The averages and standard deviation of the idle time, the queue size, the cycle and the allocation time are shown for the **2M2T** problems with equal distribution.

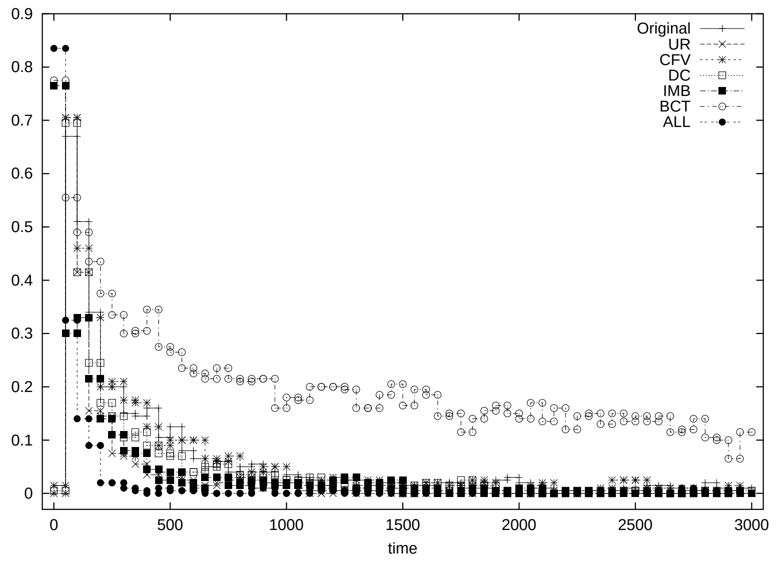
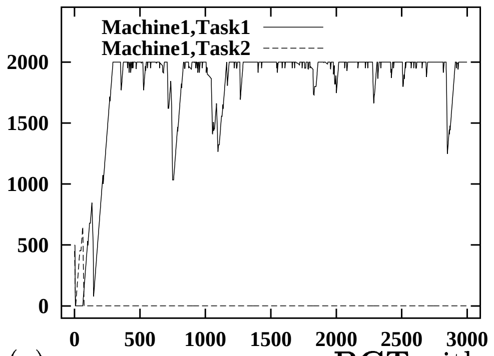
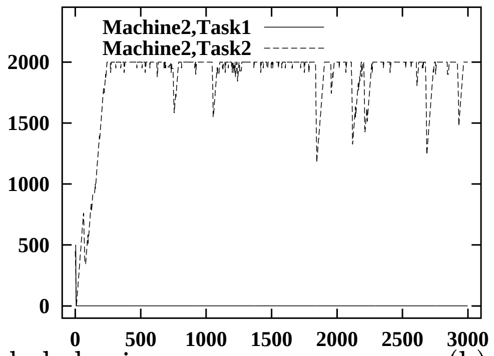


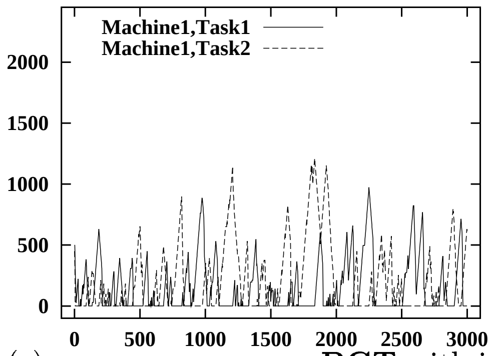
Figure 8.5: The average number of setups per machine per 50 time steps is displayed for the **2M2Tequ** problem. Values are summed up over 50 time steps and the result is taken as one point in the plot. The values are averaged over 100 simulations.

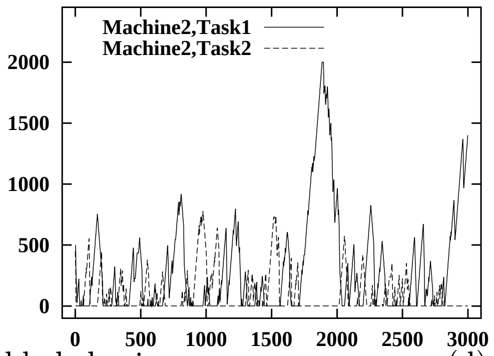
disappears. This can be seen in Figure 8.7, where the temporal behavior of the threshold values is displayed for the original system (Figures 8.7(a) and 8.7(b)) and **ALL** (Figures 8.7(c) and 8.7(d)). Each machine specializes for one type of task and does not change its specialization throughout the simulation. The fluctuation of the higher threshold values for **ALL** mainly emerges due to the additional rules **BCT** and **IMB**.





(a) **BCT** with stable behavior (b)





(c) **BCT** with instable behavior (d)

Figure 8.6: The two different cases of the temporal behavior of the threshold values for the **2M2Tequ** problem are displayed. (a) and (b) show good adaptive behavior, where each machine specializes for one type of task. This is not the case for (c) and (d), where the machines do frequently change the type of task they perform.

8.1.3 Two Machines – Four Types of Tasks (2M4T)

The **2M4T** problems contain two times as many types of tasks as machines. This makes the problem very difficult. In general, a machine should not specialize for too long on one task, as this would lead to a negligence of other tasks in the system. Thus, machines have to permanently re-adapt themselves to fulfill the current requirements of the environment.

Table 8.6 shows the experimental results of the average makespan (**MS**) and the average number of required setups per machine (**S**). In all cases, the combination of all improvements performs best, and the original system performs worst. The modification of the update rules, **UR**, and the additional rule **BCT**, decrease the number of required setups by approximately 25%, and therefore lead to a better performance. **CFV** and **DC**, The modifications on the calculation of the force value and the dominance contest,

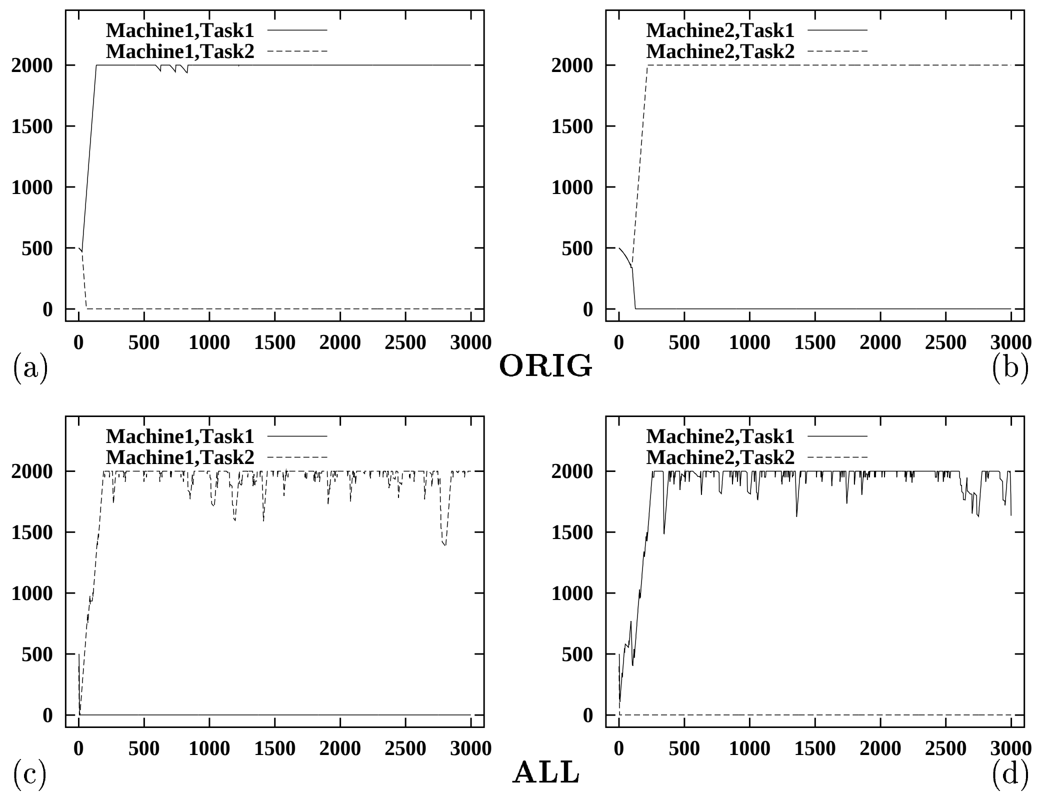


Figure 8.7: Example of the temporal behavior of the threshold values for the original system (**ORIG**, (a) and (b)) and the combination of all improvement proposals (**ALL**, (c) and (d). Both approaches succeed to adapt correctly. Each machine specializes for one type of task.

