KATHOLIEKE UNIVERSITEIT
LEUVEN

Arenberg Doctoral School of Science, Engineering & Technology
Faculty of Engineering
Department of Computer Science

Reinforcement learning enhanced heuristic search for combinatorial optimization

Tony WAUTERS

Dissertation presented in partial
fulfillment of the requirements for
the degree of Doctor
in Engineering

November 2012

Reinforcement learning enhanced heuristic search for combinatorial optimization

Tony WAUTERS

Jury:
Prof. dr. ir. Yves Willems, chair
Prof. dr. Patrick De Causmaecker, supervisor
dr. Katja Verbeeck, co-supervisor
Prof. dr. ir. Greet Vanden Berghe
Prof. dr. Ann Nowé
Prof. dr. ir. Dirk Cattrysse
Prof. dr. El-Ghazali Talbi
(University of Lille, France)
Prof. dr. Karl Tuyls
(University of Maastricht, The Netherlands)

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor in Engineering

November 2012

D/2012/7515/121
ISBN 978-94-6018-587-8

Preface

This Ph.D. thesis is a result of four years of work, starting in August 2008. I would like to thank all the people that helped me during these four years.

First of all I thank my supervisors Patrick De Causmaecker and Katja Verbeeck for their great guidance during these four years and correcting all the papers.

Secondly, I would like to thank Greet Vanden Berghe who offered me to work on interesting research projects with industrial companies. This gave me the necessary opportunities and funding to bring this Ph.D to a good end. I also thank her for the many interesting discussions and paper corrections.

Next, I would like to thank Ann Nowé, Dirk Catrysse, Karl Tuyls, El-Ghazali Talbi for accepting to be members of the jury, and for their detailed feedback which helped improving the quality of this text.

This thesis is a result of collaboration with many people and companies, for which I'm very thankful: Yailen Martinez, Elsy Kaddoum, Xiaoyu Mao, Fabio Salassa, Paul Verstraete, Vincent Van Peteghem and many more.

I would like to thank Erik Van Achter for improving the quality of this thesis and the quality of our papers.

I thank my CODeS-KAHO colleagues for their interesting discussions and entertainment: Jannes Verstichel, Wim Vancroonenburg, Mustafa Misir, Peter Demeester, Wouter Souffriau, Burak Bilgen, Koen Vangheluwe, Pieter Smet, Tim Vermeulen, Joris Maervoet and Joris Kinable.

Last but not least I would like to thank my family, my mother Annie Van Trappen and especially my better half Rebecca Dhoore for giving me support and advice, and solving my knapsack problem before traveling to an abroad conference. During the course of this Ph.D. two reinforcement learners were born, my two beloved sons Finn and Senn.

Abstract

The present thesis describes the use of reinforcement learning to enhance heuristic search for solving complex (real-world) optimization problems such as (project) scheduling, routing and assignment. Heuristic search methods are known to deliver good results in a reasonable amount of calculation time, without any guarantee of optimality. Often they require careful parameter tuning to obtain good results. Reinforcement learning methods on the other hand, learn to act in a possible unknown random environment on a trial-and-error basis. The goal of the hybridization of heuristic search and reinforcement learning is to generate intelligent search methods which are adaptive and generally applicable while keeping eventual extra overhead to a minimum.

Three levels of inclusion of reinforcement learning into heuristic search methods are defined: the direct, the metaheuristic and the hyperheuristic level. At the direct level, the reinforcement learning method searches directly for good quality solutions, while at the metaheuristic and hyperheuristic level, the reinforcement learning component is added for learning good starting solutions, good parameter values, good objective functions, good heuristics, etc. For each level, one or more learning enhanced methods are demonstrated on benchmark and/or real-world problems. A general methodology for learning permutations without any domain knowledge is described. Additionally, a method for learning to select heuristics during search is described and tested on several hard combinatorial optimization problems such as the traveling tournament problem, the patient admission scheduling problem, and the machine reassignment problem. It is shown that this learning selection method performs significantly better than selecting the heuristics at random.

From an application point of view, this thesis is mainly, though not exclusively, devoted to scheduling problems. We tackled the multi-mode resource-constrained project scheduling problem, the decentralized resource-constrained multi-project scheduling problem, the (dynamic) flexible job shop scheduling problem and a real-world production scheduling problem from the food industry.

Real-world problems often hold a rich structure allowing for learning valuable information. We show that a multi-agent reinforcement learning architecture, more specifically a network of learning automata with a common reward signal, is very well suited to design new hybrid well performing methods. Many new best results for project scheduling benchmarks are generated using the proposed GT-MAS approach.

Beknopte samenvatting

In deze thesis wordt beschreven hoe reinforcement learning kan aangewend worden om heuristische zoekmethoden te verbeteren voor het oplossen van moeilijke (praktische) optimalisatie problemen zoals bijvoorbeeld (project)plannings, routerings en toewijzingsproblemen. Heuristische zoekmethoden worden gebruikt om goede resultaten te behalen in korte tijd, maar zonder enige garantie van optimaliteit. Vaak vragen deze methoden een grondige fijnstelling van parameters voor het behalen van de beste resultaten. Reinforcement learning technieken daarentegen zullen op een trial-and-error manier leren om acties te nemen in mogelijk ongekende omgevingen. Het doel van de hybridisatie van heuristisch zoeken en reinforcement leren is het genereren van intelligente zoekmethoden die adaptief en generiek toepasbaar zijn, zonder daarbij veel overhead toe te toevoegen.

Drie niveaus voor het toevoegen van reinforcement learning aan heuristisch zoeken worden gedefinieerd: het directe, het metaheuristisch en het hyperheuristisch niveau. Op het directe niveau zal de reinforcement learning methode rechtstreeks op zoek gaan naar de beste oplossing, terwijl op het metaheuristisch en hyperheuristisch niveau, de reinforcement learning component zal ingezet worden voor het leren van goede startoplossingen, goede parameter waarden, goede doelfuncties, goede heuristieken, enz. Voor ieder niveau worden er één of meerdere door leren versterkte methoden gedemonstreerd op benchmarks en/of praktische problemen. Een algemen methode voor het leren van permutaties zonder domeinkennis wordt beschreven. Bovendien wordt er een methode beschreven voor het leren selecteren van heuristieken tijdens het zoeken. Deze methode werd uitgetest op enkele moeilijke combinatorische problemen, zoals het traveling tournament probleem, het patient admission scheduling probleem en het machine reassignment probleem. Er wordt aangetoond dat deze lerende selectie methode beter presteert dan een methode die de heuristieken volledig willekeurig gaat selecteren.

Het toepassinggebied waar in deze thesis het meeste aandacht wordt besteed

is ‘scheduling’. We hebben de volgende problemen aangepakt: het multi-mode resource-constrained project scheduling probleem, het decentralized resource-constrained project scheduling probleem, het (dynamic) flexible job shop scheduling probleem, en een productie scheduling probleem uit de voedings industrie. Praktische problemen hebben vaak een rijke structuur die toelaat om waardevolle informatie te leren. Er wordt aangetoond dat een multi-agenten reinforcement leeromgeving, meer specifiek een netwerk van leerautomaten met een gemeenschappelijk reward signaal, zeer geschikt is voor het ontwerpen van goed presterende hybride methoden. Vele nieuw beste resultaten werden behaald voor gekende projectplanningsdatasets met behulp van de voorgesteld GT-MAS aanpak.

List of abbreviations

AI	Artificial Intelligence
AON	Activity On Node
APD	Average Project Delay
ATSP	Asymmetric Traveling Salesman Problem
BSS	Basic Simple Strategy
DFJSP	Dynamic Flexible Job Shop Scheduling Problem
DPD	Deviation of the Project Delay
(D)RCMPSP	(Decentralized) Resource-Constrained Multi-Project Scheduling Problem
FJSP	Flexible Job shop Scheduling Problem
GA	Genetic Algorithm
KPI	Key Performance Indicators
LA	Learning Automata/Automaton
MARL	Multi-Agent Reinforcement Learning
MAS	Multi-Agent System
MDP	Markov Decision Problem
MES	Manufacturing Execution System
MRCPSP	Multi-mode Resource Constrained Project Scheduling Problem
PAS	Patient Admission Scheduling Problem
PBSS	Probabilistic Basic Simple Strategy
PFSP	Permutation Flow Shop Scheduling Problem
QAP	Quadratic Assignment Problem
RCPSP	Resource Constrained Project Scheduling Problem
RCPSP/GPR	Resource Constrained Project Scheduling Problem with Generalized Precedence Relations
RL	Reinforcement Learning

SA	Simulated Annealing
SGS	Schedule Generation Scheme
SOP	Sequential Ordering Problem
TD	Temporal Difference
TMS	Total Makespan
TSP	Traveling Salesman Problem
TTP	Traveling Tournament Problem

Contents

List of Figures

List of Tables

Chapter 1

Introduction

On a daily basis, people and companies are challenged with difficult problems such as finding the best route between different locations, or finding the best feasible schedule, or managing their resources. For many of these **combinatorial optimization problems**, the number of solutions grows exponentially in the input size, and finding the best solution(s) is hard. **Heuristic search** methods are very appropriate tools for solving such problems. Many heuristic search methods exist. Some of these are metaphorically inspired by phenomena from biology or physics, others use simple rules of thumb. The field of machine learning offers a variety of techniques, such as supervised learning, unsupervised learning and reinforcement learning, to develop improved and more intelligent heuristic search methods.

The aim of this thesis is to create heuristic search methods with a **Reinforcement Learning** (RL) component making them more adaptive and generally applicable to many (real-world) combinatorial optimization problems such as scheduling, planning, assignment and routing, while keeping the overhead low. Reinforcement learning (RL) is the subfield of machine learning, where one or more agents have to learn to take actions in a possibly unknown environment in order to reach some goal(s). Opposed to supervised learning, an RL agent is not told which actions are best to take, and thus has to learn by trial-and-error (e.g. child learning to ride a bike). RL methods have to deal with delayed rewards and incomplete information. Learning automata (LA) belong to this category of methods, and are extensively applied in this thesis to create intelligent optimization methods. Battiti et al. [2008] state that using an RL method in a feedback loop with a heuristic search could have many advantages. In this thesis we show that this is in fact the case.

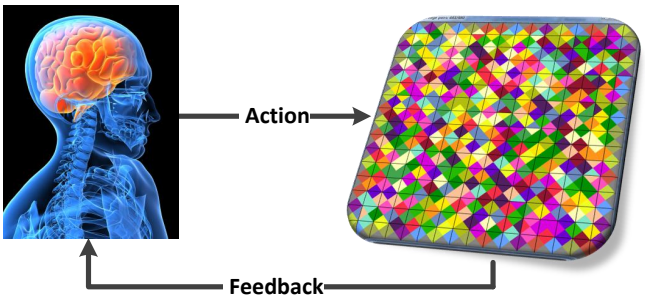


Figure 1.1: Person solving a hard edge-matching puzzle.

Consider the following illustrative example (Figure 1.1): A person is challenged with the task of solving a hard edge-matching puzzle such as the Eternity II puzzle [Wauters et al., 2012a] (*combinatorial optimization problem*). She will start with an initial solving technique, possibly based on some rule of thumb (*heuristic search method*). In the course of the solving process the human will receive feedback, e.g. how many pieces are placed by using the current solving technique. She will adapt her solving technique based on the retrieved feedback (*reinforcement learning*), i.e. the performance of the search technique. If everything goes well, she applies the current solving technique with a larger probability, if not, she decreases that probability and tries other solving techniques. A similar reasoning process will be followed by the methods described in this thesis.

1.1 Motivation

Researchers developing heuristic search methods to hard combinatorial optimization problems (e.g. scheduling problems) are faced with a number of challenges. One major challenge is to avoid convergence to a poor solution. Another challenge is to create a method applicable to different problems of various sizes and properties, while still being able to produce good quality solutions in a short amount of time. Metaheuristics [Glover, 1986, Glover and Kochenberger, 2003, Talbi, 2009] and the more recently introduced hyperheuristics [Burke et al., 2003a] try to address these issues. Hybrid systems and their various perspectives cope with these challenges even better, as shown in many recent publications for benchmark and practical problems [Blum et al.,

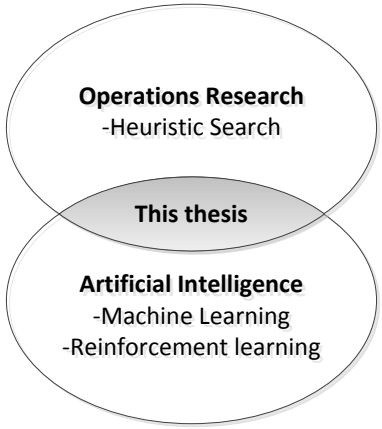


Figure 1.2: Intersection between operations research and artificial intelligence.

2008]. Successful hybridizations with integer programming and constraint programming techniques were proposed. Another possible hybridization, which is the main topic of this contribution, involves the inclusion of a Reinforcement Learning (RL) [Kaelbling et al., 1996, Sutton and Barto, 1998a] component to these meta- and hyper-heuristic methods. This idea fits in the area of intelligent optimization [Battiti et al., 2008], which can be defined as a combination of techniques from Operations Research and Artificial Intelligence (Figure 1.2). Some intelligent (learning) component aids the optimization method in order to obtain a better informed search. During the search process a learning component can adjust parameters or support the optimization method in making decisions.

In what follows, it is shown how RL and metaheuristics can be fruitfully combined. Reinforcement learning causes the algorithm to be adaptive, and as such it minimizes the weaknesses of strongly parameterized methods. As long as the algorithm is granted enough time facilitating the learning of valuable information, reinforcement learning offers interesting advantages. It does not require a complete model of the underlying problem. RL methods learn the model by gathering experience, often referred to as trial-and-error. Many model free RL methods exist. This is noteworthy since ordinarily no model is available for most combinatorial optimization problems. Some reinforcement learning methods can handle incomplete information, even though this is obviously a much harder learning task. Reinforcement learning allows applying independent learning agents, and thus, it is even applicable to fully decentralized problems.

Furthermore, it is computationally cheap in that it often uses only a single update formula at each step. Additionally, if only general problem features and no instance specific features are used, then the learned information can possibly be transfered to other instances of the same problem. Recently some type of RL algorithms that are well suited for this task have been introduced, these are called transfer learning [Taylor et al., 2007, Taylor and Stone, 2009]. Furthermore, one can build on theoretical properties showing that many RL methods converge to optimal state-action pairs under certain conditions (e.g. the policy for choosing the next action is ergodic) [Sutton and Barto, 1998a]. In this dissertation we will demonstrate that these interesting theoretical properties show good results when applied to heuristic search for combinatorial optimization.

The thesis focuses on how RL methods, more specifically learning automata, can be used for solving combinatorial optimization problems, and how they can be hybridized with existing heuristic search methods. We study the calculation time overhead of learning, and the influence of the learning parameters on the obtained results. Often a single parameter is used, i.e. the learning rate.

An interesting application domain for these RL enhanced heuristic search methods is **scheduling**. Scheduling problems are very relevant for industry, e.g. production scheduling, scheduling multiple projects in the construction sector, or just scheduling jobs on machines in a shop floor environment. Strict time and resource constraints have to be fulfilled, while simultaneously optimizing time or cost related objectives such as the makespan. Scheduling is the main application area of this thesis, including project scheduling problems, flexible job shop scheduling problems and a real world production scheduling problem in the food industry.

1.2 Structure of the thesis

The present thesis starts by discussing the concepts of reinforcement learning and heuristic search, and the combination of both. Chapter 2 describes how these two methods are related, and how they can be efficiently combined. An extensive literature overview on approaches combining both domains is given, discussing their similarities and differences. We illustrate the opportunities for inclusion of learning at three different levels, i.e. the direct level, the metaheuristic level and the hyperheuristic level (Figure 2.4). At the direct level the RL method directly learns good quality solutions, while at the metaheuristic level the RL method can learn good starting solutions, a good objective function, or good parameter values. At the hyperheuristic[1] level the RL method can be integrated in the

[1] A typical hyperheuristic in the category of 'heuristics to choose heuristics' is assumed.

selection or acceptance steps. Learning enhanced approaches to permutation problems, project scheduling problems and other combinatorial optimization problems are described in full detail in the subsequent chapters. The coverage of the RL inclusion levels by the chapters in this thesis is shown in Figure 1.3.

We start with the general task of learning permutations. Permutations are frequently part of combinatorial optimization problems and techniques for solving them. Designing a method able to learn good quality permutations without using problem specific information, would contribute to the general applicability of heuristic search methods. Chapter 3 describes such techniques for learning permutations using learning automata. These permutation learning methods can be integrated into the hyperheuristic framework to order their low-level heuristics. Hyperheuristics can also use the techniques introduced in Chapter 4 for learning which heuristic to select at each step. A single learning automaton shows to outperform the often used simple random heuristic selection method. These LA based heuristic selection methods are demonstrated on three hard combinatorial optimization problems: the traveling tournament problem, the patient admission scheduling problem and the machine reassignment problem. The traveling tournament problem is a combination of a timetabling and a routing problem. A double round robin tournament for (football, rugby, soccer) teams has to be created with minimal total traveling distance, while respecting consecutive home-away constraints. In the patient admission scheduling problem, patients have to be assigned to beds in a hospital taking into account many medical and personal constraints. The machine reassignment problem, which was the subject of a recent challenge organized by EURO/ROADEF in 2012, considers the (re-)assignment of processes to machines such that the resource loads on the machines are balanced and the costs of moving the processes are minimized.

In the further chapters of this thesis we switch to our main application domain: scheduling. Reinforcement learning enhanced heuristic search methods were applied to the multi-mode resource-constrained project scheduling problem (MRCPSP) and the decentralized resource-constrained multi-project scheduling problem (DRCMPSP). Activities of the MRCPSP can be scheduled in multiple execution modes. An execution mode defines the duration and resource requirement of an activity. The DRCMPSP incorporates scheduling multiple projects in a decentralized manner where both global and local resources are available. Details of the methods for these two well known project scheduling problems are described in Chapter 5. State-of-the-art results are achieved and many new best solutions were found.

The application of learning automata for flexible job shop scheduling problems is demonstrated in Chapter 6. The proposed methods are tested on datasets from the literature and compared to existing well performing methods. Furthermore a

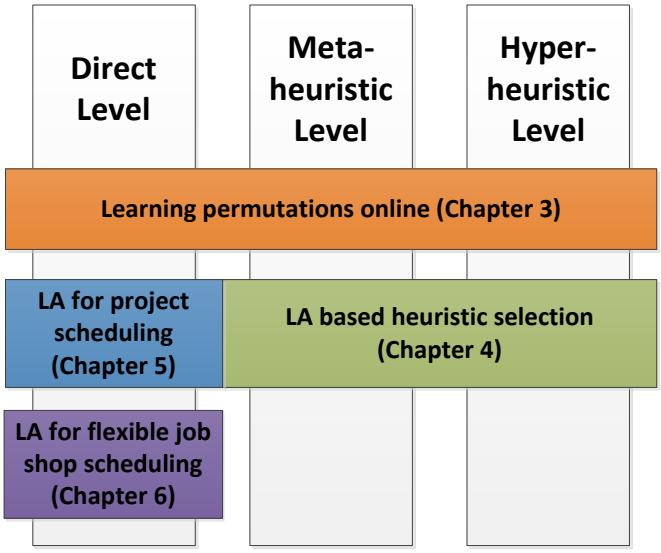


Figure 1.3: Coverage of the RL inclusion levels by the chapters in this thesis.

dynamic version of the problem is introduced with release dates, due dates, and perturbations. New problem instances are generated. The LA based method is adapted to this dynamic setting and compared to pure reactive and global optimization approaches, using several time related objective measures.

Application of the proposed techniques to a production scheduling case in the food-industries is discussed in Chapter 7. This real-world case combines multiple hard optimization problems such as routing, assignment and scheduling. The scheduling part is translated to a project scheduling problem with generalized precedence relations which can be effectively solved by a network of learning automata (as demonstrated in Chapter 5). It is shown where RL can be used to improve the developed scheduling system. In addition, machine learning techniques are employed to 'learn' correct background information, which can be used by the scheduler to obtain more realistic schedules A supervised learning technique was used on historical data to predict machine speeds.

In Chapter 8 we conclude by formulating the main achievements of this thesis in terms of the contribution to the field of heuristic search, scheduling and combinatorial optimization in general. We end this thesis by enlisting several promising directions for future work.

Chapter 2

Reinforcement learning and heuristic search

2.1 Reinforcement learning

Artificial intelligence (AI) is a broad research field in which one tries to develop intelligent machines or agents. These agents have to take actions in order to find some goals or to optimize their performance. Some of the main AI disciplines include, but are not limited to, knowledge representation, planning, natural language processing and machine learning. Machine learning can be useful to create intelligent optimization methods, as stated by Battiti et al. [2008].

Reinforcement Learning (RL) [Kaelbling et al., 1996, Sutton and Barto, 1998a] belongs to the category of machine learning algorithms. A reinforcement learning agent has the computational task of learning which action to take in a given situation (state) to achieve one or more goal(s). The learning process takes place through interaction with an environment (Fig. 2.1), and is therefore different from supervised learning methods that require a teacher. At each discrete time step an RL agent observes the current state s. In each state s the agent can take some action a from the set of actions available in that state. An action a can cause a transition from state s to another state s', based on transition probabilities. The model of the environments contains these transition probabilities. A numerical *reward signal* r is returned to the agent to inform the RL agent about the 'quality' of its actions or the intrinsic desirability of a state. The reward signal is also a part of the model of the environment. An RL agent searches for the optimal policy in order to maximize accumulated reward.

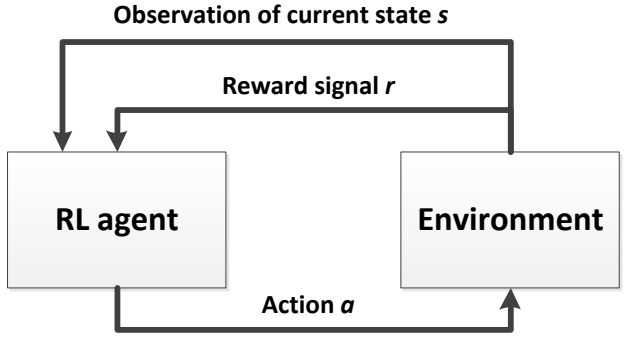


Figure 2.1: The basic reinforcement learning model.

A *policy* π maps states to actions or action probabilities. $\pi(s, a)$ denotes the probability that action a is selected in state s. An RL agent wants to maximize the expected sum of future rewards. When an infinite horizon is assumed, a discount factor γ is used to discount these future rewards. As such, less or more importance is given to rewards further into the future.

One of the main issues in RL is balancing exploration and exploitation, i.e. whether to exploit the already gathered experience or to gather completely new experience. Another important issue is the credit assignment problem, where one has to deal with delayed rewards, and thus answering the question which action from the past should receive credit for a given reward. The latter problem is currently getting limited attention in hybrid RL inspired search methods, as will be shown in Section 2.5.

2.1.1 Policy iteration methods

Two different types of reinforcement learning methods exist, namely policy iteration and value iteration. An RL agent searching directly for the optimal policy in the space of all possible policies is applying a 'policy iteration' method. One category of methods of this type are the policy gradient methods, like the REINFORCE algorithm [Williams, 1992]. Learning Automata (LA) [Narendra and Thathachar, 1989, Thathachar and Sastry, 2004] also belong to this category, and are discussed next.

Learning Automata

LA are simple reinforcement learning devices that take actions in single state environments. A single learning automaton maintains an action probability distribution p, which it updates using some specific learning algorithm or reinforcement scheme. Several reinforcement schemes with varying convergence properties are available in the literature. These schemes use information from a reinforcement signal provided by the environment, and thus the LA operates with its environment in a feedback loop. Examples of linear reinforcement schemes are linear reward-penalty, linear reward-inaction and linear reward-ϵ-penalty. The philosophy of these schemes is to increase the probability of selecting an action in the event of success and decrease it when the response is a failure. The general update scheme is given by:

$$\begin{aligned} p_m(t+1) &= p_m(t) + \alpha_{reward}\beta(t)(1-p_m(t)) \\ &\quad - \alpha_{penalty}(1-\beta(t))p_m(t) \qquad (2.1) \\ &\quad \text{if } a_m \text{ is the action taken at time } t \\ p_j(t+1) &= p_j(t) - \alpha_{reward}\beta(t)p_j(t) \\ &\quad + \alpha_{penalty}(1-\beta(t))[(|A|-1)^{-1} - p_j(t)] \qquad (2.2) \\ &\quad \text{if } a_j \neq a_m \end{aligned}$$

with $p_i(t)$ the probability of selecting action i at time step t. The constants $\alpha_{reward} \in [0,1]$ en $\alpha_{penalty} \in [0,1]$ are the reward and penalty parameters. When $\alpha_{reward} = \alpha_{penalty}$, the algorithm is referred to as linear reward-penalty (L_{R-P}), when $\alpha_{penalty} = 0$, it is referred to as linear reward-inaction (L_{R-I}) and when $\alpha_{penalty}$ is small compared to α_{reward}, it is called linear reward-ϵ-penalty ($L_{R-\epsilon P}$). $\beta(t) \in [0,1]$ is the reward received by the reinforcement signal for an action taken at time step t. $|A|$ denotes the number of actions.

Pursuit Algorithm

Another type of learning automata algorithms are the estimator algorithms. These algorithms use the past history of actions and reinforcements for updating their action probabilities. A distinguished estimator algorithm is the Pursuit Algorithm [Thathachar and Sastry, 1986]. This algorithm maintains two

additional vectors $Z(t)$ and $\eta(t)$. $Z(t)$ stores the total obtained reinforcement for each action, while $\eta(t)$ counts the number of times each action was chosen. Both vectors are used and updated at each step to generate estimates of the reward probabilities $d(t) = \frac{Z(t)}{\eta(t)}$. These estimates are equivalent to the average reinforcement for each action. The reward probability estimates are used to update the action probability vector $p(t)$ as follows:

$$p(t+1) \quad = \quad p(t) + \lambda(e_{M(t)} - p(t)) \tag{2.3}$$

where $0 < \lambda \leq 1$ is the learning parameter, and $e_{M(t)}$ is a unit vector with element at index $M(t)$ equal to 1 and all other elements 0. The greedy action $M(t)$ is determined by

$$d_{M(t)} \quad = \quad \max_i d_i(t) \tag{2.4}$$

A major advantage of the pursuit algorithm is that the reward values are not necessarily in [0,1]. Since most objective function values are not limited to [0,1], this property makes it an interesting candidate for inclusion in heuristic search methods. Further on, it converges faster than L_{R-I} while keeping the interesting theoretical properties [Thathachar and Sastry, 2004]. However, the update step requires more calculation time, thus less attractive for inclusion into a heuristic search method. A limited number of experiments with the pursuit algorithm were carried out on actual cases, but due to the lack of significant results, this technique was omitted for further research.

2.1.2 Value iteration methods

Value iteration methods are more common than policy iteration methods. Value iteration methods do not directly search for the optimal policy, instead they are learning evaluation functions for states or state-action pairs. Evaluation functions, as in the popular Q-learning algorithm [Watkins, 1989b, Watkins and Dayan, 1992] can be used to evaluate the quality of a state-action pair. The Q-learning algorithm maintains an action-value function called Q-values. The Q-learning update rule is defined by

$$Q(s,a) = Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right], \tag{2.5}$$

where s is the previous state, s' the new state, a the action taken in state s, a' a possible action in state s', r the received reward, $\alpha \in [0, 1]$ the learning rate or step size, and $\gamma \in [0, 1]$ the discount rate which indicates the importance of future rewards. In many cases the number of states or state-action pairs is too large to store, and thus the (action)-value function must be approximated. Often an artificial neural network is used to accomplish this task, but any linear or nonlinear function approximation can be applied. Q-learning is known as a Temporal Difference (TD) learning method because the update rule uses the difference between the new estimate and the old estimate of the value function. Other common TD methods are SARSA [Rummery and Niranjan, 1994, Sutton, 1996] and TD(λ) [Watkins, 1989b].

2.1.3 Multi-agent reinforcement learning

Multi-agent reinforcement learning (MARL) refers to systems where multiple reinforcement learning agents act together in a common environment [Busoniu et al., 2008b]. A large number of multi-agent applications where learning could be beneficial exist, e.g. distributed control, robotic teams, collaborative decision making, resource management, etc. In the case of resource management, the resources can be controlled centrally, but managing each resource by its own agent may add a helpful, distributed perspective to the system. The agents can work together (cooperate) or have their individual (maybe conflicting) objectives (competitive). Since the agents all share a common environment, the environment becomes non-stationary for each individual agent. The reward each agent receives is influenced by the actions of the other agents. Excellent multi-agent learning overview papers are given by Busoniu et al. [2008a], Panait and Luke [2005], Yang and Gu [2009], Tuyls and Weiss [2012].

Examples of state-of-the-art algorithms in MARL literature are Joint Action Learning [Claus and Boutilier, 1998], Nash-Q learning [Hu and Wellman, 2000, 2003] and Gradient Ascent algorithms such as the Infinitesimal Gradient Ascent algortihm [Singh et al., 2000].

2.2 Heuristic search

Heuristics are simple and fast techniques to find 'good enough' solutions for combinatorial optimization problems. They are often used when exact methods such as mixed integer linear programming or constraint programming are inadequate. Metaheuristics and hyperheuristics are more general and intelligent heuristic search methods and are described in the following sections.

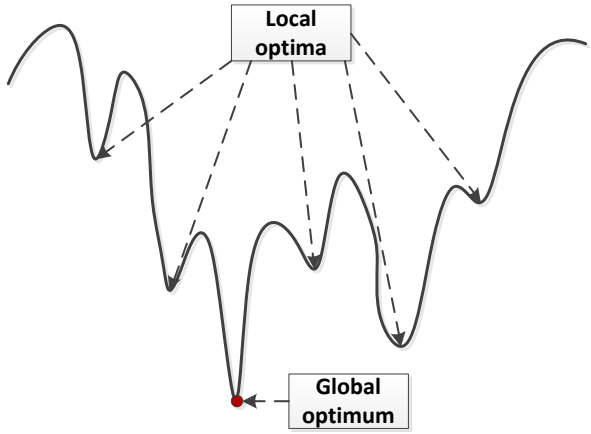


Figure 2.2: Local and global optima of a fictitious search landscape.

2.2.1 Metaheuristics and local search

Metaheuristics [Glover, 1986, Glover and Kochenberger, 2003, Talbi, 2009] are commonly used methods for solving combinatorial optimization problems. They use an objective function to intelligently guide the search towards better solutions. Compared to simple heuristics, metaheuristics employ higher level strategies to escape from a local optima. A local optimum is a solution for an optimization problem for which no improving candidate solution exists using the current set of operators. The goal is to reach the global optimum, i.e. the best solution among all local optima (Figure 2.2)[1]. Metaheuristics are often metaphorically inspired by physics or biological behaviour. Throughout this thesis we focus on single-solution based metaheuristics, also called S-metaheuristics by Talbi [2009]. S-metaheuristics keep only a single solution in memory, opposed to population based metaheuristics (P-metaheuristics) which store multiple solutions (the population).

A balance must be made between *diversification* and *intensification*. Diversification stimulates the search to find solutions in other areas of the search space, while intensification limits the search to promising regions. Common S-metaheuristic techniques are tabu search [Glover, 1986, Glover and Laguna, 1997], simulated annealing [Kirkpatrick et al., 1983b] and variable neighbourhood

[1]The search landscape is not intended to be continuous, but is just for illustration purposes.

search [Mladenovic and Hansen, 1997a], while genetic algorithms [Holland, 1975] and ant-based methods [Dorigo, 1992., Dorigo and Gambardella, 1997] belong to the P-metaheuristic class.

Local search

Local search is an S-metaheuristic method which employs one or multiple neighbourhood functions to iteratively improve upon a starting solution by moving from one solution to another. These neighbourhood functions define small *local* changes to the current solution, e.g. swapping two elements or adding an element to the solution.

In recent years the application of metaheuristics and local search has led to better understanding how to tackle complex optimization problems in practice. Still it is not clear which method is most appropriate for a given problem. Metaheuristics typically require finetuning multiple parameters. Moreover these parameters are problem or even instance specific and thus require expert knowledge. Out of the box software solutions are therefore very labour-intensive, complex and as a result very expensive. This increases the need for automated systems, for which the human part in design and tuning of heuristic search methods is limited. For example: how to a priori choose the length of the tabu list in tabu search or the cooling schedule in a simulated annealing method? Recent trends in metaheuristic research show this need for a more general approach. The hybrid methods are popular, where multiple optimization techniques are combined. Another recent trend in the area of local search is the development of so called hyperheuristics [Burke et al., 2003a], discussed in the next section.

2.2.2 Hyperheuristics

Hyperheuristics [Burke et al., 2003a] are search methods operating at a higher level than metaheuristics. Opposed to metaheuristics, hyperheuristics perform a search in a space of (meta)heuristics. Typically these hyperheuristics do not use any problem specific knowledge. Only information about the performance of the lower-level (meta)heuristics is given, e.g. the achieved improvement or the required CPU-time. In this way hyperheuristics tend to be more general than metaheuristics, by using information that is common to all optimization problems. The methods proposed in this thesis are developed towards a similar goal as hyperheuristics. Moreover, some of them can be a part of a hyperheuristic (e.g. Chapter 4). Misir [2012] gives a complete survey and the state-of-the-art of hyperheuristics.

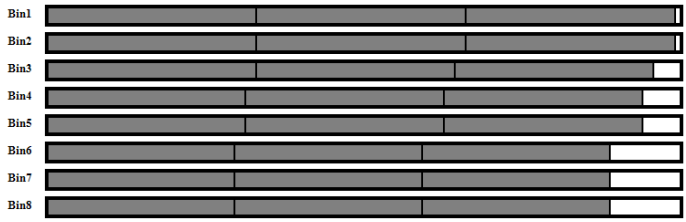


Figure 2.3: Example of a solution for the one-dimensional bin packing problem constructed with the minimal bin slack heuristic.

2.2.3 Example

To clarify the differences between a heuristic, a metaheuristic and a hyperheuristic we consider the following example. In a classical one-dimensional bin packing problem, n items with size $s_i,\ i = 1..n$ have to be packed into a minimum number (m) of bins with capacity C. For each bin $B_j,\ j = 1..m$, the sum of the sizes of the items in that bin must be smaller or equal than the bin capacity.

$$\sum_{i \in B_j} c_i \leq C \tag{2.6}$$

An example heuristic for this problem sorts the items by non-increasing size, and puts them one by one in the first available bin with sufficient free capacity. This heuristic is called first fit decreasing. Better heuristics exist, such as the best fit decreasing and the minimal bin slack heuristics [Gupta and Ho, 1999, Fleszar and Hindi, 2002]. The best fit decreasing heuristic is similar to first fit decreasing, but puts the items in the best fitting bin, i.e. the fullest bin offering sufficient free capacity. The minimal bin slack heuristic and its extensions is one of the best performing heuristics in bin packing literature. The heuristic recursively attempts to find a set of items that fits the bin capacity as much as possible. An example solution with 24 items and 8 bins is shown in Figure 2.3. The solution was constructed using the minimal bin slack heuristic.

A simple local search metaheuristic for this problem for example, makes use of an ordered list of items and places the items in order in the first bin with sufficient free capacity. A local search e.g. tries all possible swaps of items in the list to create new solutions. This is called a swap neighbourhood. A steepest descent will take the best solution (i.e. the solution with a minimal number of bins) from the swap neighbourhood to continue with, and stops when no improving solution can be found. More advanced metaheuristics like tabu search or simulated annealing can be applied.

A hyperheuristic can manage these heuristics and metaheuristics and decide on the acceptance of the solution they produce. The selection of the (meta)heuristics can be done at random or in a more intelligent way. For the acceptance part, any of the existing acceptance mechanisms can be used, e.g. improving-or-equal, which accepts solution if their quality is better than or equal to the quality of current solution.

2.3 Relationships between reinforcement learning and heuristic search

A direct link exists between reinforcement learning and search algorithms. Given a state space, an action space and a reward function, the reinforcement learning problem can obviously be reduced to a search in the space of policies. It can be seen as an online stochastic optimization problem with unknown objective function. Thus similar issues like exploration and exploitation are faced, often called diversification and intensification in metaheuristic literature.

Evolutionary algorithms like GAs are often used for solving combinatorial optimization problems. However, GAs and (reinforcement) learning are closely related [Panait et al., 2008]. Shapiro shows the usage of GAs in machine learning, and states that reinforcement learning is the most important application of genetic algorithms in machine learning. Moriarty et al. [1999] gives an overview on the usage of evolutionary algorithms for reinforcement learning. A GA receives feedback (the fitness value) from the environment similar to an RL method. Because of the use of a population, a GA can also be used for learning in multi-agent systems. The interaction between learning and evolution was studied in [Ackley and Littman, 1991].

Unlike the subject of the present thesis, where (reinforcement) learning methods are used to support heuristic search, there are methods which do exactly the opposite, i.e. using metaheuristic methods to improve learning. Bennett and Parrado-Hernández [2006] discuss this interplay between optimization and machine learning.

2.4 Opportunities for learning

Reinforcement learning methods can be utilized at different levels for solving combinatorial optimization problems. We define the following three levels (shown in Figure 2.4).

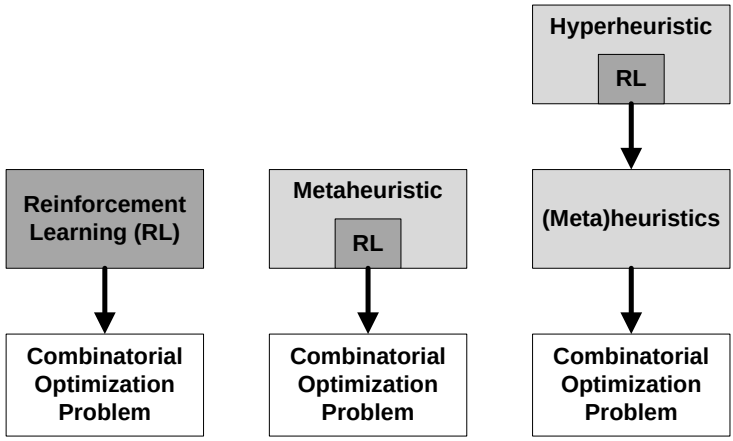


Figure 2.4: Different levels to employ reinforcement learning for combinatorial optimization.

