A Multi-Agent Learning Approach for the Multi-Mode Resource-Constrained Project Scheduling Problem

Tony Wauters, Katja Verbeeck, Greet Vanden Berghe
Vakgroep IT
KaHo Sint-Lieven
Gebroeders Desmetstraat 1
B-9000 Gent, Belgium
{FirstName.LastName}@kahosl.be

Patrick De Causmaecker
Faculty of Sciences, Department of Computerscience
K.U. Leuven Campus Kortrijk
Etienne Sabbelaan 53
B-8500 Kortrijk, Belgium
patrick.decausmaecker@kuleuven-kortrijk.be

ABSTRACT

This paper introduces a novel approach for solving the multi-mode resource-constrained project scheduling problem (MRCPSP), in which multiple execution modes are available for each of the activities of the project. The new approach applies simple agent learning devices, i.e. learning automata, to construct the project schedules. We present some comparative results, to show that our decentralized method can easily compete with the best performing algorithms for the MRCPSP.

1. INTRODUCTION

In the last few decades, the resource-constrained project scheduling problem (RCPSP) has become a popular subject in operations research. It consists of scheduling the activities from a project by respecting the resource requirements and precedence relations between the activities.
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 their own shovel need 6 days to dig a pit, while 4 people each with their own shovel and one additional wheelbarrow need only 2 days). A comprehensive survey of the project scheduling problem can be found in [3]. The latter paper presents a unifying notation, a model, a classification scheme, i.e. a description of the resource environment, the activity characteristics, and the objective function, respectively. The notation is similar to machine scheduling and allows to classify the most important models. It also introduces some exact and heuristic methods for both single and multi-mode problems. In [6] Herroelen et al. discuss the problem and its practical relevance. Kolisch and Hartmann [13] provide an update of their survey 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 comparitive results. The RCPSP is shown to be an NP-hard optimization problem [1], thus so is the MRCPSP, because it is a generalisation of the RCPSP [11]. In order to produce good quality solutions in a short amount of time, different kinds of (meta-)heuristics have been proposed to address the problem. A tabu search metaheuristic is applied in [5] for solving the MRCPSP with generalized precedence relations. In [4, 7, 16, 17] a genetic algorithm is presented for the MRCPSP.

Agent-based approaches have also been succesfully applied to the MRCPSP. In [10] two types of agents (basic and enhanced agents) are used, 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. In [9] a number of agents, each representing a different optimization algorithm including local search, tabu search, as well as several specialized heuristics, have been used to work on a population of solutions in parallel on different computers. In [8] a population learning algorithm is presented for solving both the single and the multi-mode rcpsp.
In this paper we present a novel multi-agent based approach, fundamentally different from these other agent based approaches. It is based on a graph representation in which the nodes are agents representing activities. Each agent is responsible for one activity. The agents make use of two learning devices, i.e. learning automata (LA) to construct a schedule. We use a smart coupling mechanism of rewards to learn both the order of the activities and the modes at the same time. Further on our approach is inspired by theoretical results that show how interconnected LA devices are able to find attractor points in MDP's and Markov Games [18, 19]. Altough the activity-on-node model for the MRCPSP problem as we consider here does not satisfy the Markov property as was assumed in [18, 19], good results are produced.

This paper is structured as follows. Section 2 gives a brief problem description, followed by our model and multi-agent learning algorithm in Section 3. In Section 4 we present some computational experiments and comparative results. Finally in Section 5 we draw conclusions and discuss future work.