again do not result in a significant change of performance for the same reasons given in Section 8.1.1. The other additional rule (**IMB**) reduces the makespan, but does not diminish the number of required setups. Table 8.7, where the idle time, the queue size, the cycle time and the allocation time for the **2M4Tequ** problem are given, shows a significantly shorter average queue size for the additional rules, the modification of the update rules and the combination of all improvement proposals. **BCT** and **IMB** reduce the time that machines are idle. Additionally, for **BCT** and **ALL**, tasks are allocated much faster to machines. This was predictable, as **BCT** reduces the thresholds of all machines, in case a task remains unallocated.

Figure 8.8 shows the average number of setups per machine per 100 time steps over time for the **2M4Tequ** problem. As re-adaptation is required throughout the whole simulation, the machines do not reach a state where setups are not required. Instead, each approach converges towards a cer-

MS	**2M4Tequ**	**2M4Tdif**	**2M4Tchg**	**2M4Tsin**
ORIG	5032.0 ± 377.6	4283.5 ± 495.6	4510.5 ± 474.5	4971.8 ± 326.2
UR	4081.4 ± 319.6	3765.9 ± 275.5	3771.8 ± 291.4	4093.4 ± 331.3
CFV	4942.2 ± 373.2	4270.0 ± 408.2	4477.4 ± 401.0	4980.2 ± 427.0
DC	4918.9 ± 378.0	4270.0 ± 458.5	4420.0 ± 393.9	4997.6 ± 395.3
BCT	4062.8 ± 375.5	3917.6 ± 562.1	3937.0 ± 451.2	4094.1 ± 329.1
IMB	4561.8 ± 486.1	3917.9 ± 480.3	3999.7 ± 534.3	4386.3 ± 504.9
ALL	**3893.6 ± 359.8**	**3546.3 ± 280.7**	**3570.2 ± 330.1**	**3842.3 ± 341.6**

(a)

S	**2M4Tequ**	**2M4Tdif**	**2M4Tchg**	**2M4Tsin**
ORIG	83.61 ± 7.70	60.39 ± 11.89	68.08 ± 9.93	82.55 ± 6.47
UR	55.98 ± 7.31	44.59 ± 5.94	44.70 ± 6.58	56.09 ± 7.43
BCT	57.63 ± 5.62	48.15 ± 8.73	49.74 ± 8.05	57.95 ± 6.03
IMB	72.11 ± 11.78	51.81 ± 12.24	54.48 ± 14.58	67.94 ± 13.09
CFV	82.28 ± 7.82	60.16 ± 10.45	66.34 ± 9.42	82.05 ± 7.78
DC	81.98 ± 7.29	60.76 ± 10.45	65.38 ± 9.30	82.36 ± 6.81
ALL	**49.58 ± 4.33**	**39.30 ± 4.30**	**40.10 ± 4.42**	**48.74 ± 3.84**

(b)

Table 8.6: The makespan (**MS**, (a)) and the number of required setups per machine (**S**, (b)) for the **2M4T** problems are shown for the four task generation processes **equ**, **dif**, **chg** and **sin**.

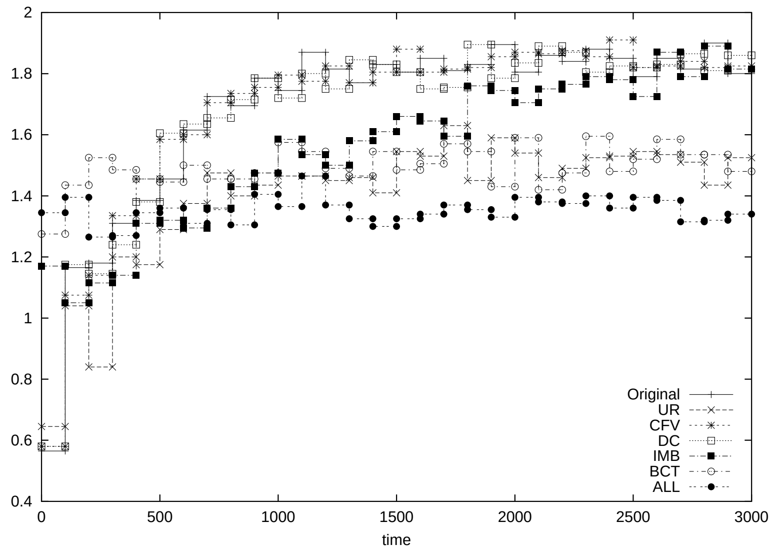


Figure 8.8: The average number of setups per machine per 100 time steps is displayed for the **2M4Tequ** problem. Values are summed up over 50 time steps and the result is taken as one point in the plot. The values are averaged over 100 simulations.

2M4Tequ	**Idle Time**	**Queue**	**Cycle Time**	**Alloc Time**
ORIG	363.9 ± 176.9	16.57 ± 10.74	926.3 ± 178.6	136.6 ± 6.8
UR	324.6 ± 124.7	**12.18 ± 7.37**	538.9 ± 142.9	114.5 ± 7.4
CFV	345.5 ± 181.1	18.67 ± 11.59	894.7 ± 167.9	137.1 ± 5.9
DC	351.4 ± 159.7	18.15 ± 11.29	888.1 ± 167.8	137.5 ± 6.6
BCT	290.9 ± 186.9	16.27 ± 8.39	485.8 ± 142.7	**6.4 ± 0.3**
IMB	**272.8 ± 132.7**	12.34 ± 8.05	611.9 ± 226.6	107.2 ± 19.2
ALL	339.3 ± 198.7	12.42 ± 5.69	**387.9 ± 115.6**	7.2 ± 0.3

Table 8.7: The averages and standard deviation of the idle time, the queue size and the cycle time are shown for the **2M4T** problem with equal distribution. The last column, **Alloc Time**, gives the average time until a task is allocated to a machine. The values are averaged over 100 simulations.

tain value, that is lower for **ALL**, **UR** and **BCT**, and higher for the other approaches.

For the **2M4T** problems, we do not show plots of the temporal behavior of the threshold values. The need to continuously re-adapt, makes the threshold values change very dynamically in time, making an anlaysis on the threshold values' dynamics very difficult and complex.

8.2 Large Problem Instances

The small problem instances provide good study casesto analyze the general behavior of the insect-based approach to the DTA, as it simplifies a detailed survey of single machine behavior. Additionally, small problem instances allow an elementary study of the temporal behavior of threshold values, which is very difficult for larger problem instances with a higher number of machines and types of tasks. However, small problems lack of representing real-world problems. In an industrial environment, like a painting facility, there are in general more machines and tasks than in the examined small problem instances. Therefore, in this section we analyze the insect-based approach for larger problems, which contain either 10 or 25 machines and 10 different types of tasks. A comparison is made between the original system, the system containing our improvement proposals, and the real world proven, market-based multi-agent system of Morley, which was explained in detail in Section 3.4.4. In addition to the Morley system, we examine a slightly modified version of that approach, that is also explained in Section 3.4.4. The modifications of existing rules, **UR**, **CFV** and **DC**, are only considered in combination, as their influence on the performance is much smaller than the influence of our additional rules, **BCT** and **IMB**.