- The **direct level** (left): RL is directly applied to the problem.
- The **metaheuristic level** (middle): RL is used as a component of a metaheuristic.
- The **hyperheuristic level** (right): RL is used as a component of a hyperheuristic.

These levels will be used to further classify the methods described in this thesis.

When metaheuristics are considered, learning may help to find good settings for various parameters or components. For example, RL methods can learn properties of good *starting solutions* or an *objective function* that guides a metaheuristic towards good quality solutions. Such an approach is adopted in [Boyan, 1998, Boyan et al., 2000], where a function is learned that is able to guide the search to good starting solutions. Another component on which learning can be applied is the *neighborhood* or *heuristic selection*. A learning method can learn which are the best neighborhoods or heuristics to construct or change a solution at any time during the search, so that in the end good quality solutions can be generated. Such an approach is applied by Zhang and Dietterich [1995] and Nareyek [2001]. When a classical hyperheuristic with acceptance and selection mechanism is used, learning can be applied to both mechanisms. A learning method can learn to select the low-level heuristics (e.g. in [Burke et al., 2003c] and [Misir et al., 2009]), or it can ascertain when to accept a move. To summarize, the components possibly involved, include but are not limited to:

- the starting solution,
- the objective function,
- the neighborhoods or heuristics selection,
- the acceptance of new solutions/moves.

All these parameters or components can be updated by the RL algorithm in an adaptive way.

Alternatively, RL methods can also be directly applied to solve optimization problems (left image of Figure 2.4). In this case they are not considered hybridized and are themselves used as a metaheuristic. The RL method learns and directly assigns values to the variables. Such approaches are investigated by Gambardella et al. [1995], Miagkikh and Punch [1999], Wauters et al. [2010] and Wauters et al. [2011]. The latter two papers are developed in the course of this thesis and described in Chapter 5. In contrast to [Gambardella et al., 1995] and [Miagkikh and Punch, 1999] which employ value iteration methods such as Q-learning, we directly search in the space of policies using a network of learning automata.

A possible indication for the benefit of including RL in a search algorithm is the presence of a random component. By replacing this random component with an RL component the algorithm develops a more intelligent decision mechanism. For example in a hyperheuristic with a simple-random selection step, the selection can be replaced by some RL method, as we proposed in [Misir et al., 2009], and further discussed in Chapter 4 of this thesis.

Yet another opportunity for learning arises when either the problem is intrinsically distributed or can be split into several subproblems. Examples of such include distributed scheduling and planning problems such as the decentralized resource-constrained multi-project scheduling problem (DRCMPSP) [Confessore et al., 2007] and [Homberger, 2009]. This problem considers scheduling multiple projects simultaneously, with each project having multiple jobs. A job requires local or global resources, which are available for either all jobs in the project or have to be shared among all projects respectively. Some local objectives can be optimized, for example the makespan of the individual projects, whereas global objectives, such as the average project delay or the total makespan can be minimized. For this kind of problems multi-agent reinforcement learning methods are appropriate. When one or more global objectives need to be optimized, the agents share a common goal and can thus be cooperative. The agents have to coordinate in order to jointly improve their decisions. Using a common reward, the agents can simply but effectively coordinate their decisions.

We applied this approach in [Wauters et al., 2010, 2012b] for the DRCMPSP, which is further described in Chapter 5 of this thesis.

Some important counter-indications might shed doubts on the applicability of RL in (meta)heuristic search. A metaheuristic operating on a combinatorial optimization problem behaves as an unknown random environment for the reinforcement learner, which makes it a hard learning task. Moreover, the Markov Property, which is the basic hypothesis for most theoretical RL results, does not hold for a heuristic search method. In heuristic search methods, the next optimal action depends on the previous actions (memory), which invalidates the Markov property. Additionally, time or information available for learning is often limited. As shown further in this thesis these arguments do not affect the applicability of RL methods in practice. A well thought design of states, actions and reward signal is needed. This will be discussed in the following sections.

2.4.1 States and actions

Before applying RL to a problem, the set of possible states and the set of possible actions available in each state, have to be defined. Many possible ways exist to accomplish this. First of all we can make a distinction between *search-dependent*, *problem-dependent* and *instance-dependent* state space definitions. A search-dependent state space definition uses observations of the search process itself, such as the current iteration, the number of successive non-improving iterations, or the total improvement over the initial solution. A problem-dependent setting is defined by the use of generic problem features, like the Resource Dilation Factor for scheduling problems, as defined by Dietterich and Zhang [2000]. An instance-dependent setting uses instance-specific features, like the number of tardy jobs in a scheduling problem, or the number of full bins in a bin-packing problem. Combinations of these three settings are also possible. When a problem-dependent or a search-dependent state space definition is used, the learned information can possibly be transfered to other instances of the same problem, or even to other problems. In many cases the properties of the solutions to the optimization problem itself cannot be used directly, due to the curse of dimensionality. It would be better to use some extracted problem features. Take for example a traveling salesman problem (TSP), where one searches for the shortest tour through a set of cities. If one should use the encoding of a complete tour directly as the state, then the number of states would grow exponentially, i.e. $n!$ with n the number of states.

The set of possible actions in each state is determined by the decisions an RL agent or multiple RL agents have to make. These actions are parameter values or components of the metaheuristic one wants to learn.

2.4.2 Reward function

Experience gathering is a prerequisite to learning, which can be achieved either online or offline. Experience is hardly present in combinatorial optimization problems. Often only a single numerical value is available, indicating the quality of a complete solution. However, reward functions are very important for an RL method in order to learn some valuable information. As stated by Dietterich and Zhang [2000], there are three requirements that a reward function should satisfy. First of all, it should give higher rewards to better solutions. Secondly, it should encourage the reinforcement learning system to find efficient search policies, i.e. search policies that involve only a few steps. Thirdly, it should be a normalized measure, in order to be transferable to new problem instances. An additional fourth requirement may be added, namely that it should be computationally efficient. When designing a reward function for a hybrid RL-metaheuristic method we do take these four requirements into account.

The RL framework is able to deal with delayed rewards, nevertheless using the final solution quality as a reward would mean that one has to repeat the search multiple times. Restarting the search is not always possible, e.g. because of limited computation time. In such a case, only a small number of runs can be performed. As an example, we refer to the winning hyperheuristic algorithm of the Cross-domain Heuristic Search Challenge (CHESC2011)[2]. The winning algorithm developed by Misir et al. [2012] had the best overall performance over a set of six problem domains, but due to the lack of sufficient time for learning, it had a very bad performance on the personnel scheduling domain. For this reason, it is necessary to use the information available during a single search run (online), and adapt the current search appropriately. An example of an immediate reward signal is the improvement achieved by a selected move type or heuristic (see Chapter 4).

2.5 Literature overview

The combination of heuristic search and (reinforcement) learning is relatively new. Only few references describe and use such a hybrid approach. Applied problem domains include, but are not limited to, scheduling, packing and routing. Table 2.1 compares the methods further discussed in this section by the used RL-method, heuristic search method and the component involving RL. The methods in bold are part of this thesis.

[2]CHESC website: `http://www.asap.cs.nott.ac.uk/external/chesc2011/`

Reference	RL method	Heuristic search	Component with RL	Problem(s)
[Zhang and Dietterich, 1995]	TD(λ)	Constructive Method	Heuristic selection	Job scheduling
[Gambardella et al., 1995]	Q-learning	Ant system	Direct	ATSP
[Miagkikh and Punch, 1999]	Q-learning	GA	Direct	ATSP and QAP
[Boyan, 1998, Boyan et al., 2000]	No RL (regression)	Local search (e.g. SA)	Obj. function, start solution	Bin packing, channel routing, SAT,...
[Moll et al., 1998]	TD(λ)	2-opt local search	Obj. function	Dial-a-ride
[Nareyek, 2001]	Utility values	CSP solver	Heuristic selection	Orc Quest and Logistics Domain
[Burke et al., 2003c]	Utility values	Hyperheuristic	Heuristic selection	Timetabling and rostering
[Bai et al., 2007]	Utility values + discount	Hyperheuristic	Heuristic selection	Course timetabling
[Misir et al., 2009]	**LA**	**Hyperheuristic**	**Heuristic selection**	**TTP**
[Wauters et al., 2009c]	**LA**	GA	**Direct**	**Project scheduling (MRCPSP)**
[Özcan et al., 2010]	Utility values	Hyperheuristic	Heuristic selection	Examination timetabling
[Wauters et al., 2010]	**LA (+Dispersion Game)**	**-**	**Direct**	**Project scheduling (DRCMPSP)**
[Wauters et al., 2011]	**LA**	**-**	**Direct**	**Project scheduling (MRCPSP)**
[Torkestani and Meybodi, 2011]	LA	-	Direct	Vertex coloring

Table 2.1: Comparison of hybrid RL-heuristic search methods.

2.5.1 Value iteration methods for heuristic search

As mentioned earlier, the methods described in this thesis employ learning automata, which is a policy iteration method. For completeness we give an overview of the existing value iteration methods for heuristic search in this section.

One of the first papers covering the combination of learning and metaheuristic search is presented by Zhang and Dietterich [1995]. A reinforcement learning method is applied to learn domain-specific heuristics for the NASA space shuttle payload processing problem, which is modeled as a job shop scheduling problem. A value function is learned offline using a temporal difference algorithm TD(λ) together with a neural network. General features of the schedules (solutions) such as the percentage of the time units with a violation are used to represent a state. The possible actions are taken from a set of repair heuristics. After learning the value function on a number of small problem instances, it is used over multiple instances of the same problem. The TD algorithm is compared to an existing method for the problem, i.e. an iterative repair method with simulated annealing. The reinforcement learning based method outperforms the iterative repair method. It is noteworthy that the value functions that were learned on small problem instances also have a very good performance on larger instances. Dietterich and Zhang [2000] provide a more detailed description and application of this approach.

Another early contribution to the application of RL for solving combinatorial optimization problems can be found in [Gambardella et al., 1995]. The paper describes the Ant-Q algorithm, which combines the Ant System and the Q-Learning algorithm, and has been successfully applied to the Asymmetric Traveling Salesman Problem (ATSP). Ant System is based on the observation of ant colonies. Each ant from a colony constructs a solution for the ATSP, called a tour. The method uses a modified version of Q-values, called AQ-values. These AQ-values are updated using a Q-learning update rule. The delayed reward, which is calculated when each ant completes a tour, is based on the best tour of the current iteration or on the best tour of all past iterations, taking each ant into account. The Ant-Q algorithm shows an interesting property. It was observed that the Ant-Q agents do not make the same tour, demonstrating the explorative character of the search method.

Miagkikh and Punch [1999] present an algorithm that combines reinforcement learning with genetic algorithms for the Asymmetric Traveling Salesman Problem (ATSP) and Quadratic Assignment Problem (QAP). For the ATSP a Q-learning [Watkins, 1989b, Watkins and Dayan, 1992] method is used to both express and update the desirability of choosing city a after city b. A state is a city,

and an action is another city following the aforementioned city in the tour. A population of RL agents with desirability values (Q-values) is formed, with each agent holding one solution. The offspring is constructed by replicating solution parts from one parent and filling in the other parts using the desirability values (updated by a q-learning update rule) of the other parent, rather than the traditional genetic crossover operators method. The reward is a weighted combination of immediate and global rewards based on the tour lengths of the new solution and the solutions of the parents. The QAP is solved using a simplified update rule that does not require a particular order as opposed to the Q-learning update rule. Competitive results compared to other population based approaches are shown for both addressed problems.

Boyan [1998] and Boyan et al. [2000] describe the STAGE algorithm, which searches for good quality solutions using two alternating phases. The first phase runs a local search method, e.g. hill climbing or simulated annealing from a starting solution until a local optimum is reached. During this phase, the search trajectory is analyzed and used for learning an evaluation function. This is achieved by training a linear or quadratic regression method using the properties or features of the visited solutions and the objective function value of the local optimum. The authors point out that in some conditions a reinforcement learning method like TD(λ) belonging to the class of the temporal-difference algorithms, may make better use of the training data, converge faster, and use less memory during training. The second phase performs hill climbing on the learned evaluation function to reach a new starting solution for the first phase. This phase enables the algorithm to learn how to find good starting solutions for a local search method. Empirical results are provided for seven large-scale optimization domains, e.g. bin-packing, channel routing, ... This demonstrates the ability of the STAGE algorithm to perform well on many problems.

Moll et al. [1998] combine aspects taken from the research by Zhang and Dietterich [1995], Boyan [1998] and Boyan et al. [2000]. A reinforcement learning algorithm TD(λ) is applied to learn a value function in an offline training phase, and then uses this function to solve other instances of the same problem. This method also uses features of solutions for representing a state. A linear function approximation algorithm is used. The method is applied to the dial-a-ride problem, and was compared to both the STAGE algorithm, and a 2-opt and 3-opt local search method. The method performs better than 2-opt and STAGE if the same calculation time is used. It was not as performant as 3-opt, but a lot faster.

Gabel [2009] gives an extensive overview of single and multi-agent RL approaches for distributed job-shop scheduling problems. Both value function-based and policy search-based RL methods are discussed, including policy gradient RL methods and Q-learning.

2.5.2 RL enhanced hyperheuristics

Hyperheuristics are perfectly suited for inclusion of learning methods. Hyperheuristics are categorized in 'heuristics to choose heuristics' and 'heuristics to generate heuristics'. In this thesis we focus on the first category. Reinforcement learning can help to choose the lower level heuristics, as discussed in the following papers.

Nareyek [2001] describes a non-stationary reinforcement learning method for choosing search heuristics. At each decision point weights are used to select the search heuristics via a probabilistic selection rule (softmax) or by randomly selecting among the choices with maximal value. Based on the increase/decrease of the objective function the weights of the search heuristics are updated using simple positive/negative reinforcement rules (e.g. incrementing/decrementing the weight value). Different selection and reinforcement method combinations are tested on two types of problems - the Orc Quest problem and problems from the Logistics Domain benchmark. Nareyek conclude that a weak positive reinforcement rule combined with a strong negative reinforcement rule works best on the tested problems.

Burke et al. [2003c] present a hyperheuristic in which the selection of low-level heuristics makes use of basic reinforcement learning principles combined with a tabu search mechanism. The reinforcements are performed by increasing/decreasing the rank of the low-level heuristics when the objective function value improves/worsens. The hyperheuristic was evaluated on various instances of two distinct timetabling and rostering problems and showed to be competitive with the state-of-the-art approaches. The paper states that a key ingredient in implementing a hyperheuristic is the learning mechanism.

An interesting study on memory length in learning hyperheuristics is performed by Bai et al. [2007]. Utility values or weights are used to select the low-level heuristics, similar to [Nareyek, 2001] and [Burke et al., 2003c]. A discount factor is added to this mechanism to discount rewards later on in the search process, and thus obtaining a short term memory. The results obtained on a course timetabling problem show that a short term memory can produce better results than both no memory and infinite memory.

Özcan et al. [2010] present a hyperheuristic with an RL selection mechanism and a great-deluge acceptance method for the examination timetabling problem. A set of exams must be assigned a timeslot and possibly a room while respecting a number of hard and soft constraints, such as the room capacity. An RL method based on utility values with simple update rules is used, similar to what was presented by Nareyek [2001]. The idea is that a heuristic is selected when it results in a lot of improving moves, and thus has a higher utility value.

When a heuristic i results in an improving move, the utility value u_i of that heuristic is incremented, and in case of a worsening move the utility value is lowered using three different rules, namely subtractive ($u_i = u_i - 1$), divisional ($u_i = u_i/2$) and root ($u_i = \sqrt{u_i}$). Upper and lower bounds are applied to the utility values to encourage exploration in further steps. Experiments are performed with different settings for the selection of the heuristics, the upper and lower bound, and the negative utility adaptation mechanism. The setting with a maximal selection (i.e. selecting the heuristic with a maximal utility value) and subtractive negative utility adaption mechanism performed the best. The method improves the performance of a non learning simple-random great-deluge hyperheuristic on the examination timetabling problem.

2.5.3 Search supported by Learning Automata

Recently, learning automata have been introduced to solve combinatorial optimization problems. Most of these applications are further described in subsequent chapters of this thesis. In [Misir et al., 2009] we have presented a heuristic selection method for hyperheuristics, which they have applied to the traveling tournament problem. Instead of using simple reinforcement rules, a learning automaton was used for the selection. In [Wauters et al., 2009c] we describe the combination of a genetic algorithm and learning automata to solve the Multi-Mode Resource-Constrained Project Scheduling Problem (MRCPSP). The GA is applied to find good activity orders, while the LA are used to find good modes, a mode being an important decision variable of the scheduling problem. This work is extended in [Wauters et al., 2011] where the GA is replaced by a network of learning automata [Thathachar and Sastry, 2004]. All decision variables (i.e. activity order and modes) of the scheduling problem (MRCPSP) are now directly chosen by multiple LA. The method produces state-of-the-art results for the MRCPSP. In [Wauters et al., 2010] we follow a very similar approach for the Decentralized Resource-Constrained Multi-Project Scheduling Problem (DRCMPSP). Multiple projects are scheduled factoring in the availability of both private and shared resources, while a global objective, i.e. the average project delay, is optimized. A network of learning automata searches for activity orders resulting in good schedules for each single project, while a dispersion game is employed to coordinate the projects. A multi-agent system with project managers and network of LA for solving the DRCMPSP was applied in [Wauters et al., 2010]. One motivating factor for organizing the activities in a project as learning automata is that theoretical convergence properties hold in both single and multi automata environments. One of the foundations for LA theory is that a set of decentralized learning automata using the reward-inaction update scheme is able to control a finite Markov Chain with

unknown transition probabilities and rewards. In [Littman, 1994], this result was extended to the framework of Markov Games. That is a straightforward extension of single-agent Markov decision problems (MDP's) to distributed multi-agent decision problems. However, the convergence properties fail to hold here since the activity-on-node model does not bear the Markov property. Good results can be achieved with the network of LA in the single project scheduling scheme, as we show in [Wauters et al., 2011]. Torkestani and Meybodi [2011] proposes a cellular learning automata for solving the minimum vertex colouring problem. Each vertex employs a learning automata which chooses its own color based solely on the colors selected by its adjacent vertices. The authors show that this learning algorithm outperforms the existing methods both in terms of running time and quality. The methods aforementioned are added to the bottom of Table 2.1.

2.6 Conclusion

A combination of reinforcement learning and heuristic search methods can lead to more general and adaptive methods. Many opportunities for combining both techniques are available. Well-considered design of the reward signal and state/action definitions is needed to get the best out of these hybrid methods.

Many hybrid approaches generate high quality results, in some cases even up to the state-of-the-art. One might notice that most early hybrid RL-metaheuristic methods use RL algorithms as Q-learning and TD(λ), which make use of delayed rewards. Recent methods, mostly applied to hyperheuristics, are using a more simple RL mechanism based on utility values operating in a single state environment, and thus do not benefit the full power of RL which deals with the problem of delayed rewards and the credit assignment problem. LA or networks of LA are also simple to apply, but have strong theoretical advantages, which are also observed in practice. In subsequent chapters these LA are extensively used to enhance heuristic search methods for general problems such as learning permutations, to more specific (project) scheduling and real world optimization problems.

Chapter 3

Learning permutations

3.1 Introduction

The process of finding good quality permutations, i.e. arrangement of objects or values into a particular order, is a recurring element in combinatorial optimization problems. The permutations may represent a full or a partial solution to such problems. Typical examples can be found in routing and scheduling. The traveling salesman problem (TSP) for instance, aims at finding a tour of minimum distance through a number of cities. A solution can be represented by a permutation, defining the order for visiting the cities. Many solutions for scheduling problems contain some permutation representation. A solution for the permutation flow shop scheduling problem (PFSP) is such an example. In the PFSP a number of jobs have to be sequenced in order to be processed on a predefined number of resources. All these problems have a search space exponential in the number of inputs n (cities, jobs, ...). Due to the very nature of permutations there are at least $n!$ different solutions. An objective function representing the quality of the solutions has to be optimized. If a solution to a problem can be represented by a permutation, then the objective function value states how good the permutation is.

In fact, we can imagine the following general problem (see Figure 3.1): given a permutation π, an objective function f can give a value for that permutation $f(\pi)$. Function f can be the objective function of an optimization problem, and it is assumed that the function is not known. It is a black box. Since all values can be normalized, we can assume that $f(\pi) \in [0, 1]$, with a value $f(\pi) = 0$ meaning the worst permutation and $f(\pi) = 1$ the best or optimal permutation.

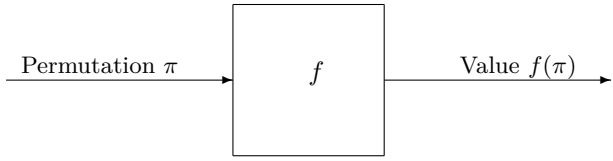


Figure 3.1: General permutation problem seen as a black box function.

In the present chapter, this general permutation problem is tackled using simple reinforcement learning devices, called Learning Automata (LA). It is shown how these methods are capable of learning permutations of good quality ($f(\pi)$ close to 1) without the use of problem specific information or domain knowledge. However it is not our primary concern to outperform existing specialized optimization methods for these problems, but only to show the strength of simple and general learning mechanisms for online permutation learning.

Several related papers have been published in the literature. Population based incremental learning (PBIL) [Baluja, 1994] is a method related to genetic algorithms for solving optimization problems. It maintains a real-valued probability vector for generating solutions. PBIL is very similar to a cooperative system of finite learning automata where the learning automata choose their actions independently and update with a common reward, i.e. all LA are updated with the same reward value. Population based methods like GAs are related to (reinforcement) learning (as discussed in Section 2.3.

COMET [Baluja and Davies, 1998] incorporates probabilistic modeling in conjunction with fast search algorithms for application to combinatorial optimization problems. The method tries to capture inter-parameter dependencies by generating a tree-shaped probabilistic network.

PermELearn [Helmbold and Warmuth, 2009] is an online algorithm for learning permutations. The approach makes use of a doubly stochastic[1] weight matrix to represent estimations of the permutations, together with exponential weights and an iterative procedure to restore double stochasticity.

The work in the present chapter shows the learning of permutations and the use of learning automata for solving optimization problems, and was published as: *Wauters, T., Verbeeck, K., De Causmaecker, P., Vanden Berghe, G. (2012). Fast permutation learning. In Hamadi, Y. (Ed.), Schoenauer, M. (Ed.), LION 6 proceedings: Vol. LNCS 7219. Learning and Intelligent OptimizatioN*

[1] A matrix is doubly stochastic if all its elements are nonnegative, all the rows sum to 1, and all the columns sum to 1.

	T0	T1	T2	T3
A0	3	4	1	3
A1	3	2	3	1
A2	3	4	2	2
A3	2	3	4	4

Table 3.1: Cost matrix for an example assignment problem of size $n = 4$

Conference. Paris, France, 16-20 January 2012 (pp. 292-306) Springer.
Different categories of permutation functions and their properties are discussed. Centralized and decentralized methods based on learning automata for online permutation learning are described. The presented methods are analyzed using some well known benchmarks, and a successful application to project scheduling is demonstrated. The methods in this chapter are applicable to all RL inclusion levels as shown in Figure 2.4.

Scheduling is one of the main application areas of this thesis. Typically, solutions for scheduling problems can be encoded as a sequence of jobs (a permutation). The methods proposed in this chapter were applied in the following scheduling papers and will be further discussed in Chapter 5. In Wauters et al. [2011] we introduce a method with multiple learning automata to cooperatively find good quality schedules for the multi-mode resource-constrained project scheduling problem (MRCPSP). A common reward is used based on the makespan of the scheduling solution. In Wauters et al. [2010] we present a method using learning automata combined with a dispersion game for solving the decentralized resource-constrained multi project scheduling problem (DRCMPSP). To date the method is part of the state-of-the-art for this problem[2].

3.2 Permutation functions

A permutation function is a function mapping permutations to values. This can take several forms. Most of them are highly non-linear. The most straightforward function is one that gives a value to each individual position. Take for example an assignment problem where a matrix defines the cost for assigning an agent to a task. A permutation, where task i at position j is performed by agent j, is a possible solution representation for this problem.

Solutions to the assignment problem with the cost matrix from Table 3.1 result in a permutation function as shown in Figure 3.2. The total cost for each

[2]Multi project scheduling problem library: http://www.mpsplib.com ; accessed on July: 16, 2012

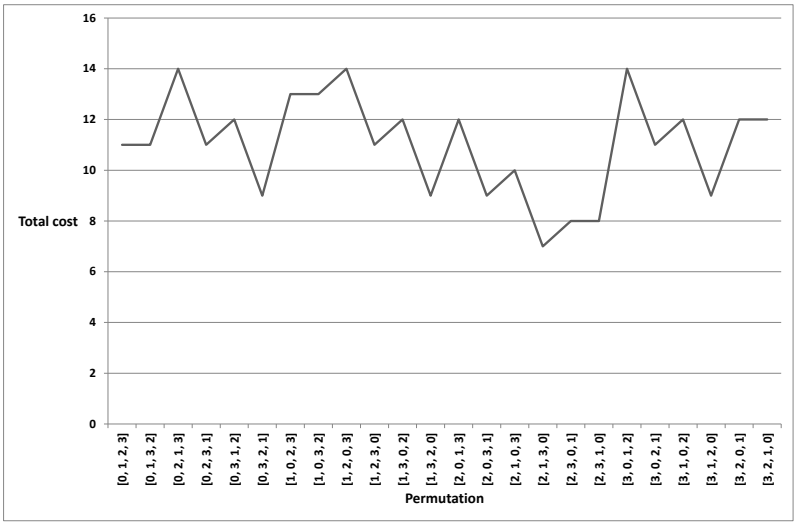


Figure 3.2: Permutation function for an example assignment problem.

permutation is plotted. The search space has an optimal solution $[2, 1, 3, 0]$ with a total cost of 7. This solution denotes that task 2 is performed by agent 0, task 1 is performed by agent 1, task 3 is performed by agent 2 and task 0 is performed by agent 3. It is noteworthy that the assignment problem is solvable in polynomial time by the Hungarian method [Kuhn, 1955], but is here only used as an example. Moreover, the solutions of the LP relaxation are always integer valued. This is because the constraint matrix is totally unimodular.

For many problems using a permutation representation the total cost or value is determined by the adjacency of elements. For example, if element A is directly followed by element B in the permutation, there is a cost of c_{AB}. These costs can be symmetric ($c_{AB} = c_{BA}$) or asymmetric ($c_{AB} \neq c_{BA}$). In this category of permutation functions we can also make a distinction between cyclic and non-cyclic functions. In cyclic permutation functions there exists a cost between the first and the last element of the permutation. A typical example where the permutation function is based on adjacency costs and is also cyclic is the TSP. In the TSP the cities are the elements in the permutation, and the cost between adjacent elements is the distance between the cities. The distance between

the first city and last city is also counted in the evaluation, which makes the permutation function for TSP cyclic.

To summarize, we distinguish the following default permutation function categories:

- **individual position**
- **adjacent positions (cyclic and non-cyclic)**

Many permutation functions use or combine elements from these default categories.

Some optimization problems have additional constraints on the permutations. For example, one can have precedence constraints, imposing that one element must occur before or after another element. Examples include the sequential ordering problem (SOP) which is a TSP with precedence constraints) and project scheduling problems. Yet another additional constraint can be that several elements must be adjacent to each other and form groups. As often done, one can incorporate these or any other additional hard constraints by adding a high penalty to the cost value of the permutation.

3.3 Learning permutations

In order to learn permutations of good quality one or more learning components are put in a feedback loop with the permutation evaluation function f (i.e. the environment), as is shown in Figure 3.3. The rest of this section describes a number of centralized and decentralized approaches for performing this learning task.

3.3.1 Naive approach

A naive and centralized approach (Figure 3.4) to learning permutations using learning automata would be to assign one action per permutation. An action being a particular decision to take by the LA. This results in a total of $n!$ actions, which is impractical for larger n both with respect to calculation time and memory usage.

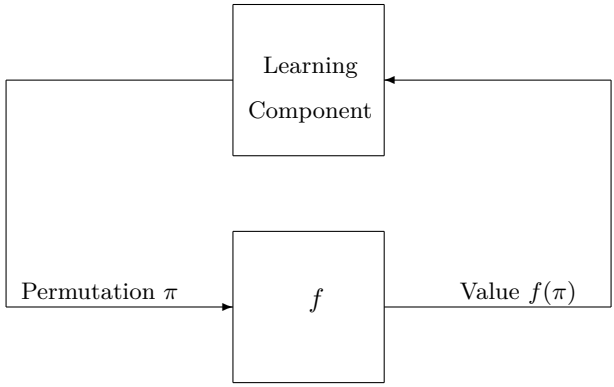


Figure 3.3: Learning component in a feedback loop with the permutation learning problem

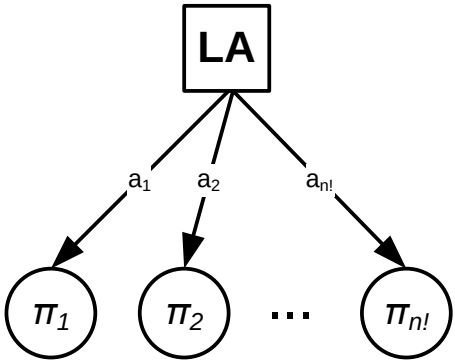


Figure 3.4: A naive approach, one LA with one action per permutation.

3.3.2 Hierarchical approach

A better approach would be to divide the learning task among different learning automata, more specifically a tree of learning automata, called a hierarchical learning automaton [Thathachar and Ramakrishnan, 1981]. An example of such a hierarchical learning automaton for $n = 3$ is shown in Figure 3.5. An LA at depth $d \in \{1, 2, \ldots, n\}$ in the tree is responsible for choosing the element at position d in the permutation. Each LA at depth d has $n + 1 - d$ actions, excluding all the actions chosen in the LA in the path from this LA to the root of the tree. The advantage of this hierarchical approach is that each individual LA has a smaller action space (maximum n). There is also a drawback. In case

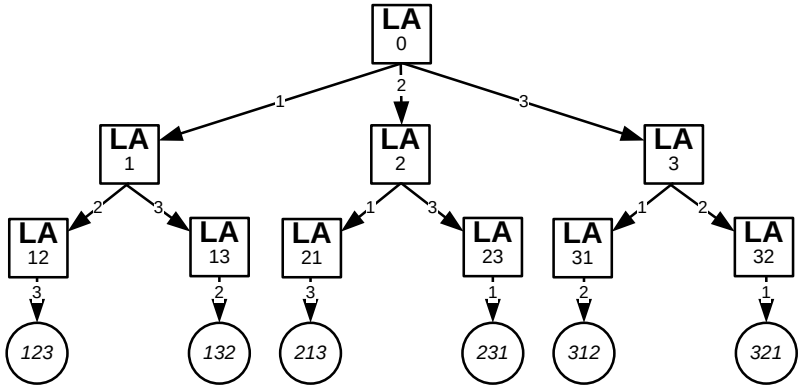


Figure 3.5: An example of a hierarchical learning automaton for $n = 3$.

of a large exploration, the whole search tree is visited in the worst case, which results in $\sum_{d=1}^{n} \frac{n!}{d!}$ LAs. Since all action selection probabilities for each LA have to be stored, this can be very memory intensive.

3.3.3 Probability matrix approach

To deal with the large memory requirements of the hierarchical approach we developed an algorithm with a compact representation. Following [Helmbold and Warmuth, 2009], we use a doubly stochastic matrix P with n rows and n columns. P_{ij} is the probability for element i to be on position j in the permutation. The approach works as follows:

1. generate a uniform doubly stochastic matrix P with: $\forall_{i=1..n} \forall_{j=1..n} P_{ij} = \frac{1}{n}$
2. select a permutation π using P
3. retrieve a reward $r = f(\pi)$ for the selected permutation
4. update P using reward r and a specific LA update rule
5. repeat from step 2 until some stopping condition is met.

These steps are now described in more detail.

Selecting a permutation from P: Several methods can be used to select a permutation from a doubly stochastic matrix. A first method is to uniformly

p1	p2	p3	p4	H	Entropy based row order
0.035	**0.947**	0.000	0.018	0.35	2
0.020	0.023	0.001	**0.956**	0.31	1
0.673	0.001	0.326	0.001	0.92	3
0.272	0.030	**0.673**	0.025	1.18	4

Table 3.2: Entropy based row order for a probability matrix of size $n = 4$. Largest probabilities indicated in bold.

select a row i and then use a probabilistic selection (e.g. roulette wheel selection) on this row for determining at which position j we have to put element i in the permutation. After that we reduce the matrix by removing row i and column j. Then we normalize the remaining rows, and repeat the process until all rows (and also all columns) have been selected once. We then have a complete permutation. Another permutation selection method, selects the rows using the entropy. The entropy is a measure of uncertainty associated with a random variable [Ihara, 1993]. The rows with small entropy are selected first.

$$\underset{i}{argmin}\ H = -\sum_{j=1..n} P_{ij} log_2 \left(P_{ij}\right)$$

The reason for using the entropy would be to select the least uncertain row first. Consider the probability matrix from Table 3.2. The second row will be used first to select a position in the permutation because it has the lowest entropy value (0.31), then the first row will be used to select a position, followed by the third and the fourth row.

Updating P with reward r: The probability matrix P is updated with an LA update scheme for each row i, and the selected action is determined by j. After updating all rows, the matrix P remains doubly stochastic, which is not the case in the PermELearn algorithm [Helmbold and Warmuth, 2009]. The PermELearn algorithm requires an extra normalizing step to make the matrix doubly stochastic again. This requires extra calculation time, which is not required by the method here proposed.

The proof for remaining doubly stochastic after an update with a linear LA update scheme is the following. Assume matrix $P(t)$ is doubly stochastic at time step t, and that we use the L_{R-I} update scheme. We need to prove that the matrix $P(t+1)$ is still doubly stochastic at time step $t+1$. Therefore, both the rows and the columns need to remain stochastic (sum to 1). At each update, we apply the LA update scheme to all rows. As a consequence, the rows automatically remain stochastic (property of the linear LA update scheme).

Only one element per column and one element per row are updated positively (Equation 2.1), while all other elements are updated negatively (Equation 2.2). To prove that the columns also remain stochastic, the increase in a column l by the positive update of element p_{kl} must equal the decrease of all other negative updates.

Proof. According to Equation 2.1, the value on row k and column l increases with:

$$\Delta^{+} = \alpha_{reward}\beta(t)(1 - p_{kl}(t)) \tag{3.1}$$

All other elements in column l are decreased (according to Equation 2.2) with:

$$\forall_{i,i\neq k}\Delta_{i}^{-} = \alpha_{reward}\beta(t)p_{il}(t) \tag{3.2}$$

In order to prove that the columns remain stochastic, the increase must be equal to the total summed decrease.

$$\Delta_{+} = \sum_{i,i\neq k}\Delta_{i}^{-} \tag{3.3}$$

$$\alpha_{reward}\beta(t)(1 - p_{kl}(t)) = \sum_{i,i\neq k}\alpha_{reward}\beta(t)p_{il}(t) \tag{3.4}$$

$$1 - p_{kl}(t) = \sum_{i,i\neq k}p_{il}(t) \tag{3.5}$$

$$1 = \sum_{i,i\neq k}p_{il}(t) + p_{kl}(t) \tag{3.6}$$

$$1 = \sum_{i}p_{il}(t) \tag{3.7}$$

Which is true given our assumptions.

□

3.3.4 Decentralized action selection approach

The following method is similar to the 'probability matrix method', but uses a more decentralized approach. Each position in the permutation is determined by an individual agent. An agent employs a learning automaton for choosing its individual position from the full set of positions. Thus, there are n agents (LA) representing the element of the permutation, with n actions each corresponding to the elements' position in the permutation. This leads to the same memory footprint as the 'probability matrix approach'. The agents play a dispersion game [Grenager et al., 2002], also called an anti-coordination game, for constructing a permutation. In a full dispersion game the number of agents is equal to the number of actions. In order to form a permutation, all agents need to select a distinct action so that the assignment of actions to agents is maximally dispersed. For example, if three agents select the following distinct actions: agent 1 selects position 2, agent 2 selects position 3 and agent 3 selects position 1, then the permutation becomes $[3, 1, 2]$.

A Basic Simple Strategy (BSS) was introduced by Grenager et al. [2002], allowing agents to select maximally dispersed actions in a logarithmic (in function of the number of agents) number of rounds, where a naive approach would be exponential. BSS does not incorporate the agents' preferences, it uses uniform selection. To take the agents' preferences into account, we introduce a probabilistic version of BSS, which we call Probabilistic Basic Simple Strategy (PBSS). The PBSS works as follows. Given an outcome $o \in O$ (selected actions for all agents), and the set of all actions A, an agent using the PBSS will:

- select action a with probability 1 in the next round, if the number of agents selecting action a in outcome o is 1 ($n_a^o = 1$).
- select an action from the **probabilistic** distribution over actions $a' \in A$ for which $n_{a'}^o \neq 1$, otherwise.

The probabilistic distribution over actions is obtained from the agents' LA probability vector. Once a permutation is constructed by playing the game, a common reward (permutation function) or individual reward can be obtained. These common or individual rewards are then given to the agents' LA, which consequently update their probabilistic distribution. Experiments have shown that this decentralized action selection approach has very similar performance characteristics to those of the 'probability matrix approach' and converges to a near-optimal permutation.

To illustrate the decentralized action selection approach and the PBSS, consider the following example permutation learning problem with $n = 3$. There are

3 agents, one for each element. Each agent has one LA with 3 actions. The actions are the positions in the permutation.

At the start of the procedure the learning automata have the following uniform probability vectors:

```
Agent 1 : [ 0.33 ; 0.33 ; 0.33 ]
Agent 2 : [ 0.33 ; 0.33 ; 0.33 ]
Agent 3 : [ 0.33 ; 0.33 ; 0.33 ]
```

Now we can start the PBSS using these probability vectors. The agents select a position (=action) in the permutation.

```
Agent 1 : position 2
Agent 2 : position 2
Agent 3 : position 3
```

Agent 3 is the only agent who has selected position 3, therefore it will reselect position 3 in the next round. Agent 1 and agent 2 have selected the same position, therefore they will reselect a position different from 3 in the next round using the following temporary probabilities.

```
Agent 1 : [ 0.5 ; 0.5 ; 0.0 ]
Agent 2 : [ 0.5 ; 0.5 ; 0.0 ]
Agent 3 : [ 0.0 ; 0.0 ; 1.0 ]
```

Then, the agents select the following positions in the permutation.

```
Agent 1 : position 2
Agent 2 : position 1
Agent 3 : position 3
```

All the positions are unique, and the PBSS is finished, resulting in the permutation $\pi = [2, 1, 3]$. The permutation is evaluated with a permutation evaluation function $f(\pi) = 0.4$. Then, the probability vectors can be updated using the linear-reward inaction update scheme with this evaluation and a predefined learning rate $\alpha = 0.1$. This update leads to the following probability vectors:

```
Agent 1 : [ 0.32 ; 0.36 ; 0.32 ]
Agent 2 : [ 0.36 ; 0.32 ; 0.32 ]
Agent 3 : [ 0.32 ; 0.32 ; 0.36 ]
```

This procedure can be repeated by selecting a permutation using PBSS and these updated probabilities.

