RAIRO-Oper. Res. 46 (2012) 23–39
DOI: 10.1051/ro/2012006

RAIRO Operations Research
www.rairo-ro.org

A POLARIZED ADAPTIVE SCHEDULE GENERATION SCHEME FOR THE RESOURCE-CONSTRAINED PROJECT SCHEDULING PROBLEM

Reza Zamani[1]

Abstract. This paper presents a hybrid schedule generation scheme for solving the resource-constrained project scheduling problem. The scheme, which is called the Polarized Adaptive Scheduling Scheme (PASS), can operate in a spectrum between two poles, namely the parallel and serial schedule generation schemes. A polarizer parameter in the range between zero and one indicates how similarly the PASS behaves like each of its two poles. The presented hybrid is incorporated into a novel genetic algorithm that never degenerates, resulting in an effective self-adaptive procedure. The key point of this genetic algorithm is the embedding of the polarizer parameter as a gene in the genomes used. Through this embedding, the procedure learns *via* monitoring its own performance and incorporates this knowledge in conducting the search process. The computational experiments indicate that the procedure can produce optimal solutions for a large percentage of benchmark instances.

Keywords. Project-scheduling, resource-constrained, heuristics.

Mathematics Subject Classification. 90B35.

1. Introduction

Market globalization makes projects more specialized and diversifies the resources needed by different activities of these projects. This increases the pressure on companies to manage their projects effectively, implying that those companies

Received December 22, 2010. Accepted March 12, 2012.

[1] Building 39, SISAT, Faculty of Informatics, Wollongong University, Wollongong, 2522 NSW, Australia. reza@uow.edu.au

Article published by EDP Sciences

which fail to do so will not remain competitive. Hence, market globalization can be considered as the main driving force of employing sophisticated software packages for project management. Modules used in these packages can range from those balancing the cost and duration of projects in activity networks [33] to those minimizing the cost in PERT networks [7]. One of the main modules needed to be imbedded in these software packages is, however, a procedure for solving the Resource-Constrained Project Scheduling Problem (RCPSP).

Being of particular interest to project managers, civil engineers, and system planners, the RCPSP is a general problem that has a variety of applications in manufacturing, production planning, project management, and various other areas in industry. Not only is the RCPSP strongly NP-hard, but it is also hard to approximate [27]. In effect, the RCPSP is an easy-to-state but hard-to-solve combinatorial optimization problem that is defined as follows:

There is a project with J activities labelled $j = 1, 2, \ldots, J$. The precedence relations between the activities are represented by the set of immediate predecessors. For the activity j, its set of immediate predecessors is represented by P_j and indicates that the activity j cannot start unless all activities in P_j have been completed. Two extra activities are added to the project, representing the "start" and the "end" of the project and shown by 0, and $J + 1$, respectively. These two activities have zero duration and no resource requirements.

The number of resource types is shown by K, and the availability of resource type k is represented by R_k. Moreover d_j and r_{jk} represent the duration of activity j and its required for resource k, respectively. The goal is to calculate the starting times of activities, S_j $j = 1, 2, \ldots, J + 1$, subject to resource and precedence constraints so that the value of S_{J+1} is minimized.

In this paper, we present a Polarized Adaptive Scheduling Scheme (PASS), which has been particularly designed to synergize the capabilities of the parallel [17] and serial (*cf.* Brooks in [1]) schedule generation schemes. Based on a given problem instance, the PASS can adaptively be adjusted through selecting a settle point between its two poles. This adjustment is controlled by a parameter, called "*polarizer*". The driving force behind employing the polarizer parameter is that using adaptive parameters in areas like Lagrangean relaxation has proved effective [41].

The polarizer parameter can vary between zero and one, and its value determines how similar to each of the two poles the PASS should perform. This parameter has a key role in the PASS performance, and when it is set to zero, the PASS exactly performs like the parallel method. On the other hand, when this parameter is set to one, the PASS exactly performs like a serial method in which the duration of activities show their priorities.

Having the capability of performing in a spectrum marked by the two poles of the serial and parallel schedule generation schemes provides a flexibility that neither of these two poles holds. With using a novel genetic algorithm that in each of its genomes embeds the polarizing parameter as a gene and never degenerates, we also incorporate the PASS into a self-adaptive search procedure that can conduct

the first two of the following three mappings used in the literature to tackle the RCPSP.