2. PROBLEM FORMULATION

The MRCPSP can be formulated as follows. A project has N activities $a_1, \ldots, a_N; a_i \in A$, with A the set of activ-

Cite as: Title, Author(s), *Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, Decker, Sichman, Sierra and Castelfranchi (eds.), May, 10–15, 2009, Budapest, Hungary, pp. XXX-XXX.

ities, where each activity can be performed in one out of a set of K modes $m_{i1}, \ldots, m_{iK}; m_{ij} \in M_i$, with M_i the set of modes for activity i. Each mode corresponds to a specific activity duration $d_{m_{ij}}$ and resource requirements ($ren^l_{m_{ij}}$ for the renewable resources and $nren^l_{m_{ij}}$ for the non-renewable resources). The dummy start and end activities 1 and N have zero duration and zero resource usage. These activities have to be scheduled according to their finish-start precedence relations. Any activity i has a set P_i of activities as its predeccessors, and also a set S_i of activites as its successors. The project has some renewable and non-renewable resources available each with their own availability. For each renewable resource l the total availability ren^l is constant throughout the problem horizon, while the non-renewable resources have a limited total usage $nren^l$.

In the MRCPSP, the objective is to find an activity order - mode combination that produces a schedule that minimises the project makespan and is subject to two hard constraints: 1) an activity should not be scheduled until all its predecessors have finished (precedence constraint) and 2) the number of assignments of a resource at any time should not be larger than the availability of that resource (resource constraint). With s_i the start time and d_i the duration of activity i we can also formulate the problem as follows:

$$min\ s_N \tag{1}$$

$$s.t.$$

$$\sum_{j=1}^{K} m_{ij} = 1 \qquad i = 1, \ldots, N \tag{2}$$

$$s_p + d_p \leq s_i \qquad \forall_i; s_p \in P_i \tag{3}$$

$$d_i = \sum_{j=1}^{K} m_{ij}\ d_{m_{ij}} \qquad i = 1, \ldots, N \tag{4}$$

$$\sum_{i=1}^{N}\sum_{j=1}^{K} m_{ij}\ nren^l_{m_{ij}} \leq nren^l \qquad \forall_l \tag{5}$$

$$\sum_{i=1}^{N}\sum_{j=1}^{K} m_{ij}\ ren^l_{m_{ij}}\ e_{it} \leq ren^l \qquad \forall_l; t = s_1, \ldots, s_N \tag{6}$$

$$\left\{\begin{matrix} e_{it}=1 & if\ s_i \leq t < s_i + d_i \\ e_{it}=0 & o.w. \end{matrix}\right\} \qquad \forall_{i,t} \tag{7}$$

$$m_{ij}, e_{it} \in \{0, 1\} \qquad \forall_{i,j,t} \tag{8}$$

$$d_{m_{ij}}, s_i \in \mathbb{N} \qquad \forall_{i,j} \tag{9}$$

$$s_0 = 0 \tag{10}$$

3. THE MULTI-AGENT LEARNING APPROACH

The (M)RCPSP can be presented with an activity-on-node diagram. It uses a graph to show the precedence relations between the activities. In Figure 1 we see an example of a project with 7 activities according to the problem description in Section 2 (1 and 7 are dummy activities) and their relations. That respresentation is the starting point, for the multi-agent based learning algorithm we developed.

Our goal is to create an activity order list and a mapping from activities to modes, which can later be used to construct a schedule. The activity order list is a permutation of all the activities, and determines in which order the schedule construction algorithm handles the activities. The mode

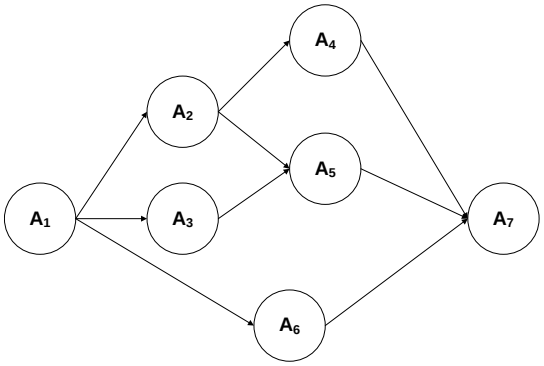

Figure 1: An example of an activity-on-node diagram for a project with 7 activities.