3.4 Experiments

As an illustration of the methods' behaviour, some experiments were performed on a fictitious permutation function and simple benchmark problems, the TSP and an assignment problem. Subsequently, an application to a more extensive multi-project scheduling problem is given, which shows the real advantage of the described methods. The following experiments report on the decentralized action selection approach unless mentioned otherwise. Similar properties were observed for the hierarchical and probability matrix approach. All results are averaged over 100 runs and the experiments were performed on an Intel Core i7 2600 3.4Ghz processor, using the Java version 6 programming language.

3.4.1 Peaked permutation function

Consider the following permutation function definition. If the decimal number of the permutation π is $dec(\pi)$ according to the factorial number system (Lehmer code) [Knuth, 1973], then the value of the permutation is defined as:

$$f(\pi) = \left(\frac{2dec(\pi)}{n!}\right)^{10} \text{ if } dec(\pi) \leq \frac{n!}{2} \tag{3.8}$$

$$= \left(\frac{2(n! - dec(\pi))}{n!}\right)^{10} \text{ if } dec(\pi) > \frac{n!}{2} \tag{3.9}$$

This function has a peak value in the middle of the permutation range. Figure 3.6 shows this permutation function for $n = 9$.

Figure 3.7 compares the average calculation time (in milliseconds) of the described approaches. For each size $n = 2..20$, 5000 iterations are performed on the peaked permutation function. The learning rate has no influence on the calculation time. All but the naive approach show good calculation time properties. Since the memory usage of the hierarchical approach is very high, we prefer the decentralized action selection approach.

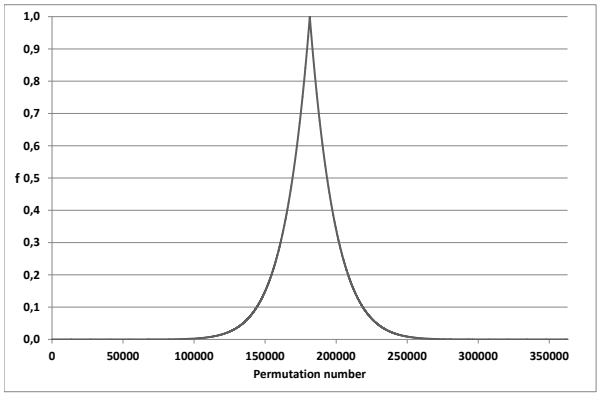


Figure 3.6: Peaked permutation function landscape $n = 9$.

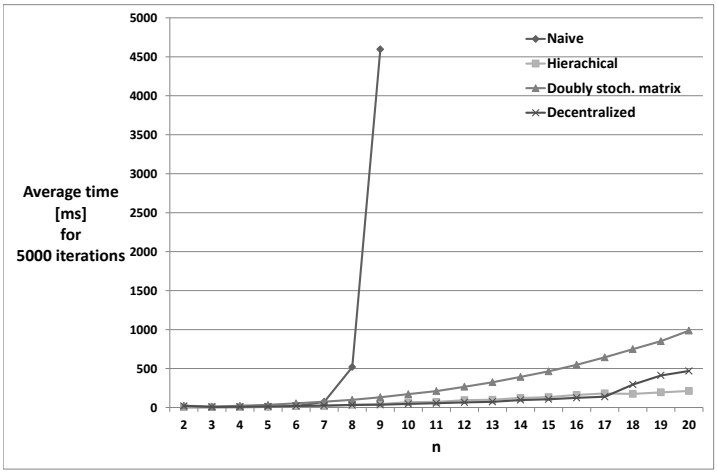


Figure 3.7: Average calculation time in milliseconds for performing 5000 iterations of the presented approaches for different permutation sizes on the peaked permutation function.

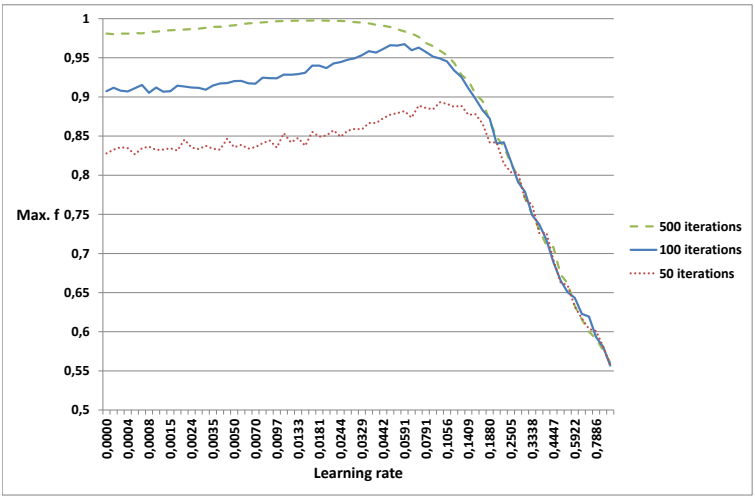


Figure 3.8: Max. objective function value for different number of iterations in the decentralized action selection approach on a peaked permutation function of size $n = 15$.

Figure 3.8 shows the maximum function value over a number of iterations (50,100,500) for different learning rates when applying the decentralized action selection approach with common reward (i.e. the permutation function value). The results show that for a particular range of learning rates, better permutations are learned compared to random sampling (i.e. learning rate equal to 0). When more iterations can be spent on learning, the difference between random sampling and learning becomes smaller. Bad performance can be observed when the learning rates are too high, and thus premature convergence is likely to occur.

Figure 3.9 shows the visited solutions with their corresponding position in the search space (permutation number) during a single run of 500 iterations on the peaked permutation function. A learning rate of 0.03 was used. At the beginning of the search, an extensive exploration of the search space is observed. After a while the search is focused towards high quality solutions (peak of the function). The duration of the exploration phase depends on the learning rate. In the experiments presented in Figure 3.9, most exploration disappears after approximately 150 iterations.

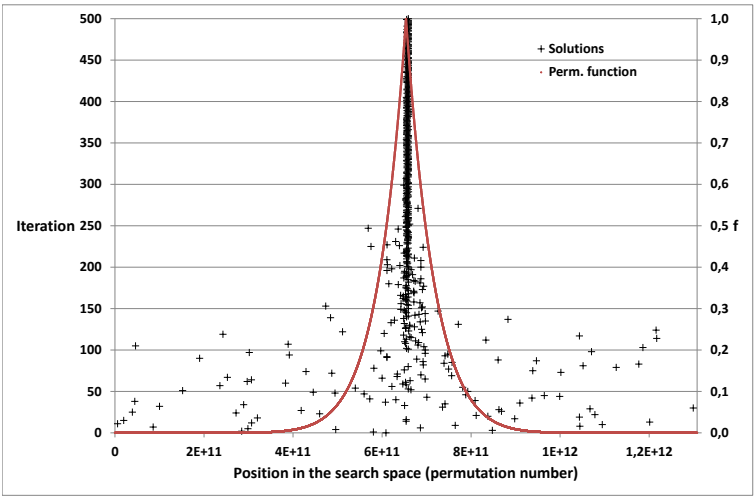


Figure 3.9: Visited solutions with their corresponding position in the search space. Run of 500 iterations on the peaked permutation function of size $n = 15$.

3.4.2 TSP

We tested the decentralized action selection approach with common reward on a number of TSP instances from TSPLIB[3]. The distance was scaled to a value between 0 and 1, such that a value of 1 corresponds to an optimal distance and 0 corresponds to an upper bound on the distance. Figure 3.10 shows the maximum function value over a number of iterations (1000, 5000, 10000, 50000) for different learning rates on a size $n = 17$ instance with name 'gr17'. For a particular range of learning rates better solution values can be observed, compared to random sampling. If more iterations are given, then the best solutions occur for lower learning rates. Again, too high learning rates lead to worse solutions.

[3]TSPLIB website: http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/ ; last check of address September: 23, 2011

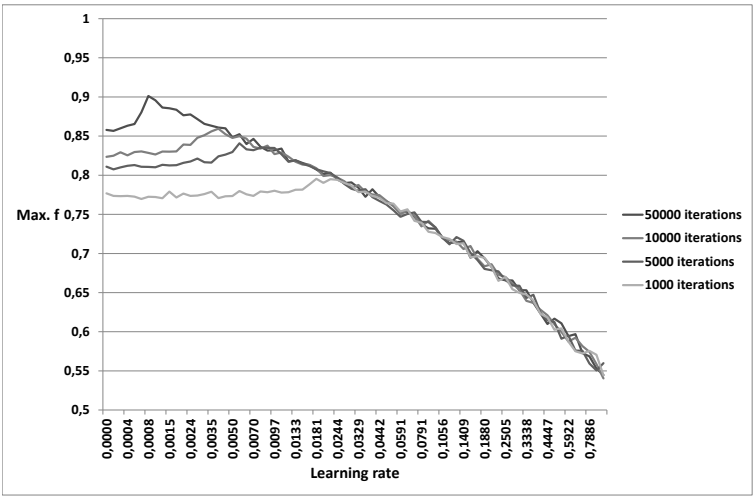


Figure 3.10: Max. objective function value for different number of iterations on a TSP instance (gr17) of size $n = 17$.

3.4.3 Assignment problem

Assignment problems belong to the category of permutation functions where the total cost of the permutation is equal to the sum of the individual costs. Therefore both individual and global rewards can be given to the agents. Figure 3.11 shows a comparison of the maximum objective function value for individual and common rewards on a random assignment problem of size $n = 9$ with the cost matrix of Table 3.3. A run of 1000 iterations is performed for each learning rate and the maximum objective function value is measured. The objective function is scaled between 0 and 1 such that 1 corresponds to the optimal solution. The results show that individual rewards produce better solutions than common rewards. For a particular range of learning rates the method performs better than random sampling, and the optimal solution can be found by using individual rewards .

	T0	T1	T2	T3	T4	T5	T6	T7	T8
A0	7	8	5	3	9	3	9	4	7
A1	3	6	9	3	2	9	6	5	7
A2	6	3	5	1	3	6	9	2	7
A3	8	1	9	3	3	6	3	6	3
A4	7	3	5	7	3	8	9	3	2
A5	4	2	8	2	7	5	4	6	4
A6	7	8	8	9	4	8	9	8	8
A7	7	4	7	8	9	8	1	3	5
A9	9	3	9	7	6	1	5	2	8

Table 3.3: Cost matrix for a random assignment problem of size $n = 9$

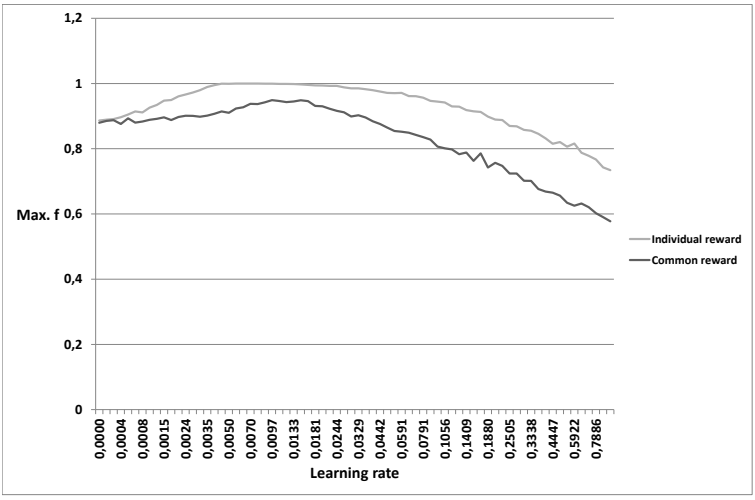


Figure 3.11: Comparison of max. objective function value for individual and common reward on a assignment problem of size $n = 9$.

3.4.4 Multi-project scheduling

The decentralized action selection method was used for solving the decentralized resource-constrained multi project scheduling problem (DRCMPSP) (See Chapter 5). The DRCMPSP was introduced by Confessore et al. [2002, 2007] and extended by Homberger [2007]. It is a generalization of the Resource Constrained Project Scheduling Problem (RCPSP) [Blazewicz et al., 1983b, Kolisch, 1996a] and can be stated as follows. A set of n projects have to be scheduled simultaneously using autonomous and self-interested decision makers. Each individual project contains a set of jobs or activities, precedence relations between the jobs, and resource requirements for executing each job. These resource requirements constitute local renewable resources, and global renewable resources shared among all projects. A global objective value must be minimized, examples include but are not limited to: the average project delay (APD), the total makespan (TMS), and the deviation of the project delay (DPD). The project delay of a project is its makespan minus the critical path duration. The remainder of the current section will concentrate on the APD objective. A constructive schedule generation scheme was applied to the DRCMPSP, requiring the project order and the job order for each project as input. The project order is a permutation of projects, while the job order is a permutation with precedence constraints. The decentralized method with the dispersion game was used to find good quality project orders, leading to schedules with a low APD objective value. Each project was represented by one project order decision maker. Instances from the multi project scheduling problem library[4] have been experimented on. To date, the method is part of the state-of-the-art for this problem, showing 104 best solutions out of 140, with respect to the average project delay objective.

3.5 Conclusion

Permutations are part of many optimization problems and solution methods. Learning these permutations online could make a heuristic search more adaptive and generally applicable. In this chapter, several centralized and decentralized methods using learning automata were studied for online learning good quality permutations. The decentralized approach uses a single learning automaton for each element in the permutation and a dispersion game to deliver the best runtime performance (calculation time and memory).

[4]Multi project scheduling problem library: http://www.mpsplib.com ; retrieved September: 23, 2011

Different permutation functions have been discussed. The capabilities of the methods have been analyzed and demonstrated on well known benchmark problems. The methods are very general because they do not use any problem specific information or domain knowledge, which makes them well suited for application within general optimization methods, like hyperheuristics. In fact the methods are applicable to all RL inclusion levels as defined in Figure 2.4. One could directly learn solutions for the problem (direct level), but also a sequence of neighborhoods or low-level heuristics (meta- and hyper-heuristic level). In the future, the very same methods can be applied to many other optimization problems and make them core elements of new intelligent optimization approaches. An application to the multi-project scheduling problem showed to be very successful. It will be discussed in Chapter 5.

Chapter 4

Learning Automata for Heuristic Selection

For many combinatorial optimization problems, multiple well performing heuristics or neighborhoods exist. Single solution based metaheuristics methods or hyperheuristics apply these heuristics/neighborhoods in an alternating manner. It is of highest importance to select and switch between these in the best possible manner in order to reach the best possible solutions. One could select heuristics in a fixed order, or use a random order, but often a better selection strategy exists. Such a strategy depends on the type of problem or even on the problem instance. Therefore it is interesting to select heuristics in a more adaptive way (compared to a static selection strategy). This is the subject of the present chapter. More specifically we will use learning automata for heuristic or neighborhood selection. Section 4.1 describes the proposed technique of using learning automata for heuristic selection. Section 4.2 demonstrates the properties of these learning heuristic selection methods. Conclusions are formulated in Section 4.3. Our main contributions on this topic were published as [Misir et al., 2009].

4.1 Heuristic selection with learning automata

Learning automata are very appropriate for learning which heuristics to select and apply at each time step. The most straightforward way is to use a single learning automaton whose action set is the set of heuristics (as shown in Figure 4.1). The LA's internal action probability vector determines the selection of a

heuristic at each time step. Another possibility is to use the methods described in Chapter 3 to learn a sequence of heuristics determining the order in which to apply heuristics.

A very important design question here is the reward signal. When do we update? Which reward value is given? The most *basic reward signal* is the following. Update with

$$\begin{aligned} &- \textit{a reward value of } 1 && \textit{if a better solution was found.} \\ &- \textit{a reward value of } 0 && \textit{otherwise.} \end{aligned} \tag{4.1}$$

This basic reward signal is simple, however it does not use much information. One could incorporate the calculation time of the heuristics or the achieved gain/loss in solution quality. Another possibility is to compare the solution quality improvement achieved by the heuristic with the average solution quality improvement over all heuristics. The latter will favor heuristics that generate large solution quality improvements, without taking into account the needed calculation time. It may be better to use a combination of these to have the best update signal. In this chapter we will always normalize the reward values between 0 and 1. One could also use a learning method which supports absolute values, such as the pursuit algorithm. In any case, no problem specific knowledge is used, which makes the method generally applicable.

Another important question is: what is the overhead of learning? This is influenced by the number of heuristics. In all conducted experiments (described in the next Section) the overhead of learning was negligible (much less than 1% of the total calculation time). A linear behaviour in the number of heuristics was observed, which is normal because an LA with linear-reward inaction update scheme was used.

4.2 Case studies

The LA based heuristic selection methods were tested on several challenging combinatorial optimization problems such as the traveling tournament problem, the patient admission scheduling problem and the machine reassignment problem (subject of the GOOGLE/ROADEF 2012 challenge). In the following sections we will describe how the LA based selection was applied to these different cases.

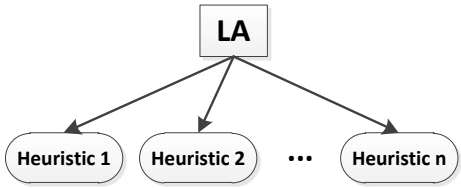


Figure 4.1: Selection of heuristics using a single learning automaton.

4.2.1 Case: the traveling tournament problem

We have proposed an LA based selection method in [Misir et al., 2009] for solving the Traveling Tournament Problem (TTP). The selection method was integrated in a classical hyperheuristic with two sub-mechanisms, i.e. the selection of low-level heuristics and the acceptance of new candidate solutions.

Problem description

The Traveling Tournament Problem (TTP) is a very hard timetabling problem. The goal is to find a feasible home/away schedule for a double-round robin tournament [Easton et al., 2001]. No optimal solutions have been found for realistic problem sizes, i.e. more than 10 teams. The optimization version of this problem considers the shortest total traveling distance (sum over all the teams). In small or middle-sized countries such as in Europe, the teams typically return home after playing another team in another place. But, in large countries, places are too far away from each other, so that returning home between matches might not be an option.

Generally, solutions for the TTP should satisfy the following two constraints : (c_1) *a team cannot play more than three consecutive games at home or away* and (c_2) *for the games between team t_i and t_j, t_i–t_j cannot be followed by t_j–t_i.*

The objective function that was used to measure the quality of the TTP solutions is defined as $f = d \times (1 + \sum_{i=1}^{2} w_i vc_i)$. With: d the total traveling distance, vc_1 the number of violations of the first constraint, w_1 the weight of c_1, vc_2 the number of violations of the second constraint, w_2 the weight of c_2. The weights w_1 and w_2 were set to 10.

Low-level heuristics

To handle the constraints and optimize the solutions for the TTP, five low-level heuristics were used in the heuristic set of the hyperheuristic. Short descriptions are:

SwapHomes	:	Swap home/away games between teams t_i and t_j
SwapTeams	:	Swap the schedule of teams t_i and t_j
SwapRounds	:	Swap two rounds by exchanging the assigned matches to r_i with r_j
SwapMatch	:	Swap matches of t_i and t_j for a given round r
SwapMatchRound	:	Swap matches of a team t in rounds r_i and r_j

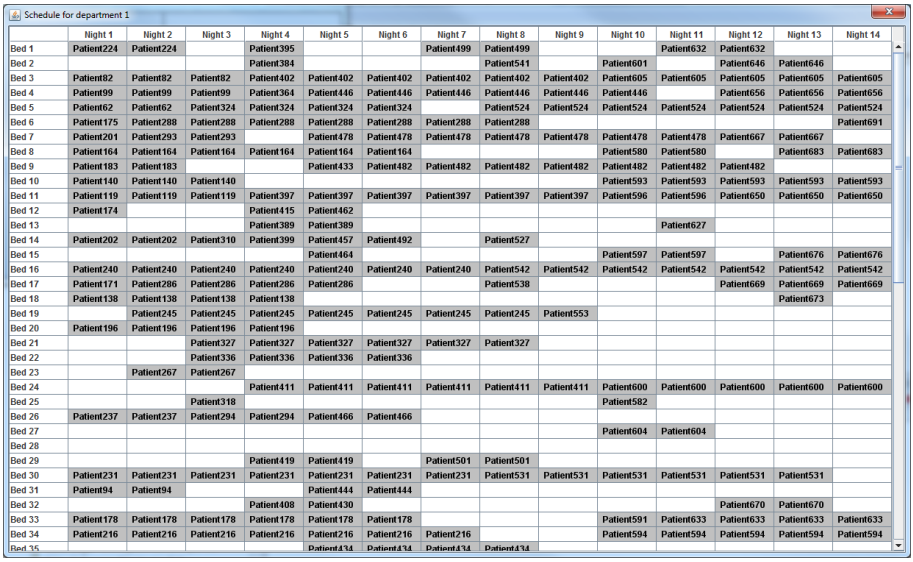
Results and experiments

The TTP instances[1] used in this study were derived from the US National League Baseball and the Super 14 Rugby League. The first group of instances (NL) incorporate $\forall_{n=2}^{8} 2n$ teams and the second set of instances (Super) includes 4 to 14 teams ($\forall_{n=2}^{7} 2n$). Several LA based heuristic selection experiments were performed on these instances, including the basic reward signal (Equation 4.1) and a probability restart policy experiment, in which the probabilities of the learning automata are reset during the search. The method showed to be competitive with existing approaches for the TTP. Some new best solutions were found for the Super12 and Super14 instances. These results were submitted and validated on the TTP benchmark website `http://mat.gsia.cmu.edu/TOURN/` (submitted on March 30, 2009).

4.2.2 Case: the patient admission scheduling problem

An LA based heuristic selection study was also performed on the patient admission scheduling problem (PAS). The PAS problem with all its real world constraints was introduced by Demeester et al. [2010]. The problem was further investigated and simplified by Ceschia and Schaerf [2011], new lower-bounds and an effective local-search procedure using simulated annealing are proposed. When extending the original work by Demeester et al. [2010], we observed improved performance by using a learning heuristic selection compared to a simple random selection mechanism on the existing heuristic set for the PAS problem [Wauters et al., 2009b].

[1] `http://mat.gsia.cmu.edu/TOURN/`



	Night 1	Night 2	Night 3	Night 4	Night 5	Night 6	Night 7	Night 8	Night 9	Night 10	Night 11	Night 12	Night 13	Night 14
Bed 1	Patient224	Patient224		Patient395			Patient499	Patient499			Patient632	Patient632		
Bed 2				Patient384				Patient541		Patient601		Patient646	Patient646	
Bed 3	Patient82	Patient82	Patient82	Patient402	Patient402	Patient402	Patient402	Patient402	Patient402	Patient605	Patient605	Patient605	Patient605	Patient605
Bed 4	Patient99	Patient99	Patient99	Patient364	Patient446	Patient446	Patient446	Patient446	Patient446	Patient446		Patient656	Patient656	Patient656
Bed 5	Patient62	Patient62	Patient324	Patient324	Patient324	Patient324		Patient524	Patient524	Patient524	Patient524	Patient524	Patient524	Patient524
Bed 6	Patient175	Patient288	Patient288	Patient288	Patient288	Patient288	Patient288	Patient288						Patient691
Bed 7	Patient201	Patient293	Patient293		Patient478	Patient478	Patient478	Patient478	Patient478	Patient478	Patient478	Patient667	Patient667	
Bed 8	Patient164	Patient164	Patient164	Patient164	Patient164	Patient164				Patient580	Patient580		Patient683	Patient683
Bed 9	Patient183	Patient183			Patient433	Patient482	Patient482	Patient482	Patient482	Patient482	Patient482	Patient482		
Bed 10	Patient140	Patient140	Patient140							Patient593	Patient593	Patient593	Patient593	Patient593
Bed 11	Patient119	Patient119	Patient119	Patient397	Patient397	Patient397	Patient397	Patient397	Patient397	Patient596	Patient596	Patient650	Patient650	Patient650
Bed 12	Patient174			Patient415	Patient462									
Bed 13				Patient389	Patient389						Patient627			
Bed 14	Patient202	Patient202	Patient310	Patient399	Patient457	Patient492		Patient527						
Bed 15					Patient464					Patient597	Patient597		Patient676	Patient676
Bed 16	Patient240	Patient240	Patient240	Patient240	Patient240	Patient240	Patient240	Patient542	Patient542	Patient542	Patient542	Patient542	Patient542	Patient542
Bed 17	Patient171	Patient286	Patient286	Patient286	Patient286			Patient538				Patient669	Patient669	Patient669
Bed 18	Patient138	Patient138	Patient138	Patient138									Patient673	
Bed 19		Patient245	Patient245	Patient245	Patient245	Patient245	Patient245	Patient245	Patient553					
Bed 20	Patient196	Patient196	Patient196	Patient196										
Bed 21			Patient327	Patient327	Patient327	Patient327	Patient327	Patient327						
Bed 22			Patient336	Patient336	Patient336	Patient336								
Bed 23		Patient267	Patient267											
Bed 24				Patient411	Patient411	Patient411	Patient411	Patient411	Patient411	Patient600	Patient600	Patient600	Patient600	Patient600
Bed 25			Patient318							Patient582				
Bed 26	Patient237	Patient237	Patient294	Patient294	Patient466	Patient466								
Bed 27										Patient604	Patient604			
Bed 28														
Bed 29				Patient419	Patient419		Patient501	Patient501						
Bed 30	Patient231	Patient231	Patient231	Patient231	Patient231	Patient231	Patient231	Patient531	Patient531	Patient531	Patient531	Patient531	Patient531	
Bed 31	Patient94	Patient94			Patient444	Patient444								
Bed 32				Patient408	Patient430							Patient670	Patient670	
Bed 33	Patient178	Patient178	Patient178	Patient178	Patient178	Patient178				Patient591	Patient633	Patient633	Patient633	Patient633
Bed 34	Patient216	Patient216	Patient216	Patient216	Patient216	Patient216	Patient216			Patient594	Patient594	Patient594	Patient594	Patient594
Bed 35					Patient434	Patient434	Patient434	Patient434						

Figure 4.2: Visualization of a schedule for a single department.

Problem description

The PAS problem, is a complex real world optimization problem where patients have to be assigned to beds in a hospital. Beds belong to a room in a certain department. Such a department can have one or more specialisms. Examples of departments are pediatrics, geriatrics, etc. The patients stay in the hospital for a number of nights and are transfered to other rooms as less as possible. The objective is to optimize several hard and soft constraints such as: availabilities of the rooms, the admission and discharge date of the patients, gender of the patients (only men, only women, mixed), age restrictions, personal room preferences (single or double room), medical equipment (oxygen machine), etc. For a more detailed problem description we refer the reader to the original PAS paper [Demeester et al., 2010].

Solution representation

A solution to this PAS problem was represented as a set of two-dimensional matrices, where each matrix represents a department. The rows in the matrices refer to the beds, while the columns denote the nights (Figure 4.2).

Low-level heuristics

The following set of existing low-level heuristics was used:

- Swap-beds (H1)
- Transfer patient to an empty bed in another department (H2)
- Transfer patient in the same department (H3)
- Transfer patient to another department (H4)

Results and experiments

Experiments were conducted on instances from the PAS dataset (`http://allserv.kahosl.be/~peter/pas/`). The dataset contains 14 instances (*testdata0.txt, testdata1.txt,...,testdata13.txt*) with varying properties (e.g. number of beds, number of departments, etc.).

Figure 4.3 shows the evolution of the heuristic selection probabilities in time when the heuristic selection mechanism with basic reward signal is used. The learning rate was set to a low $\alpha_{reward} = 0.005$ to avoid premature convergence. The heuristics above defined (H1,H2,H3,H4) have been used in the experiments. Heuristics H2 and H3 are given a higher probability by the learning method, while H1 and H4 are almost completely diminished. Heuristic H3 receives about 70% of the time during the second half of the search, which means it realized the most improvements.

4.2.3 Case: the machine reassignment problem

A metaheuristic approach with a late acceptance mechanism was developed for participation in the GOOGLE/ ROADEF 2012 challenge. The algorithm reached a first place in the qualification round of the challenge, and a fourth place in the final round of the junior category. Late acceptance, recently introduced by Burke and Bykov [2008], is a simple metaheuristic diversification mechanism (like simulated annealing). The mechanism accepts new solutions only if they are better or equal than L iterations ago. It showed to be very well performing on hard combinatorial optimization problems like exam timetabling [Burke and Bykov, 2008, Ozcan et al., 2009] and lock scheduling [Verstichel and Vanden Berghe, 2009]. Moreover, late acceptance is easy to tune because of its single parameter (L). Further experimentation on this problem showed

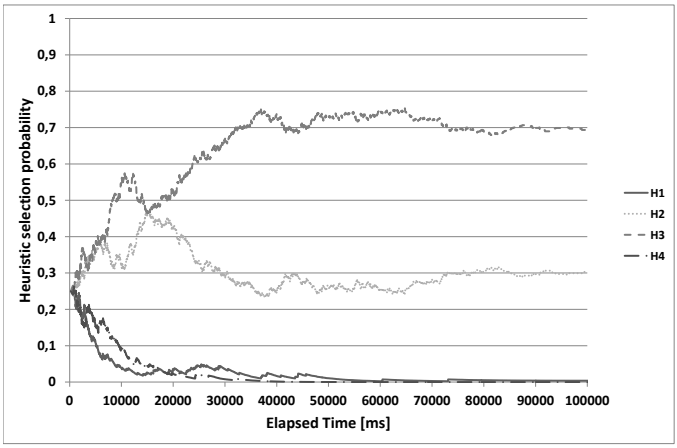


Figure 4.3: Probability-time evolution of the four heuristics with the basic learning selection method applied to PAS instance *testdata1.txt*.

that adding an LA based selection method can improve the previously obtained results.

Problem description

The machine reassignment problem, which was the subject of the GOOGLE/ROADEF 2012 challenge, considers the reassignment of processes to machines. The machines have multiple resources (e.g. RAM and CPU) each with their own capacity and safety capacity. Processes belong to a service, while the set of machines is partitioned into locations and neighborhoods[2]. The objective is to optimize the machine load and the resource balance, while keeping the cost of moving the processes to a minimum. The goal of the resource balance is to achieve a given target on the available ratio of two different resources, for example the ratio between CPU and RAM. Several hard constraints need to be respected:

- Capacity constraints: the total usage of a resource cannot exceed its capacity.

[2]Not a local search operator, but a geographical region like a city.

- Conflict constraints: all processes belonging to the same service must run on distinct machines.
- Spread constraints: the number of distinct locations where at least one process of service s should run is at least $spreadMin(s)$.
- Dependency constraints: if service s_a depends on service s_b, then each process of s_a should run in the neighborhood of an s_b process.
- Transient constraints: respect the capacity of the transient resources. Transient resources are consumed twice, once on the original machine, and once on the machine the process moved to.

For more detailed information on the problem we refer to the challenge subject paper [roa, 2012].

The resource usage of an initial solution and an optimized solution are visualized in Figure 4.4. A green colored block refers to a resource with a usage below the safety capacity, where as a red colored block refers to a resource usage higher than the safety capacity. As expected, the optimized solution contains less red blocks, i.e. less safety capacity violations.

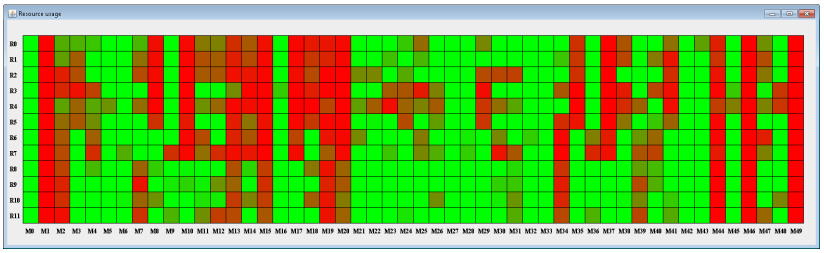
Neighborhoods

Several neighborhood functions were developed during the course of the challenge. The most important neighborhoods are:

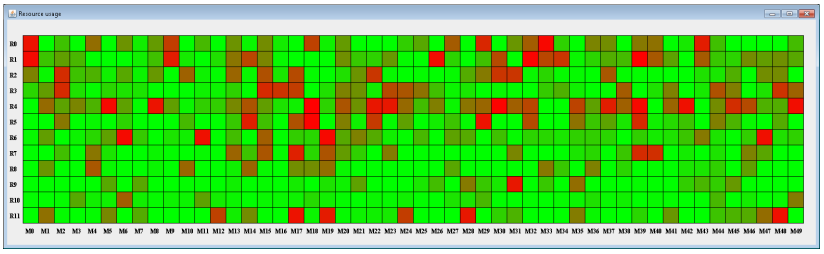
- Move a single randomly selected process to another randomly selected machine (*srm*).
- Swap the assignment of two randomly selected processes (*swap*).
- Swap the assignment of three randomly selected processes (*triswap*).

Results and experiments

Experiments with an LA based neighborhood selection method were conducted using these three neighborhood functions, and tested on the challenge problem instances. Two problem datasets were available, dataset A with small size instances (used for the qualification round of the challenge) and dataset B with large size instances (used for the final round). The large size dataset (dataset B) of the challenge will be used in further experiments. The dataset contains



(a) Before optimization



(b) After optimization

Figure 4.4: Comparison of the resource usage of two solutions for the machine reassignment problem on instance: a2_4 of the ROADEF 2012 challenge.

10 instances with varying characteristics. The number of processes varies from 5,000 to 50,000, and the number of machines ranges from 100 to 5,000. For the experiments an improving-or-equal acceptance method was used to minimize the effect of the acceptance method. Such an acceptance method, as the name already indicates, only accepts solutions when they have an objective function value equal to or better than the current solution's objective value. At each iteration we sample a single move out of the neighborhood, and apply it to the current solution.

Two learning selection methods were tested, both use an LRI learning automaton with a learning rate $\alpha_{reward} = 0.001$[3], but differ in the reward signal. The first uses the basic reward signal (Equation 4.1), i.e. update with a reward value of 1 if the solution improved (denoted as *BasicHS* in the figures). The second learning selection method updates with a reward value of 1 if the improvement was larger than the average improvement (denoted as *AAHS* in the figures). Figure 4.5 shows the comparison of average objective function values obtained by a search with these two learning neighborhood selection methods and a

[3]The learning rate value was not tuned for better performance, but set sufficiently low to avoid premature convergence.

random selection method on instance b_1 and b_2 of the machine reassignment problem. The BasicHS performs better than random selection on all instances, while the AAHS only performs better than random on instance b_2 and b_3 and even worse on instance b_4. On the complete B dataset, the basic selection method obtains an average improvement of 1.47% over random selection, while the average selection method realizes a smaller 0.35% average improvement. By changing the learning rate α_{reward} larger improvements can possibly be realized. A large difference between the behavior of both learning selection methods is observed. This behavior is shown in Figure 4.6 for the basic learning selection method and in Figure 4.7 for the above average learning selection method. It is clear that the method using the reward signal based on average improvements, prefers moves which realize a large improvement (*triswap* in this example), while the basic method almost immediately discards the *triswap* because it is not often finding improvements. Both methods eventually give a higher selection probability to *swap* moves. Summarized, both learning selection methods perform better than just randomly selecting a neighorhood.

4.3 Conclusion

We have proposed an LA based heuristic selection method using a single learning automaton. We show that using the performance of the heuristics at runtime to adjust the heuristic selection probabilities leads to a more efficient search. The most important design parameter is the reward signal, which incorporates the performance of the heuristic. The proposed selection methods are very suitable to use in the general hyperheuristic framework, because they do not use any problem specific information. However, the method can also be integrated into an existing local search where a neighborhood needs to be selected at each iteration.

Experiments were conducted on several test cases such as the traveling tournament problem, the patient admission scheduling problem, and the machine reassignment problem. Significant improvements were realized by the learning heuristic selection, when comparing it to a simple random heuristic selection mechanism. New best solutions were found on the traveling tournament problem with a hyperheuristic method, which included the LA based heuristic selection proposed in this chapter.

Further experiments can be performed with other RL methods such as the pursuit algorithm, which is able to handle absolute values. This can be interesting because most objective function values are not between 0 and 1. Early tests

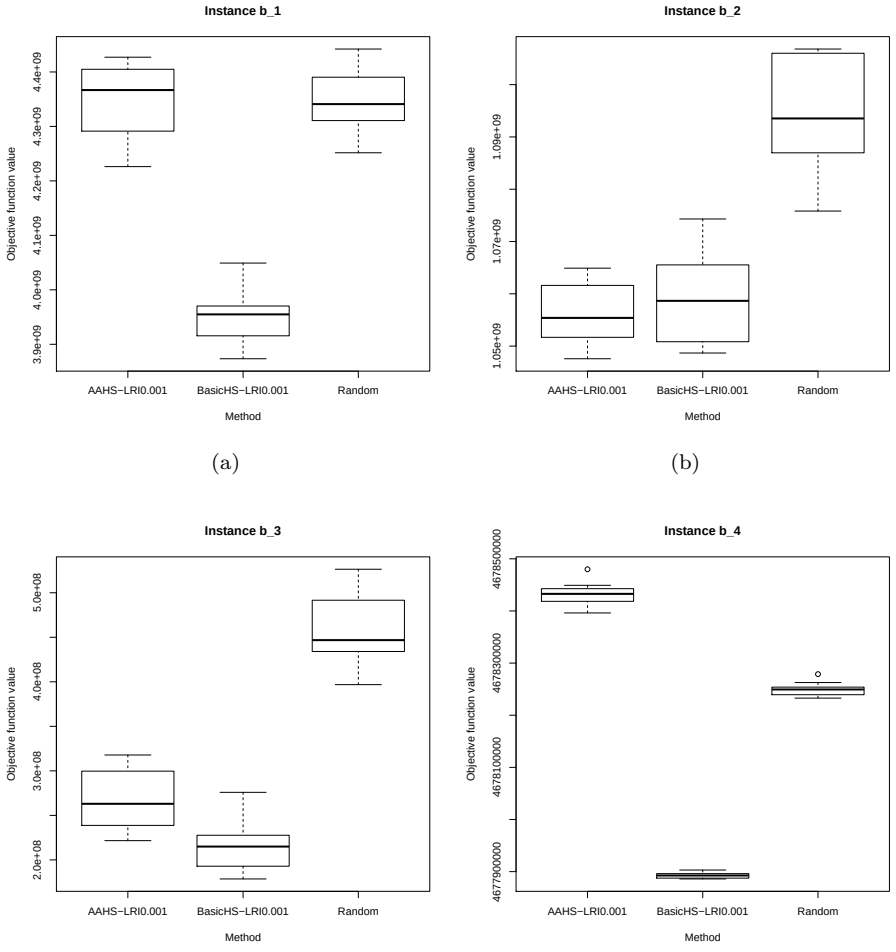


Figure 4.5: Comparison of two learning heuristic selection methods with a random heuristic selection on instances B_1 to B_4 of the machine reassignment problem.

