Noname manuscript No.
(will be inserted by the editor)

A learning-based optimization approach to multi-project scheduling

Tony Wauters · Katja Verbeeck · Patrick De Causmaecker · Greet Vanden Berghe

Abstract The present paper introduces a learning-based optimization approach to the resource constrained multi-project scheduling problem (RCMPSP). Multiple projects, each with their own set of activities, need to be scheduled. The objectives here dealt with include minimization of the average project delay and total makespan. The availability of local and global resources, precedence relations between activities, and non-equal project start times have to be considered.

The proposed approach relies on a simple sequence learning game played by a group of project managers. The project managers each learn their activity list locally using reinforcement learning, more specifically learning automata. Meanwhile, they learn how to choose a suitable place in the overall sequence of all activity lists. All the projects need to arrive at a unique place in this sequence. A mediator, who usually has to solve a complex optimization problem, now manages a simple dispersion game. Through the mediator, a sequence of feasible activity lists is eventually scheduled by using a serial schedule generation scheme, which is adopted from single project scheduling.

It is shown that the sequence learning approach has a large positive effect on minimizing the average project delay. In fact, the combination of local reinforcement learning, the sequence learning game and a forward-backward implementation of the serial scheduler significantly improves the best known results for all the MPSPLIB datasets. In addition, several new best results were obtained for both considered objectives: minimizing the average project delay and minimizing the total makespan.

KU Leuven, Department of Computer Science, CODeS, Gebroeders Desmetstraat 1, B-9000 Gent, Belgium
E-mail: tony.wauters@cs.kuleuven.be

Keywords: Multi-Project Scheduling, Learning Automata, Dispersion Games

1 Introduction

Collaborative project management is becoming common in today's globally active industries. Indeed, enterprises collaborate simultaneously with different customers or partners in various projects requiring scarce and shared resources. Collaborative or multi-project management 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 resource constrained multi-project scheduling problem (RCMPSP), which is a generalization of the resource-constrained project scheduling problem (RCPSP) [Brucker et al., 1999].

A set of n projects has to be planned simultaneously in the RCMPSP. For each project the following information is given: an earliest release time, 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. Activities of different projects may require the same shared resource at the same time. In order to enable comparing alternative solutions of a given RCMPSP, some local and global performance criteria are defined. Commonly used global criteria are the Total Makespan (TMS) and the Average Project Delay (APD). Both will be considered in this paper.

A large variety of algorithms have been developed for the RCMPSP. Exact approaches have been reported in the literature, including the early work on multi-project scheduling by Pritsker et al. [1969], who propose

a zero-one programming approach. Other mathematical approaches are presented in [Deckro et al., 1991, Chen, 1994, Vercellis, 1994]. These exact approaches are limited to solving small problem instances, whereas heuristics, mainly based on priority rules, have a better scalability. Kurtulus and Davis [1982] study a number of priority rules and introduce some new ones (e.g. SASP, MAXTWK, etc.) in order to minimize the average project delay in multi-project scheduling problems. Lova and Tormos [2001] analyse the effect of the schedule generation schemes (serial or parallel) and different priority rules (MINLFT, MINSLK, MAXTWK, SASP or FCFS) for single and multi-project problems. The average project delay is considered for the multi-project case. Goncalves et al. [2008] present a genetic algorithm for the RCMPSP in order to minimize a weighted objective function (including tardiness, earliness and flow time deviation). A random key representation of the problem is combined with a parameterized schedule generator. The approach is tested on a set of randomly generated problems, which are unfortunately not publicly available. Browning and Yassine [2010] present a comprehensive analysis of 20 priority rules for the RCMPSP. They compare the priority rules on 12,320 test problems generated with different characteristics. Kumanam and Raja [2011] propose a heuristic and a memetic algorithm for scheduling multiple projects in order to minimize the total makespan.

In addition, several decentralized approaches exist, where information asymmetry is assumed. This decentralized version is denoted as 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 addressed by integrating a metaheuristic called the centralized restart evolution strategy (RES), with an efficient decentralized electronic negotiation mechanism [Homberger, 2007, 2012]. Adhau et al. [2012] 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 (virtual) project managers negotiate for each activity to be scheduled, the present paper focuses on coordination through learning a simple sequence game managed by a trusted third party or mediator. A serial schedule generation scheme, which is adopted from single project scheduling [Kolisch and Hartmann, 1999], first builds a solution for the RCMPSP. 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. Each project manager simultaneously chooses a place in this sequence. Since all project managers should find a unique place for their project, they in fact play a dispersion game [Grenager et al., 2002]. The goal of each project manager reduces to, 1) building an efficient precedence feasible activity list locally and 2) learning a suitable place in the overall sequence of activity lists. Both goals are learned simultaneously and iteratively by using a global reinforcement signal, i.e. a value based on the APD or TMS of the schedule that was generated at the previous time step. The project managers use a network of simple reinforcement learning devices called learning automata [Thathachar and Sastry, 2004] for learning a good quality sequence of their own activities. The sequence game is played with a probabilistic version of the Basic Simple Strategy (BSS), which guarantees the players to coordinate within logarithmic time.

The contribution of this paper is threefold, 1) The approach is a fruitful application of a learning approach [Wauters et al., 2010] to RCMPSP. 2) Experiments show that the sequence learning approach has a large positive effect on the minimization of the average project delay. In fact, the combination of local reinforcement learning and the sequence learning game significantly improves the best known results for all the MPSPLIB[1]. datasets. Many new best solutions for the APD objective were produced, dominating the MPSPLIB benchmark website with 104 best solutions out of 140. In addition to the excellent performance on the APD objective, several new best solutions were generated for the TMS objective as well. Due to its heuristic nature, the learning approach scales very well. 3) The difference is investigated between the proposed collaborative scheduling approach, and one where the project managers act selfishly.

This paper is structured as follows. Section 2 describes the RCMPSP. The solution representation and encoding together with the new learning approach are given in Section 3. Section 4 presents the experiments and results. Conclusions are drawn in Section 5.

2 Problem description

The RCPSP 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.