The first mapping is involved with assigning a value to each activity in determining its priority in utilizing resources. The result of this mapping can be represented either in a form of a *random key* representation or an *activity list* [22]. While random key representation is a list of numbers each representing the priority of its associated activity explicitly, *activity list* is a precedence-feasible order of activities, with representing such priorities implicitly.

The second mapping converts a random key representation or an activity list to a schedule. For this purpose in the literature, two major generation schemes have been used: serial and parallel [19]. However, here we use a hybrid of these two schemes that is determined by a continuous parameter between zero and one. Finally, the third mapping, which has a potential to enter into the genomes as a gene but, because of its computational expenses, has currently been excluded is involved with *neighbourhood schemes*. This mapping, based on any given neighbourhood scheme, can systematically modify the priorities mentioned.

The rest of the paper is as follows. Section 2 presents the related work, and Section 3 discusses how the PASS operates. A genetic algorithm based on the PASS is presented in Section 4, and the computational results are discussed in Section 5. Finally, Section 6 provides concluding remarks and sketches a major direction for further work.

2. Related work

The procedures developed to solve the RCPSP are divided into the two main classes of exact and heuristic methods. The efficient exact methods for the problem have been presented in [3, 8–11, 15, 26, 29, 34, 39]. The main drawback of the exact methods is that they cannot be applied to practical problems and can tackle only small- and medium-sized problems.

Heuristic methods, on the other hand, at the cost of losing the guarantee of optimality can tackle practical problems and provide acceptable solutions. Heuristic procedures have been extensively studied in [6, 12, 14, 18, 21, 23, 28, 31]. Kolisch and Padman (2001) have categorised heuristics as priority-rule based, truncated branch and bound, disjunctive-arc based, and metaheuristic techniques.

Two schedule generation schemes, namely *parallel* and *serial*, comprise the building blocks of a large number of heuristics presented for the RCPSP. Theoretical results on these two building blocks have been provided in [19]. In effect, it was the development of these two schedule generation schemes which marked the beginning of the first non-exact methods for the problem, namely priority-rule-based heuristics.

While the parallel scheme builds non-delay schedules of good average quality, the serial scheme builds active schedules. The combination of two facts supports the application of the serial scheme to small- and medium-sized instances and that of the parallel scheme to large-sized instances. The first fact is that the set of active

schedules is a superset of *non-delay* schedules and the second fact is that the set of *active schedules* always contains an optimal schedule.

In the following years of the development of serial and parallel schemes, a large number of priority rules were proposed and tested. In [20], priority rules have been classified as dynamic *vs.* static, intensive information processing *vs.* light information processing, network-based *vs.* resource-based *vs.* time-based, and finally lower-bounds-based *vs.* non-lower-bounds-based.

Regardless of the schedule generation scheme employed to generate a schedule, a method called iterative forward/backward [24] has proved to be very effective when applied to the resultant schedule. The method applies serial forward backward scheduling iteratively until no further improvement in the project duration is achieved. In [35, 38], it has been shown that even if this technique, in the backward/forward direction, is applied only once to a schedule, it may still produce significant improvements. The performances of several major heuristics for the RCPSP have been investigated in [22].

Priority rules play a vital role in the performance of many procedures and since the number of these rules is limited, a *sampling* methodology is employed. In effect, *sampling* is one of the successful methodologies incorporated in the heuristics tackling the RCPSP. It makes the same priority rule capable of generating a large number of different schedules by biasing its selection of the priority rule. The sampling methods have been examined in [14]. These methods generally use the serial and/or the parallel scheme as well as one priority rule, and depending on how the biasing is achieved, they have been distinguished in three major classes: (i) random sampling, (ii) biased random sampling, and (iii) regret-based biased random sampling.

A method called ß-biased random sampling has been used in [37], which is an extension of biased random sampling. In this method, with the probability of ß the activity with the least order is selected as the next activity on the list and with the probability of $(1-ß)$, the biased random sampling is used for selecting the next activity on the list. Because of their flexibility and simplicity, random sampling and multi-pass methods have predominantly been used to tackle the RCPSP. In [35], a multi-pass method that uses random sampling and performs forward-backward passes has been presented.