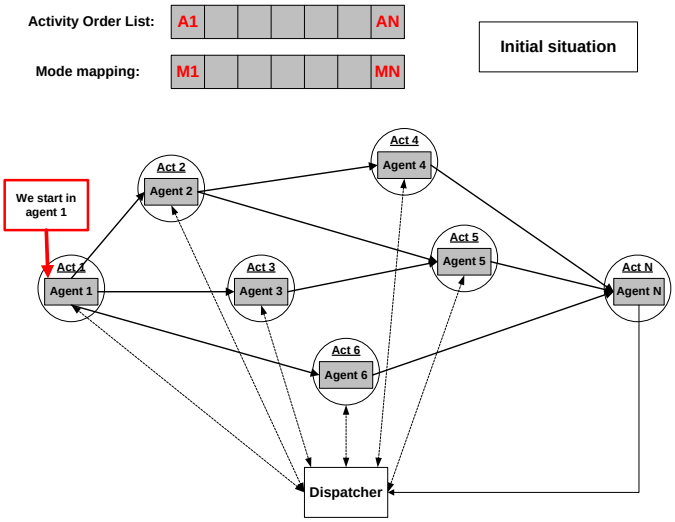

Figure 2: Initial situation, one agent in every activity node + one extra dispatching agent.

mapping determines in which mode each activity will be executed. We start by placing an agent in every activity node. Further on we add an extra dispatching agent (dispatcher) which is needed by our algorithm to construct schedules that respect the two hard constraints. In contrast to the other agents, the dispatcher does not represent an activity and only chooses an other agent to hand over the control. This initial situation is presented in Figure 2.
The main idea of the algorithm is to enable every agent to learn which decisions to make, concerning:

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

The algorithm works as follows: we start in the situation as in Figure 2, we give the control to agent 1, this agent chooses an 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. Now the control is given to $Agent_{current} \Leftarrow Agent_{next}$, which also decides upon an order to visit its successors (e.g. A_2 chooses order $|A_5, A_4|$, so it will first take A_5 and then A_4) and takes the first agent from this order ($Agent_{next}$). $Agent_{current}$ has

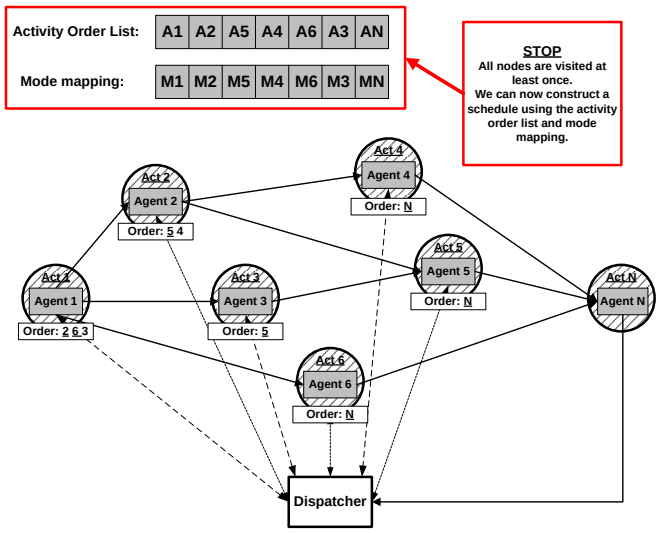

Figure 3: Final situation, all agents have been visited at least once.

not been visited before. Consequently the activity it represents is added to the activty order list, and the agent also chooses a mode which is added to the mode mapping. 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 ($Pr_{RandToVisited}$) to choose a random eligible agent from the list of already 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 changed into something else (e.g. a heuristic strategy). These steps are carried out subsequently until all agents have been visited at least once.

In addition to all the previous, the agents behave stochastically. At any time they can give the control, with a small probability Pr_{ToDisp}, to the dispatcher. This makes it possible to naturally consider all possible activity-order permutations and hence all the possible schedules.

Now we can construct a schedule with the activity order and mode mapping with a serial schedule generation scheme that uses a standard heuristic method for RCPSP (see [12] for details).

3.1 Algorithm

In this section we present the algorithm behind our approach in a more formal way by using pseudo code. In Algorithm 1 the global control of the algorithm for constructing an Activity Order List and Mode Mapping is presented. From there the control is given to individual agents which use Algorithm 2. The latter returns the next agent to visit ($Agent_{next}$) which is given control by the global control.
For clarity we left out the implementation of the method to determine if an Agent is eligble.