[1] MPSPLIB, `http://www.mpsplib.com`, January 21, 2014

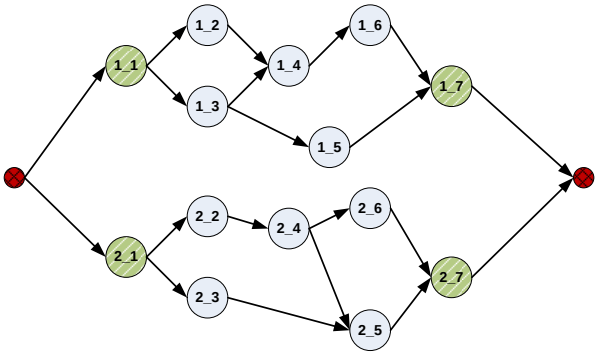

Fig. 1 Multi-Project activity-on-node diagram for an example with 2 projects (7 activities each, dummy activities included)

[2007], and on a problem notation similar to Homberger [2012]. We however do not focus on the decentralized aspect of the problem (i.e. information asymmetry is not addressed), but instead strive for the best global performance.

The RCMPSP 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. The first and last activities of the projects are dummy activities with a zero duration and no resource requirements. The dummy activities determine the start and end of the project. Figure 1 presents an example of a multi-project activity-on-node diagram with two projects and seven activities each (dummy activities included). The precedence relations are denoted by arrows.

Each project i has an arrival (or release) time ad_i, which is the earliest point in time for the project to start. Project 1 is always released at $ad_1 = 0$, while the other projects may 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 RCMPSP 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 time s_i that is equal to the starting time s_{i1} of the dummy starting activity. Similarly, each project i has a finishing time f_i 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. The makespan MS_i of a project i is defined as the difference between the project's finishing time and the project's arrival time $MS_i = f_i - ad_i$. The project delay PD_i of project i is defined as the difference between the project's makespan and its 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 by the critical path method and 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 time of all single projects. Note that the earliest arrival time over all projects is always zero, because $ad_1 = 0$. The average makespan is the average of the makespans of all the projects ($AMS = \sum_{i=1}^{n} MS_i/n$). The average project delay is the average of the project delays of all the projects ($APD = \sum_{i=1}^{n} PD_i/n$). Finally the DPD is calculated as the standard deviation of the project delays of all the projects $DPD = \sqrt{\sum_{i=1}^{n} (PD_i - APD)^2/(n-1)}$. The remainder of the paper only considers the APD and TMS objectives.

3 Learning approach

This section describes the learning approach for the RCMPSP, which is based on Wauters et al. [2011, 2012]. It provides details of the applied encoding, schedule construction, configuration and learning techniques.

3.1 Solution representation and encoding

Multi-project schedules are generated by organising 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 a precedence feasible ordered list of activities belonging to one project. When all ALs are combined according to 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 gen-

eration scheme (SGS) [Kolisch and Hartmann, 1999] 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. The technique is also named justification by Valls et al. [2005]. [Lova et al., 2000] use the forward-backward technique in a multicriteria heuristic for multi-project scheduling. The method 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 determine the order in which a new activity list is generated 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.

3.2 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 are to schedule projects either sequentially or interleaved.

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

Many hybrid combination techniques are applicable, but only the two extremes that are able to incorporate sequence information are considered in the present paper. Another option is to directly learn the sequence of all activities from all projects. This has the following drawbacks. First, it would drastically increase the search space of the learning methods. Second, as shown in this paper, the project order has a significant impact on the APD objective, and is therefore important to focus on. Furthermore, it must be noted that the interleaved method is only of interest when the number of activities among projects does not differ much.

Figure 2 illustrates the two methods: sequential and interleaved with a simple two-project example (A and B). The POL for this example determines that project A comes before project B.

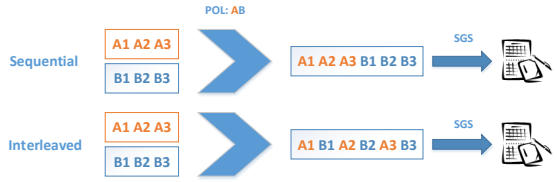

Fig. 2 Combining activity lists: sequential and interleaved

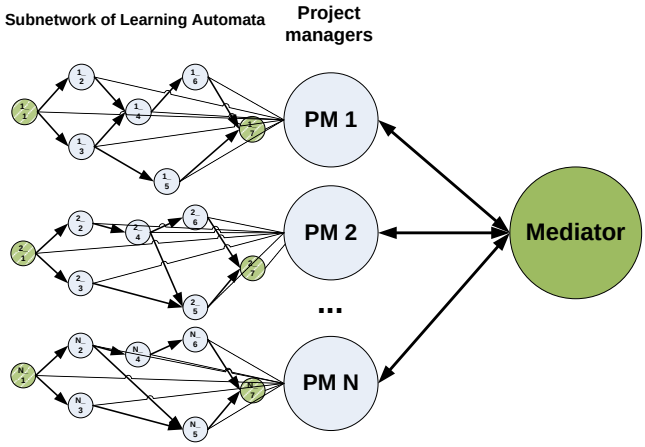

Fig. 3 Decomposition approach

The performance difference between the sequential and interleaved approach will be explained in Section 4.2.

3.3 Learning activity lists using learning automata

The problem is decomposed into two subproblems: generating a feasible activity list for each project and generating an order of projects. The first problem is solved by the project managers (one for each project), while the second is coordinated by the mediator. Figure 3 illustrates this decomposition. The present section describes the main functionality of the project managers, while Section 3.4 considers the role of the mediator.

Each project manager is responsible for generating a precedence feasible activity list for his own project. The task is performed by a multi-agent learning method. It is based on the technique presented in [Wauters et al., 2009, 2011] for learning sequences in the multi-mode resource-constrained project scheduling problem (MR-CPSP). Each activity in the project uses simple reinforcement learning devices called learning automata (LA) [Thathachar and Sastry, 2004] to learn an order of its successor activities. In contrast to [Wauters et al., 2009, 2011], multiple modes are not part of the problem addressed in this paper, therefore the learning components for the modes can be removed from the solution approach. Only the components for learning a prece-

dence feasible activity list are used.