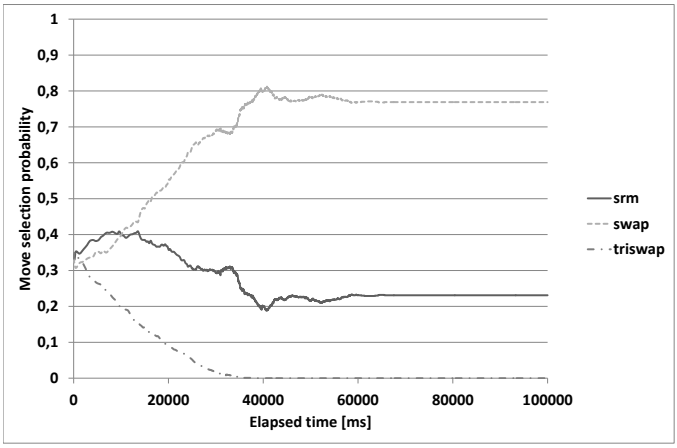


Figure 4.6: Probability-time evolution for the basic learning selection method using three neighborhoods on instance b_1 of the ROADEF 2012 challenge.

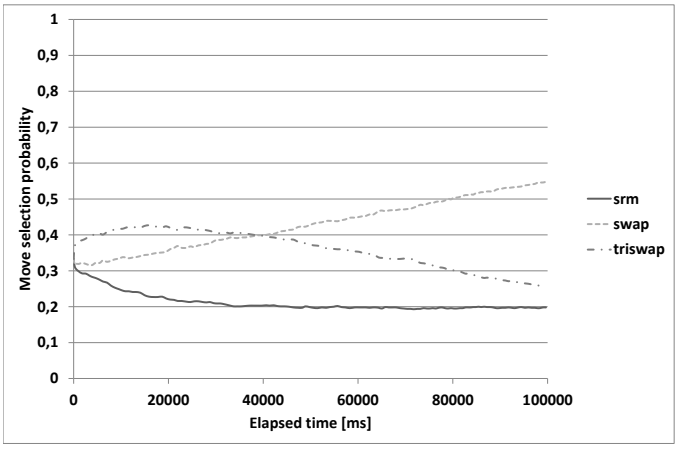


Figure 4.7: Probability-time evolution for the above average learning selection method using three neighborhoods on instance b_1 of the ROADEF 2012 challenge.

however show no significant improvements over the simple linear-reward-inaction mechanism used in this chapter.

Referring to Figure 2.4 the topics discussed in this chapter are situated at the meta- and hyper-heuristic level. Rather than intelligently selecting the heuristic to apply at each time step, the next chapter will discuss the usage of learning automata as a direct heuristic to two general project scheduling problems.

Chapter 5

Learning automata for project scheduling

Project scheduling is a research discipline with a long lasting history, mainly originated from economics and management. Standard project management techniques include CPM (Critical Path Method) and PERT (Programme Evaluation Review Technique). Most project scheduling problems are NP-hard and difficult to solve. State-of-the-art results are obtained by heuristics and metaheuristics. Applying reinforcement learning to project scheduling is rather new, but not without success. Two successful applications to project scheduling are reported in this chapter. Both problems are generalizations of the common Resource-Constrained Project Scheduling Problem (RCPSP), as shown in Figure 5.1. The first application (Section 5.1) is the multi-mode resource constrained project scheduling problem (MRCPSP), where each activity can be performed in more than one mode. A mode is a combination of resource usage and activity duration. A network of learning automata was employed for learning the activity lists and the modes. A common reward signal was used to update all LA. The second application (Section 5.2) is the decentralized resource-constrained multi-project scheduling problem, where multiple projects are scheduled together in a decentralized way by autonomous and self-interested agents. In addition to the individual resources for each project, some resources have to be shared between the projects. The methods for the MRCPSP were extended with a dispersion game to learn good project orders for this (D)RCMPSP using the techniques introduced in Chapter 3. According to Figure 2.4 the methods presented in this chapter are operating at the direct RL inclusion level, and are thus using RL directly to find good solutions.

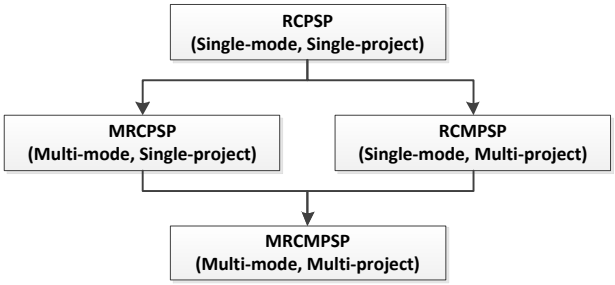


Figure 5.1: Resource-constrained project scheduling problem hierarchy.

The methods described in this chapter are slightly modified versions of the following two papers:
Wauters, T., Verbeeck, K., Vanden Berghe, G., De Causmaecker, P. (2011). Learning agents for the multi-mode project scheduling problem. Journal of the Operational Research Society, 62 (2), 281-290.

Wauters, T., Verbeeck, K., De Causmaecker, P., Vanden Berghe, G. (2012). Decentralized multi-project scheduling: a game theoretic learning approach. Technical Report KAHO-CW-09/07/2012 (Under review)

5.1 The multi-mode resource-constrained project scheduling problem

In [Wauters et al., 2009c,a, 2011] we have introduced a multi-agent reinforcement learning and local search for building schedules in the multi-mode resource-constrained project scheduling problem (MRCPSP). The idea is to learn a good constructive heuristic that can be further refined by local search techniques for large scale scheduling problems. With the latter goal in mind, a network of distributed reinforcement learning agents was set up, in which agents cooperate to jointly learn a well performing heuristic. As we will show later, our learning method generates results comparable to those obtained by the best-performing finetuned algorithms found in the literature, confirming the mentioned positive presumptions of using machine learning in search.

In the last few decades, the resource constrained project scheduling problem

(RCPSP) has become an attractive subject in operational research. It considers scheduling a project's activities while respecting the resource requirements and the precedence relations between the activities. The academic problem has many relevant real world counterparts in the building and consultancy sector, for example, resulting in a challenging list of instances with varying optimization objectives. The MRCPSP is a generalized version of the RCPSP, where each activity can be performed in one out of a set of modes, with a specific activity duration and resource requirements (e.g. 2 people each with a shovel need 6 days to dig a pit, while 4 people each with a shovel and one additional wheelbarrow need only 2 days). The RCPSP was shown to be NP-hard [Blazewicz et al., 1983a], thus so is the MRCPSP because it is a generalisation of the RCPSP. Demeulemeester and Herroelen [2002] published a comprehensive research handbook on project scheduling.
We introduce a novel multi-agent based approach [Wauters et al., 2009c,a, 2011], which is fundamentally different from the ones found in literature. It is based on a graph representation in which the nodes are agents representing activities. Each agent is responsible for one activity of the project. The agents make use of two simple reinforcement based learning devices i.e. learning automata (LA) to learn constructing good quality schedules. One of the agents' learning automata is responsible for selecting the order of its succesor activities. The second learning automaton in the agent is responsible for selecting an appropriate mode for its own activity. We use a smart coupling mechanism of rewards to learn both the order of the activities and the modes at the same time. Our approach is inspired by theoretical results that show how interconnected LA devices are capable of finding attractor points in Markov Decision Processes and Markov Games [Wheeler and Narendra, 1986, Vrancx et al., 2008].

The rest of this section gives a brief problem description, followed by related work, the model and multi-agent learning algorithm, computational experiments and comparative results on well studied benchmarks.

5.1.1 Problem formulation

The MRCPSP is a well known combinatorial optimization problem, of which several problem formulations have been proposed in the literature. Talbot [1982] gives a 0-1 formulation based on the traditional Pritsker's model of the RCPSP [Pritsker et al., 1969]. More recently, a new conceptual formulation was presented by [Van Peteghem and Vanhoucke, 2010]. We use a formulation similar to that of [Sprecher et al., 1997] and formulate it as follows. A project consists of J non-preemptive activities (jobs), including a dummy start and dummy end activity. Each activity $i \in \{1, \ldots, J\}$ can be performed in one out of a set of K_i modes. A mode represents a way of combining different

resources and the amount of resources requested. The activities of the project may require some renewable and non-renewable resources. A set R of renewable resources is available. Each renewable resource $r \in R$ has a constant availability K_r^ρ for each time period. Similarly a set N of non-renewable resources r is available, each with an availability K_r^ν for the entire project. Each mode $j \in \{1, \ldots, K_i\}$ of activity i corresponds to a specific activity duration d_{ij} and resource requirements for the resources. The requirement of activity i in mode j for resource $r \in \{1, \ldots, |R|, \ldots, |R| + |N|\}$ is k_{rij}^ρ for the renewable resources and k_{rij}^ν for the non-renewable resources. The dummy start ($i = 1$) and end ($i = J$) activities have zero duration and zero resource usage. The activities of the project have to be scheduled according to their strict finish-start precedence relations. Any activity i has a set P_i of strictly preceding activities, and also a set S_i of strictly succeeding activities.

A solution to the MRCPSP consists of start times s_i and finish times f_i for each activity i (a schedule). The objective is to find an activity order and mode combination that produces a schedule with a minimum makespan ($=f_J$). It should satisfy three hard constraints: 1) an activity should not start before all its predecessors have finished (precedence constraint), 2) the number of units used of a renewable resource should not be larger than the availability of that resource at any time, and 3) the number of units used of a non-renewable resource for the entire project should not exceed its availability. The second and third constraint are the resource constraints.

5.1.2 Related work

Brucker et al. [1999] present a unifying notation, a model, a classification scheme, i.e. a description of the resource environment, the activity characteristics and the objective function for the MRCPSP. The notation is similar to that for machine scheduling and allows for classifying the most important models. They furthermore introduce some exact and heuristic methods for both single and multi-mode problems. Herroelen et al. [1998] discuss the problem and its practical relevance. Kolisch and Hartmann [2006] provide an update of their survey of heuristics for the RCPSP that was first published in 2000. They summarize and categorize a large number of heuristics that have recently been proposed in the literature together with some detailed comparative results. Several exact methods have been developed for the MRCPSP in [Talbot, 1982, Patterson et al., 1990, Sprecher et al., 1997, Zhu et al., 2006]. None of these algorithms are useful for addressing large realistic problems, due to high calculation times. Different (meta-)heuristics have been proposed to address the problem, in order to enable producing good quality solutions in a short amount of time. Hartmann [1997], Alcaraz et al. [2003], Masao and Tseng

[1997], Lova et al. [2009], Van Peteghem and Vanhoucke [2010] all present a genetic algorithm for the MRCPSP. Other recent attempts for solving the MRCPSP include [Jarboui et al., 2008] where a combinatorial particle swarm optimization is proposed. Ranjbar et al. [2009] present an algorithm based on scatter search and path relinking. An artificial immune system has been reported in [Van Peteghem and Vanhoucke, 2009]. Jozefowska et al. [2001] and Bouleimen and Lecocq [2003] use a simulated annealing approach. A tabu search metaheuristic is applied in [Thomas and Salhi, 1998] for the RCPSP, and in [Herroelen and De Reyck, 1999] for solving the MRCPSP with generalized precedence relations.

Agent-based approaches have also been successfully applied to the MRCPSP. Knotts et al. [2000] use two types of agent (basic and enhanced agents), together with a set of different priority rules. The two agent types differ by the feasible execution mode that is selected for each resource. A basic agent, which is purely reactive, chooses the first feasible execution mode that it finds. In contrast, an enhanced agent deliberates the mode selection according to several rules. No learning is involved. Jedrzejowicz and Ratajczak-Ropel [2007] used a number of agents to work on a population of solutions in parallel on different computers. Each request represents a different optimization algorithm including local search, tabu search, as well as several specialized heuristics. Jedrzejowicz and Ratajczak-Ropel [2006] present a population learning algorithm for solving both the single and the multi-mode problems.

The contributions of this work include the application of multi-agent reinforcement learning for the project scheduling problem and the introduction of new benchmark results obtained with this method.

5.1.3 Multi-agent learning for the MRCPSP

The multi-agent learning approach is inspired by a commonly used activity network representation, namely the activity on node diagram (AON) where each activity is represented by a node in a graph, and where the edges represent the precedence relations between the activities. Figure 5.2 shows an example of a project with 7 activities according to the problem description in Section 5.1.1 (1 and 7 are dummy activities) and their relations. Activity 5 can only be executed when Activity 2 and 3 are finished.

Our goal is to generate an activity list (AL) and a mapping from activities to modes, i.e. a mode assignment (MA), which can later be used to construct a schedule. In this section, we first describe the multi-agent algorithm and its control flow. Next we discuss learning automata and how they are used in the

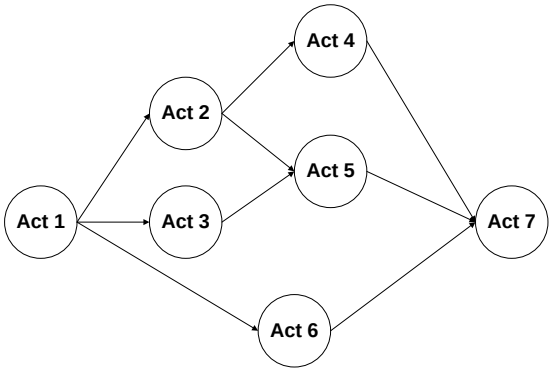


Figure 5.2: An example of an activity-on-node diagram for a project with 7 activities.

multi-agent setup, followed by more detailed information about the dispatcher agent, which is an important agent in the method.

Multi-agent setup

The activity list is a permutation of all the activities. It determines in which order the schedule construction algorithm handles the activities. The mode assignment determines in which mode each activity will be executed. The algorithm works by forwarding control from one agent to another. When an agent receives the control, the agent has to make some decisions. We start by placing an agent in every activity node. As in the AON diagram, these agents are interconnected by precedence relations on their activity nodes. Further on, we add an extra dispatching agent (dispatcher), which is employed by the algorithm for constructing schedules that respect the two hard constraints. In contrast to the other agents, the dispatcher does not represent an activity. It only selects another agent for taking over the control. This initial situation is presented in Figure 5.3.
The main idea of the algorithm is to enable every agent to learn which decisions to make, concerning:

1. the order in which to visit its successors, and
2. the mode in which the activity needs to be performed.

The algorithm works as follows: we start in the situation depicted in Figure 5.3. That is, we start from an empty activity list and an empty mode assignment.

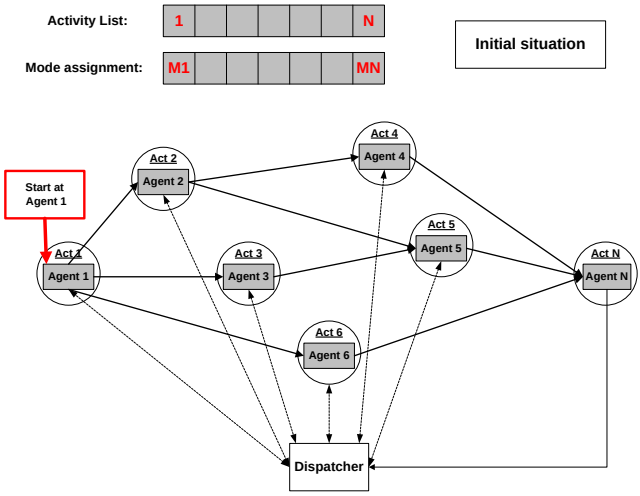


Figure 5.3: Initial situation: one agent in every activity node + one extra dispatching agent.

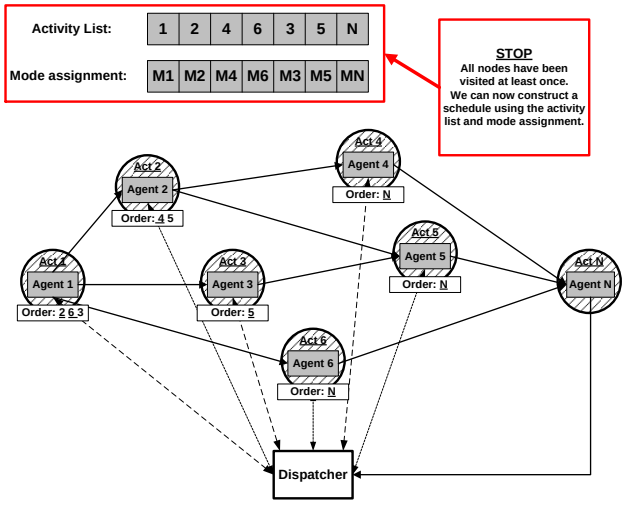


Figure 5.4: Final situation, all agents have been visited at least once.

The goal is to complete these lists. First the control is given to Agent 1. This agent chooses the order to visit its successors and picks the first agent from this order ($Agent_{next}$). Its activity is already in the activity list so it does not need to choose a mode. We set $Agent_{current} = Agent_{next}$. Now the control is given to $Agent_{current}$, which also decides in which order to visit its successors and takes the first agent from this order, which is the new $Agent_{next}$ (e.g. $Agent2$ chooses order $|4, 5|$, so it will first take $Agent4$ and then $Agent5$). $Agent_{current}$ has not been visited before. Consequently the activity it represents is added to the activity list, and the agent also chooses a mode that is added to the mode assignment. This process is continued until the agent in the last dummy node is visited. This node is special in the sense that its agent does not need to choose an activity order or a mode, but always forwards the control to the dispatcher. The dispatcher has a certain probability ($Pr_{DispToVisited}$) to choose a random eligible agent from the list of visited agents. Otherwise it chooses a random eligible unvisited agent. An agent is eligible when all the predecessors of the activity it represents have been visited. Note that this simple random dispatcher strategy can be replaced by a more intelligent one (e.g. a heuristic strategy, a learning dispatcher, ...), as is explained further. These steps are carried out subsequently until all the agents have been visited at least once. A possible final situation is shown in Figure 5.4. All the agents are visited and have chosen a successor order and a mode. A complete activity list and mode assignment have been generated.

The behavior of the agents is stochastic. At any time they can, with a small probability Pr_{ToDisp}, give the control to the dispatcher. This is a natural way to make all possible activity-order permutations reachable and hence all the possible schedules.

Now we can construct a schedule using the activity list and mode assignment with a schedule generation scheme (SGS) that uses a standard heuristic method for the RCPSP (see [Kolisch, 1996b] for details). The parallel SGS does not always lead to an optimal solution. The serial SGS does not have this defect. The extended schedule generation scheme from [Van Peteghem and Vanhoucke, 2010] makes use of the well-known forward/backward technique (introduced by [Li and Willis, 1992]) together with a new procedure to improve the mode selection. We will use the latter. When scheduling an activity $i \in \{2, \ldots, J-1\}$ there is a probability P_{modimp} for applying a mode improvement procedure. This procedure will consider other modes than the currently selected mode. If the usage of these other modes does not increase the violation of the non-renewable resource constraints, then the finishing time f_i is computed. We select the mode with the lowest f_i and set it as the current mode. We do this in the forward as well as in the backward step of the SGS. Van Peteghem and Vanhoucke [2010] show that this procedure leads to a significant improvement

in the solution quality.

In Algorithm 1, the global control of the algorithm for constructing an activity list and mode assignment is presented in pseudo-code. It gives control to individual agents that use Algorithm 2. The latter returns the next agent ($Agent_{next}$) to hand over control. For clarity we left out the description of the method that determines if an agent is eligible. This is done by checking whether all the agent's predecessors have been visited.

Algorithm 1 Global Control

Require: Project data and Algorithm parameters
Ensure: A feasible schedule w.r.t. the precedence constraints and renewable resource constraints.
initialize *ActivityList* and $ModeAssignment$
$Agent_{current} \Leftarrow Agent_1$
while Not all agents visited **do**
give the control to $Agent_{current}$
$Agent_{next}$ determined by $Agent_{current}$ using Algorithm 2
if $Agent_{next}$ is eligible **then**
$Agent_{current} \Leftarrow Agent_{next}$
else
$Agent_{current} \Leftarrow Dispatcher$
end if
end while
Schedule $\Leftarrow$ construct a schedule using the complete *ActivityList* and $ModeAssignment$
return *Schedule*

The method for constructing a schedule can now be used in an iterative way. We use the optimized schedule's quality (makespan) and mode assignments for the agents to learn which actions to take. We apply some simple learning automata which we will describe next. Note that the constructed schedules can be infeasible in terms of non-renewable resource usage, but in our experiments we noticed that at the end of a run the produced schedules are completely feasible. This is because a penalty proportional to the number of violations is given to infeasible schedules, and thus rewarding better schedules with lower violations and a better makespan, as we will describe later.

Learning a constructive heuristic

For learning the activity order and the best modes we apply the (L_{R-I}) learning automata update scheme because of its ϵ-optimality property in all stationary

Algorithm 2 Single Agent Control

Require: $ActivityList$, $ModeAssignment$
Ensure: $Agent_{next}$
  rand ⇐ random number between 0 and 1
  if $(rand < Pr_{ToDisp})$ or (this is $Agent_N$) **then**
    $Agent_{next} \Leftarrow Dispatcher$
  else
    if $Agent_{current}$ not yet visited **then**
      add $Agent_{current}$ to the $ActivityOrderList$
      $Mode \Leftarrow$ chooseMode() using Mode LA
      add the $Mode$ to the $ModeAssignment$
      $Order \Leftarrow$ chooseOrder() using Order LA
      $Agent_{next} \Leftarrow$ first Agent in $Order$
    else
      $Agent_{next} \Leftarrow$ next Agent in $Order$
    end if
  end if
  return $Agent_{next}$

environments. The learning rate (reward parameter) used for learning the activity order and the one used for learning the mode are named LRO and LRM respectively. The application of the reinforcement will be presented in what follows.

Once a schedule has been constructed, we update all the learning automata using the following reinforcements. If the makespan of the constructed schedule at instant k is:

- Better: $r(k) = 1$
- Equal: $r(k) = r_{eq}$ $(r_{eq} \in [0, 1])$
- Worse: $r(k) = 0$

Both r_{eq} and the learning rates LRO and LRM determine the learning speed. A higher r_{eq} can speed up the learning, especially for a problem like the MRCPSP where attempts to generate new schedules only rarely result in quality improvements. The settings of the two learning rates are dependent. A proper combination will be important for a good overall performance.

If the constructed schedule turns out to be infeasible in terms of the non-renewable resources, we add a high penalty value to the makespan of the schedule. This forces the method to construct feasible schedules. Although

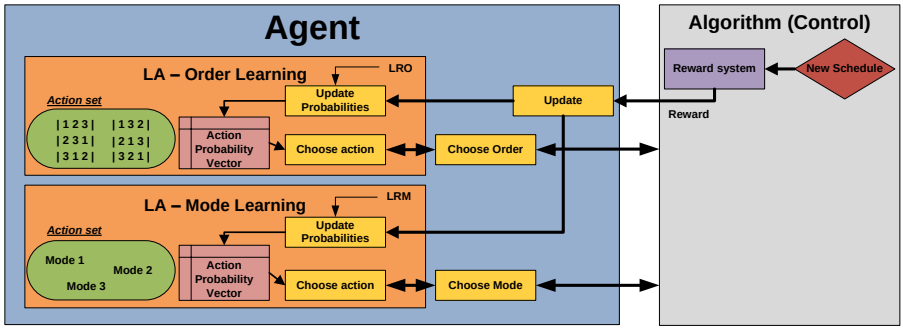


Figure 5.5: The single agent view.

we have no formal arguments to justify this theoretically, in practice not one infeasible schedule was found at the end of a run, when using such a penalty.

Note also that the mode LA is reinforced with the mode that was actually used in the constructed schedule. Since the mode optimization procedure in the SGS can change the mode that was originally selected by the mode LA.

The viewpoint of a single agent is presented in Figure 5.5. Each agent has two learning devices. When an agent is visited for the first time, the algorithm will ask the agent to choose an order for visiting its successors and a mode. For both choices the agent consults the corresponding learning automaton. These learning automata make a choice according to their probability vector (probability distribution). After all the agents have been visited at least once, the algorithm constructs a schedule. Using the information from this schedule, the reward system will update all the agents according to the basic 0-1 reward signal. The agents forward the reinforcement signal to their learning automata devices. These learning automata will then update their probability vector using the L_{R-I} method.

The dispatcher agent

The dispatcher agent assures that the algorithm will end in finite time by forwarding the control to visited and other eligible agents. The simplest strategy to apply for this dispatcher is just a random selection of agents. We added to this that the dispatcher has a certain probability ($Pr_{DispToVisited}$) of choosing a random eligible agent from the list of already visited agents, otherwise it

chooses a random eligible unvisited agent. This probability can be fixed or dynamically evolve through the process.

The next step was to apply a learning method to the dispatcher. We decided to make a dispatcher based on Q-learning [Watkins, 1989a]. In contrast to a single automaton, Q-learning can handle environments of more than one state. Q-learning learns optimal policies by exploring and exploiting the state-action space. In our case, states are presented by the agent that was visited in the previous step. The actions are the next agent to forward the control to.

Preprocessing

Before we start the learning algorithm we apply a preprocessing step. This preprocessing step consists of applying the preprocessing procedure of [Sprecher et al., 1997]. Modes that are inefficient or non-executable are excluded. Further on, preprocessing excludes redundant non-renewable resources. Sprecher et al. [1997] define a mode as inefficient if its duration is not shorter and its resource needs are not less than the other modes of the same activity. A non-executable mode is a mode for which the renewable or non-renewable resource constraints are violated. A non-renewable resource is redundant if the sum of the maximum requests for that resource is not larger than the resource's availability.

5.1.4 Experimental results

In this section we evaluate the performance of the multi-agent learning algorithm. The algorithm has been implemented in Java Version 6 Update 16 and runs on a mobile Intel Core 2 Duo T9600 2.8GHz processor, 4GB RAM. To test the performance of the algorithm, we applied it to instances of the project scheduling problem library (PSPLIB) [Kolisch and Sprecher, 1996], which is available from the ftp server of the University of Kiel (http://129.187.106.231/psplib/).

First we present the experimental results for the multi-mode RCPSP followed by the results on the single-mode version.

Multi-mode

PSPLIB contains a number of MRCPSP datasets with a number of activities ranging from 10 to 30 (J10, J12, J14, J16, J18, J20 and J30). For all except the last dataset, the optimal solutions are known. All the instances in these datasets have two renewable and two non-renewable resources. Each dataset

contains 640 instances, of which some are infeasible. We exclude the infeasible instances from the experiments.

When testing the algorithm, we found that the required number of iterations strongly depends on the initial settings. For that reason we used the algorithm in the common multi-restart way. This involves restarting the algorithm a number of times per instance and taking the best solution over all the runs. In Table 5.1 and Table 5.2, we present the results of the multi-agent based algorithm for the J10 to J20 datasets from the PSPLIB library, using the following empirically obtained parameters for all the tests: 0.4 for the order learning rate (LRO), 0.4 for the mode learning rate (LRM), $r_{eq} = 0.01$, $Pr_{ToDisp} = 0.00(0\%)$, $Pr_{DispToVisited} = 0.50(50\%)$, $P_{modimp} = 0.40(40\%)$, Random Dispatcher (RD) and $5 \times 1,000$ *iterations*.

In Table 5.3 we present the results for the J30 dataset using $5 \times 1,000$ iterations and $5 \times 5,000$ iterations, both with Random (RD) and Q-Learning Dispatcher (QLD). For this dataset the optimal solutions are not known to the research community. We therefore calculated the average procentual deviation from the best known solutions.

When considering these results for the MRCPSP, we can conclude that the multi-agent approach performs very well when comparing it to the state of the art methods from the literature. We even reach the best result for one dataset (J16).

Single mode

Since the MRCPSP is a more general definition than the RCPSP, the multi-agent learning approach is also suitable for solving the latter problem. Table 5.4 presents the results for the J120 RCPSP dataset, which is the largest dataset for RCPSP in the PSPLIB library. The tests were carried out with the same parameters as in Section 5.1.4 but with $5 \times 5,000$ iterations and without mode learning. Since the optimal solutions are not all known for this dataset, we calculate the average procentual deviation from the critical path length. We also provide the average procentual deviation from the best known solutions.

Evaluation

When looking at the single-mode RCPSP results, which reveal average performance when comparing them to the best algorithms reported in the literature [Kolisch and Hartmann, 2006], we can conclude that the power of the approach is its coupling between learning the activity order and learning

the modes. Note that our approach does not require mutual communication between the learning automata. The coupling of the LA happens through the common global reward signal. For both the RCPSP and MRCPSP, specialized Genetic Algorithms (GA) are among the best performing algorithms in the literature. When we compare our results with one of the very best GAs for the MRCPSP [Van Peteghem and Vanhoucke, 2010], the results of the multi-agent learning approach have similar quality.

We investigated the contribution of each component in our system. An example of this is shown in Figure 5.6 where we show the effect of disabling each component in the system (no mode learning, no order learning and no mode optimization procedure in the SGS). We compared this to the situation were all components are active. These tests have been run using the same settings as before on several instances. The results in Figure 5.6 are for one instance from the J30 set, but tests on other instances show similar results. As shown, all components are important. Disabling one of them reduces the average solution quality. Disabling the mode learning has the most negative impact on the quality of the solutions.

The multi-agent learning model is based on the models of [Wheeler and Narendra, 1986, Vrancx et al., 2008], where it was shown that a network of LA is capable of converging to some optimal attractor points of the underlying multi-agent game. Although the activity-on-node model for the MRCPSP problem that we consider here does not satisfy the ergodicy property that was assumed in [Vrancx et al., 2008], in practice it still produces good results. The main reason why the sufficient conditions for convergence are not satisfied is that not all nodes in the network are reachable from any other node. Not all agents are reachable from the dispatcher agent for instance. Only the eligible agents can be selected. Further study could investigate the theoretical properties of the learning scheme as presented here.

Previous contributions to the MRCPSP using agent technology include [Knotts et al., 2000, Jedrzejowicz and Ratajczak-Ropel, 2006, 2007]. We briefly point out the main differences and similarities between the work in this paper and other agent based approaches in the literature. Table 5.1 shows that our approach clearly results in solutions of better quality than those obtained by similar approaches [Jedrzejowicz and Ratajczak-Ropel, 2006, 2007]. These other approaches applied a different stopping criterion, which was a predefined running time of 5 minutes [Jedrzejowicz and Ratajczak-Ropel, 2007] and 50000 generated solutions [Jedrzejowicz and Ratajczak-Ropel, 2006]. In [Jedrzejowicz and Ratajczak-Ropel, 2007] each agent represents a different algorithm (heuristic or meta-heuristic) without any learning, while a population based method is presented in [Jedrzejowicz and Ratajczak-Ropel, 2006]. It operates on complete solutions (individuals in their population), while our approach uses simple local

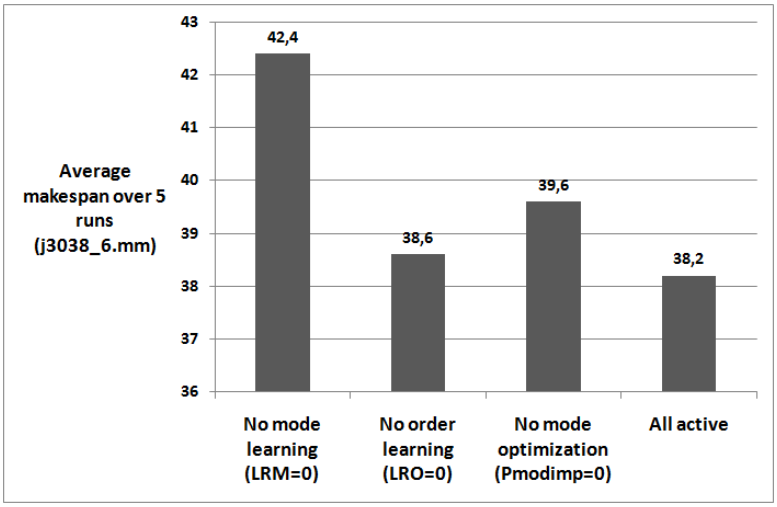


Figure 5.6: Contribution of system components.

learning to construct solutions. The main difference between our approach and [Knotts et al., 2000] is that we use learning to adapt the decisions made by the agents, while their approach uses some predefined rules.

Finally, in the future a lot of different dispatcher strategies could be tested as well. A dispatcher could be equipped with heuristic information or use more complex learning schemes than the one we use here.

5.1.5 Conclusion

A novel algorithm was developed that combines reinforcement learning and search for solving the multi-mode resource-constrained project scheduling problem (MRCPSP). Our approach integrates ideas from multi-agent systems, reinforcement learning (more in particular, learning automata) and search. We equipped our agents with the ability to learn activity orders and modes, based on past performance. A smart rewarding mechanism coupled both learning tasks intelligently, whereby we managed to obtain competitive results on most of the MRCPSP datasets from the PSPLIB Library. We even improve the results for one dataset. The big advantage of our method is that it uses simple local learning schemes, which makes our approach easy to apply.

We have shown that the combination of reinforcement learning and optimization is definitely useful and that this combination will certainly be further investigated

Table 5.1: Comparison with other approaches for the MRCPSP - Average deviation from optimal solutions (%)

	J10	J12	J14	J16	J18	J20
[Jedrzejowicz and Ratajczak-Ropel, 2007]	0.72	0.73	0.79	0.81	0.95	1.80
[Jedrzejowicz and Ratajczak-Ropel, 2006]	0.36	0.50	0.62	0.75	0.75	0.75
[Bouleimen and Lecocq, 2003]	0.21	0.19	0.92	1.43	1.85	2.10
5000 schedules:						
[Jozefowska et al., 2001]	1.16	1.73	2.60	4.07	5.52	6.74
[Alcaraz et al., 2003]	0.24	0.73	1.00	1.12	1.43	1.91
[Jarboui et al., 2008]	0.03	0.09	0.36	0.44	0.89	1.10
[Ranjbar et al., 2009]	0.18	0.65	0.89	0.95	1.21	1.64
[Van Peteghem and Vanhoucke, 2009]	0.02	**0.07**	**0.20**	0.39	0.52	0.70
[Lova et al., 2009]	0.06	0.17	0.32	0.44	0.63	0.87
[Van Peteghem and Vanhoucke, 2010]	**0.01**	0.09	0.22	0.32	**0.42**	**0.57**
Multi-Agent Learning Approach RD	0.05	0.08	0.23	**0.30**	0.53	0.70

Table 5.2: Experimental results for Multi-Agent Learning Approach RD - $5 \times 1,000$ iterations

	J10	J12	J14	J16	J18	J20
Average RE (%)	0.05	0.08	0.23	0.30	0.53	0.70
Std. Dev. RE (%)	0.49	0.61	0.99	1.09	1.40	1.64
Optimal (%)	98.70	98.17	94.74	92.18	86.23	81.59
Average runtime (s)	0.8	1.0	1.4	1.7	1.9	2.1

Table 5.3: Experimental results MRCPSP - J30

	Avg. deviation from Critical Path LB (%)	Avg. deviation from best known solutions (%)
$5 \times 1,000$ iter. RD	14.68	1.80
$5 \times 1,000$ iter. QLD	14.62	1.74
$5 \times 5,000$ iter. RD	14.0	1.26
$5 \times 5,000$ iter. QLD	13.91	1.20

Table 5.4: Experimental results RCPSP - J120

	Avg. deviation from critical path length (%)	Avg. deviation from best known solutions (%)	Avg. runtime (s)
$5 \times 5,000$ iterations	36.98	4.36	41

in the future. One of the next topics will be to test the scalability of our local learning approach.

In some cases multiple projects have to be scheduled simultaneously, taking into account the availability of shared resources. This extension to multi-project scheduling is discussed in the next section of this chapter.

5.2 The decentralized resource-constrained multi-project scheduling problem

Collaborative project management is becoming quite common in today's globally active industries. Indeed, enterprises collaborate simultaneously with different customers or partners in various projects requiring scarce and shared resources. It is a means of accelerating product development, reducing cost, and increasing quality. However, it requires careful scheduling of overlapping tasks with possible competing resource requirements. This is exactly the focus of the decentralized resource constrained multi-project scheduling problem (DRCMPSP), which is a generalization of the familiar resource-constrained project scheduling problem (RCPSP) [Brucker et al., 1999].

A set of n projects has to be planned simultaneously in the DRCMPSP. For each project the following information is given: an earliest release date, a set of activities, precedence relations between the activities and a set of local renewable resources. On top of these, a finite set of global renewable resources is available, which have to be shared by all projects. Projects are planned in a decentralized manner by autonomous and self-interested decision makers, which are typically the project managers. A project manager has the goal to minimize its local objectives. However, the local objectives of the managers are usually in conflict with each other. Activities of different projects may require the same shared resource at the same time. In order to enable comparing alternative solutions of a given DRCMPSP, some local and global performance criteria are defined. Commonly used criteria are the Total Makespan (TMS) and the Average Project Delay (APD). Both will be considered in this study.

Multi-agent systems have been applied before, for solving the DRCMPSP. Confessore et al. [2007] introduced a multi-agent system, and an iterative combinatorial auction mechanism for solving the DRCMPSP. Large multi-project instances are solved by integrating a metaheuristic called the centralized restart evolution strategy, with an efficient decentralized electronic negotiation mechanism [Homberger, 2007]. This approach was further improved by Homberger [2009]. Adhau et al. [2011] present an auction-based negotiation

approach, using a new heuristic for the winner determination problem in auctions. Some improved results for the APD objective are presented.