The method described above for constructing a schedule can now be used in an iterative way. We use the schedule's quality (makespan) for the agents to learn the actions to take. We apply some simple learning automata devices

Algorithm 1 Global Control

Input: Project data and Algorithm parameters
Output: A precedence constraint feasable schedule
  initialize $ActivityOrderList$ and $ModeMappingList$
  $Agent_{current} \Leftarrow Agent_1$
  while Not all agents visited **do**
    give the control to $Agent_{current}$
    $Agent_{next}$ determined by $Agent_{current}$ using Algo. 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 obtained $ActivityOrderList$ and $ModeMappingList$
  return $Schedule$

Algorithm 2 Single Agent Control

Input: $ActivityOrderList$, $ModeMappingList$
Output: $Agent_{next}$
  rand $\Leftarrow$ 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 $ModeMappingList$
      $Order \Leftarrow$ chooseOrder() using Order LA
      $Agent_{next} \Leftarrow$ first Agent in $Order$
    else
      $Agent_{next} \Leftarrow$ next Agent in $Order$
    end if
  end if
  return $Agent_{next}$

which we will describe in the next Section.

3.2 Learning Automata

Learning Automata are simple reinforcement learners originally introduced to study human and animal behavior. The objective of an automaton is to learn an optimal action, based on past actions and environmental feedback. Formally the automaton is described by a quadruple $\{A, \beta, p, U\}$, where $A = \{a_1, \ldots, a_r\}$ is the set of possible actions the automaton can perform, p is the probability distribution over these actions, β is a random variable between 0 and 1 representing the evironmental response, and U is a learning scheme used to update p.

A single automaton is connected in a feedback loop with its environment. Actions chosen by the automaton are given as input to the environment and the environmental response to these actions serves as input to the automaton. Several automaton update schemes with different properties have been studied. Important examples of linear update schemes are linear reward-penalty, linear reward-inaction and linear reward-ϵ-penalty. The philosophy of these schemes is essentially to increase the probability to select an action when it results in a success and to decrease it when the response is a failure. The general algorithm is given by:

$$\begin{aligned} p_m(t+1) &= p_m(t) + \alpha_{reward}(1-\beta(t))(1-p_m(t)) \\ &\quad - \alpha_{penalty}\beta(t)p_m(t) \qquad (11) \\ &\quad \text{if } a_m \text{ is the action taken at time } t \\ p_j(t+1) &= p_j(t) - \alpha_{reward}(1-\beta(t))p_j(t) \\ &\quad + \alpha_{penalty}\beta(t)[(r-1)^{-1} - p_j(t)] \qquad (12) \\ &\quad \text{if } a_j \neq a_m \end{aligned}$$

The constants α_{reward} en $\alpha_{penalty}$ are the reward and penalty parameters respectively. 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 ($L_{R-\epsilon P}$).

A motivation for using 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 with unknown transition probabilities and rewards. Recently this result was extended to the framework of Markov Games, a straightforward extension of single-agent markov decision problems (MDP's) to distributed multi-agent decision problems [15].

3.3 LA for the MRCPSP

For learning the activity order and the best modes we applied the (L_{R-I}) method because of its ϵ-optimality property in all stationary environments. The learning rate (reward parameter) that is used for learning the activity order, and the one that is used for learning the mode are named LRO and LRM. The application of the reinforcement will be presented in what follows.

After a schedule was constructed, we update all the learning automata using the following reinforcements. If the makespan of the constructed schedule was:

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

Both r_{eq} and the learning rates LRO and LRM determine the speed of learning. A higher r_{eq} can speed up the learning, especially for a problem like the MRCPSP where attempts only rarely result in improvements.
The settings of the 2 learning rates are dependent. A proper combination will be important for a good overall performance.