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 based on a specific learning algorithm or reinforcement scheme. The literature provides several reinforcement schemes with different convergence properties. 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 behind these schemes is to increase the probability to select an action when it results in a success and to decrease it when the response is a failure. In this work the actions of the LA are permutations of successor activities. Each activity maintains an LA with probability distribution p for selecting these successor permutations, and updates p with the following update scheme:

$$p_m(it+1) = p_m(it) + \alpha_{reward}(1-\beta(it))(1-p_m(it)) - \alpha_{penalty}\beta(it)p_m(it) \quad (1)$$

$$p_j(it+1) = p_j(it) - \alpha_{reward}(1-\beta(it))p_j(it) + \alpha_{penalty}\beta(it)[(r-1)^{-1} - p_j(it)] \quad (2)$$

with $p_m(it)$ ($p_j(it)$) the probability for selecting the m-th (j-th) successor permutation at iteration it. The constants α_{reward} and $\alpha_{penalty}$ 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 ($L_{R-\epsilon P}$). $\beta(it)$ is the reward received by the reinforcement signal for an activity chosen at iteration it. r is the number of successor permutations. Equation 1 is used to update the probability p_m for selecting the m-th successor permutation, where Equation 2 is used for updating all other probabilities $p_j, \forall_{j \neq m}$.

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 based on the reward-inaction update scheme is able to control a finite Markov Chain 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 [Wauters et al., 2009, 2011].

The virtual project manager uses a network of LA to build an activity list. Further on, the manager logs his own performance for which he receives information from the mediator, and updates the LA according to equations 1 and 2. The reward-inaction update scheme is applied because of the above mentioned theoretical convergence properties. The simple reinforcement signal is the following. When compared with the best schedule obtained in the schedule generation process, the newly generated schedule resulted in an objective value which was:

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

Algorithms 1 and 2 describe the manager method for producing a precedence feasible activity list. The activity list is generated by visiting one activity after another. The project manager applies Algorithm 1 for constructing an activity list using a network of learning automata. A_1 (A_N) is the first (last) dummy activity of the project. Each node in this network uses Algorithm 2. The method *chooseOrder*() in Algorithm 2 applies a fitness proportionate selection method[2] on the probabilities of the LA for selecting a permutation of the activities successors (denoted as *Order*). This permutation of successors is afterwards traversed one by one on each query to the method to return the next successor activity (A_{next}). The learning rates (α) of the LA are additional parameters of the algorithm, which balance the quality and speed of learning.
An activity is eligible if all its predecessors have been visited.

3.4 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 (POL). Based on the retrieved activity lists and the POL, the mediator constructs a sequence of activity lists (SAL). This sequence is used to build a multi-project schedule with the serial schedule generation scheme. The mediator returns the quality of the obtained schedule to the project managers.

[2] also known as roulette wheel selection

Algorithm 1 Project Manager: generating an activity list

Require: Project data and Algorithm parameters
Ensure: A precedence constraint feasible activity list
  initialize *ActivityList*
  $A_{current} \Leftarrow A_1$
  while Not all activities in *ActivityList* **do**
    $A_{next} \Leftarrow$ Algorithm 2(*ActivityList*,$A_{current}$)
    if A_{next} is eligible AND not $NULL$ **then**
      $A_{current} \Leftarrow A_{next}$
    else
      $A_{current} \Leftarrow$ random eligible Activity
    end if
  end while
  return *ActivityList*

Algorithm 2 Next activity determination

Require: *ActivityList* (partial), $A_{current}$
Ensure: A_{next}
  if $A_{current}$ is A_N **then**
    $A_{next} \Leftarrow NULL$
  else
    if first visit of $A_{current}$ **then**
      add $A_{current}$ to the *ActivityList*
      Order $\Leftarrow$ chooseOrder() using LA
      $A_{next} \Leftarrow$ first activity in *Order*
      store *Order*
    else
      $A_{next} \Leftarrow$ next activity in stored *Order*
    end if
  end if
  return A_{next}

The order determination process to construct a POL resembles a game, more specifically a dispersion game [Grenager et al., 2002] where the number of positions is equal to the number of project managers. To construct a POL, all project managers need to select a distinct position or a maximally dispersed assignment of projects to positions. For example, if three project managers select the following distinct positions: project manager 0 selects position 1, project manager 1 selects position 2 and project manager 2 selects position 0, then the POL becomes $[2, 0, 1]$.

The Basic Simple Strategy (BSS) introduced in [Grenager et al., 2002], allows to select maximally dispersed actions in a logarithmic (as a function of the number of actions) number of rounds, where a naive approach would be exponential. BSS uses a uniform selection, the method used in the current work incorporates the project managers 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 positions for all project managers, and the set of all possible positions Q, each project manager using the PBSS will:

- select position $q \in Q$ with probability 1 in the next round, if the project manager selected position q in outcome o, and if the number of project managers selecting position q in outcome o is exactly 1 ($n_q^o = 1$).
- select a position from the **probabilistic** distribution over positions $q' \in Q$ for which $n_{q'}^o \neq 1$, otherwise.

n_q^o is the number of times position q was selected in outcome o. The probabilistic distribution over positions is obtained from the project managers' experience with previous decisions. A technique from an earlier study [Wauters et al., 2012] for fast permutation learning was adopted. Where Wauters et al. [2012] focus on the general problem of learning permutations, the method is here applied to learn positions in the POL. Each project manager maintains a single learning automaton to adjust its preferences for a place in the POL. Moreover, this method requires information about the positions that were selected uniquely during the PBSS procedure. The project managers play a dispersion game, and the mediator provides the needed unique position selection information, i.e. positions $q \in Q$ in outcome o for which $n_q^o = 1$ holds.

An example with four project managers illustrates the PBSS. The first two project managers (0 and 1) have strong preferences to be first in the POL. Their preference probabilities are: [0.5, 0.25, 0.125, 0.125]. The third project manager (2) has uniform preference probabilities [0.25, 0.25, 0.25, 0.25]. The fourth project manager considers it better to be scheduled later and has preference probabilities [0.1, 0.2, 0.3, 0.4]. Based on these preference probabilities, the project managers play a PBSS. Figure 4 shows a possible outcome, i.e. the positions chosen in each of the 3 required rounds. In the first round, only project manager 3 selected position 3. Consequently, project manager 3 will select unique position 3 again in the next rounds. In the second round the first three project managers select a new position, different from 3. Project manager 2 selects the unique position 2 and will select position 2 again in the next rounds. In the third round the first two project managers select a new position different from 2 and 3. At this point all project managers have selected a unique position resulting in the POL [1, 0, 2, 3].