First, we analyze the behavior of the original insect-based system without restrictions concerning the maximum queue length. In this case, we do not consider the two market-based approaches, as, in their current form, they do not produce feasible schedules without a queue size restriction. Therefore, we additionally examine systems with a maximum queue length of 10 tasks per machine and compare the performance of the insect-based approaches to the market-based approaches. For the 10 machine problems, the expected number of tasks per time step is $\lambda = 0.4$. For the 25 machine problems we used the parameter $\lambda = 1.0$.

8.2.1 No Queue Length Restriction

10 Machines – 10 Types of Tasks (10M10T)

10M10T problems are in general comparable to **2M2T** problems, as the number of machines equals the number of tasks. Nevertheless, **10M10T** problems should be considered as more difficult than **2M2T** problems. Each machine is offered 10 different types of tasks. Thus, a machine has more choices. Consider the example of equal distribution of task arrivals. In the ideal case, each machine specializes for a different type of task. For the **2M2Tequ** problem, a machine can either make the "right" or the "wrong" choice, depending on the other machine. This means, if one machine specializes for tasks of type 1, then the right choice for the other machine should be to only accept tasks of type 2. It is not that easy for the **10M10Tequ** problem. For instance, if nine machines did already specialize for different types of tasks, then the last machine has nine possibilities to make a "wrong" choice and only one possibility to make the "right" choice.

Table 8.8 shows the experimental results of the average makespan (**MS**, (a)) and the average number of required setups per machine (**S**, (b)). Additionally, for the **10M10Tequ** problem, the idle time, the average queue size, the cycle time and the allocation time are given by Table 8.8(c). The performance of the different approaches can be separated in two groups. The original system and the modifications on the existing rules perform very poorly. They require a very long makespan and far more setups than the other approaches. A worst case for the number of performed setups can be defined as the schedule that requires a setup for each arriving task. And in fact, for the equal and the slowly changing distributions, the original system performs not so far away from this worst case. Per machine, there are in average 120 tasks released throughout the simulation. For the equal and the slowly changing distributions, the original system requires more than 80 setups per

MS	**10M10Tequ**	**10M10Tdif**	**10M10Tchg**	**10M10Tsin**
ORIG	5731.0 ± 908.0	5003.6 ± 883.3	4973.1 ± 1069.3	5648.5 ± 928.4
UR+CFV+DC	4917.2 ± 293.8	4494.0 ± 372.8	4536.9 ± 306.9	4887.5 ± 270.5
BCT	3214.0 ± 293.1	3237.1 ± 98.1	3224.3 ± 141.2	3225.8 ± 300.1
IMB	3091.6 ± 214.1	3212.7 ± 104.7	3170.9 ± 76.4	**3074.2 ± 62.7**
ALL	**3069.7 ± 41.1**	**3140.8 ± 67.6**	**3152.3 ± 70.3**	3075.4 ± 38.9

(a)

S	**10M10Tequ**	**10M10Tdif**	**10M10Tchg**	**10M10Tsin**
ORIG	82.49 ± 25.47	42.05 ± 20.55	54.06 ± 25.68	81.43 ± 24.72
UR+CFV+DC	70.36 ± 6.71	54.01 ± 11.95	56.05 ± 5.91	70.64 ± 6.25
BCT	11.66 ± 6.31	10.98 ± 3.38	12.26 ± 5.36	12.39 ± 6.61
IMB	7.83 ± 7.31	**8.71 ± 1.59**	**8.89 ± 1.76**	7.81 ± 5.82
ALL	**5.01 ± 1.13**	9.83 ± 0.96	9.80 ± 1.08	**4.88 ± 1.07**

(b)

10M10Tequ	**Idle Time**	**Queue**	**Cycle Time**	**Alloc Time**
ORIG	1077.0 ± 315.6	19.22 ± 12.37	873.3 ± 309.3	70.8 ± 21.1
UR+CFV+DC	**685.1 ± 173.8**	16.03 ± 11.01	679.3 ± 121.5	69.2 ± 4.6
BCT	851.7 ± 178.7	4.74 ± 0.86	99.2 ± 56.9	1.6 ± 0.3
IMB	851.6 ± 148.7	1.69 ± 0.37	48.6 ± 39.8	3.3 ± 4.8
ALL	919.6 ± 62.5	**1.19 ± 0.21**	**41.1 ± 4.2**	**1.0 ± 0.08**

(c)

Table 8.8: The makespan (**MS**, (a)) and the number of required setups per machine (**S**, (b)) for the **10M10T** problems without queue length restrictions are shown for the four task generation processes (**equ**, **dif**, **chg** and **sin**). (c) additionally gives the results on the idle time, the average queue size, the cycle time and the allocation time for the **10M10Tequ** problem.

machine in average. This means, for more than 75% of the tasks, a reconfiguration is required. For the mentioned two distributions, the combination of all improvement proposals on the other, requires only about 5 setups per machine, what refers to less than 4.2% of the tasks causing a reconfiguration. This difference of performance results from the additionally proposed rules **BCT** and **IMB**, which perform approximately as well as the combination of all improvement proposals. Obviously, in the case of the **10M10Tequ** problem, the time a machine stays idle is not a good performance measure. The combination of the improvement proposals that concern existing rules, **UR+CFV+DC**, has the shortest idle time, but performs very poorly anyway. Anyway, this approach results in a general improvement compared to the original system. Still, there is a big gap between the performance of **UR+CFV+DC** in comparison with **BCT**, **IMB** and **ALL**.

The big difference in performance is also observable in Figure 8.9, where the temporal distribution of setups per machine for the **10M10Tequ** problem is shown. While **BCT**, **IMB** and **ALL** converge more or less fast to a low value, the other three approaches do not succeed in reaching a stable

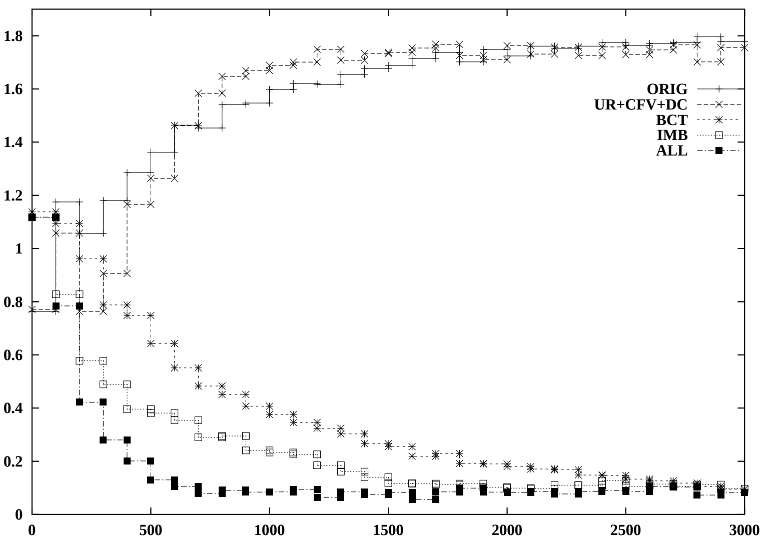


Figure 8.9: The average number of setups per machine per 100 time steps is displayed for the **10M10Tequ** problem. Values are summed up over 100 time steps and the result is taken as one point in the plot. The values are averaged over 100 simulations.

behavior and even need an increasing amount of reconfigurations throughout the simulations.