Compared to applying sampling methods, manipulating neighbourhood schemes is computationally more resource intensive but produces higher-quality solutions. It is worth mentioning that very large scale neighbourhood searches have proved to be very efficient on other problems like partitioning [30]. In effect, a large percentage of local search methods presented for the RCPSP are based on an elementary transformation changing some elements of a solution without altering its general structure.

In [4], the transformation mechanisms for neighbourhood techniques, typically used in metaheuristics, have been categorized into four categories: (i) *substitution*, (ii) *swapping*, (iii) *shifting*, and (iv) *inversion*. Generally, with each metaheuristic, different neighbourhood techniques can be employed to change its activity list or

random key representation. In [2], a simulated annealing algorithm is presented that employs an efficient neighbourhood generation technique to change an activity list, and in [5], an ant colony optimization algorithm for the RCPSP has been presented that effectively directs the stochastic building of solutions toward attaining high-quality solutions. The simultaneous cooperation of agents in solving the RCPSP can also be found in an architecture presented in [16].

Incorporating proper intelligence into local search through integrating it with other search mechanisms is the core of hybrid approaches dealing with the RCPSP. As an effective hybrid, ANGEL [36] combines nature-inspired algorithms of ant colony optimization and genetic algorithms with the local search strategy, employing both forward and backward scheduling.

As it has been mentioned in [40], one of the main factors in achieving peak performance for the RCPSP is self-adaptation of a search method to a problem instance. In [13], a self-adapting genetic algorithm has been presented that makes use of two different decoding mechanisms as well as an additional gene in the representation that determines the decoding mechanism employed.

A parameterized schedule generator has been presented in [25] that employs a genetic algorithm and aims at generating schedules that can vary between non-delay and active schedules. The same with the parallel schedule generation scheme, it is a pure time-incrementing procedure and in neither of its stages employs any activity-incrementing mechanism. That is why its operations have no similarity with those of the serial schedule generation scheme. This is in contrast with the PASS that uses a polarizer parameter to combine the serial and parallel schedule generation schemes.

For its generated schedules to vary between non-delay and active schedules, that algorithm defines the delay of each iteration in the same chromosome on which the priorities of activities are represented and, considering the number of activities of the project as n, it needs n parameters for representing these delays. These n parameters are aimed at making the generated schedules parameterized in the sense that no resource is kept idle for more than a predefined period if it could start processing some activity. This is also in contrast with the PASS, which needs only a single parameter: the polarizer.

3. The polarized adaptive scheduling scheme (PASS)

The PASS is a hybrid schedule generation scheme that integrates the serial and parallel generation schemes into a single stochastic technique. It has been designed to behave in a way in which none of the two generation schemes can do. Moreover, neither any genetic algorithm based on the PASS can degenerate nor any local search built upon it can be trapped in local optimality. The reason simply is that two different runs of the PASS with the same value of the polarizing parameter and the same set of priorities of the activities can lead to slightly different schedules.

We describe the PASS in two phases. In the first phase, a general description is provided with clarifying the key terms needed for describing the details, and in

the second phase, the detailed operations of the PASS and its similarities with the serial and parallel schedule generation schemes are discussed.

3.1. A general description and clarifying the key terms

Before providing a general description, we briefly describe the essence of the serial and parallel generation schemes. The serial method is an activity-based schedule generation scheme. To find a schedule in an activity-based algorithm, firstly activities are prioritized and a topologic order is determined. Then, based on the order found, activities, one at a time, are examined and start in the earliest possible time allowed by both resource and precedence constraints. The starting of activities, one after another, proceeds until all activities are scheduled.

In an activity-based algorithm, time is not forwarded systematically and, instead, all possible time-slots are implicitly examined to find the starting time of the activity at hand. In other words, to examine whether an activity is resource-feasible, any activity-based algorithm has to use a profile of resource usage in the entire duration of the project and update it whenever an activity is scheduled.