Figure 5 shows the performance of PBSS in function of the number of project managers n, i.e. necessary number of rounds to find a maximally dispersed solution. For each number of project managers ($n \in \{5, 10, 20, 50, 100, 500, 1000\}$) a PBSS was performed 100 times with non-uniform probability vectors. A logarith-

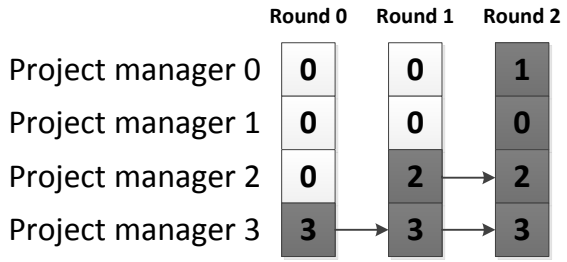

Fig. 4 Example of the PBSS procedure with 4 project managers. Grey squares indicate unique positions.

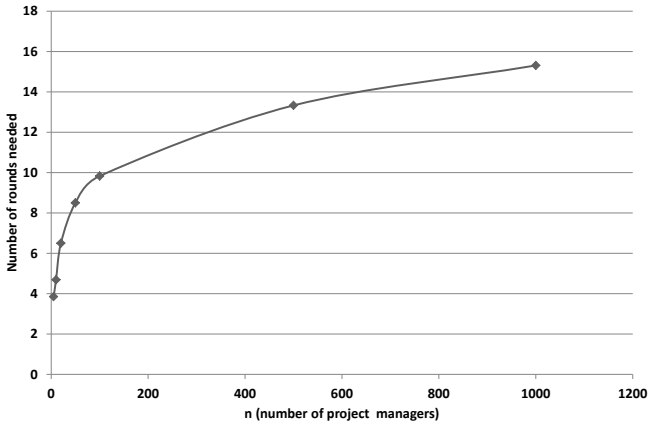

Fig. 5 Logarithmic performance of the probabilistic basic simple strategy

mic behavior for PBSS[3] can be observed. 1000 project managers on average require only 15.31 rounds for finding a POL.

3.5 Full approach

Algorithm 3 describes the full learning approach. The following steps are repeated in each iteration of the learning approach:

- generate a feasible activity list for each project.
- generate a project order list (POL).
- generate a new schedule
- update the learning automata.

The algorithm stops when a predefined maximum number of schedule generations is exceeded. Due to the forward-backward SGS method, multiple schedules can be generated at each iteration. The learning approach behaves randomly at the start, while after a few iterations schedules of higher quality are generated.

[3] The logarithmic behavior for the uniform BSS was already shown by Grenager et al. [2002].

Algorithm 3 Learning approach

Require: A RCMPSP
Ensure: A feasible multi-project schedule
  Initialize project managers and mediator
  while Maximum number of schedule generations not exceeded **do**
    {***generate a feasible activity lists for each project***}
    for all Project Managers **do**
      Generate a precedence feasible activity list (see Algorithm 1)
      Send the generated activity list to the mediator
    end for
    {***generate a project order list (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
    {***generate a new schedule***}
    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
    {***update the learning automata***}
    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 $\beta(it) = 1$
        $obj_{best} = obj_{new}$
      end if
    end for
  end while
  return $SCHED_{best}$

4 Experiments and results

The present section presents the experimental results of the proposed learning approach on standard benchmark problems for the RCMPSP. The performance is compared to the state of the art for both the APD and TMS objectives.

4.1 Problem Instances

The proposed approach is assessed for the same 140 (60+80) DRCMPSP instances as in [Homberger, 2012] and [Adhau et al., 2012]. These instances are available from the Multi-Project Scheduling Problem Li-

brary[4]. To the best of the authors' knowledge, this is the only publicly available multi-project scheduling benchmark website, where problem instances can be downloaded and results of different methods can be compared. The instances were originally designed for decentralized methods, but can also be used to evaluate centralized methods. The library contains 20 datasets with 140 instances based on multiple RCPSP instances[5]. 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 1 shows the problem datasets and their properties, taken from Homberger [2012]. 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, 2012].

4.2 Sequential vs interleaved scheduling

Figure 6 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 between the produced schedules can be noticed. Figure 7 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. Similar graphs have been generated 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. This shows that when the APD objective is considered, the project order is very important, and thus motivates the dispersion game approach.

4.3 Comparison with best known results

The experiments in this section were executed on an Intel Core 2 Duo PC (3.3Ghz, 4GB RAM), and all the methods are implemented in the Java 1.6 programming language. The stopping criterion was set to 100,000 schedule generations, so as to enable fair comparison with the 100,050 schedule generations used by all the methods compared by Homberger [2012]. After several preliminary experiments, a good value for the learning rate was determined: $\alpha_{reward} = 0.001$. Higher values converge too quickly to suboptimal solutions, while lower values are not able to learn anything useful within the given stopping criterion. This value [$\alpha_{reward} = 0.001$] was used for updating all the learning automata. A sequential activity list combination method is selected when APD is the main objective, while an interleaved activity list combination method is selected for the TMS objective.

The first analysis considers the APD objective. The learning approach is compared with the best solutions reported in the literature, obtained by either centralized (RES, SASP) or decentralized approaches (CMAS/ES, CMAS/SA, MAS/CI, DMAS/ABN) [Homberger, 2012, Adhau et al., 2012]. The APD results are shown in Table 2, which denotes the average values over all instances in each problem subset APD_{AV}, the average calculation time for the learning approach in seconds, and the percental difference with the literature's best. The learning approach obtains an average APD improvement over all instances of 24.5% compared to the best in the literature. It is interesting to notice that the achieved improvements are even better (up to 41%) for the very large instances (up to 20 projects with each 120 activities to be planned), indicating good scalability. Figure 8 compares the average project delay over all instances of the learning approach with all the methods from the literature. It is clear that a significant APD improvement over other methods from the literature is realized. Many new best APD solutions were found, showing 104 best APD solutions out of 140 instances on the MPSPLIB benchmark website (last check on June 17, 2013). It must be noted that the literature's best was often found by decentralized methods assuming information asymmetry. These new best results, calculated by the centralized learning approach, may somehow be

[4] MPSPLib, `http://www.mpsplib.com`, January 21, 2014

[5] PSPLib, `http://www.om-db.wi.tum.de/psplib/main.html`, January 21, 2014

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 1 Problem datasets and their properties [Homberger, 2012]. *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.