Figure 8.10 shows four cases for the dynamics of the threshold values for the **10M10Tequ** problem. **ORIG** and **ALL** are compared, and for each approach the threshold values of two machines are illustrated. Displaying only two of the ten machines surely can not represent the complete system. Nevertheless, it can give a reasonable idea of the system behavior. For the original system, the threshold values are very instable. This is indicated by the Figures 8.10(a) and 8.10(b). For approximately the first 300 time steps, machines are able to specialize on one type of task. Afterwards, they start performing other types of tasks. For instance, this can happen due to tasks that are waiting to be allocated already for a long time and therefore have a big stimulus intensity. Even if the respective threshold value is very high, the stimulus intensity of these tasks can overcome the threshold, so that a machine bids for them. It could also be possible that there is currently no task of the specialized type in the system, so that the respective machine stays idle. The result is that throughout the whole simulation, the machines do not specialize for any particular type of task, confirmed by the fact that for no type of task a low threshold value is reached. The machines frequently start setting up for and processing different types of tasks, so that no type is really preferred. However, if all improvement proposals are applied, these

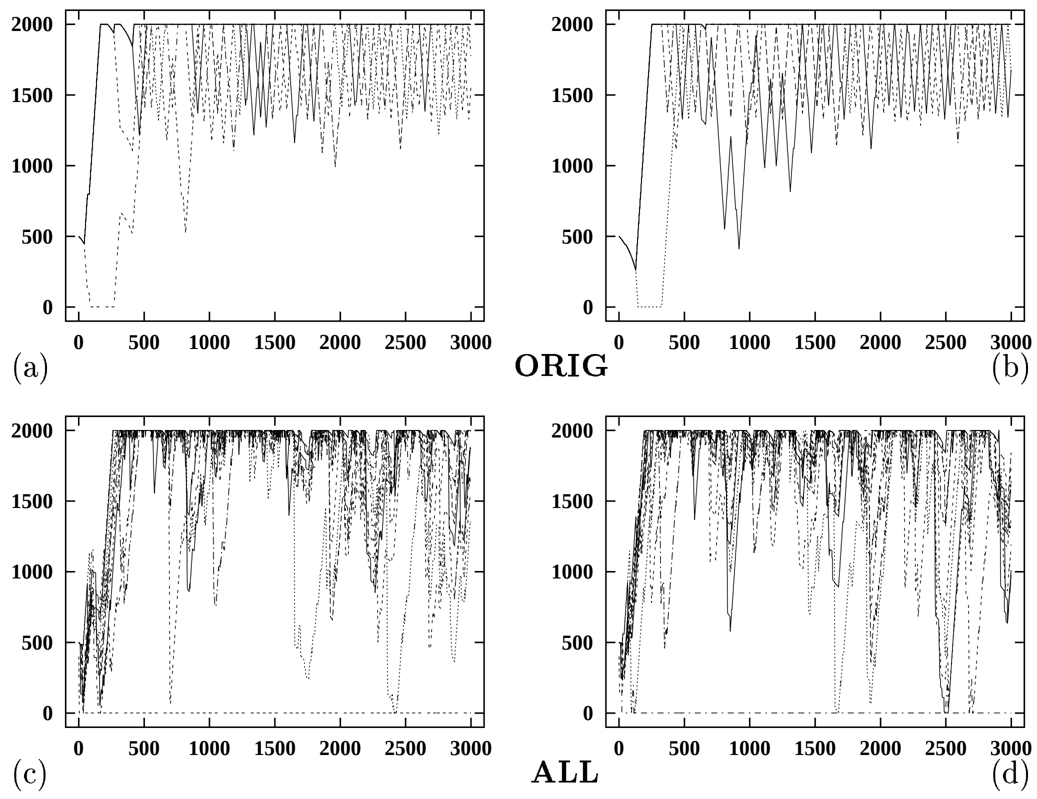


Figure 8.10: The figures show the dynamics of the threshold values throughout the simulation. A comparison is made between the original system ((a) and (b)) and the combination of all improvement proposals ((c) and (d)) for the **10M10Tequ** problem without restrictions on the queue length. For each approach, two representative machines have been chosen.

problems are overcome. The machines, that apply all improvement proposals and serve as example for the Figures 8.10(c) and 8.10(d), succeed to specialize on one task only. And even though other thresholds reach very low values, the machines continue performing the one task they are specialized for.

25 Machines – 10 Types of Tasks (25M10T)

In the same way we compared the **10M10T** problems to the **2M2T** problems, we may compare the **25M10T** problems to the **4M2T** problems. The **25M10T** problems contain more machines than different types of tasks. Thus, it can be considered as less difficult than **10M10T** problems. Table 8.9 summarizes the experimental results for the **25M10T** problems. In general, the performance of the approaches may be split in two groups again, as done for the **10M10T** problems. While the original system and the modifications on the existing rules perform rather poorly again, the additional rules, **BCT** and **IMB**, and the combination of all improvement proposals, result in a significantly better performance. As the number of machines is bigger than the number of types of tasks, only very few setups are required for all approaches. Thus, the difference in performance emerges in the difference of the time that machines stay idle. This is shown by the Table 8.9(c), which displays the idle time, the queue size, the cycle time and the allocation time for the **25M10Tdif** problem. For this problem, the combination of the modification on existing rules, **UR+CFV+DC**, performs particularly poor, having a makespan that is approximately 17% worse than the original system. The reason for this is the extremely long time that machines stay idle. We assume, that this emerges from the fact that machines seem to prefer staying idle rather than performing setups. Due to the modifications, setups are prevented whenever it is possible. Obviously, in the case of the **25M10Tdif** problem this results in a stiff behavior that lacks the ability to adapt. This loss of performance for the modification proposals occurs mainly for easier problems, where the number of machines exceeds the number of different types of tasks. For more difficult problems the modifications all resulted in an improved performance.

For the **25M10T** problems a study of the dynamics of the threshold values is not required. For all approaches, machines specialize very quickly. This specialization is probably too rigid for the original system and the combination of the modification on existing rules. However, an analysis of the threshold values' dynamics does not provide a deeper insight in this problem.

8.2.2 Maximum Queue Length of 10 Tasks

So far, the experiments only concerned problems without any restrictions on the maximum number of tasks that machines are allowed to put in their queue. In this section we add this restriction by limiting the queue length to a maximum of 10 tasks, and we additionally compare the examined insect-based approaches to two market-based approaches, explained in detail in Section 3.4.4.