On the contrary, the parallel method is a time-based algorithm. To find a solution *via* a time-based algorithm, again all of the activities should be prioritized, but the priority list is not required to be precedence-feasible. By forwarding time units systematically, all activities whose predecessors have been completed are considered and resources are allocated to each activity based on its priority. Hence, a time-based algorithm does not need to keep the resource profile and instead solely needs resource levels in the current instance of time.

The same with the serial and parallel generation schemes, the PASS requires each activity to have a priority, and it is based on these priorities that, subject to the precedence and resource constraints, the procedure finds a schedule. The PASS starts with a null schedule, in which none of the activities have started yet, and proceeds through creating consecutive *partial schedules* until a *full* schedule is generated.

Each partial schedule classifies all activities of the project into three different categories, namely completed, activated, and not-yet-started, with the combination of completed and activated activities showing the scheduled activities. In the process of converting a null schedule to a full schedule, each partial schedule differs from its previous partial schedule by having one or more extra completed activities. In effect, a full schedule is a partial schedule in which the set of completed activities comprises the entire activities of the project.

Each partial schedule is constructed from the amendment of its previous partial schedule in a *stage* marked by time τ representing the earliest completion time of one or several activities together in that previous partial schedule. Hence, it is the combination of these recently completed activities that is considered for identifying each stage. In each partial schedule, the term *eligible* activity is used for any activity whose entire predecessors have been completed and, therefore, it can start if its resource requirements are available. Since by the completion of an

activity (or several activities together), their successors may become eligible, at each stage the eligible activities are updated and are kept sorted based on their priorities.

Three sets, namely *Activated* (A), *Frontier* (F), and *Unsaddled* (U) play distinct roles in the operations of the PASS. At each stage, the union of the sets F and U represents all eligible activities. In other words, in each of the stages, activities whose entire predecessors have already been completed at time τ are either in the F or U set. The difference between these two sets is that whereas the set F includes eligible activities which, in the case of availability of resources, can start immediately, the set U is comprised of activities that, despite being eligible, have been temporarily prevented from starting.

The prevention of activities of U from starting is aimed at keeping the resources available for possible urgent activities that may become eligible in the next stages. By urgent activities we mean those activities that their postponing in the corresponding setting can increase project duration. Since the main responsibility of the activities placed in the set U is to maintain some resources for future urgent activities, as soon as these activities become resource-infeasible they will leave U and enter in F. It should be noted that when an activity, in U, becomes resource-infeasible, it has fulfilled its responsibility of permitting other activities to use the resources it has kept idle.

Unlike the sets F and U which are involved with eligible activities, the set A represents all activities that are activated. Whenever an activity in the set A is completed, some new activities may become eligible, which consequently enter in either the F or U set. The polarizer parameter, which is between zero and one, plus a random component collectively determine whether an activity should enter in the F or in the U set. The larger the value of the polarizer parameter, the larger the percentage of eligible activities that enter in the U set.

It should be noted that if in all stages, all eligible activities move into the F set, the PASS behaves like the parallel method with the same input priorities. On the other hand, if all activities move into the U set, the PASS behaves like the serial method with the duration of activities set as their priorities.

In the PASS, not every activity entering the set U is delayed and, therefore, the completion of an activity in U can mark a new stage. For instance, suppose that the current value of τ is 80, and the first activity in the set A is supposed to be completed at time 86. Now if there is an activity the set U that can be comleted at time 83, such an activity can use idle resources and become completed before the first activity in A finds the chance of completion. In this case, τ will change from 80 to 83, and not to 86. The rationale behind advancing time to 83 is that based on the completion of this activity, the algorithm may be able to find an activity that becomes precedence-feasible at time 83 with all its required resources being available, and such a ready-to-start activity can be added to the set A at time 83.

Hence, if any activity in the set U can be completed before an activity in the set A is completed, it is the completion of this activity in the set U that marks the new stage. To find such an activity, the procedure removes resource-infeasible

activities from the set U and calculates the earliest completion time of activities left in the set. In the cases where this calculated value is smaller than the earliest completion time of activities in the set A, the corresponding activities in the set U is considered as completed. Such completed activities leave the set U without entering in the set F, updating resources based on their resource requirements. Hence, only those activities of U are delayed whose required resources can assist other activities to start in later stages and others are considered as started as soon as they enter U. This will be further elucidated in the pseudocode presented later.