Rather than having all project managers negotiate for each activity to be scheduled, the present work focuses on coordination through learning a simple sequence game managed by a trusted third party or mediator agent. A serial schedule generation scheme, which is adopted from single project scheduling [Kolisch and Hartmann, 1998], first builds a solution to the DRCMPSP. Instead of giving the serial scheduler one activity list, as in single project scheduling, it is here presented with a sequence of activity lists, one for each project. The observation to be made here is that the order of the different activity lists in the sequence has a non-neglectable effect on the quality of the resulting schedule[1]. Each project manager simultaneously chooses a place in this sequence. Since all project managers should have a unique place, they in fact play a dispersion game [Grenager et al., 2002]. As such, the goal of each project manager boils down to, 1) building an efficient precedence feasible activity list locally and 2) learning a suitable place in the overall sequence of activity lists. In the proposed method, both goals are learned simultaneously and iteratively by using a global reinforcement signal, i.e. the APD or TMS of the schedule that was generated at the previous time step. The project managers locally use a network of simple reinforcement learning devices called learning automata for learning an efficient activity list. This technique was adopted from our learning technique for the single project version discussed in Section 5.1. The sequence game is played with a probabilistic version of the Basic Simple Strategy (BSS), which guarantees the players to coordinate within logarithmic time. This technique was adopted from previous work on fast permutation learning (Chapter 3).

Experiments show that the sequence learning game approach (GT-MAS) has a large positive effect on the minimization of the average project delay. In fact, the combination of local reinforcement learning, the sequence learning game and a smart forward-backward implementation of the serial scheduler significantly improves the best known results for all the MPSPLIB datasets. Many new best solutions for the APD objective were produced, dominating the MPSPLIB benchmark website with 104 best solutions out of 140. Besides the APD successes, also the TMS objective generated several new best solutions. Due to the usage of linear update rules and a mechanism with logarithmic properties for playing the sequence game, the learning approach scales very well.

In the course of this thesis, the GT-MAS method was adopted as a reference in a comparative study by Mao [2011]. Two objectives are used in this study,

[1]This may seem counter intuitive since all projects should benefit from being scheduled first. However, projects have different release times and some projects detoriate the solution more when being scheduled late than others.

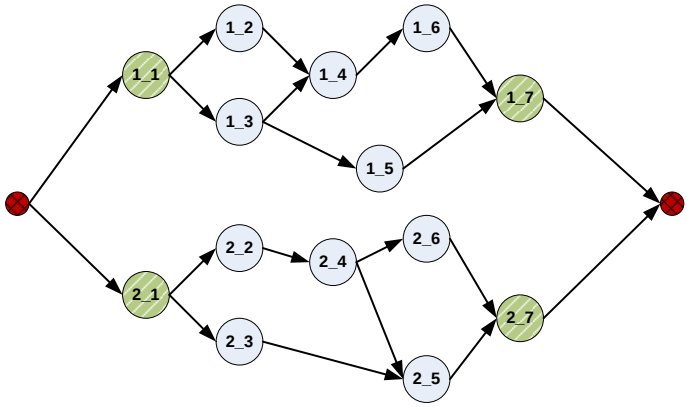


Figure 5.7: Multi-Project activity-on-node diagram for an example with 2 projects (7 activities each, dummy activities included)

i.e. the average project delay and the total squared resource utilization (TSRU) which is a resource leveling objective. GT-MAS outperforms the other methods concerning the APD objective.

This section is structured as follows. Section 5.2.1 describes the DRCMPSP. The solution representation and encoding together with the new GT-MAS approach are given in Section 5.2.2. Section 5.2.3 presents the experiments and results. Conclusions are drawn in Section 5.2.4.

5.2.1 Problem description

The RCPSP problem considers a single project in which activities need to be scheduled according to their precedence relations and resource requirements. The literature covers many models for generalizing the RCPSP to a multi-project version. The present work relies on the DRCMPSP originally introduced by Confessore et al. [2007], and on a problem notation similar to [Homberger, 2009].

The DRCMPSP is described by a set of N projects i, $i = 1..N$. Each project i consists of J_i non-preemptive activities with specific finish-start precedence relations. Each activity j of project i has a duration of d_{ij} time periods. As is common in the RCPSP literature, the first and last activities of the projects are dummy activities, which have a zero duration, no resource requirements. They determine the start and end of the project. Figure 5.7 presents an example of a multi-project activity-on-node diagram with two projects with seven activities each (dummy activities included), also showing the preference relations.

Each project i has an arrival (or release) date ad_i, which is the earliest point in time for the project to start. Project 1 always starts at $ad_1 = 0$, while the other projects start at $ad_i \geq 0$. A set L_i of local renewable resources is available for each project i. A constant maximum capacity of R_{il} units is associated with each local resource $l \in L_i$. On top of these, a set G (with $|G| \geq 1$) of global renewable resources is to be shared among all projects. Accordingly, a constant maximum capacity of R_g units is associated with each global resource $g \in G$. Each activity j of project i requires r_{ijl} units of local resource l and r_{ijg} units of global resource g.

A solution for the DRCMPSP must define a start time (or finish time) s_{ij} (f_{ij}) for each activity j of each project i, with $f_{ij} = s_{ij} + d_{ij}$. This solution is precedence feasible if it respects all precedence relations, and is resource feasible if at each time period t, the applied resources do not exceed the maximum capacity of the global and local resources. A solution that is both precedence and resource feasible is simply called feasible.

Once a solution is constructed, i.e. a multi-project schedule, its quality can be evaluated based on both local and global criteria. Each project i has a starting date s_i that is equal to the starting time s_{i1} of the dummy starting activity. Similarly, each project i has a finishing date f_i that is equal to the finishing time f_{iJ_i} of the dummy finishing activity. Commonly used local or private criteria are the makespan and the project delay. In contrast to the single project scheduling problem in the previous section, where the arrival date of the project was zero $ad_i = 0$, the makespan MS_i of a project i is here defined as the difference between the project's finishing date and the project's arrival date $MS_i = f_i - ad_i$. The project delay PD_i of project i is defined as the difference between the project's makespan and the critical path duration $PD_i = MS_i - CPD_i$, with CPD_i the critical path duration of project i. The critical path duration can be determined using the well known critical path method and it is a lower bound for the project makespan [Willis, 1985]. Commonly used global criteria are the total makespan (TMS), the average makespan (AMS), the average project delay (APD), and the standard deviation of the project delay (DPD). The total makespan is the difference between the latest finish time and the earliest arrival date of all single projects. Note that the earliest arrival date over all projects is always zero, because $ad_1 = 0$. The average makespan is the average of the makespans of all the projects ($AMS = \frac{\sum_{i=1}^{N} MS_i}{N}$). The average project delay is the average of the project delays of all the projects ($APD = \frac{\sum_{i=1}^{N} PD_i}{N}$). Finally the DPD is calculated as the standard deviation of the project delays of all the projects $DPD = \sqrt{\frac{\sum_{i=1}^{N} (PD_i - APD)^2}{N-1}}$. The remainder of the chapter only considers the APD and TMS objectives.

5.2.2 Game theoretic multi-agent learning approach

This section describes the game theoretic multi-agent learning approach (GT-MAS) to the DRCMPSP. It provides details of the applied encoding, schedule construction, multi-agent configuration and learning techniques.

Solution representation and encoding

Multi-project schedules are generated by organizing each project's activity list in a sequence of activity lists. This sequence consists of one activity list (AL) for each project, together with a project order list (POL). An AL is an ordered list of activities belonging to one project that is precedence feasible. When all ALs are combined as defined by the POL into one large AL, a combined sequence of activity lists (SAL) is obtained.

Once a SAL is generated, it serves for constructing a multi-project schedule with the serial schedule generation scheme (SGS) [Kolisch and Hartmann, 1998] in a multi-pass iterative forward-backward technique. The iterative forward-backward technique was introduced by [Li and Willis, 1992] and is known to produce good quality schedules. Lova et al. [2000] used the forward-backward technique in a multi-criteria heuristic for multi-project scheduling. The technique alternates between forward and backward passes until no more improvement can be made. The forward pass constructs schedules from the SAL, where activities start as early as possible (most left). The finish times of the activities in the forward schedules are then used to generate a new activity list that is subsequently used in a backward pass. The backward pass constructs schedules where activities start as late as possible (most right), while not exceeding the TMS of the preceding forward schedule. This procedure continues until no further objective improvements can be made. For the present purpose, the project managers jointly learn how to offer the schedule generator the best possible SAL.

Combining activity lists

The projects' activity lists and a project order list enable many possible ways for building a sequence of activity lists. The two most natural ways schedule projects either sequentially or interleaved. Obviously many hybrid combination techniques are equally applicable.

- **Sequential:** schedule all activities from the first project in the POL, then all activities from the second project in the POL, ...

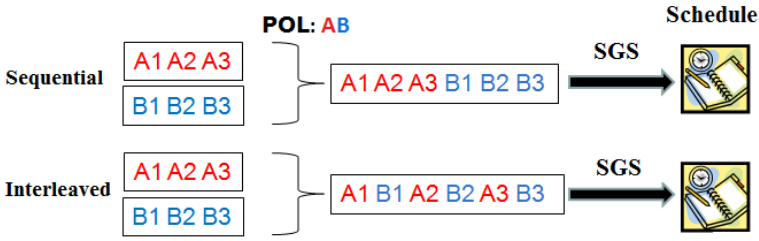


Figure 5.8: Combining activity lists: sequential and interleaved

- **Interleaved:** schedule the first activity from the first project in the POL, then schedule the first activity from the second project in the POL, ...

Figure 5.8 illustrates these two methods with a simple two-project example (A and B). The POL for this example determines that project A comes before project B.

The performance difference between the sequential and interleaved approach, will be studied in Section 5.2.3.

Learning activity lists using learning automata

A DRCMPSP environment is introduced with two main agent types: the project manager agents and the mediator agent. The approach consists of one single mediator agent while each project is represented by an individual project manager agent. Figure 5.9 illustrates the multi-agent configuration. In this section the main functionality of the project managers is described, while in Section 5.2.2 the role of the mediator will be discussed.

Each project manager agent is responsible for scheduling its own project. Its first responsibility is to learn to schedule its own activities efficient and feasible. Each activity in the project uses simple reinforcement learning devices called learning automata to learn an order of its successor activities. It is based on the technique for learning the multi-mode resource-constrained project scheduling problem (DRCMPSP) presented in Section 5.1, but without the modes.

A motivation for organizing the activities in a single project as a network of learning automata is that nice theoretical convergence properties are proven to hold in both single and multi automata environments. One of the principal contributions of LA theory is that a set of decentralized learning automata using the reward-inaction update scheme is able to control a finite Markov Chain

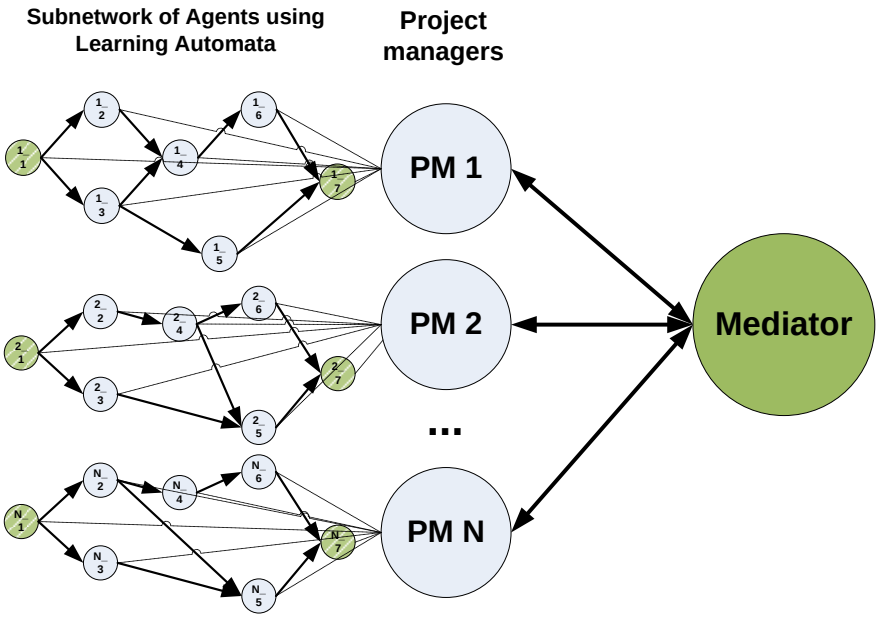


Figure 5.9: Multi-Agent configuration

with unknown transition probabilities and rewards. This result was extended to the framework of Markov Games, which is an extension of single-agent markov decision problems (MDP's) to distributed multi-agent decision problems [Littman, 1994]. Although the convergence results do not hold here because the activity-on-node model does not have the Markov property, good results are achieved with the network of LA in the single project scheduling scheme.

The project manager agent uses its network of LA to build an activity list. Further on, the manager logs its own performance for which it receives the information from the mediator, and updates the LA. The reward-inaction update scheme is applied because of the above mentioned theoretical convergence properties. The simple reinforcement signal used is the following. If the newly generated schedule resulted in an APD which was:

- Better: update all the LA with reward $\beta(t) = 1$
- Worse: update all the LA with reward $\beta(t) = 0$

A sequence learning game

The mediator collects the separate activity lists produced by the project managers and starts a process for determining the project order list. Based on the retrieved activity lists and the POL, it constructs a sequence of AL (SAL), which is used to build a multi-project schedule (with the serial schedule generation scheme). The mediator then sends the quality of the obtained schedule back to the project managers.

The order determination process to construct a POL can be seen as a game, more specifically as a dispersion game [Grenager et al., 2002] where the number of agents n is equal to the number of actions k. To construct a POL, all manager agents need to select a distinct action (an order here) or a maximally dispersed assignment of actions to agents. For example, if three agents select the following distinct actions: agent 1 selects action 2, agent 2 selects action 3 and agent 3 selects action 1, then the POL becomes $[3, 1, 2]$.

The Basic Simple Strategy (BSS) introduced by Grenager et al. [2002], allows agents to select maximally dispersed actions in a logarithmic (as a function of the number of agents) number of rounds, where a naive approach would be exponential. The original version uses a uniform selection, the method used in the current work incorporates the agents' preferences. This is achieved by introducing a probabilistic version of this BSS, called the Probabilistic Basic Simple Strategy (PBSS). The PBSS works as follows. Given an outcome $o \in O$ respecting the selected actions for all agents, and the set of all actions A, an agent using the PBSS will:

- select action a with probability 1 in the next round, if the number of agents selecting action a in outcome o is 1 ($n_a^o = 1$).
- select an action from the **probabilistic** distribution over actions $a' \in A$ for which $n_{a'}^o \neq 1$, otherwise.

The probabilistic distribution over actions is obtained from the experience the project manager has with previous decisions. This technique described in Chapter 3 for fast permutation learning was used. Each project manager agent maintains a single learning automaton to adjust its preferences for a place in the POL. Moreover, this method requires information about the actions that were selected uniquely during the PBSS procedure. The managers in the multi-agent system play the dispersion game, and the mediator provides the needed unique action selection information, i.e. for which actions $a \in A$ in outcome o the $n_a^o = 1$ holds. Figure 5.10 shows the performance of the probabilistic version of BSS in function of the number of agents n. It also results in a logarithmic

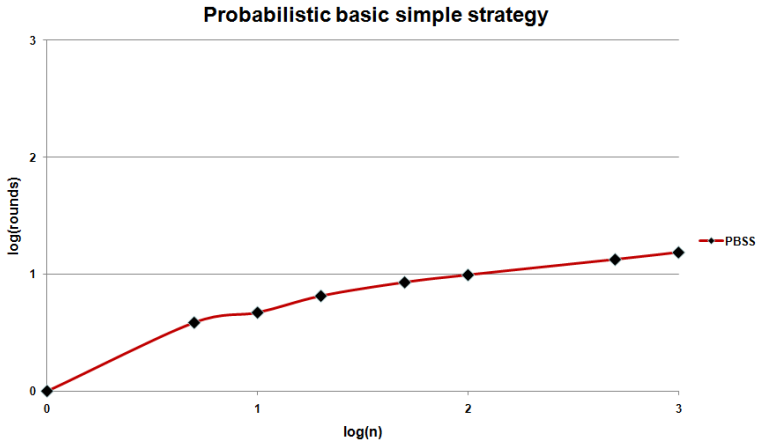


Figure 5.10: Average number of rounds to play in a dispersion game with n agents. Note that the axes have a logarithmic scale.)

behaviour as can be observed.

Algorithm 3 describes the full multi-agent system.

Summarized, the task of the mediator is to inform the project managers about the impact of their decisions and to coordinate the sequence learning game. The mediator does not take any decisions itself, it just deterministically constructs a schedule (SGS) using the decisions made by the project managers. The time required by the mediator is the time needed to send the messages and to construct a schedule.

5.2.3 Experiments and results

The present section presents the experimental results of the proposed game theoretic multi-agent learning method on standard benchmark problems for the DRCMPSP.

Problem instances

The proposed GT-MAS approach is assessed for the same 140 (60+80) DRCMPSP instances used by Homberger [2009] and Adhau et al. [2011]. These instances are available from the Multi-Project Scheduling Problem Library

Algorithm 3 Mediator Agent Control

Require: A DRCMPSP
Ensure: A feasible multi-project schedule
  Initialize project managers and mediator
  while Maximum number of schedule generations not exceeded **do**
    for all Project Managers **do**
      Generate a precedence feasible activity list (see Algorithm **??**)
      Send the generated activity list to the mediator
    end for
    {***determine the POL by playing a dispersion game***}
    while Not all project managers have chosen a different position in the POL **do**
      for all Project Managers **do**
        Choose a position in POL (=action) using PBSS strategy based on POL performance logs
        Send chosen position to mediator
      end for
      Mediator informs project managers about the actions that have been selected once.
    end while
    Mediator uses the POL and activity lists to create a SAL
    Mediator uses the SAL in a forward-backward SGS method to create a multi-project schedule $SCHED_{new}$
    if $SCHED_{new}$ has lower objective value than $SCHED_{best}$ **then**
      $SCHED_{best} = SCHED_{new}$ (store as new best)
    end if
    Mediator sends the obj_{new} to the project managers
    for all Project Managers **do**
      Update the learning automaton used for POL determination
      Update its sub-network of LA:
      if $obj_{new} < obj_{best}$ **then**
        Update all its LAs with a reward $\beta(t) = 1$
        $obj_{best} = obj_{new}$
      end if
    end for
  end while
  return $SCHED_{best}$

(`http://www.mpsplib.com` last check of address: 18 June 2012). The library contains 20 datasets with 140 instances based on multiple RCPSP instances (PSPLib, R. Kolisch, `http://129.187.106.231/psplib/`, 18 June 2012). The number of projects, is one out of $N = 2, 5, 10, 20$. The number of activities per project i, is one of $J_i = 30, 90, 120$. Thus the largest instances consist of 20×120 activities $= 2400$ activities. The library contains two types of datasets. The 'normal', ones which incorporate global and local resources with varying capacity, and the 'AgentCopp' (AC) instances where all the resources are global and have the same capacity. The normal datasets contain 5 instances per dataset, while the AC instances contain 10 instances per dataset. The instances vary in terms of arrival dates, and the ratio between the number of global and local resources. Table 5.5 shows the problem datasets and their properties, taken from [Homberger, 2009]. An additional column representing the total problem size (number of projects times number of jobs per project) was added for further scalability experiments. More information on the characteristics of the instances and the generation of the problems can be found in [Homberger, 2009].

Sequential vs interleaved scheduling

Figure 5.11 shows the schedules produced with the same SAL, for both the sequential and the interleaved method. They were applied to an instance with 10 projects, and 90 activities each (mp90_a10_nr2). The graphs show the number of activities executed per time step for all 10 projects (stacked on top of each other). The schedule on the left is the sequential one and it has $APD = 15.6$ and $TMS = 164$, while the schedule on the right (interleaved) has $APD = 22.9$ and $TMS = 148$. When the two scheduling methods are compared, considerable differences in the produced schedules can be noticed. Figure 5.12 shows the difference between sequential and interleaved scheduling for different global objectives like APD, TMS, DPD and AMS. The graph shows average values over $10,000$ randomly generated multi-activity lists for one instance of the datasets, but similar graphs can be shown for other instances. In general, the sequential scheduling method produces schedules with lower APD and higher TMS, while the interleaved scheduling method produces schedules with higher APD and lower TMS.

Comparison with best known results

The experiments in this section were carried out using the following setup. The tests were executed on an Intel Core 2 Duo PC (3.3Ghz, 4GB RAM, Windows Vista), and all the methods were implemented in the Java 1.6 programming

Problem subset	*NOI*	Characterization per instance			*Size*
		N	J_i	$(\|G\|;\|L_i\|)$	
MP30_2	5	2	30	(1;3) or (2;2) or (3;1)	60
MP90_2	5	2	90	(1;3) or (2;2) or (3;1)	180
MP120_2	5	2	120	(1;3) or (2;2) or (3;1)	240
MP30_5	5	5	30	(1;3) or (2;2) or (3;1)	150
MP90_5	5	5	90	(1;3) or (2;2) or (3;1)	450
MP120_5	5	5	120	(1;3) or (2;2) or (3;1)	600
MP30_10	5	10	30	(1;3) or (2;2) or (3;1)	300
MP90_10	5	10	90	(1;3) or (2;2) or (3;1)	900
MP120_10	5	10	120	(1;3) or (2;2) or (3;1)	1200
MP30_20	5	20	30	(1;3) or (2;2) or (3;1)	600
MP90_20	5	20	90	(1;3) or (2;2) or (3;1)	1800
MP120_20	5	20	120	(1;3) or (2;2) or (3;1)	2400
MP90_2AC	10	2	90	(4;0)	180
MP120_2AC	10	2	120	(4;0)	240
MP90_5AC	10	5	90	(4;0)	450
MP120_5AC	10	5	120	(4;0)	600
MP90_10AC	10	10	90	(4;0)	900
MP120_10AC	10	10	120	(4;0)	1200
MP90_20AC	10	20	90	(4;0)	1800
MP120_20AC	10	20	120	(4;0)	2400

Table 5.5: Problem datasets and their properties [Homberger, 2009]. *NOI* is the number of instances, N is the number of projects, J_i is the number of activities of project i, $|G|$ is the number of global resources, $|L_i|$ is the number of local resources of project i, *Size* is the total number of activities.

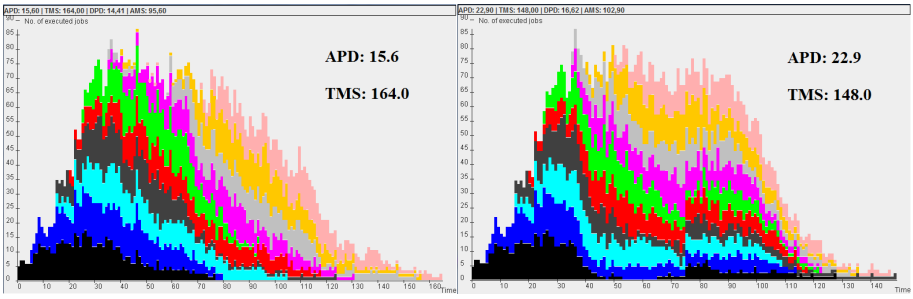


Figure 5.11: Schedule comparison, sequential (left) vs interleaved (right) activity list combination methods. Problem instance: mp_j90_a10_nr2.

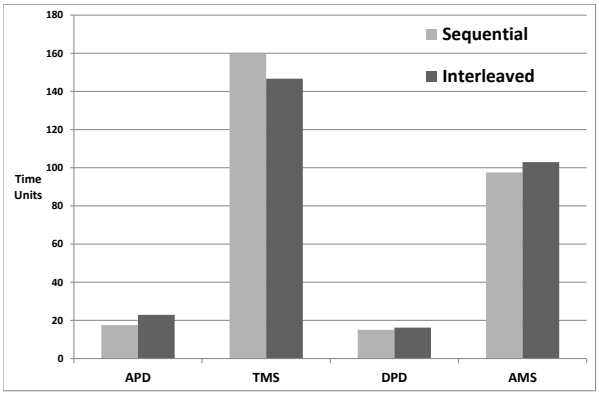


Figure 5.12: Performance comparison between sequential and interleaved scheduling. Problem instance: mp_j90_a10_nr2.

language. The stopping criterion was set to 100,000 schedule generations, in order to enable comparing with the 100,050 schedule generations used by all the methods in [Homberger, 2009]. A learning rate $\alpha_{reward} = 0.001$ for updating all the learning automata is used. A sequential activity list combination method is used when the APD objective is considered, while an interleaved activity list combination method is used for the TMS objective.

The first analysis considers the APD objective. The GT-MAS approach is compared with the best found solutions, either centralized or decentralized, in the literature. The APD results are shown in Table 5.6. The table shows that GT-MAS obtains an average APD increase of 24.5% compared to the best in the literature. It is interesting to see that for the very large instances (up to 20 projects with each 120 activities to be planned) the achieved improvements are even better (up to 41%), indicating good scalability. Figure 5.13 compares the average project delay over all instances of GT-MAS with all the methods from the literature. It is clear that GT-MAS realizes a significant APD improvement over other methods from the literature Many new best APD solutions were found, showing 104 best APD solutions out of 140 instances on the MPSPLIB benchmark website (last check on July 18, 2012).

When the total makespan objective (TMS) is considered, the GT-MAS approach performs best with an interleaved activity list combination method. Table 5.7 shows a TMS result of the GT-MAS method using the interleaved method.

	Literature	GT-MAS	
Problem subset	(APD_{AV})	(APD_{AV})	Percentual Difference
MP30_2	12.4	**11.2**	9.7%
MP90_2	5.6	**5.3**	5.4%
MP120_2	60.8	**49.4**	18.8%
MP30_5	16.7	**15.4**	7.8%
MP90_5	8.9	**7.8**	12.4%
MP120_5	65.1	**48.5**	25.5%
MP30_10	84.4	**52.0**	38.4%
MP90_10	46.1	**31.8**	31.0%
MP120_10	131.0	**100.0**	23.6%
MP30_20	177.8	**111.4**	37.3%
MP90_20	30.2	**17.6**	41.8%
MP120_20	31.8	**28.2**	11.3%
MP90_2AC	127.8	**104.3**	18.4%
MP120_2AC	47.0	**35.2**	25.1%
MP90_5AC	287.8	**244.6**	15.0%
MP120_5AC	247.5	**178.8**	27.8%
MP90_10AC	244.9	**169.4**	30.8%
MP120_10AC	151.0	**96.9**	35.8%
MP90_20AC	146.4	**85.4**	41.7%
MP120_20AC	237.1	**158.6**	33.1%

Table 5.6: Comparison of average project delay with the best in the literature

The GT-MAS approach is able to keep up with the best in literature. Even a slight improvement can be observed on some datasets. Figure 5.13 compares the total makespan over all instances of GT-MAS with all the methods from the literature. GT-MAS performs best of all decentralized methods. Only RES, which is a centralized method, performs slightly better. Several new best TMS solutions were obtained. Moreover, the GT-MAS approach generated some pareto non-dominated solutions, which improve on both objectives (APD and TMS). Figure 5.15 illustrates a multi-objective comparison of GT-MAS with other methods on instance j90_a10_nr5.

The result mentioned in Table 5.6 and Table 5.7 have been validated and uploaded to the Multi-Project Problem Library website.

Problem subset	Literature (TMS_{AV})	GT-MAS (TMS_{AV})	Percentual Difference
MP30_2	63.6	**63.4**	0.3%
MP90_2	**107.2**	107.6	-0.4%
MP120_2	**169.0**	173.4	-2.6%
MP30_5	89.2	**87.2**	2.2%
MP90_5	**123.6**	123.8	-0.2%
MP120_5	**182.2**	191.0	-4.8%
MP30_10	180.6	**180.4**	0.1%
MP90_10	**180.4**	182.4	-1.1%
MP120_10	**279.8**	287.8	-2.9%
MP30_20	**327.4**	330.8	-1.0%
MP90_20	161.6	**161.4**	0.1%
MP120_20	187.6	**186.0**	0.9%
MP90_2AC	**232.2**	235.7	-1.5%
MP120_2AC	**139.2**	147.2	-5.7%
MP90_5AC	**538.2**	551.7	-2.5%
MP120_5AC	**480.7**	493.1	-2.6%
MP90_10AC	**458.3**	469.4	-2.4%
MP120_10AC	**350.7**	357.3	-1.9%
MP90_20AC	**285.9**	286.7	-0.3%
MP120_20AC	**506.4**	512.9	-1.3%

Table 5.7: Comparison of total makespan with the best in the literature

Effect of the sequence game

Figure 5.16 shows the difference between a completely randomly chosen SAL (random feasible activity lists and random project order), and a SAL constructed with the sequence game (dispersion game). The figure shows average APD evolution per 1000 iterations, for one DRCMPSP instance. The noticeable APD increase is due to learning a good sequence via a dispersion game.

5.2.4 Conclusion

A new multi-agent approach (GT-MAS) to the decentralized resource-constrained multi-project scheduling problem (DRCMPSP) was proposed. Agents are called project manager agents, and they learn a sequence to order their activities using a network of simple reinforcement learners, i.e. learning automata. In the meantime, project managers also play a sequence game, in

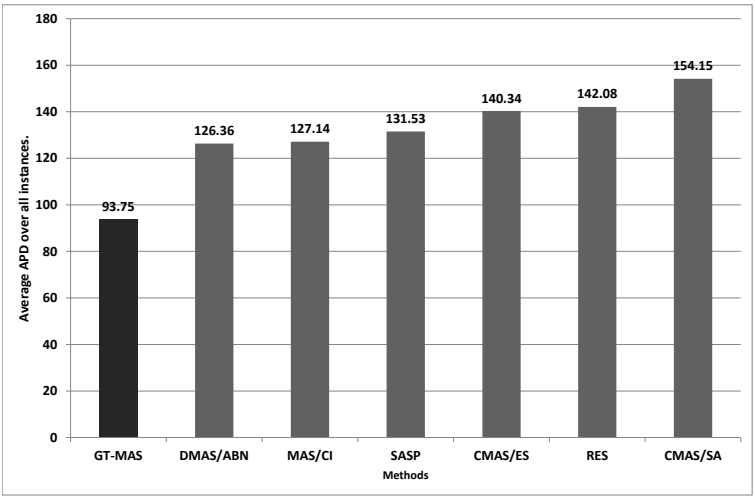


Figure 5.13: Comparison of algorithms with respect to average project delay over all problem instances.

which they all need to select a distinct action, representing their position in the overall project order list. The outcome of this game determines the order of the projects' activity lists for building a multi-project schedule. A dispersion game was played with a probabilistic version of the basic simple strategy, which is a method with logarithmic performance characteristics. An additional mediator agent was used to coordinate the dispersion game, and to build a multi-project schedule using a well known serial schedule generation procedure. The serial schedule generation procedure was also applied into a smart forward-backward mechanism that is known to improve the schedule's time criteria. The mediator agent has the task to inform the project managers about the quality or the impact of their decisions, but does not take any decisions himself. Combining the separate activity lists in a sequential way by playing the sequence game (opposed to an interleaved way), leads to smaller average project delays.

The proposed GT-MAS approach was evaluated on the MPSPLIB benchmark, and was able to generate schedules with an average project delay superior to all the previous best results. Concerning the total makespan objective the algorithm shows to be competitive with the state-of-the-art, and delivers better results than other decentralized methods. Many new best schedules on the DRCMPSP

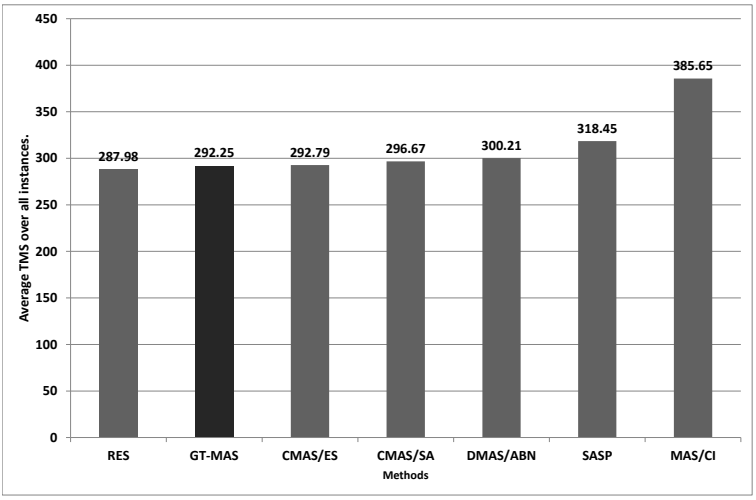


Figure 5.14: Comparison of algorithms with respect to total makespan over all problem instances.

instances were found, some of them even improved on both considered objectives (average project delay and total makespan). The experiments conducted on the MPSPLIB benchmark clearly revealed the algorithm's scalability by showing very good results on the largest instances.

It would be interesting to investigate the influence of different performance measures (such as individual project delays) to define the project manager's preferences in the dispersion game. In addition a multi-objective optimization version of the DRCMPSP, where pareto non-dominated solutions need to be found, opens perspectives for further improvement.

5.3 Conclusion

In this chapter it was shown how RL is capable of learning good quality solutions for two general project scheduling problems. More specifically a network of learning automata coupled by a common reward signal was employed. Basic $0-1$ reward signals upon improvement and a linear reward-inaction update

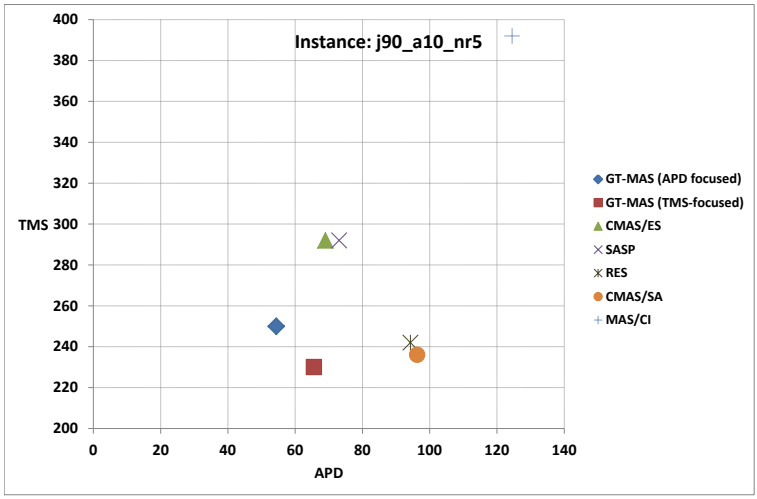


Figure 5.15: Multi-objective comparison on instance j90_a10_nr5.

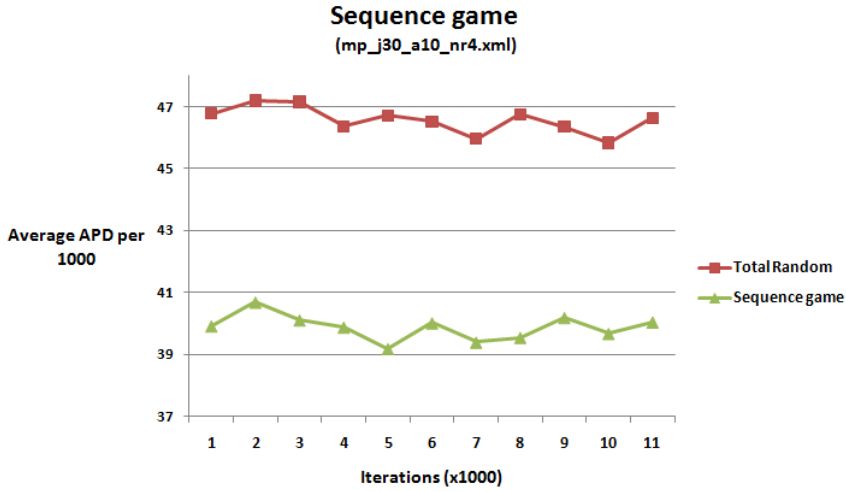


Figure 5.16: Effect of the sequence game

mechanism are sufficient to deliver state-of-the-art results and many new best solutions. In addition the techniques from Chapter 3 were used to (anti-)coordinate the project managers in the decentralized multi-project scheduling problem.

The methods described in this chapter can be applied to other scheduling or combinatorial optimization problems in a similar fashion. One such example is the flexible job shop scheduling discussed in the next chapter.

Chapter 6

Learning automata for flexible job shop scheduling

6.1 Introduction

Scheduling jobs on machines plays a crucial role in manufacturing environments. Jobs have to be processed on time as much as possible, while the available resources have to be used in an efficient manner. Some machine scheduling problems are polynomial solvable using a simple heuristic rule, while others are strongly NP-hard. For example the case with only one machine and a total weighted completion time objective is optimally solvable with a simple heuristic rule, i.e. the weighted shortest processing time. Real world scheduling problems are often more complex than this, including properties as: multiple machines, non-identical machines, multiple execution modes, not all jobs are released at the same time, machines can have breakdowns, changeovers, etc. Many of these properties are considered in this chapter. We propose an effective learning method for the flexible job shop scheduling problem (FJSP), and an adapted version for the dynamic problem extension.

In Chapter 5 state-of-the-art learning enhanced search methods were introduced for solving two general project scheduling problems, i.e. the multi-mode resource-constrained project scheduling problem (MRCPSP) and the decentralized resource-constrained multi-project scheduling problem (DRCMPSP). These problems are generalizations of a large class of scheduling problems from practice. For example a flexible job shop scheduling problem with multiple production lines or machines can be modeled as an MRCPSP. A typical production scheduling

problem with multiple orders for different customers can be modeled as a DRCMPSP, where the orders are projects and the operations are activities.

In cooperation with Elsy Kaddoum[1] and Yailen Martinez[2] a comparative study was performed on different learning and optimization methods for the flexible job shop scheduling problem with due-dates, release-dates and perturbations [Kaddoum et al., 2010]. The goal was to study the differences and similarities between purely reactive decentralized approaches and global optimization approaches. Another question we tried to answer is: is complete or partial rescheduling in case of a dynamic environment more suitable than trying to adapt online (during execution)?