10 Machines – 10 Types of Tasks (10M10T)

Table 8.10 shows the experimental results for the **10M10T** problems. In comparison to the same problem without a restriction on the queue size, the original system and the modifications on existing rules perform much better with respect to the makespan. This indicates that if there is no restriction

MS	**25M10Tequ**	**25M10Tdif**	**25M10Tchg**	**25M10Tsin**
ORIG	4697.2 ± 679.5	4372.0 ± 1399.6	4953.4 ± 483.7	4822.1 ± 517.4
UR+CFV+DC	4801.6 ± 505.0	5116.6 ± 1783.1	4992.2 ± 498.4	4808.4 ± 551.6
BCT	3257.6 ± 415.7	3403.0 ± 322.4	3425.3 ± 130.7	3445.9 ± 588.2
IMB	3091.5 ± 104.7	3085.2 ± 44.0	3115.8 ± 95.6	3087.1 ± 82.0
ALL	**3063.3 ± 23.8**	**3070.6 ± 31.1**	**3071.0 ± 34.0**	**3061.5 ± 23.5**

(a)

S	**25M10Tequ**	**25M10Tdif**	**25M10Tchg**	**25M10Tsin**
ORIG	2.01 ± 0.47	2.07 ± 0.63	2.19 ± 0.48	2.11 ± 0.59
UR+CFV+DC	1.79 ± 0.82	**1.73 ± 0.43**	**1.81 ± 0.43**	1.98 ± 3.20
BCT	1.67 ± 0.36	1.81 ± 0.36	2.15 ± 0.36	1.92 ± 0.40
IMB	**1.60 ± 0.22**	1.88 ± 0.32	2.66 ± 0.28	**1.72 ± 0.22**
ALL	1.86 ± 0.21	2.24 ± 0.26	2.85 ± 0.29	1.97 ± 0.22

(b)

25M10Tdif	**Idle Time**	**Queue**	**Cycle Time**	**Alloc Time**
ORIG	2225.42 ± 1302.94	4.06 ± 1.00	122.12 ± 76.00	1.19 ± 0.52
UR+CFV+DC	2931.84 ± 1664.24	6.98 ± 3.05	172.14 ±112.24	1.13 ± 0.42
BCT	1320.64 ± 293.74	2.91 ± 0.69	56.04 ± 20.35	**0.43 ± 0.06**
IMB	1027.62 ± 53.68	1.56 ± 0.28	37.44 ± 3.78	0.57 ± 0.06
ALL	**1003.74 ± 46.06**	**1.19 ± 0.21**	**33.00 ± 2.48**	0.45 ± 0.03

(c)

Table 8.9: The makespan (**MS**, (a)) and the number of required setups per machine (**S**, (b)) for the **10M10T** problems without queue length restrictions are shown for the four task generation processes (**equ**, **dif**, **chg** and **sin**). (c) shows the idle time, the average queue size, the cycle time and the allocation time for the **25M10Tdif** problem.

on the queue size, for these approaches some machines accept more than 10 tasks in their queue. With the restriction, this is not possible any more. At first sight, it seems that the restriction stabilizes the original system as well as the modifications on existing rules. But, q more detailed view on Table 8.10(c) – where the idle time, the queue size, the cycle time and the allocation time are given for the **10M10Tequ** problem – gives a different impression. The allocation time of tasks is dramatically increased. Without the restriction, tasks are allocated to machines after about 70 time steps for those two approaches. On the other side, with the restriction, tasks have to wait for an allocation for approximately 470 time steps for the **ORIG**, and 370 time steps for **UR+CFV+DC**. In comparison to the same problem without restriction, the performance of **BCT** degrades for all **10M10T** problems. The makespan remains constant only for the **10M10Tdif** and **10M10Tchg** problems. But for all problems, the number of required setups increases very much. In general, **ALL** and **IMB** perform much better than all other approaches and the two market-based approaches are always among the worst performing approaches, where **MORLEY1** shows the worst performance with respect to both, the makespan and the number of required setups, for all problems.

MS	**10M10Tequ**	**10M10Tdif**	**10M10Tchg**	**10M10Tsin**
ORIG	4796.8 ± 633.4	4499.6 ± 474.5	4730.0 ± 245.0	4992.4 ± 385.1
UR+CFV+DC	4553.9 ± 375.8	4138.1 ± 311.6	4200.0 ± 301.3	4579.1 ± 326.8
BCT	3918.4 ± 834.7	3308.3 ± 497.3	3292.3 ± 350.7	3791.0 ± 791.9
IMB	3120.3 ± 268.1	3109.6 ± 41.0	3113.3 ± 29.9	**3058.2 ± 36.5**
ALL	**3068.2 ± 34.3**	**3103.4 ± 42.3**	**3109.2 ± 43.0**	3072.6 ± 33.4
MORLEY1	5547.5 ± 152.1	5451.1 ± 153.7	5453.6 ± 167.7	5576.7 ± 150.8
MORLEY2	4587.2 ± 132.7	4375.0 ± 134.5	4390.8 ± 128.8	4589.9 ± 152.6

(a)

S	**10M10Tequ**	**10M10Tdif**	**10M10Tchg**	**10M10Tsin**
ORIG	81.74 ± 25.88	72.03 ± 17.24	79.94 ± 8.27	89.22 ± 13.70
UR+CFV+DC	75.20 ± 15.09	60.44 ± 10.35	62.62 ± 9.95	75.96 ± 11.36
BCT	46.73 ± 35.74	20.07 ± 21.40	20.80 ± 16.76	42.61 ± 33.85
IMB	11.04 ± 13.47	10.71 ± 2.40	**10.37 ± 2.14**	6.86 ± 3.91
ALL	**5.03 ± 1.30**	**10.41 ± 1.33**	10.47 ± 1.45	**5.26 ± 1.33**
MORLEY1	97.03 ± 4.85	85.04 ± 12.70	85.08 ± 12.53	97.19 ± 4.85
MORLEY2	67.08 ± 4.03	58.11 ± 8.14	58.07 ± 6.29	66.07 ± 3.46

(b)

10M10Tequ	**Idle Time**	textbfQueue	**Cycle Time**	**Alloc Time**
ORIG	240.78 ± 206.84	7.41 ± 2.28	789.26 ± 303.54	471.54 ± 217.40
UR+CFV+DC	208.75 ± 141.10	7.98 ± 2.82	673.98 ± 177.50	371.91 ± 131.51
BCT	455.42 ± 344.14	5.32 ± 1.21	358.60 ± 328.07	166.50 ± 205.40
IMB	780.26 ± 201.17	2.57 ± 0.37	65.80 ± 96.44	12.51 ± 52.74
ALL	908.19 ± 68.06	**1.61 ± 0.34**	**40.68 ± 4.42**	**1.09 ± 0.08**
MORLEY1	**190.35 ± 34.16**	8.76 ± 2.39	1047.64 ± 107.63	662.63 ± 94.64
MORLEY2	305.36 ± 59.03	6.31 ± 1.40	661.10 ± 89.90	440.16 ± 83.22

(c)

Table 8.10: The makespan (**MS**, (a)) and the number of required setups per machine (**S**, (b)) for the **10M10T** problems with the queue length restriction of at most 10 tasks for a machine's queue are shown for the four task generation processes (**equ**, **dif**, **chg** and **sin**). (c) shows the idle time, the average queue size, the cycle time and the allocation time for the **10M10Tequ** problem.

Figure 8.11 shows the average queue size of all machines for the different approaches. **MORLEY1** already reaches the limit of 10 tasks in the queue of each machine after approximately 1000 time steps. In general, we can say that once the limit is reached for all machines, it is very difficult to obtain a good performance any more. The explanation for this is very easy. In case, all machines have a full queue, tasks cannot be allocated at all any more, until at least one machine finishes one task. In that moment, a lot of tasks are waiting to be allocated. For the insect-based approaches, this means that tasks will probably have a high stimulus intensity, and thus, an allocation is practically forced. For **MORLEY1** on the other side, this means that a task that is the longest time in the system, will be allocated to the first available machine. A distinction between whether the allocation requires a setup or not is not made at all. Unfortunately, this appears to be a vicious circle in

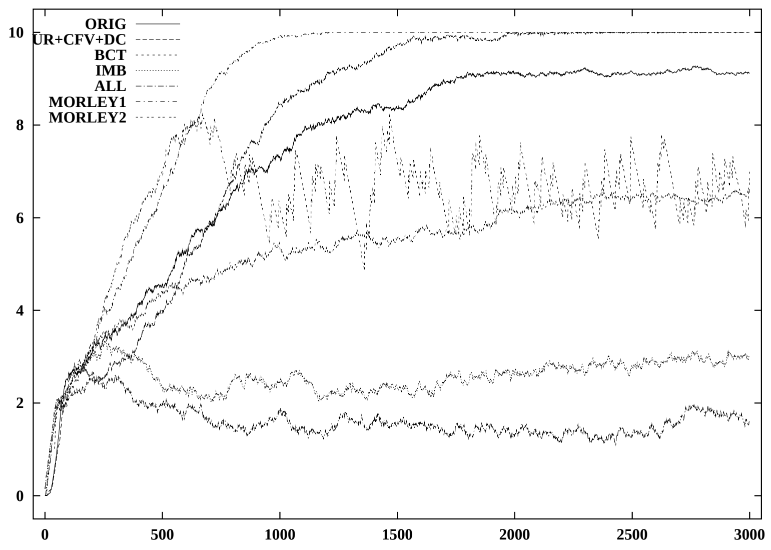


Figure 8.11: The average queue size per machine is displayed for the **10M10Tequ** problem. The five insect-based and the two market-based approaches are compared.

the sense that the longer this situation persists, the more tasks will wait for allocation. For **MORLEY2** this situation cannot arrive in any case, as by definition of the algorithm, tasks may only be allocated to machines, if at least one machine has a nearly empty queue.