The viewpoint of a single agent is presented in Figure 4. Each agent has two learning devices. When the agent is visited for the first time, the algorithm will ask an agent to choose an order to visit 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). When 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 reinforcement (reward) rules mentioned above (Equation 11 and 12). 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.

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 11 and run on an Intel Core 2 Duo E8400 3.0GHz processor, 4GB RAM. To test the performance of the algorithm, we applied it to instances of the project scheduling problem library (PSPLIB) [14], 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 in Section 4.1. In Section 4.2 we consider the single-mode version.

4.1 Multi-Mode

The PSPLIB library 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 from these datasets have two renewable and two nonrenewable resources. Each dataset contains 640 instances, of which some are infeasible. We exclude the infeasible instances from the tests.

When testing the algorithm we found that the required number of iterations depends largely 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 on the same instance and taking the best solution over all the runs. In Table 1,2,3 and 4 we present the results of the multi-agent based algorithm for the J10 to J20 datasets from the above mentioned PSPLIB library, using the following parameters for all the tests: 0.01 for the order learning rate (LRO), 0.2 for the mode learning rate (LRM), $r_{eq} = 0.05$, 0% Pr_{ToDisp}, 5% $Pr_{RandToVisited}$ and different $Restarts \times Iterations$ combinations each with a total of 100,000 iterations. For these $Restarts \times Iterations$ combinations we used: $5 \times 20,000$ iterations (5 restarts with 20,000 schedule constructions each), $10 \times 10,000$ iterations

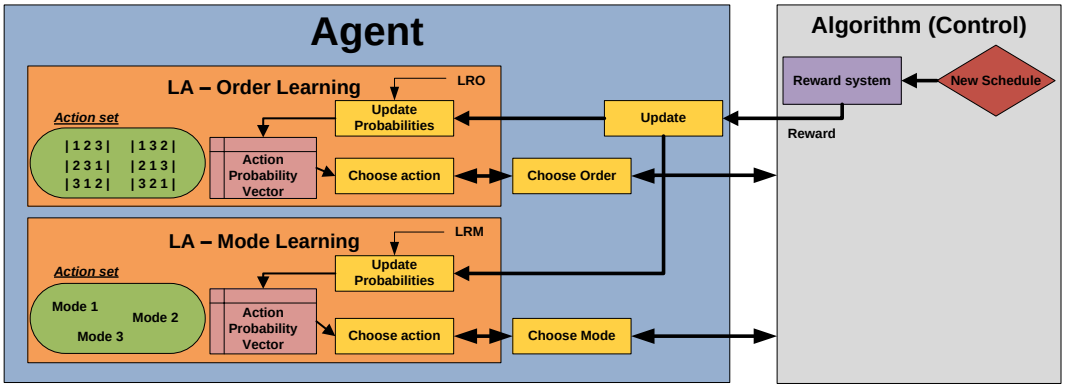

Figure 4: The single agent view.

and a tuned combination (see later). The results have been evaluated in terms of the average procentual deviation from optimum over all the instances or the relative error (RE), the maximum RE, the standard deviation over all RE, the % of optimal solutions found, and the average runtime in seconds. We compared our algorithm with an other agent based approach [9], a population learning algorithm (PLA) [8] and a simulated annealing algorithm [2]. For the population learning algorithm we took the results for 50,000 schedule constructions and for 2 PLA runs, which is similar to the total of 100,000 iterations of the agent based learning approach.

In Table 5 we present the results for the J30 dataset using $5 \times 20,000$ iterations and $5 \times 50,000$ iterations. For this dataset the optimal solutions have not been found by the research community. We therefore calculated the average procentual deviation from the best known solutions.

The learning rates LRO and LRM have been determined empirically by measuring the average performance of some learning rate combinations on several instances from the different datasets. In Figure 5 we present the average resulting makespan of different learning rate combinations for the J2054_2 instance. Here it seems that the $[0.01 - 0.2]$ and $[0.2 - 0.2]$ combinations perform best. Similar conclusions have been made when considering other instances. In any case, the LRM must be large enough (e.g. $LRM = 0.2$) to give good results in the limited interval of 20,000 iterations. This is probably due to the importance of choosing proper modes in the MRCPSP. We also added the learning rate combination $[0.0 - 0.0]$ which means that the agents do not learn, but select random actions. As we expected the method without learning performs the worst.