3.2. The detailed operations and similarities with the serial and parallel generation schemes

Having clarified the notions of partial schedule, stage, eligible activities, A, U and F sets, and the variable τ, we now describe how, at each stage, the sets A, U and F are manipulated and the value of τ is increased until all activities are completed. We also show that while in some periods of generating a schedule, the PASS performs like the serial method, in the other periods, it operates similar to the parallel method.

In presenting the detailed operations of the PASS, first we have to emphasize that the similarities between the operations of the PASS with those of the parallel and serial methods stem from the fact that in some stages, the PASS utilizes the entire resource profile whereas in the other stages it uses only a snapshot of this profile. Resource profile, Π_Profile, which shows the amounts of resources left during the project execution, is used in the serial method and its snapshot, Π_τ, which shows resource levels at time τ, is used in the parallel method. In effect, in the parallel method, there is no need to know about the history of resource usage, Π_Profile, and only does the current availability of resources matters, Π_τ.

By having the capability of using both Π_Profile and Π_τ and switching from one to another, the PASS can adapt itself to the problem at hand, and present a behaviour between those of the parallel and serial schemes. It performs both in time-based and activity-based levels and can be considered as a time-activity-based procedure. It is time-based because during its operations, it takes time forward; and it is activity-based because the set U does not let all eligible activities start and, in the middle of time-based operations, manages to start them in an activity-based manner, based on their durations as their priorities.

Activities in the set U are manipulated *via* a structure that synchronizes the operations between the two different parts of the PASS to avoid the possibility of left-shifting. This structure uses idle resources and enables the PASS to consider an activity in the set U completed. The feature of considering some activities of U completed not only prevents left shifting but increases the speed of the procedure as well.

Figure 1 presents a C-type pseudocode for the PASS, with the assumption that the value of the polarizer parameter, which is between 0 and 1, has been set to a user-defined value. It is worth mentioning that the proper value of the

polarizer parameter, which is normally less than 0.15, is determined by in the genetic algorithm which controls the PASS. In effect, this parameter is one of the genes of the encoding employed in the genetic algorithm presented later.

As is shown in Figure 1, the pseudocode starts by initializing the three sets of A, F and U to a null set. Then line 4 sets the current time, τ, to zero and line 5 finds the immediate successors of activity 0. As stated, activity 0 shows the start of the project and, therefore, its immediate successors are all activities of the project which have no predecessor. These activities are the only eligible activities at time 0 and that is why line 6 places all of them in the set F. The main loop of the procedure starts at line 7 and ends at line 48. As is shown, the loop updates the sets A, F, and U in different stages until the ending activity, $(J + 1)$, is placed either in the set A or in the set U. The updating of the sets A, F, and U are performed as follows.

line 9 sorts activities in the set F based on their priorities. Then, in lines 10 through 17, similar to what is performed in the parallel method, a compact set of activities, called C, that can start at time τ are created. The term compact indicates that adding any extra activity from the set F to the set C causes resource-infeasibility for the set C. In other words, the term compact signifies that the set C is the largest possible set that can include its current activities, and adding any other activity of F to C will cause the lack of possibility of starting all activities of C together at time τ.

Unlike in the parallel method, not all activities of the compact set selected start immediately. In effect, based on the polarizer parameter, line 18 divides the compact set of activities, C, into C_A and C_U, with the contents of C_A stating immediately, and hence joining the set A, and the contents of C_U joining the set U.

The routine used in line 18 for splitting C, into C_A and C_U operates as follows. Among the members of the compact set, C, each member with a probability shown by the polarizer parameter enters in the set C_U and otherwise enters in the set C_A. For instance, if the polarizer parameter is 0.15, every member of C with 15 percent chance enters in the set C_U and with 100-15, 85, percent of chance enters in the set C_A. This random component makes the method stochastic and causes two different runs of the PASS with the same input to generate slightly different schedules.