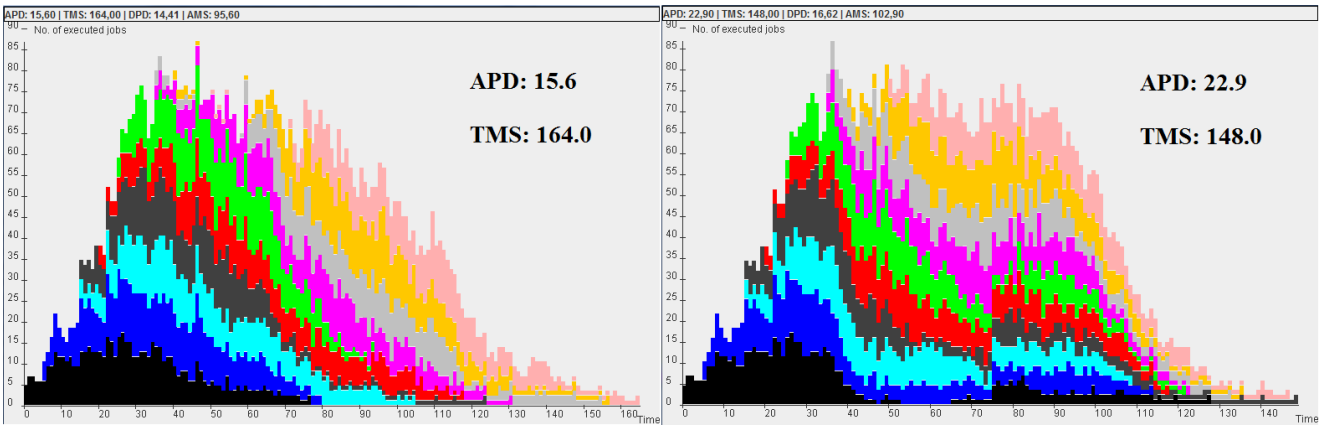

Fig. 6 Schedule comparison, sequential (left) vs interleaved (right).

used as a benchmark for decentralized methods in the future.

When the total makespan objective (TMS) is considered, the learning approach performs best with an interleaved activity list combination method. Table 3 shows TMS results obtained with the interleaved method. The learning approach is able to keep up with the best in the literature. Even a slight improvement can be observed on some datasets. Figure 9 compares the total makespan over all instances. The performance of the learning approach is very competitive with the other methods reported in the literature. Only RES [Homberger, 2007] reaches a slightly better average TMS over all instances. Nevertheless, several new best TMS solutions were obtained. Moreover, the learning approach generated some Pareto non-dominated solutions, which improve on both objectives (APD and TMS). Figure 10 illustrates a multi-objective comparison of both the TMS and APD results obtained with other methods on instance j90_a10_nr5.

The results mentioned in Table 2 and Table 3 have been validated and uploaded to the Multi-Project Problem Library website (`http://www.mpsplib.com`).

4.4 Comparison with priority rules

This section compares the learning-based optimization approach with well known priority rules for the RCMPSP. [Browning and Yassine, 2010] summarizes all priority rules for the RCMPSP. All priority rules from Browning

	Literature	Learning approach		
Problem subset	(APD_{AV})	(APD_{AV})	Time (s)	Difference
MP30_2	12.4	**11.2**	21.4	9.7%
MP90_2	5.6	**5.3**	83.4	5.4%
MP120_2	60.8	**49.4**	184.2	18.8%
MP30_5	16.7	**15.4**	79.7	7.8%
MP90_5	8.9	**7.8**	297.2	12.4%
MP120_5	65.1	**48.5**	673.6	25.5%
MP30_10	84.4	**52.0**	250.8	38.4%
MP90_10	46.1	**31.8**	1448.6	31.0%
MP120_10	131.0	**100.0**	2110.6	23.6%
MP30_20	177.8	**111.4**	868.4	37.3%
MP90_20	30.2	**17.6**	3305.2	41.8%
MP120_20	31.8	**28.2**	6946.4	11.3%
MP90_2AC	127.8	**104.3**	144.3	18.4%
MP120_2AC	47.0	**35.2**	154.2	25.1%
MP90_5AC	287.8	**244.6**	553.1	15.0%
MP120_5AC	247.5	**178.8**	871.1	27.8%
MP90_10AC	244.9	**169.4**	1414.5	30.8%
MP120_10AC	151.0	**96.9**	2618.5	35.8%
MP90_20AC	146.4	**85.4**	4252.1	41.7%
MP120_20AC	237.1	**158.6**	8586.0	33.1%

Table 2 Comparison of average project delay with the best in the literature

	Literature	Learning approach		
Problem subset	(TMS_{AV})	(TMS_{AV})	Time (s)	Difference
MP30_2	63.6	**63.4**	21.4	0.3%
MP90_2	**107.2**	107.6	76.8	-0.4%
MP120_2	**169.0**	173.4	180.2	-2.6%
MP30_5	89.2	**87.2**	64.0	2.2%
MP90_5	**123.6**	123.8	260.6	-0.2%
MP120_5	**182.2**	191.0	558.0	-4.8%
MP30_10	180.6	**180.4**	196.2	0.1%
MP90_10	**180.4**	182.4	899.0	-1.1%
MP120_10	**279.8**	287.8	1463.4	-2.9%
MP30_20	**327.4**	330.8	635.6	-1.0%
MP90_20	161.6	**161.4**	1961.8	0.1%
MP120_20	187.6	**186.0**	2848.2	0.9%
MP90_2AC	**232.2**	235.7	144.3	-1.5%
MP120_2AC	**139.2**	147.2	156.4	-5.7%
MP90_5AC	**538.2**	551.7	548.1	-2.5%
MP120_5AC	**480.7**	493.1	842.2	-2.6%
MP90_10AC	**458.3**	469.4	1220.2	-2.4%
MP120_10AC	**350.7**	357.3	1857.1	-1.9%
MP90_20AC	**285.9**	286.7	2521.3	-0.3%
MP120_20AC	**506.4**	512.9	5040.6	-1.3%

Table 3 Comparison of total makespan with the best in the literature

and Yassine [2010] have been implemented and tested on the MPSPLib benchmark. It was chosen because it also considers the average project delay as an objective. The priority rules are listed in Table 4. A parallel schedule generation scheme is used. For more details about these priority rules we refer the reader to Browning and Yassine [2010].