Many tabu search approaches have been proposed in the literature [Brandimarte, 1993, Hurink et al., 1994, Barnes and Chambers, 1995, Nowicki and Smutnicki, 1996, Dauzère-Pérès and Paulli, 1997, Brucker and Neyer, 1998] Recently a genetic algorithm was introduced for solving the FJSP [Pezzella et al., 2008] improving on existing GA approaches for this problem. To the best of our knowledge no RL approaches were applied to this problem. However, a reinforcement learning approach to the classic (non-flexible) job shop scheduling problem was proposed by Zhang and Dietterich [1995].

In what follows the flexible job shop scheduling problem with all its constraints and objectives is described. It is shown how learning automata can be successfully applied to this NP-hard problem, similarly to what has been done for the MRCPSP. It is NP-hard because it is a generalization of the classical job shop scheduling problem which is also NP-hard [Garey et al., 1976]. RL methods, more specifically learning automata, are employed at the direct RL inclusion level (left image of Figure 2.4). As in previous chapters, a simple common $0-1$ reward signal is used to update the learning automata.

6.2 Problem description

The flexible job shop scheduling problem (FJSP)(also called the multi-mode job shop problem [Brucker and Neyer, 1998]) in this study consists of performing a set of n jobs $J = \{J_1, J_2, \ldots, J_n\}$ on a set of m machines $M = \{M_1, M_2, \ldots, M_m\}$. A job J_i has an ordered set of o_i operations $O_i = \{O_{i,1}, O_{i,2}, \ldots, O_{i,o_i}\}$. Each operation $O_{i,j}$ can be performed on any among a subset of available of machines ($M_{i,j} \subseteq M$), also called *execution modes* further in this text. Since the machines are not fixed in advance, the problem actually includes a routing or assignment

[1]Systèmes Multi-Agents Coopératifs (SMAC), IRIT, Université Paul Sabatier, Toulouse, France

[2]Computational Modeling Lab (CoMo), Vrije Universiteit Brussel, Brussels, Belgium

component. Executing operation $O_{i,j}$ on machine M_k takes $p_{i,j,k}$ processing time (or duration). Operations of the same job have to respect the finish-start precedence constraints given by the operation sequence. A machine can only execute one operation at a time. An operation can only be executed on one machine and cannot leave it before the treatment is finished. In contrast to the MRCPSP, operations in the FJSP do not have more than one predecessor operation, and only renewable resources are considered.

6.2.1 Dynamic environment

The basic problem formulation was extended with the following features, and tested in a dynamic environment. A job J_i is released at time r_i and is due at time d_i. A machine M_k can have perturbations (e.g. breakdowns) causing already started operations to suspend their execution. The interrupted operation can continue when the perturbation is finished. Once an operation has started on a machine it can not move to another machine. In a way, this can be considered as a limitation with respect to the basic FJSP.

To create a dynamic environment, problem information is released in a step by step manner. This means that jobs in the future are not known in advance, neither are machine perturbations. Information about job i is only known at time $t = r_i$. Such a release of information is called an event. There are three possible events:

- Job release event.
- Machine perturbation started event.
- Machine perturbation finished event (the duration of the perturbation is not known in advance).

In the dynamic environment it is allowed that operations can be rescheduled as long as they have not been started yet. This dynamic extended version is further denoted as DFJSP, to distinguish it from the flexible job shop scheduling problem (FJSP).

6.2.2 Objectives

We denote the scheduled start and completion time of an operation $O_{i,j}$ as $s_{i,j}$ and $c_{i,j}$. The completion time of a job c_i is equal to the completion time of its latest operation c_{i,o_i}. The tardiness of a job J_i is $T_i = max(c_i - d_i, 0)$. If a job

J_i has a tardiness larger than zero ($T_i > 0$), then we say that it is tardy and $U_i = 1$ else $U_i = 0$. The following objectives are used:

- $C_{max} = max\{C_i | 1 \leq i \leq n\}$: makespan or completion time of the last job that leaves the system,
- $T_{max} = max\{T_i | 1 \leq i \leq n\}$: maximum tardiness,
- $\overline{T} = (1/n) \sum_{i=1}^{n} T_i$: mean tardiness, and
- $T_n = \sum_{i=1}^{n} U_i$: the number of tardy jobs.

No weights are used, all jobs are equally valued. The most common objective for this problem is the makespan objective.

6.3 Learning approach

As in the method applied to the MRCPSP in Chapter 5, an agent with two learning automata is used for each operation in order to generate a precedence feasible operation list. One LA is used for selecting the next job to select an operation from, while the other is used for selecting the machine to execute the operation. An additional learning automaton (LA_0) is added for selecting the job to start with. Figure 6.1 shows the LA used for learning a precedence feasible operation list. The arrows denote the actions each LA can take. For example, the LA for operation $O_{1,1}$ can choose between the next operation of the same job ($O_{1,2}$), or the next operation of Job 2. Using this LA configuration, no solutions are excluded from the search space. All LA are equipped with a linear reward-inaction update scheme (L_{R-I}). As described in previous chapters of this thesis, the L_{R-I} has interesting theoretical properties, which also show good results for practical problems. In what follows we will describe how the LA will learn a precedence feasible operation list and how they will learn the execution modes. Both have to be learned simultaneously in order to reach good quality solutions.

6.3.1 Learning a precedence feasible operation list

In order to learn a precedence feasible operation list, the learning method starts in LA_0 and chooses a job. The first unvisited operation of the chosen job is added to the end of the operation list. The corresponding LA has the following choices: go to the next operation of the same job, or go to another job. Each

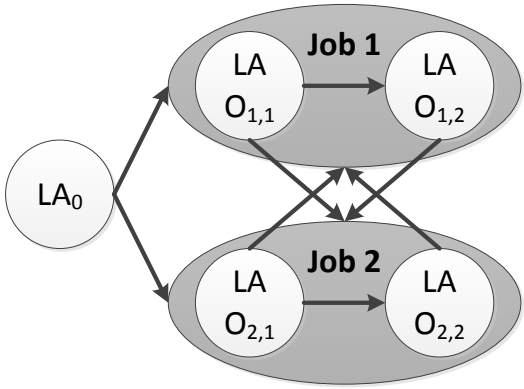


Figure 6.1: Learning automata configuration for learning a precedence feasible operation list. Example with 2 jobs, each job having 2 operations.

time a job is chosen, it automatically goes to the next unvisited operation of that job. If a selected job has no unvisited operations left, a random job with at least one unvisited operation is chosen.

6.3.2 Learning execution modes

For each operation a second corresponding LA is responsible for learning the execution mode of that operation, i.e. on which machine the job will be performed. The LA has one action per execution mode, with a maximum of m (number of machines) actions.

6.3.3 Schedule generation

Schedules are generated with a serial schedule generation scheme (serial-SGS), adopted from the project scheduling literature [Kolisch and Hartmann, 1998]. This well known list based scheduling technique is perfectly suited for use in combination with heuristics. The serial-SGS requires a precedence feasible operation list as input to generate a schedule. Each operation in the operation list is scheduled at the first available time where all the required resources are available for the whole duration of the operation. The operation duration and resource requirements are determined by the chosen execution modes. The serial-

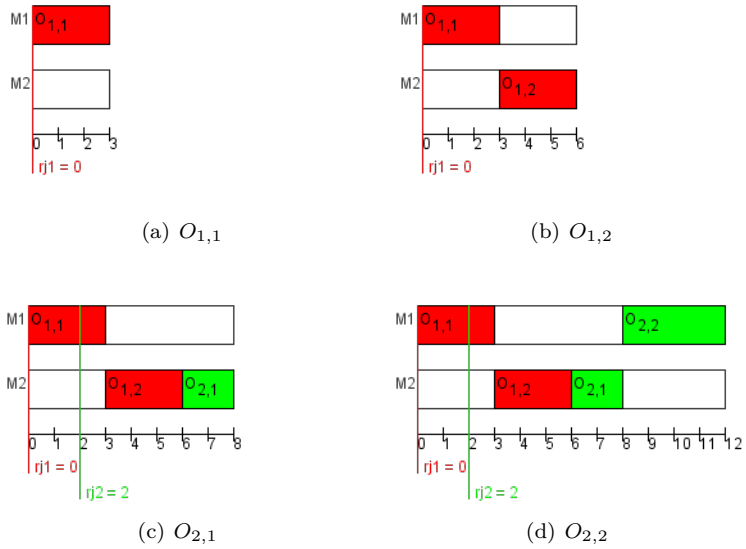
SGS is augmented with the effective forward/backward procedure introduced by Li and Willis [1992]. The procedure alternates between forward and backward steps. In each step the operation list is resorted according to the schedule that was obtained in the previous forward/backward step. The first step performs a forward pass on the input operation list, resulting in a forward schedule. The second step, resorts the operation list according to the completion times of the operations in the forward schedule, and performs a backward pass on this resorted operation list. The third step, resorts the operation list according to the completion times of the operations in the backward schedule, and performs a forward pass on this resorted operation list. This procedure can now continue until no more improvements are realized between subsequent forward passes.

SGS example

Consider the following example to illustrate the serial-SGS with forward/backward procedure. A problem instance with two jobs (J_1 and J_2) and two machines (M_1 and M_2). Each job has two operations. Job 1 has a release date $r_1 = 0$. Job 2 has a release date $r_2 = 2$. Assume the execution modes are already fixed to:

- $p_{1,1,1} = 3$ for operation $O_{1,1}$ (processing time of 3 units on machine M_1),
- $p_{1,2,2} = 3$ for operation $O_{1,2}$,
- $p_{2,1,2} = 2$ for operation $O_{2,1}$,
- $p_{2,2,1} = 4$ for operation $O_{2,2}$.

We start the SGS procedure with the following operation list: $O_{1,1}$,$O_{1,2}$,$O_{2,1}$,$O_{2,2}$. Figure 6.2 shows how this operation list is scheduled by the SGS in a forward pass, resulting in a schedule with makespan $C_{max} = 12$. We start with operation $O_{1,1}$. Job 1 has a release date $r_1 = 0$, and the execution mode of operation $O_{1,1}$ has a processing time of 3 time units. Given an empty schedule we can schedule $O_{1,1}$ at time $t = 0$. The next operation in the operation list is $O_{1,2}$, which needs to be scheduled after $O_{1,1}$ (precedence relation). Operation $O_{1,2}$ is therefore scheduled at time $t = 3$. Operation $O_{2,1}$ must be scheduled after $t = 2$, because job 2 has a release date of $r_2 = 2$. However, operation $O_{1,2}$ occupies machine M_2 during $t = [3, 6]$, causing $O_{2,1}$ to be scheduled at $t = 6$. The last operation in the list is operation $O_{2,2}$ which must be scheduled after $O_{2,1}$ and is therefore scheduled at $t = 8$. All operations are now scheduled, the resulting schedule has a makespan $C_{max} = 12$.



(a) $O_{1,1}$ (b) $O_{1,2}$

(c) $O_{2,1}$ (d) $O_{2,2}$

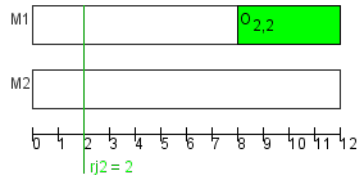
Figure 6.2: Serial-SGS example - first forward step, resulting in a schedule with makespan $C_{max} = 12$.

Resorting the operation list according to the completion times of the operations in the schedule of Figure 6.2, gives the following operation list: $O_{2,2}$,$O_{2,1}$,$O_{1,2}$,$O_{1,1}$. We now perform a backward pass on this operation list, shifting all operations to the right not further than the $t = 12$ (the makespan of the forward schedule). The steps of this backward pass are shown in Figure 6.3. The backward pass does not take the release dates into account.

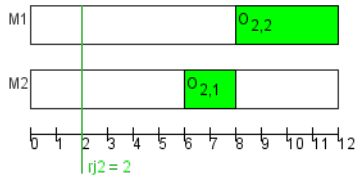
Again we resort the operation list, but this time according to the start times of the operations in the backward schedule (Figure 6.3). The resulting operation list is: $O_{1,1}$,$O_{2,1}$,$O_{2,2}$,$O_{1,2}$. Rebuilding the schedule using the SGS in a forward pass, results in the schedule of Figure 6.4. The schedule has a makespan of $C_{max} = 8$, which is 4 time units less than the first forward pass. As illustrated in this example, the forward/backward procedure can lead to more compact schedules.

6.3.4 Reward signal

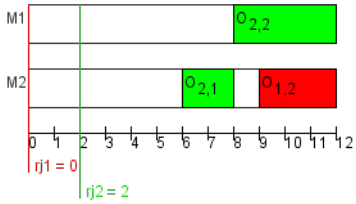
Similar to our work on the MRCPSP (Chapter 5), a common reward signal is used to immediately update the LA after generating a schedule with the chosen



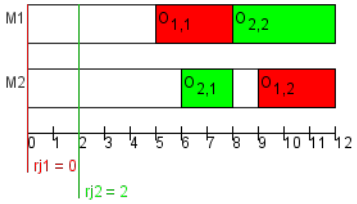
(a) $O_{2,2}$



(b) $O_{2,1}$

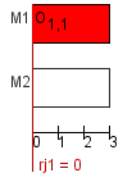


(c) $O_{1,2}$

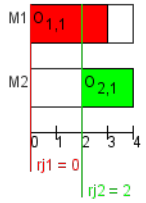


(d) $O_{1,1}$

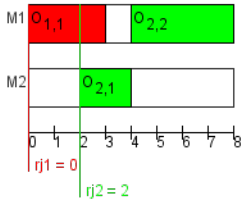
Figure 6.3: Serial-SGS example - backward step.



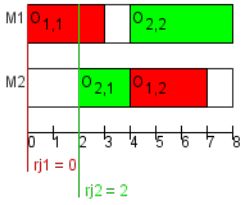
(a) $O_{1,1}$



(b) $O_{2,1}$



(c) $O_{2,2}$



(d) $O_{1,2}$

Figure 6.4: Serial-SGS example - second forward step, resulting in a schedule with makespan $C_{max} = 8$.

operation list and execution modes. Update with a

- reward of 1, if the schedule improved,
- reward of $r_{equal} \in [0, 1]$, if the schedule was equal to the previous best,
- reward of 0, otherwise.

This reward signal stimulates learning improved schedules. The r_{equal} parameter must be set to a small value, in order to avoid premature convergence. In all the following experiments we set $r_{equal} = 0.01$.

6.3.5 Application to the dynamic FSJP

In order to also apply the learning approach from the FJSP to the dynamic FSJP, we only need to change the reward signal and relearn on every new event. This reward signal is then not based on improvements of the complete schedule, but on improvements of the partial schedule. A partial schedule is a schedule where not all operations are scheduled yet (here because of unknown future information). For example: the makespan is the finishing time of the latest operation in the partial schedule.

At every time step the problem is checked for new events. If no events occurred then nothing has to be done, else the 'future' has to be (re-)scheduled during a number of learning iterations (LIT). As described by the DFJSP, operations can not be re-scheduled if they have already started. The dynamic execution can be described as in Algorithm 4, and is further called the 'RollingTime' approach.

Let us illustrate the dynamic execution with the following small example. Consider a DFJSP problem with 2 jobs (J_1 and J_2), each job consisting of two operations. J_1 has a release date at time $r_1 = 0$, while J_2 has a release date of $r_2 = 2$. The operation execution modes are:

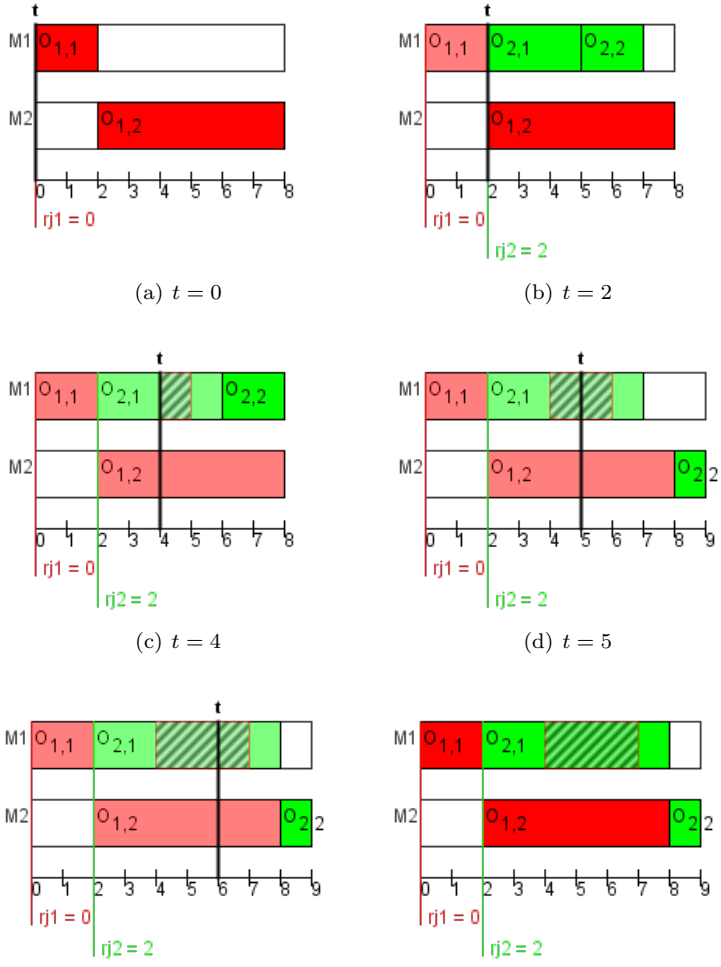
- Job 1 operation 1 ($O_{1,1}$): $p_{1,1,1} = 2$
- Job 1 operation 2 ($O_{1,2}$): $p_{1,2,2} = 6$
- Job 2 operation 1 ($O_{2,1}$): $p_{2,1,1} = 3$
- Job 2 operation 1 ($O_{2,2}$): $p_{2,2,1} = 2$ or $p_{2,2,2} = 1$

Note that the only operation with more than one execution mode is $O_{2,2}$. Furthermore, a breakdown will occur at time $t = 4$ on machine M_1, the breakdown will end at time $t = 7$.

The partial schedules generated by the RollingTime approach for this dynamic example are shown in Figure 6.5. At time step $t = 0$ only the operations of job J_1 are known and scheduled, resulting in a partial schedule with makespan $C_{max} = 8$ shown in Figure 6.5(a). At time step $t = 2$, job J_2 is released, and all operations of this job are scheduled on machine M_1 (Figure 6.5(b)). $O_{2,2}$, the operation with two available execution modes, was scheduled on machine M_1 because the other operation mode would result in a schedule with a higher makespan. At time step $t = 4$, a breakdown on machine M_1 occurs (Figure 6.5(c)). The duration of the breakdown is unknown and operation $O_{2,2}$ stays on machine M_1, because changing to machine M_2 would result in a larger makespan. At time step $t = 5$ the breakdown has not been sorted out yet (Figure 6.5(d)). Since operation $O_{2,1}$ was already in execution at the start of the breakdown, the duration of operation $O_{2,1}$ was extended with one time slot, causing operation $O_{2,2}$ to move from machine M_1 to machine M_2. At time step $t = 6$ the breakdown is still ongoing (Figure 6.5(e)), again leading to an extended duration of operation $O_{2,1}$. The breakdown finishes at $t = 7$, resulting in the final schedule with makespan $C_{max} = 9$ shown in Figure 6.5(f). Note that if $O_{2,2}$ would not have changed its execution mode, the makespan would be $C_{max} = 10$.

Algorithm 4 Dynamic execution of the learning algorithm

Require: Machine information
Ensure: Feasible schedule
  $t \Leftarrow 0$ **time step**
  $E \Leftarrow \emptyset$ **set of events**
  $S \Leftarrow emptySchedule$ **current schedule**
  while not all operations executed **do**
    $E \Leftarrow getEvents(t)$ **get all events a time t**
    if $E \neq \emptyset$ **then**
      $S \Leftarrow reschedule(S, E, t)$
    end if
    $t++$
  end while
  return S



(a) $t = 0$ (b) $t = 2$

(c) $t = 4$ (d) $t = 5$

(e) $t = 6$ (f) Final schedule

Figure 6.5: Evolution of the partial schedules during the RollingTime approach.

Dataset	No. Instances	n	m	o_i
Brandimarte [1993]	10	[10-20]	[4-15]	[5-15]
Dauzère-Pérès and Paulli [1997]	18	[10-20]	[5-10]	[5-25]
Barnes and Chambers [1995]	21	[10-15]	[11-18]	-
Hurink et al. [1994]	129	[6-30]	[5-15]	-

Table 6.1: Datasets and their characteristics

6.4 Experimental results

6.4.1 Datasets from the literature

The proposed learning algorithm is tested against a large number of problem instances from the literature (`http://www.idsia.ch/~monaldo/fjsp.html`). Multiple sets of problem instances without release dates, due dates and perturbations have been considered. Table 6.1 shows these problem datasets and their main characteristics: the number of instances in the dataset, the number of jobs n, the number of machines m and the number of operations per job o_i.

6.4.2 DFJSP datasets

Since no datasets are available for the DFJSP, we generated new datasets based on existing FJSP datasets. Two problem categories were generated: instances with perturbations and instances without perturbations. Five problem instances were generated for each FJSP instance from the Brandimarte dataset in the following manner.

Release dates and due dates

The release dates and due dates are uniformly generated between lower and upper bound values. The lower bound LB is the best known lower bound for the FJSP instance which is used as a basis. The upper bound UB is obtained by applying a simple greedy algorithm to the FJSP instance. This greedy algorithm assigns each operation to the first available machine from the list of eligible machines.

Perturbations

Perturbations or machine breakdowns are unpredictable events and are not known in advance to the scheduler. The inter-arrival times of the perturbations follow a Poisson distribution which is defined by the mean inter-arrival time. The mean inter-arrival time λ_{ia} is calculated as: $\lambda_{ia} = \frac{UB}{z}$, where z is the upper bound division factor. A higher z value leads to shorter inter-arrival times, and thus more perturbations. The instances have been generated with $z \in [2, 10]$.

The duration of the perturbations are generated using an Erlang Distribution and rounded to the nearest integer. An Erlang Distribution is defined by two parameters: the shape parameter k which is a non-negative integer, and the rate parameter λ which is a non-negative real number. The following parameter values were used: $k = 6$ and $\lambda = 2$, leading to a mean perturbations duration of 3 time units.

6.4.3 Effect of learning

The learning rate α_{reward} is the only parameter of the proposed method. Figure 6.7 shows the influence of the learning rate on the performance of the algorithm. 10 runs of $10,000$ iterations were conducted on instance Mk01 from the Brandimarte dataset [Brandimarte, 1993]. The best known makespan for instance Mk01 is 40. The learning rate values range from 0 to 1 and increase in steps of 0.01. A learning rate between 0.1 and 0.4 shows an improved performance over no learning ($\alpha_{reward} = 0$). On the other hand, too high learning rates > 0.5 result in bad performance, even worse than without learning. Similar behaviour was observed while solving other instances of the FJSP problem. The time needed to perform a run of $10,000$ learning iterations on instance Mk01 with 10 jobs and 6 machines is about 1 second on a PC with Intel Core 2 Duo 2.8GHz processor.

Note that, despite of the randomly generated instances, the network of LA is still capable of learning valuable information.

Figure 6.8 shows an iteration-cost comparison between a search with learning ($\alpha_{reward} = 0.2$) and a search without learning, on instance Mk01 and for $100,000$ iterations. The learning search method is able to find more and better solutions, and for this instance even the best known solution with a makespan $C_{max} = 40$. The schedule found by the learning approach is shown in Figure 6.6.

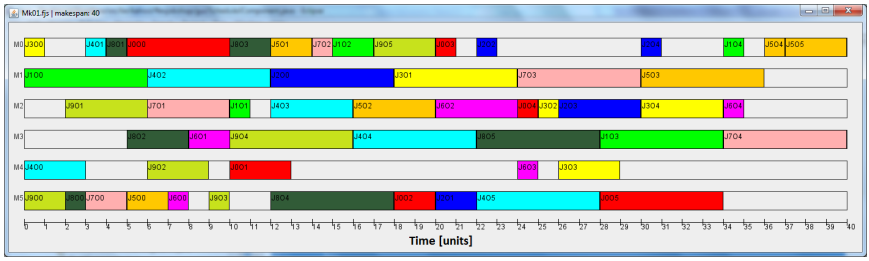


Figure 6.6: Schedule with makespan $C_{max} = 40$, generated by the learning approach for instance Mk01. The graphical user interface (GUI) was developed by the author during the course of this thesis.

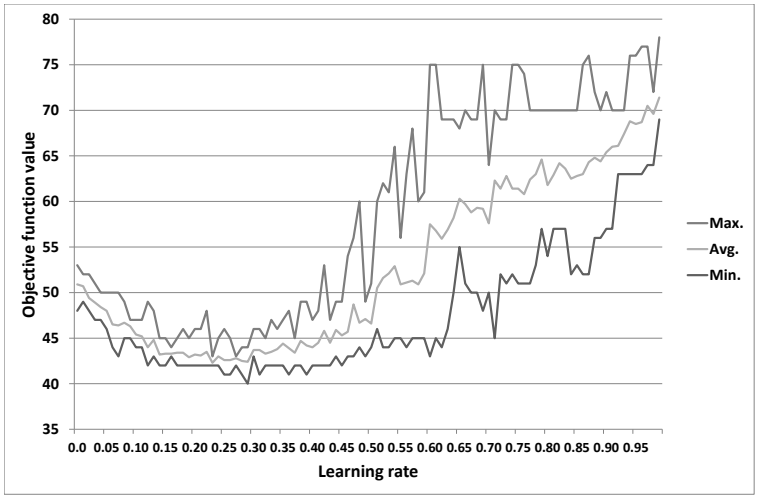


Figure 6.7: Influence of the learning rate on the objective function value. Minimum, average and maximum values on instance Mk01 from the Brandimarte dataset for different learning rate settings.

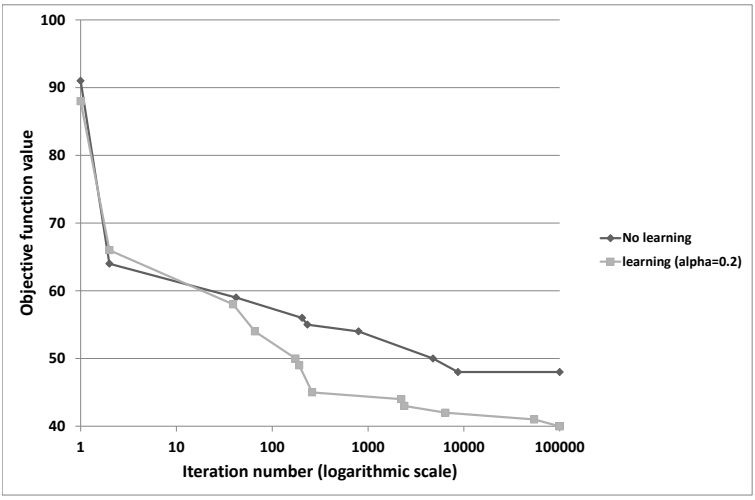


Figure 6.8: Iteration-cost comparison between learning and no learning, on instance Mk01 from the Brandimarte dataset.

6.4.4 FJSP - Comparison to other approaches

Table 6.2 compares the learning approach to other approaches from the literature, when applied to the Brandimarte instances. GA is the genetic algorithm from Pezzella et al. [2008], while M.G. denotes the effective approach of Mastrolilli and Gambardella [1996]. The learning approach (denoted as LA in the table) performed 100,000 learning iterations with a learning rate $\alpha_{reward} = 0.3$. The results show to be competitive with these state-of-the-art approaches, reaching similar solution values on many instances. For instance Mk09 the learning approach performs even better than the GA.

6.4.5 Comparison of global knowledge and present knowledge

An interesting experiment is to study the effect of losing global knowledge in a dynamic setting (DFJSP) on various objectives. We compare the learning approach with global knowledge where the perturbations and job releases are known in advance to the RollingTime approach where only the information up to the present time is known.

Name	n	m	LB	LA	GA	dev (%)	M.G.	dev (%)
Mk01	10	6	36	40	40	0,00	40	0,00
Mk02	10	6	24	27	26	3,85	26	3,85
Mk03	15	8	204	204	204	0,00	204	0,00
Mk04	15	8	48	62	60	3,33	60	3,33
Mk05	15	4	168	175	173	1,16	173	1,16
Mk06	10	15	33	63	63	0,00	58	8,62
Mk07	20	5	133	146	139	5,04	144	1,39
Mk08	20	10	523	523	523	0,00	523	0,00
Mk09	20	10	299	307	311	-1,29	307	0,00
Mk10	20	15	165	224	212	5,66	198	13,13

Table 6.2: Comparison to other approaches on the Brandimarte dataset. LB denotes the best known lower bound, LA are the result of our learning approach, GA are the result of Pezzella et al. [2008], and M.G. are the results of Mastrolilli and Gambardella [1996].

Figure 6.9 shows the difference between both approaches for the makespan objective, on the complete DFJSP dataset with upper bound division factor $z = 5$. The figure is normalized such that the global approach is always 100%. Both approaches use a learning rate $\alpha_{reward} = 0.2$. The global approach performs $10,000$ learning iterations, while the RollingTime approach performs 100 (re-)learning iterations at each rescheduling step. The average total number number of (re-)learning iterations is about $10,000$. As expected, the RollingTime approach with only present knowledge performs worse than an approach with global knowledge. The average difference between both approaches considering the makespan objective, is about $5,23\%$.

Figure 6.10 shows the influence of the z parameter on the average makespan for the global and present knowledge (RollingTime) approaches. A higher z leads to more perturbations and thus to increased makespan values. The lack of global knowledge induces a constant average makespan increase of about 5% for all z values, indicating that the impact on the makespan of losing global knowledge is not influenced by the perturbation frequency. Similar results are observed for all considered objectives. For example the average tardiness $\overline{T}$ objective as shown in Figure 6.11.

A radarplot with multiple objectives is shown in Figure 6.12. The figure compares the global approach with the RollingTime approach. Each objective axis is scaled such that the largest value corresponds to 100%. The RollingTime approach has a significant performance degradation for each objective, which is a result of losing global knowledge.

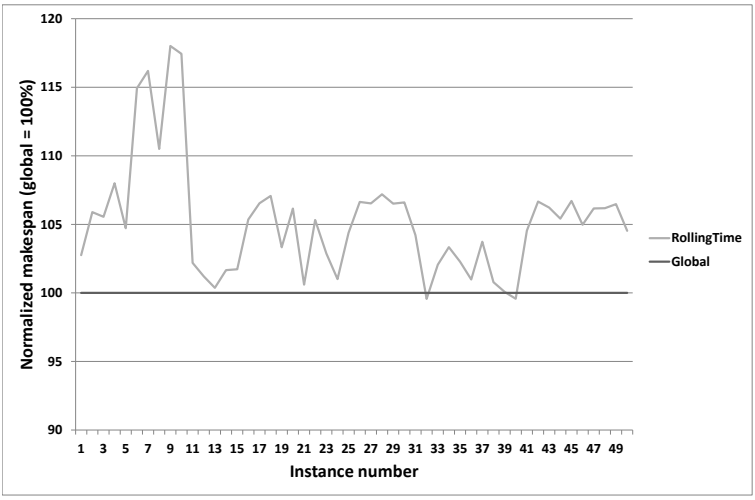


Figure 6.9: Comparison of the average makespan objective between the global and present knowledge (RollingTime) approaches on all 50 instances of the DFJSP dataset with $z = 5$.

6.5 Conclusion

A learning approach to the flexible job shop scheduling problem was proposed. The approach uses simple reinforcement learning devices, called learning automata for learning good sequences of operations, and simultaneously learning good execution modes. The learning approach shows to be competitive with other well performing methods from the literature including a recently proposed genetic algorithm. Similar to the methods in the previous chapter, the methods in this chapter can be seen as a learning heuristic which is applied directly to the problem, and thus operating at the direct level according to Figure 2.4.

A dynamic version of the flexible job shop scheduling problem with release dates, due dates and perturbations was introduced. New benchmark instances were generated, and an adapted version of the learning approach (RollingTime) were proposed. Experiments on these instances show the effect of the lack of global knowledge in a dynamic setting. This dynamic problem extension was a step towards real-world cases. In the next chapter we will discuss the application of these methods to an actual real-world production scheduling case in the food

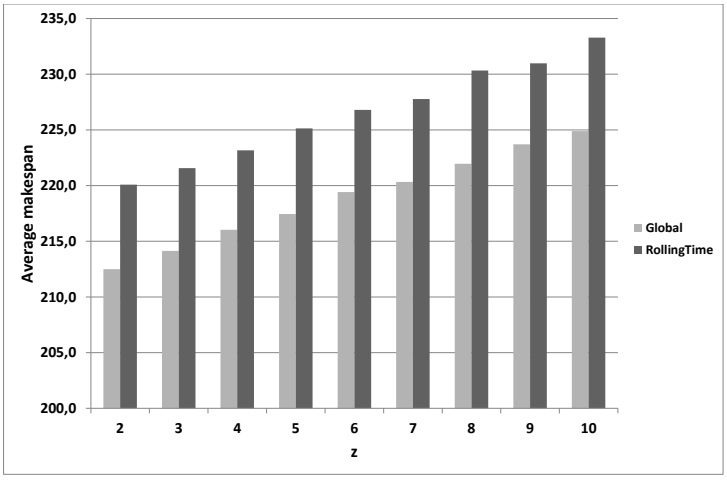


Figure 6.10: Influence of the z parameter on the average makespan objective for the global and present knowledge (RollingTime) approaches, considering all instances.

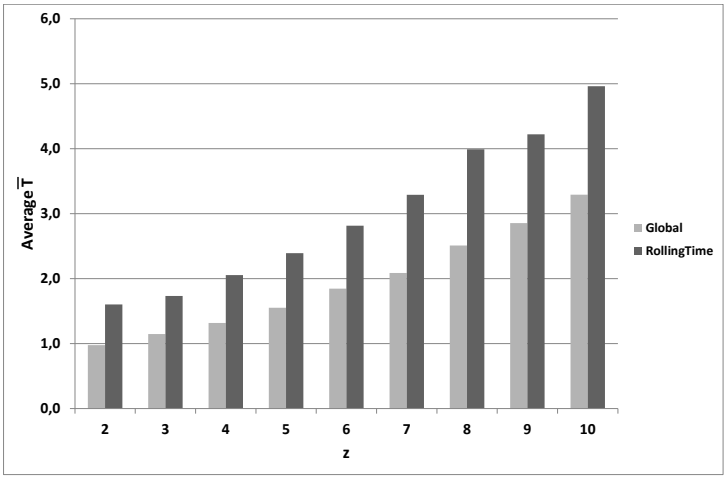


Figure 6.11: Influence of the z parameter on the average $\overline{T}$ objective for the global and present knowledge (RollingTime) approaches, considering all instances.

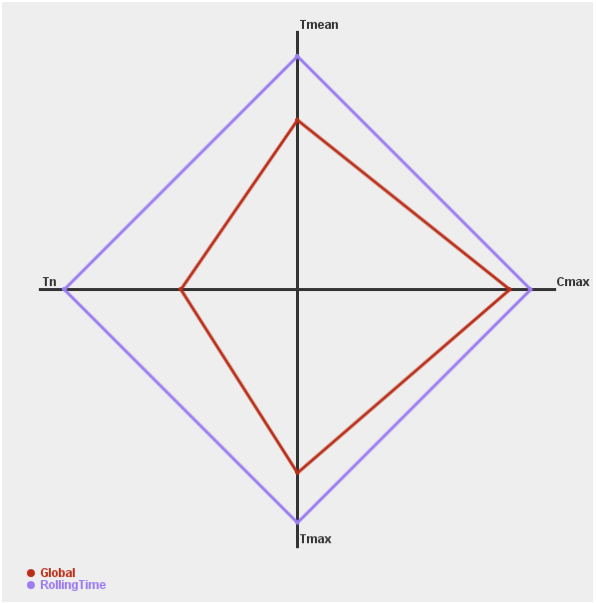


Figure 6.12: Radarplot with multiple objectives comparing the global and present knowledge (RollingTime) approaches. Each objective is scaled according to the largest value (100%).

industry.

Chapter 7

Beyond the benchmarks: application to a real-world scheduling problem

7.1 Introduction

Manufacturing Execution Systems (MES) incorporating a scheduling component have the potential of contributing considerably to the efficiency of a factory. A manufacturing execution system is a control system for managing and monitoring work-in-process on a factory floor. MES systems are being increasingly integrated with enterprise resource planning (ERP), hereby connecting the control and management levels automatically and continuously. The overall aim is that the costumer requests are processed fast and that available resources are being used most efficiently. At the same time, waiting times for materials waste, set-up times and preparation times are reduced while simultaneously managing product variation and (product) exceptions on the plant floor [Kletti, 2007].

MES systems are perfectly suited for supporting real-world production scheduling for the so-called process industry, where manufacturing is subject to formulas and manufacturing recipes. The primary production processes are either continuous, or occur on an indistinguishable batch of materials. Examples of the process industries include food, beverages, chemicals, pharmaceuticals, petroleum, paints and coatings etc. In general, the production process can be divided in two stages. First raw materials are processed into (intermediate)

products. Next, finished food products are packed. This chapter concentrates on food processing companies. In these environments, the scheduler has to deal with a combination of discrete, continuous and batch processes, and is typically complicated by particular characteristics. For instance, the physical routes between workcenters in the plant layout, the high diversity of products (different tastes, different packaging, custom labels, ...), complex product recipes, and many more. More information about the food industry and its characteristics can be found in Akkerman and van Donk [2009].

Although production scheduling is a widely studied topic [Hermann, 2006], it has received little attention in real (food) processing industries [Akkerman and van Donk, 2009]. Nevertheless, a number of related papers can be found in the academic literature. Christou et al. [2007] and Ferreira et al. [2009] developed a practical production planning approach for a real application in the beverage industry. Blanc et al. [2008] described a holonic approach for an MES, and illustrated this approach on a real industrial application. Bilgen and Günther [2010] introduced a so called 'block planning approach' for production scheduling in the fast moving consumer industry. Some recent real life industrial case studies using mathematical programming techniques can be found. Entrup et al. [2005] introduce three MILP model formulations and apply it for a weekly production planning on an industrial case study of yogurt production. A combination of a discrete and continuous representation of time is used. Kopanos et al. [2010] propose a MILP model for the simultaneous lot-sizing and production scheduling problem in a multiproduct yogurt production line of a diary plant in Greece. Kapanos et al. [2011] show a MIP for the production planning and scheduling of parallel (single-stage) continuous processes in the presence of sequence-dependent switchover times and costs for product families, and sequence-independent switchovers for products belonging to the same family.