The pseudocode shows that no activity joining U can start unless it has become either resource-infeasible in some stage or it has become resource-feasible in its entire duration since it has been delayed. Whereas in the first case, the delayed activity has permitted other activities to use the resources it has kept idle, in the second case no activity has been able to use these idle resources. That's why in the second case, the delayed activity will be considered as completed. Hence, any activity joining U in some stage, will either be delayed and join the set F in later stages, in lines 23 and 24, or it will be considered as completed, at line 41.

line 23 finds the resource-infeasible activities of the set U and after placing these activities in the set F, at line 24, they are removed from the set U, at line 25. It should be noted that, as line 21 represents, any activity joined the set U has

```
 1 PASS()
 2 {
 3   A=F=U=∅;
 4   τ=0;
 5   Find I0, the set of immediate successors of activity 0;
 6   F=F ⋃ I0;
 7   do
 8   {
 9     Sort activities in F based on their priorities;
10     for all j in F, based on their priorities
11     {
12        If (j is resource feasible)
13        {
14            C = C ⋃ {j}; F = F\{j};
15            Update ∏τ ;
16        }
17     }
18     Based on the routine presented, split C into CA and CU;
19     A= A ⋃ CA, and set the starting time of activities in CA as τ;
20     Update the ∏-Profile based on the activities in CA
21     U = U ⋃ CU;
22     Update ∏τ;
23     U1 = the set of resource infeasible activities of U
24     F= F ⋃ U1;
25     U=U\U1
26     TA = min_{j∈A} {Sj + dj} & TU = min_{j∈U} {S'j + dj}
27     if (TA < TU)
28     {
29        τ= TA;
30        Remove Completed activities from A;
31        Update ∏τ;
32     }
33     else if (TA > TU)
34     {
35        τ= TU ;
36        Sort activities in U based on their priorities;
37        for all j in U, based on their priorities
38        {
39         if (j is resource feasible and can be completed at TU)
40         {
41          Set the starting time of j to τ minus its duration & U= U\{j};
42          Update ∏-Profile;
43         }
44        }
45     }
46     if (T= TU)do lines 29 through 31 as well as lines 36 through 44;
47     Update F;
48   }until (J+1∈ A ⋃ U)
49 }
```

FIGURE 1. The C-type pseudo-code of the PASS.

been initially resource-feasible and when, in a stage, it becomes resource-infeasible, it means that, at the cost of being delayed, it has reserved some resources and permitted other activities to start. That is why as soon as an activity becomes resource-infeasible, it joins the set F and leaves the set U, as lines 24 and 25 show.

Before describing the rest of the pseudocode, it should be emphasized that the main loop, which includes lines 7 through 48, is repeated for each single stage and the sets A, F, and U are updated in these stages. In each stage, line 26 computes the *release time* associated with the sets A and U, shown by T_A and T_U, respectively, with the release time of a set showing the time that the first activity in the corresponding set can be completed. Considering the facts that S_j shows the starting time, d_j shows the duration, and S'_j shows the time of joining activity j to the set U, at line 26, $S_j + d_j$ shows the completion time of activity j, and $S'_j + d_j$ shows the completion time of activity j assuming that it has been started as soon as it has joined the set U. In general, since activities joining U can be delayed, the real starting time of activity j can be greater than S'_j.

It is also worth noting that before computing T_U, all resource-infeasible activities have left the set U, at line 25, and only those activities have been considered that are still resource-feasible. In the cases where a set is empty, line 26 returns its release time as zero. Since the time associated with each stage is shown with τ, depending on the comparison made at line 27, either line 29 or line 35 updates the value of τ based on the values of T_A or T_U, respectively.

The comparison made at line 27 not only determines how time is forwarded but it leads to three different ways of updating the sets as well. When T_A is less than T_U, the completed activites at time τ are removed from the set A and Π_τ is updated, lines 30-31, and when T_A is greater than T_U, activities in the set U are sorted based on their priorities and all resource-feasible activities which can be completed at time T_U are considered completed and leave the set U, lines 37 through 44. The only case left is when T_U is equal to T_A, which is treated in line 46 where all operations performed in lines 29 through 31 and those in lines 36 through 45 will be performed jointly.

line 47 does the final updating needed for the next stage, and is involved with adding new eligible activities to F. The reason for this updating is that the completion of one activity (or several activities together) leads to the creation of new eligible activities and lines 30 and/or 41 have managed to complete some activities.