Figure 11 shows the average APD over all instances for the considered priority rules and compares it with the learning approach. The learning approach performs better than any of the priority rules. Furthermore, if one would combine all priority rules, it would give an average APD of 124.72, which is still significantly worse than the result of the learning approach. The best priority rules are LCFS and SASP, while the worst priority rules are FCFS and WACRU.

4.5 Effect of the sequence game

Figure 12 shows the difference between a completely random SAL (random feasible activity lists and random project order), and a SAL constructed with the se-

FCFS	first come first served
SOF	shortest operation first
MOF	maximum (longest) operation first
MINSLK	minimum slack
MAXSLK	maximum slack
SASP	shortest activity from shortest project
LALP	longest activity from longest project
MINTWK	minimum total work content
MAXTWK	maximum total work content
RAN	random
EDDF	earliest due date first
LCFS	last come first served
MAXSP	maximum schedule pressure
MINLFT	minimum late finish time
MINWCS	minimum worst case slack
WACRU	weighted activity criticality and resource utilization
TWK-LST	MAXTWK and earliest late start time (2-phase rule)
TWK-EST	MAXTWK and earliest early start time (2-phase rule)
MS	maximum total successors
MCS	maximum critical successors

Table 4 Implemented priority rules from Browning and Yassine [2010].

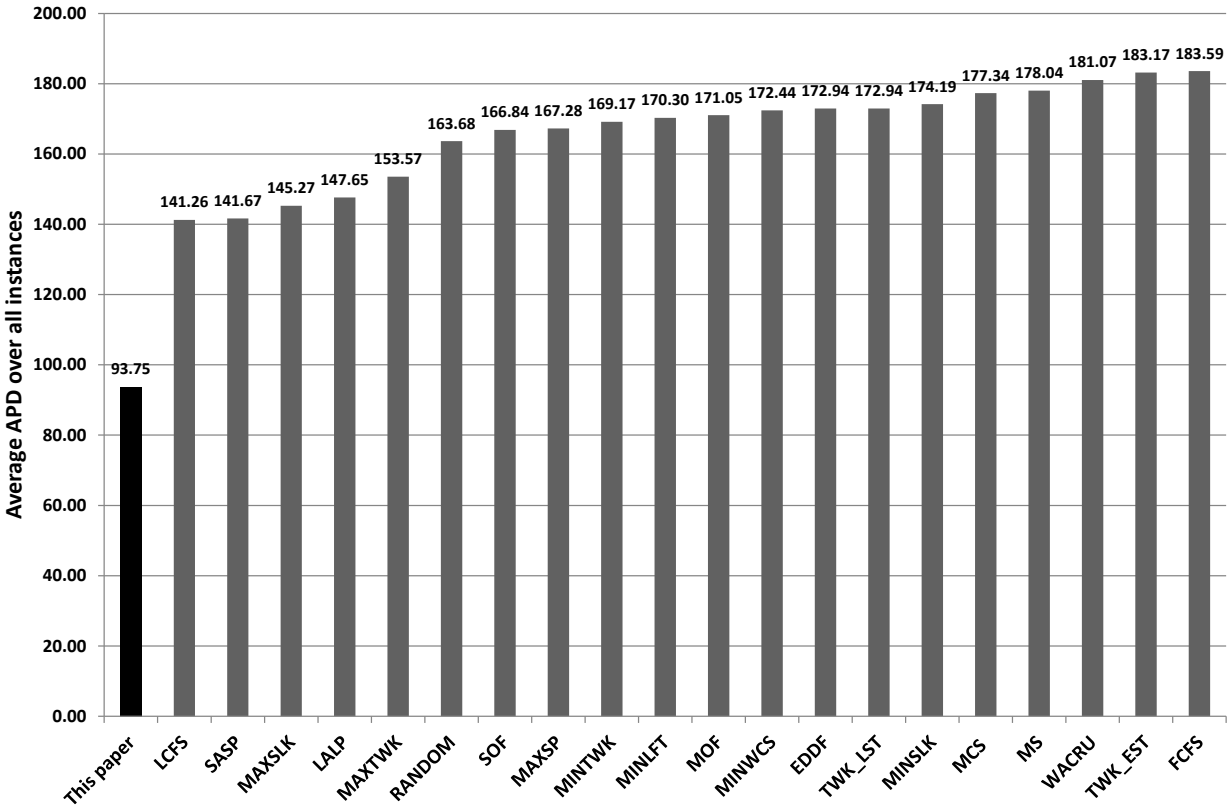

Fig. 11 Comparison of the learning approach with priority rules from Browning and Yassine [2010].

quence game (dispersion game). The figure shows average APD evolution per 1000 iterations, for one RCMPSP instance (j30_a10_nr4). The noticeable APD increase is due to learning a good sequence via a dispersion game.

4.6 Global vs individual reward

In real world situations, each project manager can have individual objectives and goals. Whereas, the RCMPSP studied in the present papers considers a global objective to be optimized. The proposed approach can be classified as a central method, since all scheduling infor-

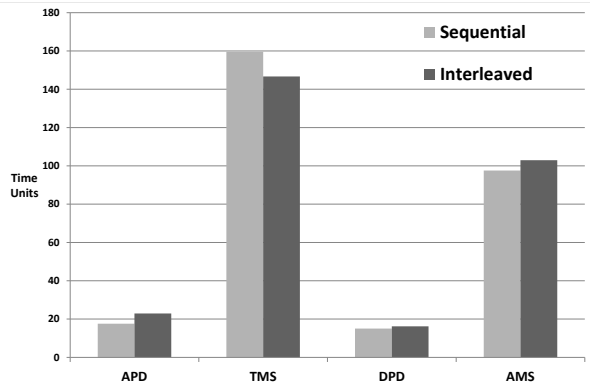

Fig. 7 Performance comparison between sequential and interleaved scheduling

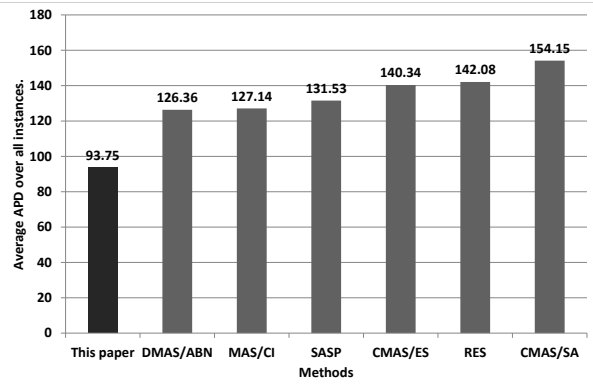

Fig. 8 Comparison of algorithms with respect to average project delay over all problem instances.