In joint cooperation with MESware nv[1], we developed an integrated approach to refine the scheduling kernel inside their products. The overall challenge the customers of the company are faced with is to cope with an increasing demand for flexible, customized and faster production. At the same time, the company also aims at serving larger customers. At this time the company and its clients do not require optimal schedules. The schedules have to be generated fast enough and must be of sufficient quality compared to manual practice. Explicit requirements for the company are: i) being able to schedule multiple orders simultaneously, ii) being able to map the different processing steps to workcenters present in the plant layout of the customer, while satisfying the relations between these steps (i.e. routing), iii) being able to feed the scheduler

[1]MESware nv is a software company that offers generic MES solutions.
Adress: Beversesteenweg 561 B2, B-8800, Roeselare, Belgium.
Website: `http://www.mesware.be`

with up-to-date information on the production process, iv) being able to include stock capacities and sequence depending set-up times, v) being able to work with customer specific objectives and finally vi) being able to integrate with personnel scheduling.

The overall solution approach developed in this work consists of a generic scheduling framework in which industry specific problems can be included by means of preprocessing or decision modules. A make-to-order setting is assumed, where products are only produced when there is an order for it. Techniques from local search are used to optimize the decisions taken in these modules. Secondly, we use machine learning techniques to improve the background knowledge on key performance indicators (KPIs). Processing times, set-up times, breakdowns, etc were brought in. We then show how the modular framework is used to implement the first three requirements, and how it can be used to further address the remaining ones. Currently the scheduler operates in a fully deterministic setting, but there are plans to make it more dynamic.

The main contributions of this chapter include academic and practical components. Academically we show that a combination of Artificial Intelligence and Operational Research methods allows dealing generically with complex problem definitions provided by industry. We point at a set of practical issues that are underestimated when considering applications of an academic approach. Examples include how to grasp accurate data (expert knowledge); how to get accurate information about the execution; how to make a scalable generic scheduler for all sorts of industrial processes. These issues have been addressed in the framework.

The remainder of this chapter is organized as follows. In the next section the generic scheduling framework is described at an abstract level. Each different module is discussed afterwards in a separate section. The application of our approach to real-life data from the food processing industry is illustrated. We conclude with a discussion and, finally, indicate some directions for further research.

This chapter is a slightly adapted version of *Wauters, T., Verbeeck, K., Verstraete, P., Vanden Berghe, G., De Causmaecker, P. (2012). Real-world production scheduling for the food industry: an integrated approach. Engineering Applications of Artificial Intelligence, 25 (2), 222-228.*

7.2 A high level overview

Real world scheduling is a highly complex challenge with many constraints. It seemed therefore a good idea to decompose the scheduling task into several steps as shown in Figure 7.1. We can distinguish the following steps: a preprocessing step, a decision step, a translation step, a schedule generation step, and finally an optimization step.

The *preprocessing step* checks the correctness of the incoming orders, and makes some preparations, e.g. it can calculate all the possible assignments of operations to workcenters for manufacturing the demanded products. This is a routing problem that is discussed in Section 7.3. Choosing an actual route is left for the decision step.

The *decision step*, which is a part of the optimization step, assigns values to the decision variables. In the case study here discussed, the decision variables are the possible routes, and the sequence or order in which the orders will be scheduled. These decisions can be made by an intelligent component such as a reinforcement learning agent. This step is discussed in Section 7.4.

The *translation step* uses the background information, the incoming orders, and the decision variables to formulate a scheduling problem. We use a general scheduling problem, namely the resource-constrained project scheduling problem with generalized precedence relations (RCPSP/GPR) [Franck et al., 2001]. It encompasses all the resource and time constraints. Each order is translated into one project scheduling problem. Multiple orders will then compose a multi-project scheduling problem.

The *schedule generation step* solves the scheduling problem formulated in the previous step, which is a multi-project RCPSP/GPR. The multiple scheduling problems are solved one by one by the given sequence of orders (chosen in the decision step). The individual scheduling problems can be solved by any applicable method from the literature [Bartusch et al., 1988, De Reyck and Herroelen, 1998, Schwindt, 1998, Franck et al., 2001]. For the RCPSP/GPR, we use a scheduling method based on priority rules, together with a serial schedule generation scheme with unscheduling rules. Section 7.5 describes the translation step, together with the schedule generation step. The schedule generation step provides some feedback on the quality of the solution that was generated. This information is fed back to the optimization step, so that better local decisions can be made and better solutions (schedules) can be found iteratively.

Finally, the *optimization step* searches for good quality solutions in a guided way by receiving information from the previous steps in a feedback loop. Many optimization methods can serve as guiding mechanism. For example, steepest

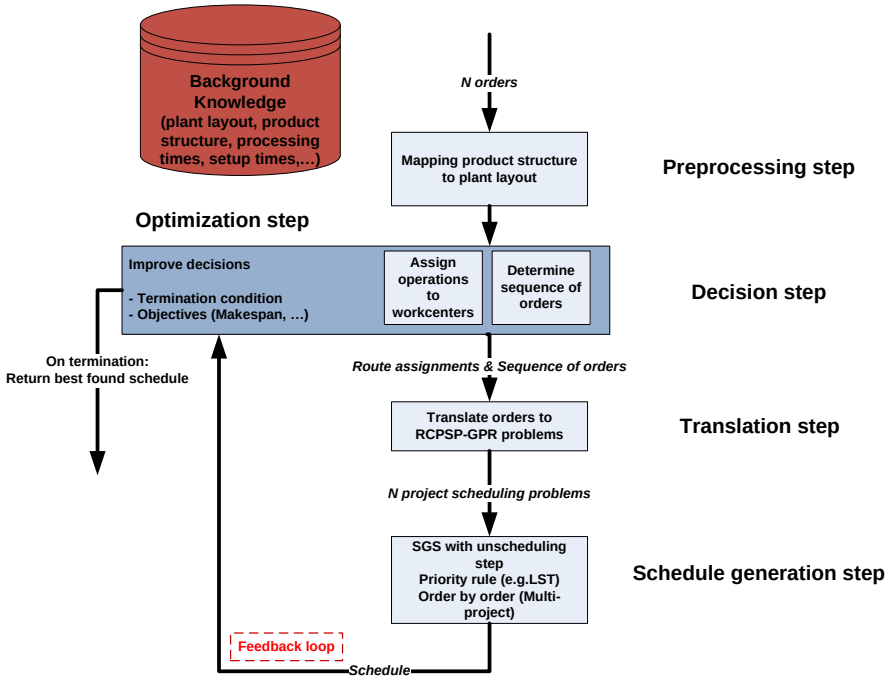


Figure 7.1: A high level overview of the integrated approach to the food processing case

descent, simulated annealing [Kirkpatrick et al., 1983a], tabu search [Glover and Laguna, 1997] and variable neighborhood search [Mladenovic and Hansen, 1997b] are appropriate. This step also holds the termination condition and the set of objectives (e.g. makespan, weighted tardiness, ...). In addition, reinforcement learning enhanced heuristic search methods, as discussed in previous chapters of this thesis, are perfectly suited for this task. Note, as we discuss later, that an internal optimization algorithm is also possible inside the schedule generation step, which itself can be determined by a decision variable.

In all the steps we assume the presence of *background knowledge*. It holds static and dynamic information about the plant layout, process relations, product structure, processing times, setup times, machine breakdown or maintenance, personnel, working hours and equipment. The accuracy of this information strongly influences the applicability of the scheduler. Section 7.6 discusses the latter topic.

7.3 Routing

7.3.1 Routing problem description

In order to manufacture the demanded products, operations have to be assigned to workcenters. This leads to the following routing problem. Consider a plant layout containing all the connections in the plant, i.e. connections between workcenters, buffers and storage zones. The connections contain information about process relations and the possible time lags in between. Two types of connections are present, i.e. end-end and end-start connections. An end-end connection between two connected workcenters means that the operations on these two workcenters must end at the same time (excluding time lags). An end-start connection means that the second operation can not start before its predecessor operation finishes. The product structure, which is the recipe of the product or the bill of materials, is also given. This product structure contains a number of process steps and the order relation in between. Figure 7.2 shows an example of such a product structure with its process steps. The example shows that processes can have multiple input products in a certain proportion. The default form of the product structure is a tree and the plant layout is a graph. Both end-end and end-start connected workcenters are allowed. In fact, the problem is to find one or all mappings from process steps to workcenters, with the restriction that the workcenters should be able to perform the required processes.

By decomposing the product structure into linear substructures, this routing problem can be solved recursively. We start at the root process of the tree defined by the product structure. We move along the tree towards the leafs and add the vertices to the substructure. When arriving in a vertex q with more than one child vertex, we end the current substructure with vertex q and start a new substructure (recursive call) for each subtree starting from the child vertices of q. Vertex q is added as the first vertex of the new substructure, followed by its child vertex. This method results in the following set of substructures for the example given in Figure 7.2: $\{A = \{1, 2, 3, 4\}, B = \{4, 5, 7\}, C = \{4, 6, 8\}\}$. These substructures are also connected, e.g. A is the parent substructure of substructures B and C.

Once the set of linear substructures is generated, we search for all feasible routes through the plant layout for each substructure. A feasible route is a mapping from process steps to workcenters, respecting the following two conditions. The workcenters must be able to perform the required process, and two consecutive processes must be mapped to connected workcenters. We have two kinds of connections between workcenters: direct connections and connections through

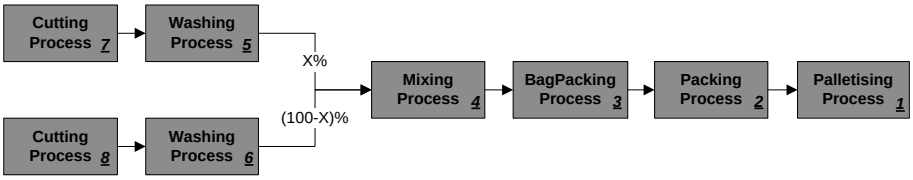


Figure 7.2: Example of the process steps in a product structure

storages and buffers (indirect). When searching for feasible routes, we prefer the direct connections. Indirect connections are considered when no direct connections are available. A depth-first search is used to find all the feasible routes for each linear substructure. When only direct connections are present, we have a worst case time complexity of $O(b^d)$, with d the number of process steps, and b the average number of feasible connections for each workcenter (which is in practice not very high). When the runtime of the depth-first search would become problematic or when costs are introduced on the connections between workcenters, then more informed search methods like A* or IDA* are recommended.

The last step in this routing method combines the feasible routes from the substructures into a route for the whole product structure. This is done by selecting a feasible route for each substructure such that the overlapping vertices (process steps) of the substructures are mapped to the same workcenter. For end-end connected production steps, an extra requirement is necessary. Sibling substructures with end-end parts at the start of their routes must be mapped to different workcenters.

7.3.2 Special routing case

We distinguish a special case of the routing problem, in which only end-end connections are allowed. We define the product structure as a directed graph $G(V_G, E_G)$ with labeled vertices. We further have the plant layout, which is also a directed graph $H(V_H, E_H)$ with a list of labels on the vertices. The labels in G are the process steps, while the labels in H are the process steps supported by the workcenters. Only end-end connected workcenters are allowed. The problem is to find a mapping f from the vertices of G to a subset of the vertices of H, respecting the following constraints. If two vertices $v1, v2 \in V_G$ are connected by a direct edge in G, then the corresponding vertices $f(v1), f(v2) \in V_H$ must also be connected by a direct edge in H. A second constraint is that the labels of mapped vertices must have a common label, $\forall v \in V_G \ : \ \exists l_i(v) \in$

$L(v), l_j(f(v)) \in L(f(v)) \;:\; l_i(v) = l_j(f(v))$ with $L(v)$ the set of labels of vertex v and with $l_i(v)$ label i of vertex v. The problem described here is a variant of the induced subgraph isomorphism problem, with directed edges and labels on the vertices. Ullmann [1976] proposed a backtracking algorithm for the subgraph isomorphism problem. It is one of the most commonly used algorithms for exact graph matching. It has a worst case time complexity of $O\left(N!\ N^2\right)$. Better (sub)graph isomorphism algorithms exist for matching larger graphs. More recently Cordella et al. [2004] described the VF2 algorithm with a worst case time complexity of $O\left(N!\ N\right)$. Because of its simplicity, we adapted the algorithm of Ullmann [1976], to handle our specific problem with directed edges and vertex labels that have to match. Experiments have shown that the backtracking algorithm is sufficiently fast for the real world routing problem in this study (less than $10ms$ for the largest test case).

7.4 Decision and optimization

The decision step should assign values to decision variables. An initial schedule is constructed through subsequent steps (translation, schedule generation). The schedule is fed back to the optimization step. The initial decision variable assignments can be chosen randomly or by a simple heuristic procedure.

The decision variables in the current approach include: a route and a sequence for processing the orders. A route is the mapping from process steps to workcenters. The list of possible routes was determined in the preprocessing step. Many other decision variables could be added, for example, the algorithm or priority rule that will be used to solve the translated scheduling problem (see Section 7.5). This fits perfectly into the idea behind hyperheuristics [Burke et al., 2003b]. Hyperheuristics are search methods operating on a higher level. Unlike metaheuristics, which are searching in the space of solutions, hyperheuristics are searching in the space of low-level heuristics or metaheuristics. In other words they are searching for the best method to find good solutions quickly at each part of the optimization process. We will further discuss hyperheuristics in Section 7.8.

The heuristic used for generating an initial schedule works as follows. For mapping the process steps to workcenters, we choose the routes with the largest free capacity and we maximally disperse routes for different orders. Dispersion is arranged by first selecting a random route, and subsequently selecting routes that are maximally different from the previous routes (ties are broken randomly). A random sequence of orders is generated. If tardiness needs to be considered, sorting the orders according to the 'earliest due date'

first rule is more appropriate. The decision step is actually a sub step of the optimization step. We have chosen to guide the optimization by a local search algorithm, which can be any metaheuristic, hyperheuristic or RL enhanced heuristic (as was explained in the preceding chapters of this thesis). For each decision variable a neighborhood is defined. These neighborhoods contain a set of solutions, i.e. one for each alternative value of the decision variable. Some of the solutions in the neighborhood can violate constraints, these are infeasible and are not considered further. The neighbourhoods can be intelligently selected at each step by the methods proposed in Chapter 4. The latter would lead to more adaptive methods that are less sensitive to specificities of particular cases and predefined settings and thus more generally applicable. For example, in some situations it can be that changing the order sequence has no effect on the schedule, and that it is better to not use a neighbourhood changing the sequence. This would be the case when the orders are selected to run on distinct workcenters. The methods from Chapter 4 would reduce the probability of selecting such a 'bad' move, and leaving more time for more effective moves.

An objective value is determined by information of the constructed schedules. This could be a time-dependent or a cost-dependent objective. Examples of time-dependent objectives are: lateness, tardiness, total completion time or total makespan. Other objectives are the total cost or a resource leveling objective. The makespan objective is used in this research, i.e. the difference between the start time of the earliest operation and the end time of the latest operation. The choice of the objective does not influence the general structure of our approach. The optimization stops when a termination condition is met. Possible termination conditions include a certain number of iterations without improvement, the computation time or a quality level. At termination, the system returns the best found schedule.

7.5 Translation to RCPSP/GPR

The real world constraints, variables and objectives are translated to a scheduling problem. The resource-constrained project scheduling problem with generalized precedence relations (RCPSP/GPR) is a very general scheduling problem, in which resource and time constraints can easily be formulated. An example of these constraints are the time-lags between consecutive activities, which are explained later in the present text. Each order is translated into a RCPSP/GPR. Multiple orders will then compose a multi-project scheduling problem with shared resources. First the RCPSP/GPR is described, followed by the conversion of the real world constraints to this theoretical scheduling problem.

The RCPSP/GPR is a type of resource-constrained project scheduling problem with minimal and maximal time lags, also called generalized precedence relations or temporal constraints [Franck et al., 2001]. Neumann and Schwindt [1997] describe make-to-order production scheduling as one of the most obvious application fields of such a problem. We use a similar notation as the one described by Franck et al. [2001]. The RCPSP/GPR consists of a project with n real activities $(1, \ldots, n)$. Two dummy activities (0 and $n+1$) are added to represent the start and the completion of the project. Let p_i be the processing time or duration and S_i the start time of activity i. Neither of the time values are limited to integer values, but can be any moment or timespan, since we are working with real world data. In fact, we use a continuous-time implementation. S_0 is set to the earliest point in time from which scheduling activities can start, and S_{n+1} is equal to the project duration or makespan.

Minimum and maximum time lags are defined between the activities of the project. $d_{ij}^{\min} > 0$ denotes a minimum time lag between activities i and j $(i \neq j)$. Their start times are related as $S_j - S_i \geq d_{ij}^{\min}$. For a maximum time lag $d_{ij}^{\max} > 0$ between activities i and j, the relation between start times is defined as $S_j - S_i \leq d_{ij}^{\max}$. If all the time lags are respected in a schedule, we say that the schedule is time-feasible.

Activities require resources to be executed. A set R of renewable resources is available. Renewable resources are resources that can be re-used when a previous activity is finished. These resources k have a capacity $R_k > 0$. A resource capacity r_{ik} is required for carrying out activity i on resource k, where $0 \leq r_{ik} \leq R_k$. If at each time t, the resource capacities are respected in a schedule, we say that the schedule is resource-feasible.

If a schedule is both time- and resource-feasible, then it is called feasible. The question whether or not a feasible schedule exists is an NP-complete decision problem [Bartusch et al., 1988]. Finding a feasible solution that minimizes the project duration is strongly NP-hard. Therefore, we have chosen to use heuristic methods for solving this problem.

Table 7.1 shows how concepts of the real world problem map to a theoretical scheduling problem (RCPSP/GPR). We translate each order to a project. Process steps are translated to activities. Workstations and equipment are translated into renewable resources. The resources have a capacity of $R_k = 1$ in most cases, but larger capacities are possible (e.g. a palletising workcenter that can handle input from multiple lines). Routing relations (i.e. synchronization of start or finish times), time lags and product structure are translated into generalized precedence relations. Mapping the process steps to workcenters was determined in the decision step as discussed in the previous section. The processing times are set.

Real world problem	Theoretical project scheduling problem
Order	Project
Process step	Activity
Workcenter	Renewable resource
Equipment	Renewable resource
Product structure	Generalized precedence relations
Routing relations	Generalized precedence relations
Time lags	Generalized precedence relations

Table 7.1: Translation to a theoretical project scheduling problem.

In what follows, we describe an example of the translation to generalized precedence constraints. Two consecutive process steps PS_1 and PS_2 are assigned to be executed on workcenters WC_1 and WC_2 respectively. If PS_1 is translated to activity i and PS_2 to activity j, and WC_1 and WC_2 have a time lag of TL, then we define the following minimum and maximum time lags, depending on the type of connection between WC_1 and WC_2.

- End-start connection: $d_{ij}^{\min} = p_i + TL$
- End-end connection: $d_{ij}^{\min} = TL$ and $d_{ij}^{\max} = TL$

Once translated, i.e. into an RCPSP/GPR problem, the solution is generated sequentially project by project. The sequence was determined in the decision step. The RCPSP/GPR problems can be solved with techniques from the literature. Several branch-and-bound methods have been proposed for this scheduling problem [Bartusch et al., 1988, De Reyck and Herroelen, 1998, Schwindt, 1998]. Franck et al. [2001] introduce truncated branch and bound methods and metaheuristics such as tabu-search and genetic algorithms. Different priority rules have been presented in the literature [Franck and Neumann, 1998, Kolisch, 1996c] and can be used: smallest 'latest start time' first (LST rule), 'minimum slack time' first (MST rule), 'most total successors first' (MTS rule), ... Priority rules have been shown to be fast, and they produce solutions of acceptable quality (especially for large problems). Priority rule methods are used together with a serial schedule generation scheme to successively schedule the activities of a project. From Franck et al. [2001], we selected the priority rule method with LST rule and applied it to the investigated problem.

The methods described in Chapter 5, where a network of learning automata is employed for solving two general project scheduling problems, can also be applied to this RCPSP/GPR. Especially the method applied to the multi-project scheduling problem (DRCMPSP) is very appropriate, since we have multiple projects here (one project for each order).

K-Nearest Neighbor (k=10)	Model Tree	Neural Network
50,9%	50,4%	40,4%

Table 7.2: Comparison of regression techniques for predicting machine speed values for a production history of 1 year. The values represent the percentage RMSE improvement over a naive predictor.

7.6 Feeding the scheduler with accurate background knowledge

In various steps of the approach, different background knowledge is required. The accuracy of background knowledge is of highest importance for obtaining a good scheduling method. For example, adequate processing times, i.e. the time needed to process a certain quantity of a certain product on a certain workcenter. Mostly these values are estimated using theoretical machine speeds. This is reasonable as a first estimate, but in practice these values are not very accurate. We use the production history to determine more accurate machine speeds. We will shortly describe this method.

We developed a method based on regression techniques, for adjusting the machine speed values by using the production history. Experiments have been carried out on a confidential dataset provided by a costumer of MESware. The dataset contains logs of executed operations over a period of one year. First of all, we translated the information history of executed operations to a set of instances. Each executed operation corresponds to one instance with multiple parameter values (workcenter, product type, quantity, ...) and one continuous class value (i.e. machine speed). Afterwards, we fed these instances to a regression technique in order to build a model from the data. The model was then used to adjust the machine speed values in the background knowledge. We experimented with multiple regression techniques: neural networks, regression trees, model trees [Wang and Witten, 1996] and K-nearest neighbor methods [Atkeson et al., 1997]. Table 7.2 shows the results of three regression techniques for a production history of 1 year. Model trees and K-nearest neighbor methods performed best in terms of root mean squared error (RMSE) performance. We compared the methods to a naive predictor. A naive predictor would always predict the average observed machine speed. The K-nearest neighbor based predictor method, appeared to meet the quality expectations (50% better than a naive predictor) on the real test cases, and has by now been promptly integrated in the scheduler at the production site. A screenshot of the utility application using the k-nearest neighbour algorithm is added in the Appendix A (Figure A.1).

7.7 Illustrative example

We illustrate the approach by means of an average scale real world example. We have obtained data from a plant layout with about 20 workcenters, 3 parallel packing lines, and a lot of storage zones and buffers. Both end-end and end-start connections are present. Almost all workcenters are different in speed and capability. A number of heterogeneous sets of production orders with different sizes (from 10 to 100 in increments of 10, and from 100 to 1000 in increments of 100 production orders) are prepared, using a set of 5 different product types given by MESware (all tree structures). The orders consist on average of 10 process steps (including production and packaging). All these orders contain one or more mixing steps where multiple intermediate products are required as input.

Figure 7.3 shows the average calculation times in milliseconds for generating an initial feasible schedule for a set of heterogeneous orders of a certain size. The schedule generation step applies LST as a priority rule. The tests are performed on an Intel core 2 Duo 2.8Ghz processor, and the program is written in the c♯ programming language. All background information is loaded in advance, database connection times are excluded. We can see that even for the largest set of 1000 production orders the algorithm needs less than a second to find a first feasible schedule (initial solution). This includes the preprocessing (routing), decision and schedule generation step. At this moment a typical customer of MESware is scheduling less than 100 orders a week. The tested approach needs less than 100 milliseconds for scheduling this number of orders, and thus leaving enough potential for optimization and for scaling to larger clients. The performance scales linearly with the number of orders because the orders are scheduled one after the other. The number of process steps per order will have a larger impact on the system, and influences the runtime of both the routing and schedule generation steps.

For an improved schedule, i.e. a schedule with lower makespan than the initial one, the calculation time depends on several factors. One such factor is the termination condition. We use the following termination condition: stop when the number of consecutive loops (iterations) without significant ($> x_{min}\%$) improvement exceeds a limit L_{max}. Other factors that influence the calculation time are the intial solution, the number of orders and the number of alternative routes. The local search method used is a simple steepest descent which optimizes the total makespan objective (i.e. minimize the makespan over all projects).

Generating an optimized schedule for a set of 30 orders takes approximately 10 seconds, when we use $x_{min} = 0.01\%$ and $L_{max} = 3$ as the termination condition.

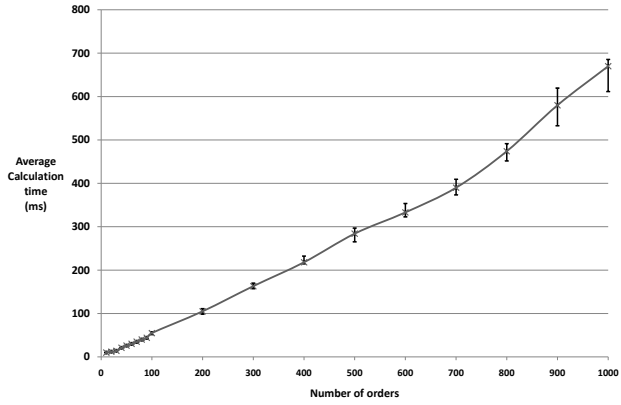


Figure 7.3: A simple scaling analysis: average calculation time (with error bars) in function of the numbers of production orders.

The progress of this optimization in terms of total makespan is presented in Figure 7.4. The figure shows that the optimization further improves on the initial solution. In our tests we noticed that many improvements are realized by changing the routes. The sequence of orders has less influence on the schedule makespan when all the orders are similar. Note that we could easily replace the steepest descent algorithm by a more intelligent optimization algorithm (tabu-search, simulated annealing, genetic algorithm, RL enhanced heuristic search, ...), which would enable further improvements, at the cost of some computation time.

7.8 Conclusion

In this chapter we presented an integrated approach for production scheduling, and demonstrated its applicability to the food processing industry. In these environments, scheduling has to deal with a combination of discrete, continuous and batch processes, and is typically complicated by particular characteristics. For instance, the physical routes between workcenters in the plant layout, the high diversity of products (different tastes, different packaging, custom labels, ...), complex product recipes, and many more. We decomposed the real world scheduling task into several sub tasks or steps that can be performed by different

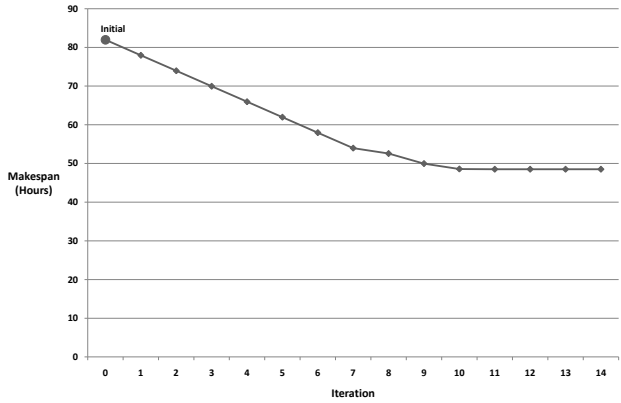


Figure 7.4: Optimization progress starting from an initial schedule for a set of 30 heterogeneous orders

components. This allowed us to develop methods for each subtask separately as well as to easily extend the approach with new components. We initially modeled the scheduling problem as a very general resource-constrained project scheduling problem with generalized precedence relations. This enabled us to model a variety of industry related constraints in separate modules. A popular priority rule based method is used to solve the theoretical scheduling problem. The priority rule method could be replaced by any solution technique, including the LA based methods described in previous chapters of this thesis. Especially the techniques for general project scheduling problems (Chapter 5) are well suited.

The presented approach allows scheduling multiple production orders at once, which was actually an important requirement of the industrial partner. Dealing with any possible plant layout was another essential requirement. Therefore, a routing problem is solved in a separate module, i.e. process steps are mapped onto workcenters. By putting this in a separate module, deciding on a specific route has become an explicit decision variable. By applying local search, new and better decisions can be explored for this module. As Figure 7.4 shows, deciding on a route is a well chosen decision variable since considerable improvements of the schedule can be reached by changing this decision variable.

Another decision variable determines an explicit sequence in which the customer

orders should be scheduled. This module should be extended further, so as to address sequence dependent set-up times. Other modules can be added to decide on, for instance, whether or when to include stock capacity into the production. This module could be designed as a reinforcement learning module [Sutton and Barto, 1998b] that learns in which situations stock should be included or not. In the special setting of food industries, keeping products in stock or in refrigerators is subject to timing constraints. This kind of constraints are perfectly expressible using the generalized precedence relations as well.

Many other decision variables could be taken into account. One could even make a decision variable to represent the choice of the actual theoretical scheduling algorithm used. This idea fits into a trend in search methodologies, called hyperheuristics [Burke et al., 2003b]. The idea behind hyperheuristics is to build generic search methods independent from problem or domain specific characteristics. As such, a hyperheuristic framework is concerned with learning at an abstract level which low level algorithm is most suited for a problem at hand. It would be interesting to investigate whether hyperheuristics could improve the generic applicability of our approach.

One of the remaining requirements is to feed the scheduler with up-to-date information on the production process. This was addressed by replacing the default machine speeds by better estimates. These estimates result from a regression algorithm that uses historical information. For now, only estimations of the processing times are fed back to the system, but this can straightforwardly be extended to other KPIs.

The project discussed in this chapter can be considered as a good exercise for bridging the gap between real world cases and innovative research. When working with a company, strict timing and robustness requirements are set. One can not just implement the state-of-the-art methods without careful testing these additional requirements. Real world implementation and research differ a lot from this point of view. Our approach is fast enough for practical use and scales well to a large number of production orders. It is currently integrated into a commercial software package that is running at several production sites. Many opportunities for improvement and the inclusion of learning techniques, as the one described in this thesis, are left to investigate in further research. However, the necessary conditions for realizing these opportunities have already been foreseen in the developed framework, making it easy to plug-in new learning components.

In the future we will extend our approach by including extra functionality and additional real world constraints. Of particular interest will be a complete integration of stock and buffers, the use of sequence dependent set-up times, customer specific objectives and the integration with personnel scheduling.

An additional challenge to be considered is grouping and splitting orders on workcenters or production lines.

Chapter 8

Conclusion

Methods for solving combinatorial optimization problems in practice, and more specifically real-world scheduling problems, offer many opportunities for including learning approaches. First of all the data is continuously changing, requiring the methods to be adaptive enough. Second, the applied heuristic search methods take random decisions, e.g. the selection of the next heuristic to apply. Third, in practice the data is often based on default values, whereas the real values can be significantly different. In order to allow a larger applicability of heuristic search methods in practice, more general methods need to be created. In the present thesis reinforcement learning enhanced heuristic search methods for combinatorial optimization were studied in order to develop intelligent and more adaptive heuristic methods capable of addressing these issues. In this chapter we summarize the main results and contributions.

8.1 Contributions

The goal of this dissertation was successfully realized by applying simple reinforcement learning devices, called learning automata. We showed that these learning automata can contribute to the growing fields of intelligent optimization, scheduling, heuristic search and reinforcement learning in the following ways.

- It is shown how reinforcement learning can be applied to combinatorial optimization problems and hybridized with meta-/hyper-heuristic search. Three different levels of RL inclusion are defined. The direct level, the

meta-heuristic level, and the hyperheuristic level. At each of these levels experimental research was performed and demonstrated on benchmark and real world problems. It was shown that replacing a random component in a heuristic search procedure by one or more learning automata can improve the results.

- A general LA based scheduling methodology was described, which is applicable to many scheduling problems from benchmarks and practice.
- The proposed methods show to be well performing on hard optimization problems, such as scheduling, routing and assignment problems. Further on they show to deliver state-of-the-art results for the multi-mode resource constrained project scheduling problem (MRCPSP), and the decentralized resource-constrained multi-project scheduling problem (DRCMPSP) (Chapter 5). Many new best solutions for the MPSPLIB benchmark[1] have been presented in this dissertation, significantly improving on previous best results (up to 41%).
- The proposed GT-MAS method for the DRCMPSP has been adopted as a reference in the multi-project scheduling literature by [Mao, 2011].
- A new method using LA was introduced for learning good quality permutations online, when no problem specific knowledge is provided (Chapter 3). Combining a decentralized action selection with a dispersion game technique and a common reward signal leads to a fast and memory efficient permutation learning method. Where existing permutation learning techniques require an additional normalization procedure, this is not required by the LA based approach proposed in this dissertation. Experimental results show interesting properties, i.e. improved performance for a particular range of learning rate values.
- It is also shown that LA can be used efficiently for selecting heuristics or neighbourhood functions during meta-/hyper-heuristic search (Chapter 4). Such an LA based heuristic selection method can be added to many existing heuristic search techniques, and as a result be applied to manifold optimization problems. Application to three hard optimization cases shows improved results over random heuristic selection. This approach generated new best results on some instances of the traveling tournament problem[2].
- We show how LA can be successfully applied to the flexible job shop scheduling problem and a more realistic extended problem with release

[1] Best results submitted to `http://www.mpsplib.com`

[2] Best results obtained on March 30, 2009 and submitted to `http://mat.gsia.cmu.edu/TOURN/`

dates and due dates (Chapter 6). Application of LA based rescheduling to a dynamic flexible job shop environment with machine perturbations is also tested and compared to more reactive methods on newly generated instances. Multiple objectives, such as the makespan or the tardiness, have been considered.

- The proposed LA based methods require only a single parameter (the learning rate) as input, avoiding the labour intensive finetuning process of most heuristic search methods.
- Scaling is one of the important factors of heuristic search methods. We showed that a network of learning automata, when applied to scheduling, scales very well towards large instances.
- Most of the proposed methods in this thesis require only a simple $0 - 1$ reward signal and a linear reward-inaction learning automaton update scheme to reach good results in practice. This simple mechanism requires little extra calculation time, thus introducing less overhead when enhancing heuristic search with RL. Further on, these simple reward signals contain no problem specific information, which contributes to the general applicability of the proposed methods.
- Many RL inspired hyper-heuristic methods currently use simple utility value mechanisms, which are not theoretically supported. The proposed LA based techniques in this thesis for use in hyper-heuristics are theoretically substantiated by learning automata theory.
- We developed a successful application to a real-world production scheduling problem in the food industry (Chapter 7). This complex problem was tackled with a combination of heuristic search and techniques from AI. We have shown that this real-world problem offers many opportunities for including the LA based methods proposed in the rest of this thesis.

To summarize, we have shown that the application of reinforcement learning components such as learning automata, can make a search method more adaptive, can improve on static search methods, are easy to apply, and require little extra calculation time. Moreover, they can be either applied directly to the problem or added to an existing heuristic search technique.

8.2 Further Directions

Reinforcement learning offers many opportunities for enhancing heuristic search procedures as was discussed in Chapter 2. Some issues were not addressed during the course of this thesis, therefore we give some directions for further research and for extending the work of this thesis.

New hybridizations with RL
Still many possible inclusions of RL into heuristic search are left to investigate. Think about the integration of RL into the neighbourhood functions of a metaheuristic, to learn when to accept a new solution (even if it is worse), or to learn when to stop the heuristic search.

Transfer learning
Techniques to transfer learned information to other RL tasks were recently introduced in the RL literature. However, transferring learned information to solve similar combinatorial optimization problems has hardly been studied. A possible requirement for successfully applying transfer learning, is the usage of problem or instance independent features.

Simultaneously learning multiple parameters
Simultaneously learning multiple parameters or components at the meta- or hyper-heuristic level offers interesting future perspectives. In Chapter 5 execution modes and the order of activities were learned simultaneously for the MRCPSP. This was at the direct level. All the methods in this thesis at the two higher levels use RL to learn only a single thing (e.g. the heuristic selection probabilities). Learning two or even more parameters simultaneously could bring some extra challenges. These learning mechanisms will possibly have to coordinate, as in multi-agent systems, to reach a good combined performance setting. What if they have conflicting goals (e.g. calculation time vs quality)?

Multi-objective optimization
Some combinatorial optimization problems tackled in this thesis had multiple objective functions. For example, in the decentralized resource-constrained multi-project scheduling problem we studied both the total makespan and average project delay objectives. However, in all cases the reward signal had a single numerical value. But how would we deal with a multi-dimensional reward signal containing information about multiple objectives? How can learning be beneficial in the search for nondominated solutions (or Pareto optima) that offer different trade-offs of the competing criteria? A straightforward solution could be to have separate learning devices for each objective and to combine their decisions with some voting or bidding process. A recent study by Liao

and Wu [2012] takes a first step in this direction by using learning automata for continuous multi-objective optimization.

Parallel execution
When the problems become too large to be solved on a single processor, parallel execution could be a solution. Multi-agent systems are perfectly suited for parallel execution, hence are the methods proposed in this thesis. A possible advantage for this is that the communication between the agents is very low, i.e. only a single reward value is sent to the agents.

Reward signal
All the methods in this thesis employ a basic $0-1$ immediate reward signal, while most RL methods are capable of dealing with delayed rewards. As discussed earlier in this thesis, it is not clear which step of a heuristic search leads to a final high quality solution. Additionally it is possible that a heuristic search has very limited time to perform only a few runs, or even one. If one would only use the quality of the final solutions (delayed reward), this would result in hardly any feedback to learn something valuable.

States
Most methods described in this thesis use no explicit[3] state definition, which is not a problem since they operate online (while searching) and a state would be visited only once. However, the definition of a state space is part of the RL problem. Before using states into these RL enhanced heuristic search methods, one has to answer the following question: what is the state of a heuristic search procedure? The total improvement? The calculation time? Or the number of new best solutions found? Splitting the search into phases could bring a possible solution, but that requires defining phases.

Real-world integration Next to the topics discussed in this thesis, we were challenged with multiple real-world scheduling problems in the automotive, joinery and food industry. In this experience we can state that the gap between scheduling in the literature and scheduling in practice has a long way to go. Often the software companies offer a manual scheduling component or in the best case a simple heuristic solution. The scheduling software developers seem not to be familiar with state-of-the-art scheduling approaches. To bridge the gap, a translation of the real-world problem to a general project scheduling problem seems to be a first step in the right direction, as was shown by a preliminary implementation in Chapter 7 of this thesis.

[3] A networks of LA, used in this thesis, is a way to create multiple states in a more implicit manner.

Appendix A

Machine speed prediction tool

This appendix contains a screenshot (Figure A.1) of the machine learning prediction tool we developed for the real-world production scheduling case (described in Chapter 7). The tool predicts machine speeds based on historical data. The k-nearest neighbour algorithm with $k = 10$ was used.