To determine the number of restarts together with the number of iterations per restart we did some experimental tests on the hardest instances for every dataset(i.e. instances for which our approach performed the worst in earlier tests). For every hard instance we performed 20 runs for some $Restarts \times Iterations$ combinations. These combinations all have a total of 100,000 iterations. We averaged the procentual difference with the optimum over the 20 runs. All tests have been executed using the default parameter values mentioned in the beginning of this Section. In Figure 6 we see the results for some hard J20 instances, which shows us that the $10 \times 10,000$ combination is the best performing for this dataset. We can draw similar conclusions from Figure

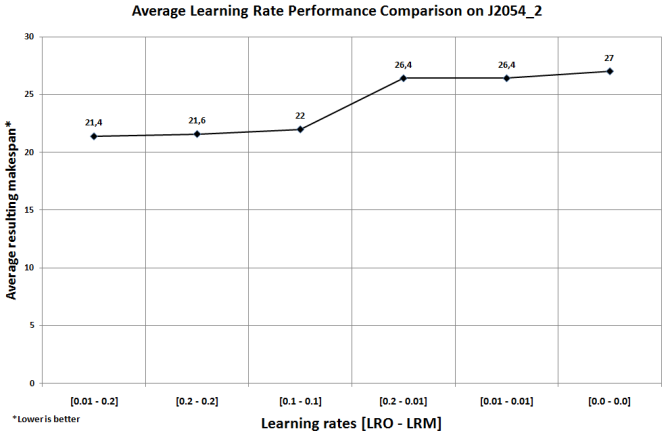

Figure 5: A comparison of different learning rate combinations for the J2054_2 instance.

7 and 8 for the J30 and J10 dataset. $5 \times 20,000$ seems the best combination for J30, while $50 \times 2,000$ appears to be the best for the J10 dataset. Using these best performing $Restarts \times Iterations$ combinations for every dataset, we obtained the 'Tuned' results from Figure 4. In general we can see that larger problem instances need more iterations, taking into account the fixed 100,000 iterations this automatically results in fewer restarts.

When considering these results for the MRCPSP we can conclude that the multi-agent approach performs very well when comparing it to the methods from the literature. We even reach 100% optimality for the J10 dataset when using the Tuned version of the algorithm.

4.2 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. In Table 6 we present 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 4.1 but only $5 \times 5,000$ iterations. Since not all the optimal solutions are 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 solu-