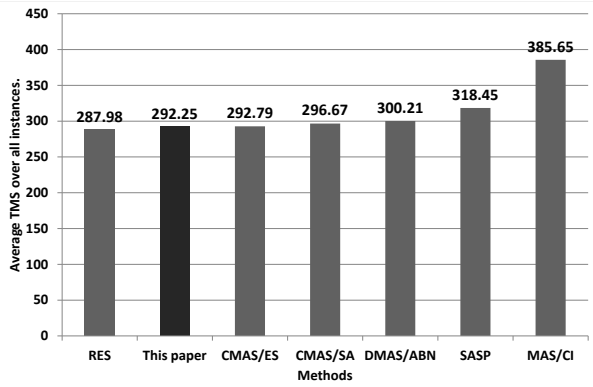

Fig. 9 Comparison of algorithms with respect to total makespan over all problem instances.

mation is known by the mediator. The decisions about activity priorities are made locally by each project manager, and can be parallelized in order to speed up decision making. Moreover, the project managers themselves do not receive scheduling information about the other projects. Only a single 0-1 global reward signal is provided to them. In order to assess the importance of global knowledge, an experiment was conducted to study the difference between a system with a global

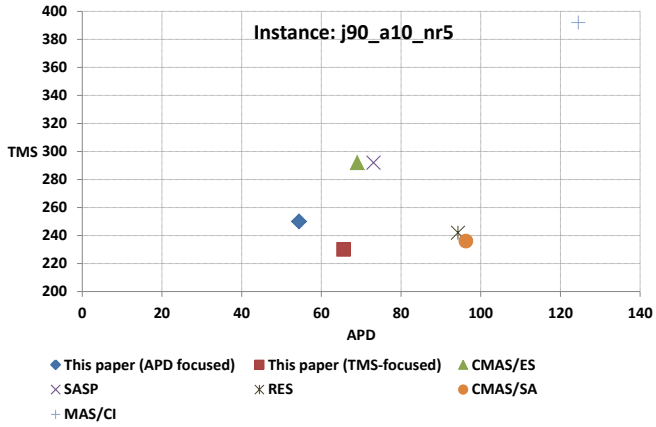

Fig. 10 Multi-objective comparison on instance j90_a10_nr5.

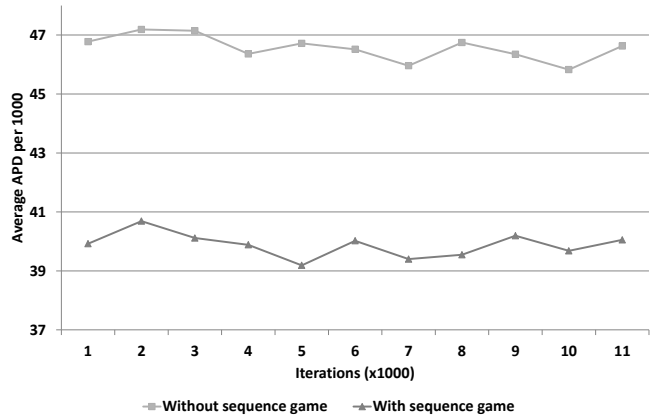

Fig. 12 Effect of the sequence game on instance j30_a10_nr4.

reward signal and one with an individual reward signal. The project managers receive a binary reward signal based on the APD objective value in the global reward signal test. Contrastingly, the test with individual reward signal provides a binary reward signal based on the managers' individual project delay (PD_i). The project managers do not know how the global system is evolving during the search. Both reward signals are $\beta(t) = 1$, if the objective value was improved, and $\beta(t) = 0$ otherwise.

Figure 13 shows the difference in solution quality (APD) between the two aforementioned systems. The following settings were used: stopping criterion $= 100,000$ schedule generations, and learning rate $\alpha_{reward} = 0.001$ for updating all the learning automata. The figure clearly shows the negative impact of losing global knowledge on all problem subsets. The solutions obtained with the global reward signal are on average 30% better than the solutions obtained with the individual reward signal. This result, showing the importance of the global reward signal, was somehow expected since the project managers have conflicting goals. The result also shows

the importance of a good coordination mechanism among the project managers, in this work handled by a dispersion game.

5 Conclusion

A new learning-based optimization approach to the resource-constrained multi-project scheduling problem (RCMPSP) was introduced. Project managers learn a sequence of activities using a network of learning automata. In the meantime, project managers also play a sequence game, in 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 was used to coordinate the dispersion game, and to build a multi-project schedule using a serial schedule generation procedure. 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 learning 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. Many new best schedules on the benchmark 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.

In addition to the presented sequential and interleaved activity list combination methods, one could test the whole spectrum of combinations. A multi-objective optimization version of the RCMPSP, where Pareto non-dominated solutions need to be found, opens perspectives for further improvement.

References

S. Adhau, M.L. Mittal, and A. Mittal. A multi-agent system for distributed multi-project scheduling: An auction-based negotiation approach. *Engineering Applications of Artificial Intelligence*, 25(8):1738 – 1751, 2012.

T. R. Browning and A. A. Yassine. Resource-constrained multi-project scheduling: Priority rule performance revisited. *International Journal of Production Economics*, 126(2):212 – 228, 2010.

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

V.Y.X. Chen. A 0-1 programming model for scheduling multiple maintenance project at a copper mine. *EJOR*, 76 (1):176–191, 1994.

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.

R. F. Deckro, E.P. Winkofsky, J. E. Hebert, and R. Gagnon. A decomposition approach to multi-project scheduling. *European Journal of Operational Research*, 51(1):110 – 118, 1991.

J.F. Goncalves, J.J.M. Mendes, and M.G.C. Resende. A genetic algorithm for the resource constrained multi-project scheduling problem. *European Journal of Operational Research*, 189(3):1171 – 1190, 2008.

T. Grenager, R. Powers, and Y. Shoham. Dispersion games: general definitions and some specific learning results. In *In AAAI 2002*, pages 398–403. AAAI Press, 2002.

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

J. Homberger. A (μ, λ)-coordination mechanism for agent-based multi-project scheduling. *OR Spectrum*, 34 (1):107–132, 2012.