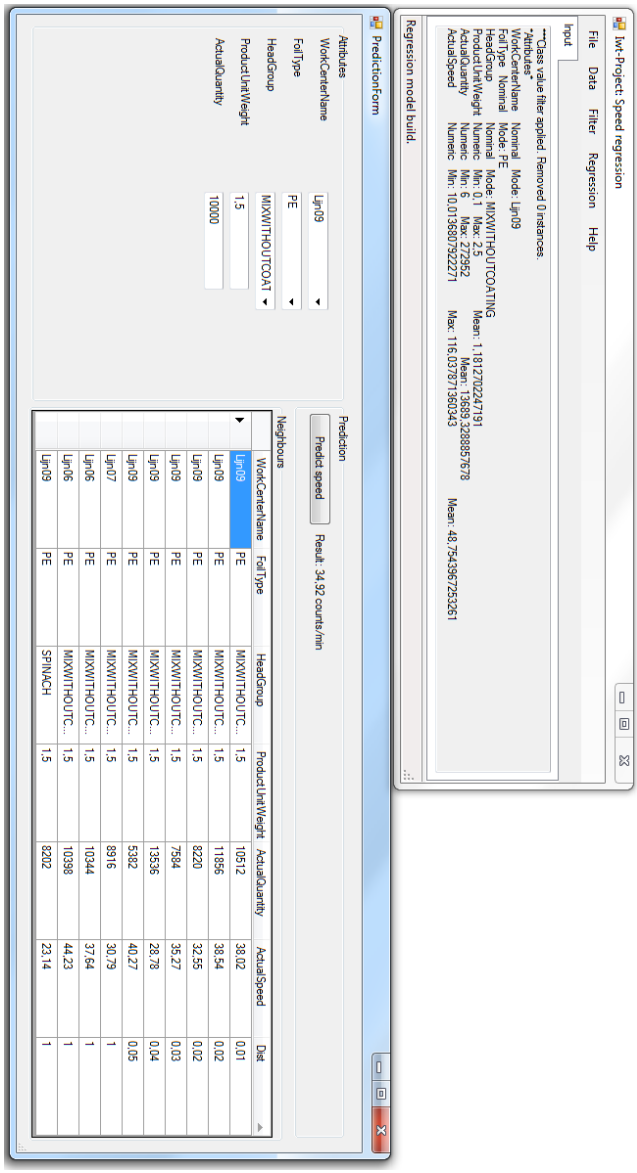


Figure A.1: Screenshot of the machine speed prediction tool.

Bibliography

Google roadef/euro challenge 2011-2012: Machine reassignment, 2012. `http://challenge.roadef.org/2012/files/problem_definition_v1.pdf`, accessed July 2, 2012. pages 54

D. Ackley and M. Littman. Interactions between learning and evolution. In *Artificial Life II*, 1991. pages 15

S. Adhau, M.L. Mittal, and A. Mittal. A multi-agent system for distributed multi-project scheduling: An auction-based negotation approach. *Engineering Applications of Artificial Intelligence*, Available online, 2011. doi: http://dx.doi.org/10.1016/j.engappai.2011.12.003. pages 77, 85

R. Akkerman and D. P. van Donk. Analyzing scheduling in the food-processing industry: structure and tasks. *Cogn Tech Work*, 11:215–226, 2009. pages 118

J. Alcaraz, C. Maroto, and R. Ruiz. Solving the multi-mode resource-constrained project scheduling problem with genetic algorithms. *Journal of the Operational Research Society*, 54:614–626, 2003. pages 64, 76

C. G. Atkeson, A. W. Moore, and S. Schaal. Locally weighted learning. *Artificial intelligence review*, 11:11–73, 1997. pages 128

R. Bai, E. K. Burke, M. Gendreau, G. Kendall, and B. Mccollum. Memory length in hyper-heuristics: An empirical study. In *Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Scheduling (CI-Sched 2007)*, 2007. pages 20, 23

S. Baluja. Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning. Technical Report CMU-CS-94-163, School of Computer Science, Carnegie Mellon University, 1994. pages 28

S. Baluja and S. Davies. Fast probabilistic modeling for combinatorial optimization. In *Proceedings of the fifteenth national/tenth conference*

on Artificial intelligence/Innovative applications of artificial intelligence, AAAI '98/IAAI '98, pages 469–476, Menlo Park, CA, USA, 1998. American Association for Artificial Intelligence. ISBN 0-262-51098-7. URL `http://dl.acm.org/citation.cfm?id=295240.295718`. pages 28

J. W. Barnes and J. B. Chambers. Solving the job shop scheduling problem with tabu search. *IIE Transactions*, 27 (2):257–263, 1995. pages 98, 108

M. Bartusch, R.H. Möhring, and F.J. Radermacher. Scheduling project networks with resource constraints and time windows. *Annals of Operations Research*, 16:201–240, 1988. pages 120, 126, 127

R. Battiti, M. Brunato, and F. Mascia. *Reactive Search and Intelligent Optimization*, volume 45 of *Operations research/Computer Science Interfaces*. Springer Verlag, 2008. pages 1, 3, 7

K. P. Bennett and E. Parrado-Hernández. The interplay of optimization and machine learning research. *J. Mach. Learn. Res.*, 7:1265–1281, December 2006. ISSN 1532-4435. URL `http://dl.acm.org/citation.cfm?id=1248547.1248593`. pages 15

B. Bilgen and H.O. Günther. Integrated production and distribution planning in the fast moving consumer goods industry: a block planning application. *OR Spectrum*, 32:927–955, 2010. pages 118

P. Blanc, I. Demongodin, and P. Castagna. A holonic approach for manufacturing execution system design: An industrial application. *Eng. Appl. of AI*, 21(3):315–330, 2008. pages 118

J. Blazewicz, J. K. Lenstra, and A. H. G. Rinnooy Kan. Scheduling projects subject to resource constraints: Classification and complexity. *Discrete Applied Mathematics*, 5:11–24, 1983a. pages 63

J. Blazewicz, J.K. Lenstra, and A.H.G. Rinnooy Kan. Scheduling subject to resource constraints: classification and complexity. *Discrete Applied Mathematics*, 5:11–24, 1983b. pages 44

C. Blum, M.J.B. Aguilera, A. Roli, and M. Samples, editors. *Hybrid metaheuristics: an Emerging Approach to Optimization*. Springer, 2008. pages 2

K. Bouleimen and H. Lecocq. A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version. *European Journal of Operational Research*, 149:268 – 281, 2003. pages 65, 76

J. Boyan. *Learning Evaluation Functions for Global Optimization*. PhD thesis, Carnegie-Mellon University, 1998. pages 16, 20, 22

J. Boyan, A. W. Moore, and P. Kaelbling. Learning evaluation functions to improve optimization by local search. *Journal of Machine Learning Research*, 1:2000, 2000. pages 16, 20, 22

P. Brandimarte. Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations Research*, 41 (3):157–183, 1993. pages 98, 108, 109

P. Brucker and J. Neyer. Tabu-search for the multi-mode job-shop problem. *OR Spektrum*, 20:21–28, 1998. pages 98

P. Brucker, A. Drexl, R. Mohring, K. Neumann, and E. Pesch. Resource-constrained project scheduling: Notation, classification, models and methods. *European Journal of Operational Research*, 112:3–41, 1999. pages 64, 77

E. Burke, E. Hart, G. Kendall, J. Newall, P. Ross, and S. Schulenburg. Hyper-heuristics: An emerging direction in modern search technology. In *Handbook of Metaheuristics*, pages 457–474. Kluwer Academic Publishers, 2003a. pages 2, 13

E. Burke, G. Kendall, J. Newall, E. Hart, P. Ross, and S. Schulenburg. *Hyper-Heuristics: An Emerging Direction in Modern Search Technology*, chapter 16, pages 457–474. Handbook of Metaheuristics, Springer, 2003b. pages 124, 132

E. K. Burke and Y. Bykov. A late acceptance strategy in hill-climbing for exam timetabling problems (extended abstract). In *Proceedings of PATAT 2008 conference*, Montreal, Canada, August 2008. pages 52

E. K. Burke, G. Kendall, and E. Soubeiga. A tabu-search hyperheuristic for timetabling and rostering. *Journal of Heuristics*, 9:451–470, December 2003c. ISSN 1381-1231. doi: 10.1023/B:HEUR.0000012446.94732.b6. URL `http://portal.acm.org/citation.cfm?id=965845.965864`. pages 16, 20, 23

L. Busoniu, R. Babuska, and B. De Schutter. A Comprehensive Survey of Multiagent Reinforcement Learning. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 38(2):156–172, 2008a. doi: 10.1109/TSMCC.2007.913919. URL `http://dx.doi.org/10.1109/TSMCC.2007.913919`. pages 11

L. Busoniu, R. Babuska, and B. De Schutter. A comprehensive survey of multi-agent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 38 (2):156–172, 2008b. pages 11

S. Ceschia and A. Schaerf. Local search and lower bounds for the patient admission scheduling problem. *Computers and Operartions research*, 38: 1452–1463, 2011. pages 50

I.T. Christou, A.G. Lagodimos, and D. Lycopulou. Hierarchical production planning for multi-product lines in the beverage industry. *Prod Plan Control*, 18:367–376, 2007. pages 118

Caroline Claus and Craig Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *In Proceedings of National Conference on Artificial Intelligence (AAAI-98*, pages 746–752, 1998. pages 11

G. Confessore, S. Giordani, and S. Rismondo. An auction based approach in decentralized project scheduling. In *Proc. of PMS 2002 - International Workshop on Project Management and Scheduling. Valencia*, pages 110–113, 2002. pages 44

G. Confessore, S. Giordani, and S. Rismondo. A market-based multi-agent system model for decentralized multi-project scheduling. *Annals of Operational Research*, 150:115–135, 2007. pages 17, 44, 77, 79

L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. A (sub)graph isomporhism algorithm for matching large graphs. *IEEE Transactions on pattern analysis and machine intelligence*, 26 (10):1367–1372, 2004. pages 124

S. Dauzère-Pérès and J. Paulli. An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search. *Annals of Operations Research*, 70:281–306, 1997. ISSN 0254-5330. URL `http://dx.doi.org/10.1023/A:1018930406487`. 10.1023/A:1018930406487. pages 98, 108

P. Demeester, W. Souffriau, P. De Causmaecker, and G. Vanden Berghe. A hybrid tabu search algorithm for automatically assigning patients to beds. *Artif. Intell. Med.*, 48:61–70, January 2010. ISSN 0933-3657. doi: http://dx.doi.org/10.1016/j.artmed.2009.09.001. URL `http://dx.doi.org/10.1016/j.artmed.2009.09.001`. pages 50, 51

E. Demeulemeester and W. Herroelen. *Project scheduling: a research handbook*, volume 49 of *International Series in Operations Research & Management Science*. Kluwer, 2002. pages 63

T. G. Dietterich and W. Zhang. Solving combinatorial optimization tasks by reinforcement learning: A general methodology applied to resource-constrained scheduling. *Journal of Artificial Intelligence Research*, 2000. pages 18, 19, 21

M. Dorigo. *Optimization, Learning and Natural Algorithms*. PhD thesis, Politecnico di Milano, Italie, 1992. pages 13

M. Dorigo and L.M. Gambardella. Ant colony system : A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1 (1):53–66, 1997. pages 13

K. Easton, G.L. Nemhauser, and M. Trick. The traveling tournament problem description and benchmarks. In *Proceedings of the 7th International Conference on Principles and Practice of Constraint Programming*, pages 580–584, London, UK, 2001. Springer-Verlag. ISBN 3-540-42863-1. pages 49

M. Entrup, H.O. Günther, P. van Beek, M. Grunow, and T Seiler. Mixed-integer linear programming approaches to shelf-life-integrated planning and scheduling in yoghurt production. *International Journal of Production Research*, 43:5071–5100, 2005. pages 118

D. Ferreira, R. Morabito, and S. Rangel. Solution approaches for the soft drink integrated production lot sizing and scheduling problem. *European Journal of Operational Research*, 196:697–706, 2009. pages 118

K. Fleszar and K. S. Hindi. New heuristics for one-dimensional bin-packing. *Computers & Operations Research*, 29(7):821 – 839, 2002. ISSN 0305-0548. doi: 10.1016/S0305-0548(00)00082-4. URL `http://www.sciencedirect.com/science/article/pii/S0305054800000824`. pages 14

B. Franck and K. Neumann. Resource-constrained project scheduling with time windows: Structural questions and priority-rule methods. Technical report, Technical Report WIOR-492, Institute for Economic Theory and Operations Research, University of Karlsruhe, 1998. pages 127

B. Franck, K. Neumann, and C. Schwindt. Truncated branch-and-bound, schedule-construction, and schedule-improvement procedures for resource-constrained project scheduling. *OR Spektrum*, 23:297–324, 2001. pages 120, 126, 127

T. Gabel. *Multi-agent Reinforcement Learning Approaches for Distributed Job-Shop Scheduling Problems*. PhD thesis, Universität Osnabrück, Deutschland, 2009. pages 22

Luca M. Gambardella, Marco Dorigo, and Université Libre De Bruxelles. Ant-q: A reinforcement learning approach to the traveling salesman problem. pages 252–260. Morgan Kaufmann, 1995. pages 17, 20, 21

MR Garey, DS Johnson, and R. Sethi. The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1 (2):117–129, 1976. pages 98

F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13(5):533 – 549, 1986. ISSN 0305-0548. doi: 10.1016/0305-0548(86)90048-1. URL `http://www.sciencedirect.com/science/article/pii/0305054886900481`. <ce:title>Applications of Integer Programming</ce:title>. pages 2, 12

F. Glover and G. A. Kochenberger. *Handbook of metaheuristics*. Springer, 2003. pages 2, 12

F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997. pages 12, 121

T. Grenager, R. Powers, and Y. Shoham. Dispersion games: general definitions and some specific learning results, 2002. URL `citeseer.ist.psu.edu/grenager02dispersion.html`. pages 36, 78, 84

J.N.D. Gupta and J.C. Ho. A new heuristic algorithm for the one-dimensional bin-packing problem. *Production Planning & Control: The Management of Operations*, 10 (6):598–603, 1999. pages 14

S. Hartmann. Project scheduling with multiple modes: a genetic algorithm. *Annals of Operations Research*, 102:111–135, 1997. pages 64

D. P. Helmbold and M. K. Warmuth. Learning permutations with exponential weights. *Journal of Machine Learning Research*, 10:1705–1736, 2009. pages 28, 33, 34

J. W. Hermann, editor. *Handbook of production scheduling*, volume 89 of *International Series in Operations Research & Management Science*. Springer, 2006. pages 118

W. Herroelen and B. De Reyck. The multi-mode resource-constrained project scheduling problem with generalized precedence relations. *European Journal of Operational Research*, 119:538–556, 1999. pages 65

W. Herroelen, B. De Reyck, and E. Demeulemeester. Resource-constrained project scheduling: a survey of recent developements. *Computers and Operations Research*, 25:297–302, 1998. pages 64

J.H. Holland, editor. *Adaptation in Natural and Artificial Systems*. University of Michigan press, 1975. pages 13

J. Homberger. A multi-agent system for the decentralized resource-constrained multi-project scheduling problem. *International Transactions in Operational Research*, 14:565–589, 2007. pages 44, 77

J. Homberger. A (μ, λ)-coordination mechanism for agent-based multi-project scheduling. *OR Spectrum*, DOI - 10.1007/s00291-009-0178-3, 2009. pages xix, 17, 77, 79, 85, 87, 88, 89

Junling Hu and Michael P. Wellman. Experimental results on q-learning for general-sum stochastic games. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, pages 407–414, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc. ISBN 1-55860-707-2. URL `http://dl.acm.org/citation.cfm?id=645529.657798`. pages 11

Junling Hu and Michael P. Wellman. Nash q-learning for general-sum stochastic games. *JOURNAL OF MACHINE LEARNING RESEARCH*, 4:1039–1069, 2003. pages 11

J Hurink, B Jurish, and M. Thole. Tabu search for the job shop scheduling problem with multi-purpose machines. *OR-Spektrum*, 15:205–215, 1994. pages 98, 108

S. Ihara. *Information Theory for Continuous Systems*. World Scientific, 1993. pages 34

B. Jarboui, N. Damak, P. Siarry, and A. Rebai. A combinatorial particle swarm optimization for solving multi-mode resource-constrained project scheduling problems. *Applied Mathematics and Computation*, 195:299–308, 2008. pages 65, 76

P. Jedrzejowicz and E Ratajczak-Ropel. *Population Learning Algorithm for the Resource-Constrained Project Scheduling*, volume 92 of *International Series In Operations Research & Management Science*, chapter 11, pages 275 – 296. Springer US, 2006. pages 65, 74, 76

P. Jedrzejowicz and E. Ratajczak-Ropel. Agent-based approach to solving the resource constrained project scheduling problem. 4431/2007(8th International Conference, ICANNGA 2007):480–487, 2007. pages 65, 74, 76

J. Jozefowska, M. Mika, R. Rozycki, G. Waligora, and J. Weglarz. Simulated annealing for multi-mode resource-constrained project scheduling. *Annals of Operations Research*, 102:137–155, 2001. pages 65, 76

E. Kaddoum, Y. Martinez, T. Wauters, K. Verbeeck, P. De Causmaecker, G. Vanden Berghe, M. Gleizes, and J. Georgé. Adaptive methods for flexible job shop scheduling with due-dates, release-dates and machine perturbations. In *International Conference on Parallel Problem Solving From Nature, Workshop on Self-tuning, self-configuring and self-generating search heuristics*, Krakow, Poland, September 2010. pages 98

L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996. pages 3, 7

G. Kapanos, L Puigjaner, and C.T. Maravelias. Production planning and scheduling of parallel continous processes with product families. *Industrial and Engineering Chemical Research*, 50:1369–1378, 2011. pages 118

S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science. New Series*, 220 (4598):671–680, 1983a. pages 121

S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220 (4598):671–680, 1983b. pages 12

J. Kletti. *Manufacturing Execution System - MES*. Springer, 2007. pages 117

G. Knotts, M. Dror, and B. C. Hartman. Agent-based project scheduling. *IIE Transactions*, 32:387–401, 2000. pages 65, 74, 75

D. E. Knuth. *The Art of Computer Programming, Volume 3: Sorting and Searching*. Addison-Wesley, 1973. pages 38

R Kolisch. Serial and parallel resource-constrained project scheduling methods revisited: theory and computation. *European Journal of Operational Research*, 90:320–333, 1996a. pages 44

R. Kolisch. Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation. *European Journal of Operational Research*, 90:320–333, 1996b. pages 68

R. Kolisch. Efficient priority rules for the resource-constrained project scheduling problem. *Journal of Operations Management*, 14:179–192, 1996c. pages 127

R. Kolisch and S. Hartmann. Heuristic algorithms for solving the resource-constrained project scheduling problem: Classification and computational analysis. 1998. pages 78, 81, 101

R. Kolisch and S. Hartmann. Experimental investigation of heuristics for resource-constrained project scheduling: An update. *European Journal of Operational Research*, 174:23–37, 2006. pages 64, 73

R. Kolisch and A. Sprecher. Psplib - a project scheduling problem library. *European Journal of Operational Research*, 96:205–216, 1996. pages 72

G. Kopanos, L. Puigjaner, and M.C. Georgiadis. Optimal production scheduling and lot-sizing in diary plants: The yogurt production line. *Industrial and Engineering Chemical Research*, 49:701–718, 2010. pages 118

H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955. pages 30

K. Y. Li and R. J. Willis. An iterative scheduling technique for resource-constrained project scheduling. *European Journal of Operational Research*, 56:370–379, 1992. pages 68, 81, 102

H. Liao and Q. Wu. Multi-objective optimization by learning automata. *Journal of Global Optimization*, pages 1–29, 2012. ISSN 0925-5001. URL `http://dx.doi.org/10.1007/s10898-012-9973-5`. 10.1007/s10898-012-9973-5. pages 138

M.L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *In Proceedings of the Eleventh International Conference on Machine Learning*, pages 157–163. Morgan Kaufmann, 1994. pages 25, 83

A. Lova, C. Maroto, and P. Tormos. A multicriteria heuristic method to improve resource allocation in multiproject scheduling. *European Journal of Operational Research*, 127:408–424, 2000. pages 81

A. Lova, P. Tormos, M. Cervantes, and F. Barber. An efficient hybrid genetic algorithm for scheduling projects with resource constraints and multiple execution modes. *International Journal of Production Economics*, 117(2): 302–316, 2009. pages 65, 76

X. Mao. *Airport under Control: Multiagent scheduling for Airport Ground Handling*. PhD thesis, University of Tilburg, The Netherlands, 2011. pages 78, 136

M. Masao and C. C. Tseng. A genetic algorithm for multi-mode resource constrained project scheduling problem. *European Journal of Operational Research*, 100:134–141, 1997. pages 64

M Mastrolilli and LM. Gambardella. Effective neighbourhood functions for the flexible job shop problem. *Journal of Scheduling*, 3:3–20, 1996. pages xx, 111, 112

B. De Reyck and W. Herroelen. A branch-and-bound procedure for the resource-constrained scheduling problem with generalized precedence relations. *European Journal of Operational Research*, 111:152–174, 1998. pages 120, 127

V. V. Miagkikh and William F. Punch, III. An approach to solving combinatorial optimization problems using a population of reinforcement learning agents, 1999. pages 17, 20, 21

M. Misir. *Intelligent Hyper-Heuristics: A Tool for Solving Generic Optimisation Problems*. PhD thesis, CODeS Research Group, Department of Computer Science, KU Leuven University, Belgium, 2012. pages 13

M. Misir, T. Wauters, K. Verbeeck, and G. Vanden Berghe. A new learning hyper-heuristic for the traveling tournament problem. In *Proceedings of Metaheuristic International Conference*, 2009. pages 16, 17, 20, 24, 47, 49

M. Misir, K. Verbeeck, P. De Causmaecker, and G. Vanden Berghe. An intelligent hyper-heuristic framework for chesc 2011. In *The 6th Learning and Intelligent OptimizatioN Conference (LION12)*, Paris, France, 2012. pages 19

N. Mladenovic and P. Hansen. Variable neighborhood search. *Computers and Operations Research*, 24(11):1097 – 1100, 1997a. ISSN 0305-0548. doi: 10.1016/S0305-0548(97)00031-2. URL `http://www.sciencedirect.com/science/article/pii/S0305054897000312`. pages 13

N. Mladenovic and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24(11):1097 – 1100, 1997b. pages 121

Robert Moll, Andrew G. Barto, Theodore J. Perkins, and Richard S. Sutton. Learning instance-independent value functions to enhance local search. In *Advances in Neural Information Processing Systems*, pages 1017–1023. MIT Press, 1998. pages 20, 22

D. E. Moriarty, A. C. Schultz, and J. J. Grefenstette. Evolutionary algorithms for reinforcement learning. *Journal of Artificial Intelligence Research*, 11: 241–276, 1999. pages 15

K. Narendra and M. Thathachar. *Learning Automata: An Introduction*. Prentice-Hall International, Inc, 1989. pages 8

A. Nareyek. Choosing search heuristics by non-stationary reinforcement learning. In *METAHEURISTICS: COMPUTER DECISION-MAKING*, pages 523–544. Kluwer Academic Publishers, 2001. pages 16, 20, 23

K. Neumann and C. Schwindt. Activity-on-node networks with minimal and maximal time lags and their application to make-to-order production. *OR Spektrum*, 19:205–217, 1997. pages 126

E. Nowicki and C. Smutnicki. A fast taboo search algorithm for the job shop problem. *Management Science*, 42:797–813, 1996. pages 98

E. Ozcan, Y. Bykov, M. Birben, and E. K. Burke. Examination timetabling using late acceptance hyper-heuristics. In *Proccedings of the 11th conference on Congress of Evolutionary Computation*, Trondheim, Norway, May 2009. pages 52

E. Özcan, M. Misir, G. Ochoa, and E. K. Burke. A reinforcement learning - great-deluge hyper-heuristic for examination timetabling. *Int. J. of Applied Metaheuristic Computing*, pages 39–59, 2010. pages 20, 23

L. Panait, K. Tuyls, and S. Luke. Theoretical advantages of lenient learners: An evolutionary game theoretic perspective. *Journal of Machine Learning Research*, 9:423–457, 2008. pages 15

Liviu Panait and Sean Luke. Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, 11:387–434, 2005. ISSN 1387-2532. doi: 10.1007/s10458-005-2631-2. URL `http://dx.doi.org/10.1007/s10458-005-2631-2`. pages 11

J. Patterson, F. Talbot, R. Slowinski, and J. Weglarz. Computational experience with a backtracking algorithm for solving a general class of precedence and resource-constrained scheduling problems. *European Journal of Operational Research*, 49 (1):68–79, 1990. pages 64

F. Pezzella, G. Morganti, and G. Ciaschetti. A genetic algorithm for the flexible job-shop scheduling problem. *Computers & Operations Research*, 35:3202 – 3212, 2008. pages xx, 98, 111, 112

A. A. B. Pritsker, L. J. Watters, and P. M. Wolfe. Multiproject scheduling with limited resources: A zero-one programming approach. *Management Science*, 16:93–107, 1969. pages 63

M. Ranjbar, B. De Reyck, and F. Kianfar. A hybrid scatter search for the discrete time/resource trade-off problem in project scheduling. *European Journal of Operational Research*, 193:35–48, 2009. pages 65, 76

G. A. Rummery and M. Niranjan. On-line q-learning using connectionist systems. Technical Report CUED/F-INFENG/TR 166, Engineering Department, Cambridge University, 1994. pages 11

C. Schwindt. A branch-and-bound algorithm for the resource constrained project duration problem subject to temporal constraints. Technical report, Technical report WIOR-544, Institute for Economic Theory and Operations Research, University of Karlsruhe, 1998. pages 120, 127

J. Shapiro. Genetic algorithms in machine learning. pages 15

Satinder Singh, Michael Kearns, and Yishay Mansour. Nash convergence of gradient dynamics in general-sum games. In *In Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, pages 541–548. Morgan, 2000. pages 11

A. Sprecher, S. Hartmann, and A. Drexl. An exact algorithm for the project scheduling with multiple modes. *OR spektrum*, 19:195–203, 1997. pages 63, 64, 72

R. S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In D.S. Touretzky, M.C Mozer, and M.E. Hasselmo, editors, *Advances in Neural Information Processing Systems: Proceedings of the 1995 Conference*, pages 1038–1044. MIT Press, Cambridge, MA, 1996. pages 11

R. S. Sutton and A. G. Barto. *Reinforcement learning: an introduction*. MIT Press, 1998a. pages 3, 4, 7

R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998b. pages 132

E.G. Talbi. *Metaheuristics: From Design to Implementation*. John Wiley and Sons, 2009. pages 2, 12

F. B. Talbot. Resource constrained project scheduling with time-resource tradeoffs: The nonpreemptive case. *Management Science*, 28:1197–1210, 1982. pages 63, 64

M. E. Taylor and P. Stone. Transfer learning for reinforcement learning domains: A survey. *J. Mach. Learn. Res.*, 10:1633–1685, December 2009. ISSN 1532-4435. URL `http://dl.acm.org/citation.cfm?id=1577069.1755839`. pages 4

M. E. Taylor, P. Stone, and Y. Liu. Transfer learning via inter-task mappings for temporal difference learning. *Journal of Machine Learning Research*, 8(1): 2125–2167, 2007. pages 4

MAL Thathachar and KR Ramakrishnan. A hierarchical system of learning automata. *IEEE Transactions On Systems, Man, And Cybernetics*, 11 (3): 236–241, 1981. pages 32

M.A.L. Thathachar and P.S. Sastry. Estimator algorithms for learning automata. In *Proc. Platinum Jubilee Conf. Syst. Signal Processing*. Dept. Electrical Engineering, IISC, Bangalore, 1986. pages 9

M.A.L. Thathachar and P.S. Sastry. *Networks of Learning Automata: Techniques for Online Stochastic Optimization*. Kluwer Academic Publishers, 2004. pages 8, 10, 24

P. R. Thomas and S. Salhi. A tabu search approach for the resource constrained project scheduling problem. *Journal of heuristics*, 4:123–139, 1998. pages 65

J. A. Torkestani and M. R. Meybodi. A cellular learning automata-based algorithm for solving the vertex coloring problem. *Expert Syst. Appl.*, 38(8): 9237–9247, August 2011. ISSN 0957-4174. doi: 10.1016/j.eswa.2011.01.098. URL `http://dx.doi.org/10.1016/j.eswa.2011.01.098`. pages 20, 25

Karl Tuyls and Gerhard Weiss. Multiagent learning: Basics, challenges, prospects. *AI Magazine*, 2012. pages 11

J. R. Ullmann. An algorithm for subgraph isomporhism. *Journal of the Association for Computing Machinery*, 23 (1):31–42, 1976. pages 124

V Van Peteghem and M Vanhoucke. An artificial immune system for the multi-mode resource-constrained project scheduling problem. WORKING PAPER 09/555, 2009. pages 65, 76

V Van Peteghem and M Vanhoucke. A genetic algorithm for the preemptive and non-preemptive multi-mode resource-constrained project scheduling problem. *European Journal of Operational Research*, 201:409–418, January 2010. pages 63, 65, 68, 74, 76

J. Verstichel and G. Vanden Berghe. A late acceptance algorithm for the lock scheduling problem. *Logistic Management*, 5:457–478, 2009. pages 52

P Vrancx, K Verbeeck, and A Nowé. Decentralized learning in markov games. *IEEE Transactions on Systems, Man and Cybernetics*, 38(4):976 – 981, August 2008. pages 63, 74

Y. Wang and I. H. Witten. Induction of model trees for predicting continuous classes, working paper 96/23. Technical report, Department of Computer Science, The University of Waikato, 1996. pages 128

C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, Cambridge University, 1989a. pages 72

C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992. pages 10, 21

C.J.C.H. Watkins. *Learning from Delayed Rewards*. PhD thesis, Cambridge University, 1989b. pages 10, 11, 21

T. Wauters, K. Verbeeck, G. Vanden Berghe, and P. De Causmaecker. A multi-agent learning approach for the multi-mode resource-constrained project scheduling problem. In Decker, Sichman, Sierra, and Castelfranchi, editors, *Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, May 2009a. pages 62, 63

T. Wauters, K. Verbeeck, B. Bilgin, and P. Demeester. Learning automata for hyperheuristic selection. In *Proceedings of the national conference of the Belgian Operational Research Society (ORBEL)*, Leuven, Belgium, February 2009b. pages 50

T. Wauters, J. Verstichel, K. Verbeeck, and G. Vanden Berghe. A learning metaheuristic for the multi mode resource constrained project scheduling problem. In *Proceedings of the Third Learning and Intelligent OptimizatioN Conference (LION3)*, 2009c. pages 20, 24, 62, 63

T. Wauters, K. Verbeeck, P. De Causmaecker, and G. Vanden Berghe. A game theoretic approach to decentralized multi-project scheduling (extended abstract). In *Proc. of 9th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, number R24, 2010. pages 17, 18, 20, 24, 29

T. Wauters, K. Verbeeck, G. Vanden Berghe, and P. De Causmaecker. Learning agents for the multi-mode project scheduling problem. *Journal of the operational research society*, 62 (2):281–290, February 2011. pages 17, 20, 24, 25, 29, 62, 63

T. Wauters, W. Vancroonenburg, and G. Vanden Berghe. A guide-and-observe hyper-heuristic approach to the eternity ii puzzle. *Journal of Mathematical Modelling and Algorithms*, 11 (3):217–233, 2012a. pages 2

T. Wauters, K. Verbeeck, P. De Causmaecker, and G.et Vanden Berghe. Decentralized multi-project scheduling: a game theoretic learning approach. *Technical Report 09/07/2012*, 2012b. pages 18

R. M. Wheeler and K. Narendra. Decentralized learning in finite markov chains. *IEEE Transactions on Automatic Control*, AC-31:519 – 526, 1986. pages 63, 74

R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Machine Learning*, pages 229–256, 1992. pages 8

R.J. Willis. Critical path analysis and resource constrained project scheduling - theory and practice. *European Journal of Operational Research*, 21 (2): 149–155, 1985. pages 80

E. Yang and D. Gu. Multi-robot systems with agent-based reinforcement learning: evolution, opportunities and challenges. *International Journal of Modelling, Identification and Control*, 6 (4):271–286, 2009. pages 11

W. Zhang and T. Dietterich. A reinforcement learning approach to job-shop scheduling. In *In Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 1114–1120. Morgan Kaufmann, 1995. pages 16, 20, 21, 22, 98

G. Zhu, J. Bard, and G. Tu. A branch-and-cut procedure for the multimode resource-constrained project-scheduling problem. *Journal on Computing*, 18 (3):377–390, 2006. pages 64

Curriculum vitae

October 8, 1986	Born in Lokeren, Belgium
2004-2007	Bachelor of Science in Industrial Engineering: Electronics-ICT, KAHO Sint-Lieven, Gent, Belgium
2007-2008	Master of Science in Industrial Engineering: Electronics-ICT, KAHO Sint-Lieven, Gent, Belgium
2008-present	Research associate at the Department of Industrial Engineering, Computer Science, KAHO Sint-Lieven, Gent, Belgium

Publications

Articles in internationally reviewed journals

- Wauters, T., Vancroonenburg, W., Vanden Berghe, G. (2012). A guide-and-observe hyper-heuristic approach to the Eternity II puzzle. Journal of Mathematical Modelling and Algorithms, 11(3), 217-233.
- Wauters, T., Verbeeck, K., De Causmaecker, P., Vanden Berghe, G. (2012). Decentralized multi-project scheduling: a game theoretic learning approach. Technical Report KAHO-CW-09/07/2012 (Under review)
- Wauters, T., Verbeeck, K., Verstraete, P., Vanden Berghe, G., De Causmaecker, P. (2012). Real-world production scheduling for the food industry: an integrated approach. Engineering Applications of Artificial Intelligence, 25(2), 222-228.
- Wauters, T., Verbeeck, K., Vanden Berghe, G., De Causmaecker, P. (2011). Learning agents for the multi-mode project scheduling problem. Journal of the Operational Research Society, 62(2), 281-290.

Book chapters, internationally recognised scientific publisher

- Wauters, T., Verbeeck, K., De Causmaecker, P., Vanden Berghe, G. (2013). Boosting Metaheuristic Search Using Reinforcement Learning. In: El-Ghazali T. (Eds.), Hybrid Metaheuristics, Chapt. 17. Springer Berlin Heidelberg, 433-452.
- Misir, M., Wauters, T., Verbeeck, K., Vanden Berghe, G. (2012). A Hyper-heuristic with Learning Automata for the Traveling Tournament

Problem. In: Caserta M., Voss S. (Eds.), Metaheuristics: Intelligent Decision Making. Springer.

Papers at international conferences and symposia, published in full in proceedings

- Wauters, T., Verbeeck, K., De Causmaecker, P., Vanden Berghe, G. (2012). Fast permutation learning. In : LION 6 proceedings, LNCS 7219, (Hamadi, Y., Schoenauer, M. (Eds.)). Learning and Intelligent OptimizatioN Conference. Paris, France, 16-20 January 2012 (pp. 292-306). Springer.
- Wauters, T., Vancroonenburg, W., Vanden Berghe, G. (2010). A two phase hyper-heuristic approach for solving the Eternity II puzzle. In : Proceedings of the International Conference on Metaheuristics and Nature Inspired Computing. International Conference on Metaheuristics and Nature Inspired Computing. Djerba Island, Tunisia, 27-31 October 2010 (pp. 1-3).
- Kaddoum, E., Martinez, Y., Wauters, T., Verbeeck, K., De Causmaecker, P., Vanden Berghe, G., Gleizes, M., Georgé, J. (2010). Adaptive methods for flexible job shop scheduling with due-dates, release-dates and machine perturbations. International Conference on Parallel Problem Solving From Nature, Workshop on Self-tuning, self-configuring and self-generating search heuristics (Self* 2010). Krakow, Poland, 11-15 September 2010.
- Wauters, T., Verbeeck, K., Vanden Berghe, G., De Causmaecker, P. (2009). A multi-agent learning approach for the multi-mode resource-constrained project scheduling problem. In : Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009), (Decker, Sichman, Sierra, Castelfranchi (Eds.)). Autonomous Agents and Multiagent Systems. Hungary, 10-15 May 2009 (pp. 1-8). International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org).

Meeting abstracts, presented at international conferences and symposia, published or not published in proceedings or journals

- Vancroonenburg, W., Wauters, T. (2012). A late acceptance metaheuristic for the machine reassignment problem. European Conference on

Operational Research (EURO2012). Vilnius, Lithuania, 8-11 July 2012.

- Salassa, F., Wauters, T., Vancroonenburg, W., Della Croce, F., Vanden Berghe, G. (2012). The Eternity II puzzle: a matheuristic approach. International Symposium on Combinatorial Optimization. Athens, Greece, 17-21 April 2012.

- Wauters, T., Verbeeck, K., De Causmaecker, P., Vanden Berghe, G. (2012). A decentralized multi-project approach. In : Proceedings of the 13th International Conference on Project Management and Scheduling.. International Conference on Project Management and Scheduling. Leuven, 1-4 April 2012.

- Wauters, T., Verstraete, P., De Causmaecker, P., Vanden Berghe, G., Verbeeck, K. (2011). The MESware scheduler: an adaptive production scheduling system with complex constraints. In : Proceedings of BNAIC 2011. BNAIC. Gent, Belgium, 3-4 November 2011.

Meeting abstracts, presented at local conferences and symposia, published or not published in proceedings or journals

- Wauters, T., Vancroonenburg, W., Salassa, F., Della Croce, F., Vanden Berghe, G. (2012). The Eternity II puzzle: a matheuristic approach. ORBEL. Bruxelles, 2-3 February 2012.

- Wauters, T., Verstraete, P., Verbeeck, K., Vanden Berghe, G., De Causmaecker, P. (2011). Production scheduling for the food industry: An integrated approach. ORBEL. Ghent, Belgium, 10-11 February 2011 (89-90).

- Vancroonenburg, W., Wauters, T., Vanden Berghe, G. (2010). A real world 1D stock cutting problem: Exact and heuristic algorithms. ORBEL. Liege, 28-29 January 2010.

- Wauters, T., Verbeeck, K., Bilgin, B., Demeester, P. (2009). Learning automata for hyperheuristic selection. ORBEL. Leuven, 5-6 February 2009.

Arenberg Doctoral School of Science, Engineering & Technology
Faculty of Engineering
Department of Computer Science
CODeS Group
Celestijnenlaan 200A box 2402
B-3001 Heverlee