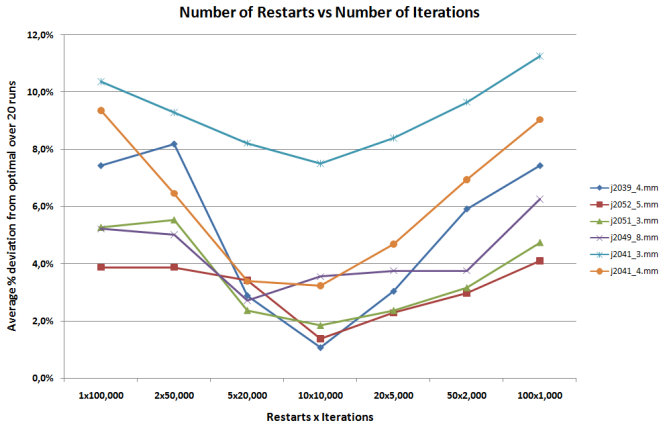

Figure 6: Number of restarts vs number of iterations for some hard J20 instances

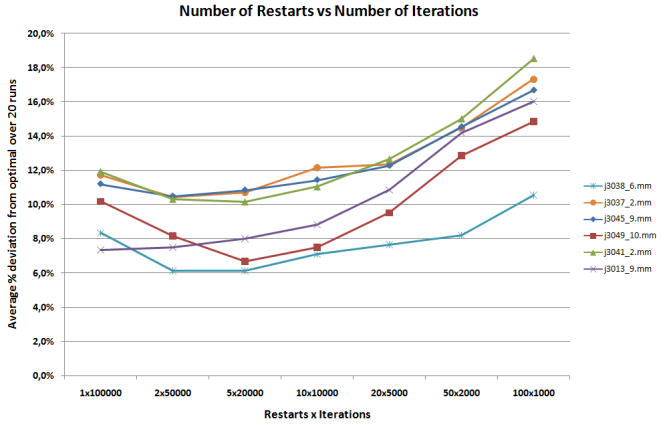

Figure 7: Number of restarts vs number of iterations for some hard J30 instances

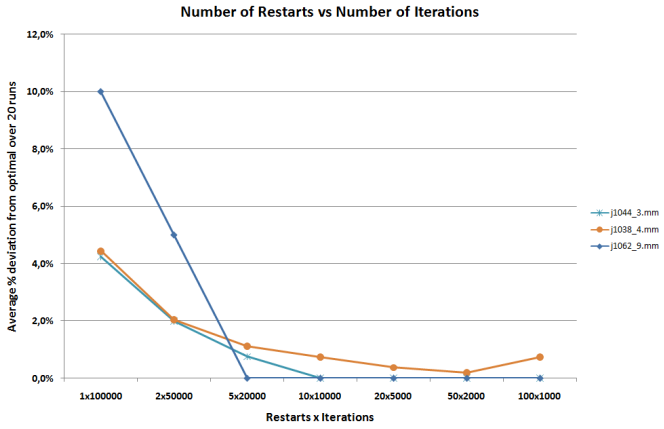

Figure 8: Number of restarts vs number of iterations for some hard J10 instances

tions.

When looking at these single-mode RCPSP results, which reveal average performance when comparing it to the best algorithms reported in the literature, we can conclude that the power of the approach is its coupling between learning the activity order and learning the modes.

Note that although our approach is distributed, it 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 MR-CPSP, 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 MR-CPSP [17], the results of the multi-agent learning approach have similar quality and even performs slightly better on some multi-mode datasets. However this comparison is not completely fair, because we did use more schedule constructions ($> 5,000$).

5. CONCLUSIONS

In this paper we have presented a novel approach for solving the multi-mode resource-constained project scheduling problem (MRCPSP) using agents. The agents make use of simple learning automata for learning both the activity orders and the mode assignments simultaneously.

Based on the results presented in this paper, we can conclude that the multi-agent approach performs very well when comparing it to the methods from the literature, especially to other agent-based and learning methods. In the future we will speed up the learning ($\pm 5,000$ iterations), by parameter tuning or incorporation of heuristic information (e.g. dispatcher strategy), so we can make a fair comparison with the most competitive algorithms, which are specialized Genetic Algorithms for the MRCPSP.

Instead of only testing the multi-agent approach on benchmarks, we expect that the presented approach is also capable of handling real practical problems.

The 'rough-and-ready' aspect of the experimental configuration coupled with the good results, strongly suggests a promising future for further research and the practical application of learning automata to several scheduling problems. Further on, this method can be applied to problems where one needs to find a permutation of elements which is restricted by precedence constraints, as in the Precedence Constraint Traveling Salesman Problem (PCTSP) or the Sequential Ordering Problem. Finally, we will also investigate how to relate the developed algorithm to the theoretical frameworks [18, 19] for interconnected LA learning devices.

6. REFERENCES

[1] J. Blazewicz, J. Lenstra, and A. R. Kan. Scheduling projects subject to resource constraints: Classification and complexity. *Discrete Applied Mathematics*, 5:11–24, 1983.

[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.

[3] P. Brucker, A. Drexl, R. Mohring, K. Neumann, and E. Pesch. Resource-constrained project scheduling: Notation, classification, models and methods. *EJOR*, 112:3–41, 1999.