R. Kolisch and S Hartmann. Heuristic algorithms for the resource-constrained project scheduling problem: Classification and computational analysis. In Jan Weglarz, editor, *Project Scheduling*, volume 14 of *International Series in Operations Research & Management Science*, pages 147–178. Springer US, 1999.

S. Kumanam and K. Raja. Multi-project scheduling using a heuristic and memetic alogrithm. *Journal for Manufacturing Science and Production*, 10 (3-4): 249256, 2011.

I. Kurtulus and E. W. Davis. Multi-project scheduling: Categorization of heuristic rules performance. *Management Science*, 28(2):161–172, 1982.

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.

M.L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *In Proceedings of the Eleventh International Conference on Machine*

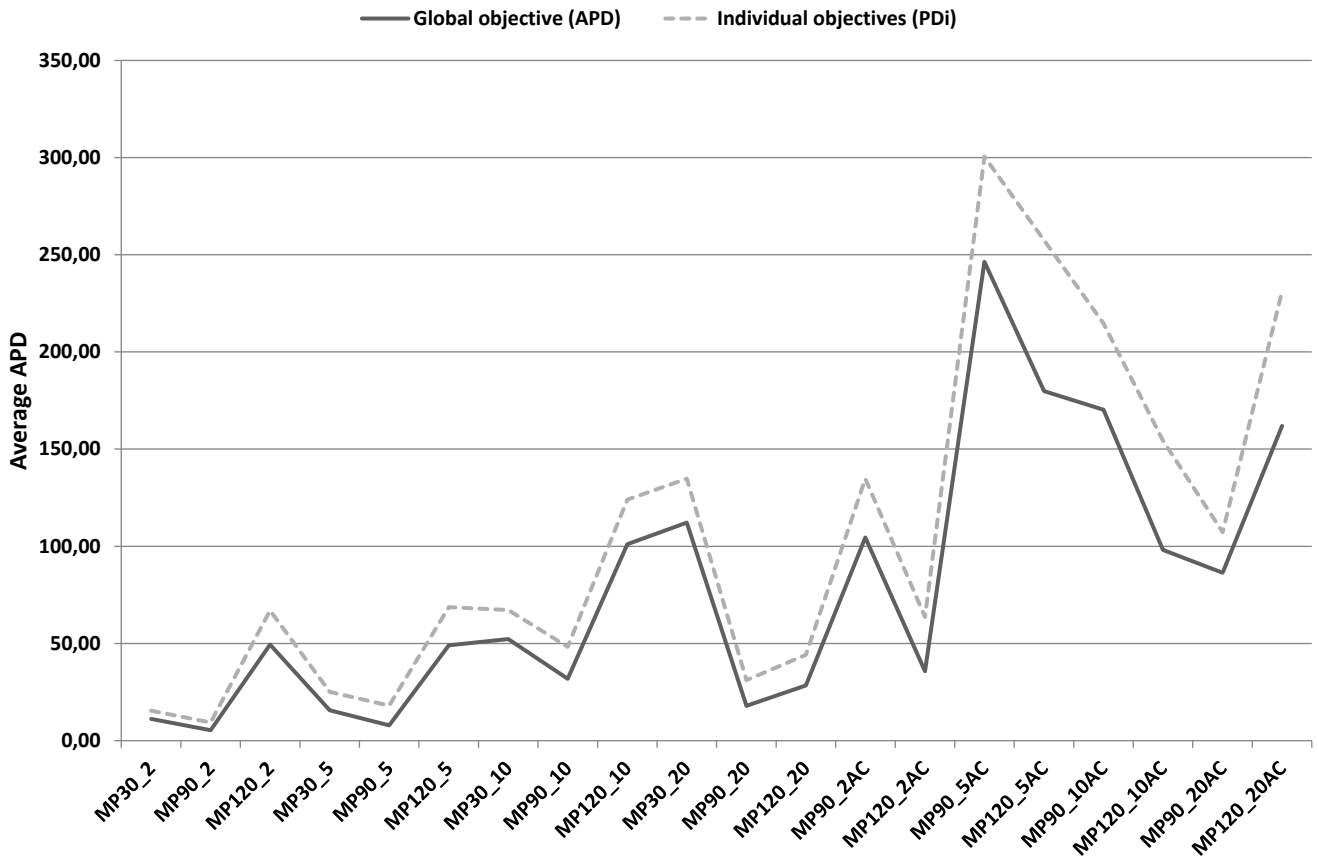

Fig. 13 Global (APD) vs individual (PD_i) reward signal for each problem subset.

Learning, pages 157–163. Morgan Kaufmann, 1994.

A. Lova and P. Tormos. Analysis of scheduling schemes and heuristic rules performance in resource-constrained multiproject scheduling. *Annals of Operations Research*, 102(1-4):263–286, 2001.

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.

A. A. B. Pritsker, L. J. Watters, and P. M. Wolfe. Multiproject scheduling with limited resources: A zero-one programming approach. *Management Science*, 16(1): 93–108, 1969.

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

V. Valls, F. Ballestin, and S. Quintanilla. Justification and RCPSP: A technique that pays. *European Journal of Operational Research*, 165(2):375 – 386, 2005.

C. Vercellis. Constrained multi-project plannings problems: A lagrangean decomposition approach. *European Journal of Operational Research*, 78(2):267 – 275, 1994.

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 *Optimisation in Multi-Agent Systems Workshop at the Conference on Autonomous and Multi-Agent Systems (AAMAS 2009)*, Budapest, Hungary, May 2009.

T. Wauters, K. Verbeeck, P. De Causmaecker, and G. Vanden Berghe. A game theoretic approach to decentralized multi-project scheduling (extended abstract). In van der Hoek, Kaminka, Lesprance, Luck, and Sen, editors, *Proc. of 9th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, number R-24, pages 1415–1416, Toronto, Canada, May 2010.

T. Wauters, K. Verbeeck, G. Vanden Berghe, and P. De Causmaecker. Learning Agents for the Multi-Mode Project Scheduling Problem. *Journal of Operational Research Society: special issue on heuristic optimization*, 62:281 – 290, 2011.

T. Wauters, K Verbeeck, P Causmaecker, and G. Vanden Berghe. Fast permutation learning. In Youssef Hamadi and Marc Schoenauer, editors, *Learning and Intelligent Optimization*, Lecture Notes in Computer

Science, pages 292–306. Springer Berlin Heidelberg, 2012.

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