Except **MORLEY1**, only **UR+CFV+DC** reaches the state where all machines have a full queue. **ORIG** converges towards about eight tasks per queue. The shortest queues are required by **ALL** and **IMB**, who both converge at approximately two tasks per queue.

25 Machines – 10 Types of Tasks (25M10T)

25M10T with a maximum queue length of 10 tasks is the last class of problems we analyze and we will briefly summarize the results. Table 8.11 shows the results for the makespan and the number of required setups. Additionally, for the equal distribution of task arrival rates, the idle time, the queue size, the cycle time and the allocation time is displayed.

All insect-based approaches perform very well. Even **ORIG** and **UR+CFV+DC**, who performed very poorly for the same problem without restriction, reach a good performance. For these two approaches, the problem that some machines have very big queues is prevented. Though they require more setups, the makespan is significantly reduced. One of the main effects of the restriction is the reduction of the idle time. Single machines do not differ so much any more with respect to their queue sizes. Thus,

MS	**25M10Tequ**	**25M10Tdif**	**25M10Tchg**	**25M10Tsin**
ORIG	3097.8 ± 153.9	3077.3 ± 42.6	3369.1 ± 211.6	3118.8 ± 171.0
UR+CFV+DC	3203.6 ± 100.3	3169.2 ± 100.6	3286.0 ± 96.4	3199.1 ± 109.1
BCT	3069.4 ± 26.1	3071.5 ± 25.5	3086.8 ± 38.3	3071.3 ± 25.3
IMB	**3064.8 ± 24.2**	3076.0 ± 32.1	3078.8 ± 32.5	3069.3 ± 27.8
ALL	3065.3 ± 23.0	**3066.7 ± 26.4**	**3067.0 ± 27.6**	**3061.2 ± 20.0**
MORLEY1	5541.1 ± 95.0	5443.9 ± 104.6	5463.4 ± 98.3	5566.0 ± 100.3
MORLEY2	3829.6 ± 89.9	3764.1 ± 88.7	3791.8 ± 83.7	3847.5 ± 94.7

(a)

S	**25M10Tequ**	**25M10Tdif**	**25M10Tchg**	**25M10Tsin**
ORIG	4.70 ± 7.83	3.53 ± 2.83	16.49 ± 9.80	7.45 ± 10.26
UR+CFV+DC	10.77 ± 7.70	8.24 ± 7.45	15.01 ± 6.16	11.64 ± 7.48
BCT	1.78 ± 0.42	1.96 ± 0.41	2.78 ± 0.50	2.01 ± 0.43
IMB	**1.64 ± 0.22**	**1.90 ± 0.29**	**2.75 ± 0.35**	**1.77 ± 0.27**
ALL	1.92 ± 0.29	2.26 ± 0.26	2.93 ± 0.29	2.04 ± 0.21
MORLEY1	9.91 ± 15.23	8.38 ± 10.65	32.97 ± 14.42	20.42 ± 24.86
MORLEY2	8.33 ± 7.75	6.39 ± 4.87	21.57 ± 9.16	17.52 ± 13.80

(b)

25M10Tequ	**Idle Time**	**Queue**	**Cycle Time**	**Alloc Time**
ORIG	946.17 ± 111.27	1.28 ± 0.79	58.43 ± 54.05	5.91 ± 24.52
BCT	1014.94 ± 40.91	2.02 ± 1.81	33.06 ± 2.33	0.30 ± 0.02
IMB	1017.45 ± 43.12	0.83 ± 0.78	31.87 ± 1.70	0.30 ± 0.02
UR+CFV+DC	**871.01 ± 178.10**	0.78 ± 0.74	95.86 ± 43.64	14.04 ± 10.84
ALL	1004.25 ± 36.68	**0.74 ± 0.77**	**30.91 ± 1.70**	**0.26 ± 0.01**
MORLEY1	886.55 ± 178.72	1.38 ± 1.30	85.74 ± 124.73	14.80 ± 69.02
MORLEY2	878.22 ± 149.46	1.53 ± 1.28	69.60 ± 49.20	3.65 ± 17.19

(c)

Table 8.11: The makespan (**MS**, (a)) and the number of required setups per machine (**S**, (b)) for the **25M10T** problems with the queue length restriction of at most 10 tasks for a machine's queue are shown for the four task generation processes (**equ**, **dif**, **chg** and **sin**). (c) shows the idle time, the average queue size, the cycle time and the allocation time for the **25M10Tequ** problem.

for the situation where all tasks are allocated, there are no machines with very large queue sizes, causing other machines to remain idle for a long time.

The market-based approaches again perform poorly, both, in terms of the makespan and the number of setups. **MORLEY1** requires a makespan that is nearly twice as long as for the insect-based approaches.

As indicated by Table 8.11(c), in average only few tasks are put in a machine's queue.