Table 1: Experimental results MRCPSP

	J10	J12	J14	J16	J18	J20
Average deviation from optimal (RE) (%):						
P. Jedrzejowicz and E. Ratajczak (2007) [9]	0.72	0.73	0.79	0.81	0.95	1.80
P. Jedrzejowicz and E. Ratajczak (2006) [8]	0.36	0.50	0.62	0.75	0.75	**0.75**
K. Bouleimen and H. Lecocq (2003) [2]	0.21	0.19	0.92	1.43	1.85	2.10
Multi-Agent Learning Approach ($5 \times 20,000$ ***)***	**0.04**	**0.11**	**0.28**	**0.34**	**0.45**	0.81
Multi-Agent Learning Approach ($10 \times 10,000$ ***)***	**0.01**	**0.02**	**0.17**	**0.23**	**0.36**	**0.72**
Multi-Agent Learning Approach (Tuned)	**0.00**	**0.02**	**0.11**	**0.18**	**0.36**	**0.72**
Average runtime (s)	5	6.5	7.5	9	10	11.5

Table 2: Experimental results MRCPSP - $5 \times 20,000$

	J10	J12	J14	J16	J18	J20
Average RE (%)	**0.04**	**0.11**	**0.28**	**0.34**	**0.45**	0.81
Std. Dev. RE (%)	0.55	0.79	1.08	1.20	1.36	1.83
Max RE (%)	11.11	8.70	7.69	8.70	12.50	10.71
Optimal (%)	99.44	97.99	93.47	91.82	88.41	80.40

Table 3: Experimental results MRCPSP - $10 \times 10,000$

	J10	J12	J14	J16	J18	J20
Average RE (%)	**0.01**	**0.02**	**0.17**	**0.23**	**0.36**	**0.72**
Std. Dev. RE (%)	0.17	0.28	0.80	0.93	1.14	1.67
Max RE (%)	4	4.76	5	6.25	6	10
Optimal (%)	99.81	99.63	95.64	93.64	90.40	82.03

Table 4: Experimental results MRCPSP - Tuned

	J10	J12	J14	J16	J18	J20
Average RE (%)	**0.00**	**0.02**	**0.11**	**0.18**	**0.36**	**0.72**
Std. Dev. RE (%)	**0.00**	0.29	0.63	0.82	1.14	1.67
Max RE (%)	**0.00**	5.56	5.26	7.14	6	10
Optimal (%)	**100**	99.63	96.73	94.91	90.40	82.03

Table 5: Experimental results MRCPSP - J30

	Average deviation from best known solutions (%)	Max RE (%)	Average runtime (s)
$5 \times 20,000$ iterations	2.03	16.13	39
$5 \times 50,000$ iterations	1.10	11.90	157

Table 6: Experimental results RCPSP - J120

	Average deviation from critical path length (%)	Average deviation from best known solutions (%)	Average runtime (s)
$5 \times 5,000$ iterations	36.98	4.36	120

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

[5] W. Herroelen and B. De Reyck. The multi-mode resource-constrained project scheduling problem with generalized precedence relations. *EJOR*, 119:538–556, 1999.

[6] 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.

[7] J.Alcaraz and C. Maroto. A new genetic algorithm for the multi-mode resource-constrained project scheduling problem. page 4, 2002.

[8] P. Jedrzejowicz and E. Ratajczak. *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.

[9] 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.

[10] M. D. G. Knotts. Agent-based project scheduling. *IIE Transactions*, 32:387–401, 2000.

[11] R. Kolisch. Project scheduling under resource constraints - efficient heuristics for several problem cases. *Physica-Verlag*, 1995.

[12] R. Kolisch and S. Hartmann. Heuristic algorithms for solving the resource-constrained project-scheduling problem: Classification and computational analysis. *Handbook on recent advances in project scheduling*, 1998.

[13] 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.

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

[15] 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.

[16] M. Masao and C. Tseng. A genetic algorithm for multi-mode resource constrained project scheduling problem. *EJOR*, 100:134–141, 1997.

[17] V. Van Peteghem and M. Vanhoucke. A genetic algorithm for the multi-mode resource-constrained project scheduling problem. Working paper, January 2008.

[18] 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.

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