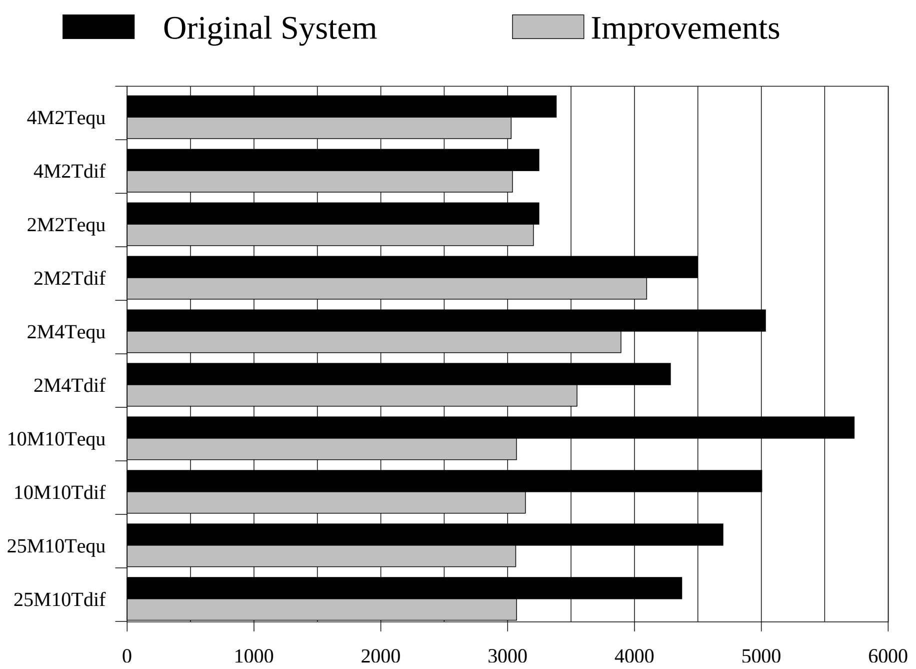
Chapter 9

Conclusions

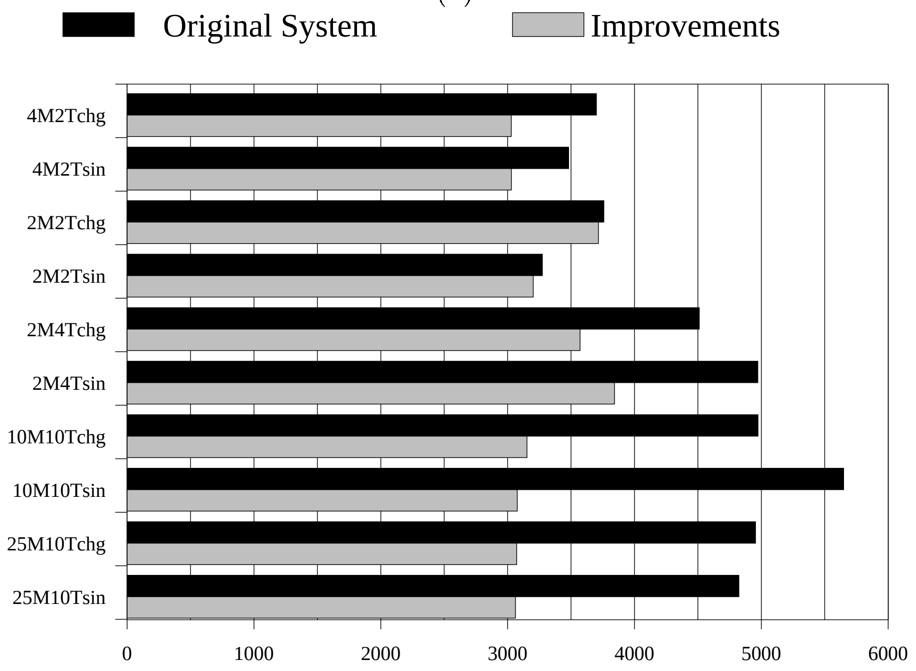
In this master thesis, we have presented a insect-based multi-agent system for a dynamic scheduling problem (DTA). Our approach is based on the work presented in [12, 9] and makes use of a threshold model that is inspired by the methodology of division of labor in social insects. The DTA simulates factory-like conditions, where reconfigurations of machines are crucial, as they are related to a high cost in time. Applying the threshold model to the DTA, results in a robust multi-agent system with the ability to adapt to changing demands. We have proposed five improvements on the original system. Three improvement proposals concern existing rules, modifying the update rules for threshold values (**UR**), the calculation of the force value (**CVF**) and the dominance contest (**DC**). The other two proposals are additional rules to speed up the adaptation process. One aims at a reduced idle time for machines (**IMB**) and the other aims at a reduced cycle time for tasks (**BCT**). We have tested our proposals on an extensive set of experiments and highlighted their effect in comparison to the original system. Figure 9.1 summarizes the results for the makespan for all problems without restriction on the queue size, comparing the combination of all improvement proposals and the original system. Small problem instances were analyzed in order to give a detailed insight into single machine performance and the temporal behavior of the threshold values. Large problem instances are meant to simulate factory environments more realistically. For large problem instances, we additionally compared the insect-based approach to a simple version of a real-world proven market-based approach by Morley [32], and a slightly modified version of this approach. These two market-based approaches, especially the one that was not modified, generally perform poorly. For the **10M10T** problems with ten machines and ten types of tasks, the modified market-based approach performs better than the original insect-based approach, but still worse than any of the improvements. For the **25M10T** problems with 25 machines and ten types of tasks, the two market-based approaches have the worst performance. In most of the problem instances, each of the five proposals results in a better overall performance, compared to the original system. The improvement with the biggest influence is **IMB**. Applying **IMB** always results in a significantly shorter makespan and a much faster adaptation. **BCT** also increases adaptation speed. But for the **2M2T** problem with two machines and two types of tasks, **BCT** exhibits instable behavior. It causes the machines to frequently change the type of task they perform, and thus, disables them to specialize for one type. However, for most problem instances, the performance achieved by **BCT** is comparable to the one of **IMB**. The modifications of existing rules, and among them especially the modification of the update rules (**UR**), also result in an improved overall performance for most cases. Though, the difference to the original system is not as big as for the new rules. In all examined problems, the combination of all proposals performs better than the original system, mostly with remarkable difference.

One major problem for the experimentation was the lack of comparison to other approaches to the same problem, which are different from the insect-based approach. We made a comparison with two market-based approaches, but their performance was not comparable to the one achieved by the insect-based approach, that applies all improvements. It is difficult to determine the quality of the insect-based approach after comparison to only one different approach. However, a study on scheduling problems has shown, that the DTA is a very specific problem, that was not studied very much so far. The main difference between the DTA and other, more thoroughly studied scheduling problems, is the consideration of reconfigurations and setup times. Due to this difference, we could not find other approaches.

Additionally, we would like to study the behavior of the system, if the various parameters in the system are dynamically changing in time. This refers to the parameters of the update rules and the improvement proposals **BCT** and **IMB**. So far, the studies we have made about this, resulted in a very instable system and were therefore not taken into consideration in experiments.



(a)



(b)

Figure 9.1: (a): Summary of the results for the makespan. The original system is compared to the combination of all improvements for all **equ** and **dif** problems without queue size restriction. (b): The same as (a) for the **chg** and **sin** distributions.

Bibliography

[1] C. Blum. ACO Applied to Group Shop Scheduling: A Case Study on Intensification and Diversification. In M. Dorigo, G. Di Caro, and M. Sampels, editors, *Proceedings of ANTS 2002 – Third International Workshop on Ant Algorithms*, volume 2463 of *Lecture Notes in Computer Science*, pages 14–27. Springer Verlag, Berlin, Germany, 2002.

[2] C. Blum. Metaheuristics for Group Shop Scheduling. Technical Report TR/IRIDIA/2002-23, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium, 2002. Also: Thesis to obtain the Diplôme d'Etudes Approfondies en Sciences Appliquées (DEA).

[3] E. Bonabeau, M. Dorigo, and G. Théraulaz. *Swarm Intelligence: From Natural to Artificial Systems.* Oxford University Press, New York, NY, 1999.

[4] E. Bonabeau and G. Théraulaz. Swarm Smarts. *Scientific American*, 282(3):72–79, March 2000.

[5] E. Bonabeau, G. Théraulaz, and J.-L. Deneubourg. Quantitative Study of the Fixed Threshold Model for the Regulation of Division of Labour in Insect Societies. In *Proceedings Roy. Soc. London B*, volume 263, 1996.

[6] E. Bonabeau, G. Théraulaz, and J.-L. Deneubourg. Fixed Response Thresholds and the Regulation of Division of Labor in Insect Societies. *Bulletin of Mathematical Biology*, 60:753–807, 1998.

[7] F. Bousquet, C. Cambier, and P. Morand. Distributed Artificial Intelligence and Object-Oriented Modelling of a Fishery. *Mathematical and Computer Modelling*, 20(8):97–107, 1994.

[8] B. Bullnheimer, R. F. Hartl, and C. Strauss. An Improved Ant System Algorithm for the Vehicle Routing Problem. *Annals of Operations Research*, 89:319–328, 1999.

[9] M. Campos, E. Bonabeau, G. Théraulaz, and J. L. Deneubourg. Dynamic Scheduling and Division of Labor in Social Insects. *Adaptive Behavior 2000*, pages 83–96, 2000.

[10] V. Cicirello and S. Smith. Randomizing dispatch scheduling policies. In *The 2001 AAAI Fall Symposium: Using Uncertainty Within Computation*, November 2001.

[11] V. A. Cicirello and S. F. Smith. Improved Routing Wasps for Distributed Factory Control. In *The IJCAI-01 Workshop on Artificial Intelligence and Manufacturing, Working Notes*, pages 26–32, Seattle, WA, 2001. AAAI SIGMAN.

[12] V. A. Cicirello and S. F. Smith. Wasp-like Agents for Distributed Factory Coordination. Technical Report CMU-RI-TR-01-39, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, December 2001.

[13] V. A. Cicirello and S. F. Smith. Wasp Nests for Self-Configurable Factories. In *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 473–480, Montreal, Canada, May 2001. ACM Press.

[14] A. Colorni, M. Dorigo, V. Maniezzo, and M. Trubian. Ant System for Job-Shop Scheduling. *JORBEL—Belgian Journal of Operations Research, Statistics and Computer Science*, 34:39–53, 1994.

[15] D. Costa and A. Hertz. Ants Can Colour Graphs. *J. Op. Res. Soc.*, 48:295–305, 1997.

[16] J.-L. Deneubourg, S. Aron, S.Goss, and J.-M. Pasteels. The Self-Organizing Exploratory Pattern of the Argentine Ant. *J. Insect Behavior*, 3:159–168, 1990.

[17] J.-L. Deneubourg, S. Goss, A Sendova-Franks, C. Detrain, and L. Chretien. The Dynamics of Collective Sorting Robot-like Ants and Ant-like Robots. In *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior*, pages 356–363. MIT Press, 1991.

[18] G. Di Caro and M. Dorigo. AntNet: Distributed Stigmergetic Control for Communications Networks. *Journal of Artificial Intelligence Research*, 9:317–365, 1998.

[19] M. Dorigo and L. M. Gambardella. Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem. *IEEE Trans. Evol. Comp.*, 1:53–66, 1997.

[20] E. H. Durfee and V. Lesser. Negotiating Task Decomposition and Allocation Using Partial Global Planning. *Distributed Artificial Intelligence*, 2:229–244, 1989.

[21] G. Färber. Prozessrechentechnik – Manuskript zur Vorlesung, Lehrstuhl für Realzeit-Computersysteme, Technische Universität München, Stand Wintersemester 1999/2000.

[22] J. Ferber. *Multi-Agent Systems - An Introduction to Distributed Artificial Intelligence.* Addison-Wesley, 1999.

[23] L. M. Gambardella, È. D. Taillard, and M. Dorigo. Ant colonies for the quadratic assignment problem. *Journal of the Operational Research Society*, 50(2):167–176, 1999.

[24] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison Weslay, 1989.

[25] P. P. Grassé. La Reconstruction du Nid et les Coordinations Interindividuelles chez *Bellicositermes Natalensis et Cubitermes sp.* La théorie de la stigmergie: Essai d'interprétation du comportement des termites constructeurs. *Insectes Sociaux*, 6:41–81, 1959.

[26] J. H. Holland. *Adpatation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, MI, 1975.

[27] C. Iffenecker. Un Système Multi-Agents pour le Support des Activités de Conception de Produits., Thèse d'Université, Université Paris 6, 1992.

[28] S. Lander, V. R. Lesser, and M. E. Connell. Conflict Resolution Strategies for Cooperating Expert Agents. In S. M. Deen, editor, *Cooperating Knowledge Based Systems 1990*, pages 183–198. Springer-Verlag, 1991.

[29] V. Maniezzo and A. Colorni. The Ant System applied to the Quadratic Assignment Problem. Technical Report 28, IRIDIA/94-28, Université Libre de Bruxelles, Belgium, 1994.

[30] R. P. McAfee and J. McMillan. Auctions and Bidding. *Journal of Economic Literature*, 25:699–738, 1987.

[31] M. Mitchell. *An Introduction to Genetic Algorithms.* Complex Adaptive Systems. MIT-Press, Cambridge, 1996.

[32] R. E. Morley and C. Schelberg. An analysis of a plant-specific dynamic scheduler. In *Proceedings of the NSF Workshop on Dynamic Scheduling*, Cocoa Beach, Florida, 1993.

[33] S. Nouyan. Agent-Based Approach to Dynamic Task Allocation. In M. Dorigo, G. Di Caro, and M. Sampels, editors, *Proceedings of ANTS 2002 – Third International Workshop on Ant Algorithms*, volume 2463 of *Lecture Notes in Computer Science*, pages 28–39. Springer Verlag, Berlin, Germany, 2002.

[34] R. C. Plowright and C. M. S. Plowright. Elitism in Social Insects: A Positive Feedback Model. *Interindividual Behavioral Variability in Social Insects*, pages 419–431, 1988.

[35] S. J. Rassenti, V. L. Smith, and R. L. Bulfin. A Combinatorial Auction Mechanism for Airport time slot Allocation. *Bell Journal of Economics*, 13:402–417, 1982.

[36] G. E. Robinson. Regulation of Division of Labor in Insect Societies. *Ann. Rev. Entomol.*, 37:637–665, 1992.

[37] T. Sandholm. An Implementation of the Contract Net Protocol Based on Marginal Cost Calculations. In *Proceedings of the 11th National Conference on Artificial Intelligence*, pages 256–263, Menlo Park, CA, USA, July 1993. AAAI Press.

[38] L. S. Shapley and M. Shubik. The Assignment Game I: The Core. *International Journal of Game Theory*, 1:111–130, 1972.

[39] K. P. Sycara. Multiagent Systems. *The AI Magazine*, 19(2):79–92, 1998.

[40] G. Théraulaz, E. Bonabeau, and J.-L. Deneubourg. Response Threshold Reinforcement and Division of Labour in Insect Societies. In *Proceedings of the Royal Society of London B*, volume 265, pages 327–335, 1998.

[41] G. Théraulaz, S. Goss, J. Gervet, and J. L. Deneubourg. Task Differentiation in Polistes Wasp Colonies: A Model for Self-Organizing Groups of Robots. In *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior*, pages 346–355. MIT Press, 1991.

[42] V. Volterra. Variation and Fluctuations of the Number of Individuals of Animal Species living together. *Animal Ecology*, 1926.

[43] C. Waldspurger, T. Hogg, B. Huberman, J. Kephart, and S. Stornetta. Spawn: A Distributed Computational Economy. *IEEE Transactions on Software Engineering*, 18(2):103–117, February 1992.

[44] C. A. Waldspurger and W. E. Weihl. Lottery Scheduling: Flexible and Proportional-Share Resource Management. In *Operating Systems Design and Implementation (OSDI)*, pages 1–12. Usenix, ACM SIGOPS, IEEE TCOS, 1994.

[45] W. E. Walsh, M. P. Wellman, P. R. Wurman, and J. K. MacKie-Mason. Some Economics of Market-Based Distributed Scheduling. In *International Conference on Distributed Computing Systems*, pages 612–621, 1998.

[46] M. P. Wellman. A Market-Oriented Programming Environment and its Application to Distributed Multicommodity Flow Problems. *Journal of Artificial Intelligence*, pages 1–23, 1993.

[47] E. O. Wilson. *The Insect Societies.* Belknap Press of Harvard University Press, Cambridge, Mass., 1971.

[48] E. O. Wilson. The Relation Between Caste Ratios and Division of Labour in the Ant Genus Pheidole (Hymenoptera: Formicidae). *Behav. Ecol. Sociobiol.*, 16:89–98, 1984.

[49] M. Wooldridge and N. Jennings. Intelligent Agents: Theory and Practice. *The Knowledge Engineering Review*, 10, 1995.