Through consecutive stages, the pseudocode completes eligible resource-feasible activities one after another until it manages to place the end activity, $J + 1$, either in the set A or in the set U, and this marks the end of the project, tested at line 48. The output of the pseudocode is the starting times of activities calculated at lines 19 and 41. As is shown, whereas the starting time of any activity in C_A has been set to τ, at line 19, the starting time of any activity scheduled at line 41 has been set to τ minus its duration. This indicates that line 41 has scheduled activity j at a time strictly smaller than τ.

The PASS has been incorporated into a genetic algorithm to compute proper values for its polarizer parameter and its required input priorities. By keep-

ing the values of the polarizer parameter and the priorities (of activities) that have performed superbly and combining them through crossover operations, this PASS-based genetic algorithm operates in the direction of obtaining high quality solutions.

4. The PASS-based genetic algorithm

With having the complementary goals of maintaining diversity and converging towards high-quality solutions, a genetic algorithm has been used which controls the value the polarizer parameter as well as the priorities of activities. For the development of this genetic algorithm, two requirements have been considered: (i) an encoding mechanism for the representation of genomes, and (ii) a decoder that can convert a genome to a solution. In line with evolutionary process, coding space is separated from solution space and while modification occurs on the coding space, evaluation is performed on the solution space, with the codes being decoded by a decoder to their corresponding solutions. It is worth mentioning that in some typical optimization problems like the Travelling Salesman Problem (TSP) no decoder is required, and both coding space and solution space include various permutation of cities, but for the RCPSP, coding space is different from solution space.

As well as a decoder, we also need an initial pool and a mechanism for modifying the content of such a pool for further generations. Before describing the encoding mechanism, the decoder, the initial pool, and the mechanism of modifying the pool, the following key feature of the employed genetic algorithm requires highlighting. The employed genetic algorithm learns the best value of the polarizer parameter for a given problem instance by forcing the genomes to be extended with extra information representing the value of the corresponding polarizer parameter.

As stated in the literature survey, high-quality solutions presented for the RCPSP are generally the result of three consecutive mappings. The employed genetic algorithm controls the first two of these three mappings. Whereas the first mapping is about generating priorities of activities, and the second mapping transforms these priorities into the starting times of activities, the third mapping transforms the starting times into better starting times through a given neighbourhood scheme.

With respect to the employed genetic algorithm, the first mapping is done through performing crossover operations on the activity lists that have performed well. These activity lists implicitly show those priorities of activities that have led to suitable schedules. On the other hand, in the direction of performing the second mapping, the employed genetic algorithm adapts its behaviour to the problem at hand through the fine-tuning of the polarized parameter.

The encoding mechanism used for the representation of a genome is involved with $n+1$ genes, from which the first n genes show the priorities of activities, and the last gene represents the value of the polarizer parameter. It is through this

gene that the best value of the polarizer parameter is computed in an evolutionary process consisting of crossover operations and the survival of the fittest principle.

An entire genome is converted to a solution through a decoder. The employed decoder is a combination of the PASS followed by the single application of a backward/forward technique [35]. In the backward/forward technique, each activity's finish (start) time of a forward (backward) schedule establishes the activity's priority when the next backward (forward) schedule is computed.

The PASS, as the major part of the employed decoder, requires having access to the priorities of activities as well as the value of the polarizer parameter, and the corresponding genome provides such access. An activity list has been used to represent priorities. It should be noticed that any activity list, which is normally used in the serial schedule generation scheme, can be converted to a set of priorities, with the position of each activity in the list showing the priority of the corresponding activity. That is why in the employed encoding, an activity list occupies the first n genes.

Having described the employed encoding mechanism and the decoder, now we describe the initial pool and the mechanism of modifying the pool for further generations. With having the balancing goals of maintaining variety and converging towards high-quality solutions, mutation and crossover operators are the most typical mechanisms used in the modifications of pools. Whereas crossover operators exchange useful information between genomes and are the major driving force of changes, mutation operators, in general, can be used in background and are solely aimed at diversifying the search. In the employed genetic algorithm, the initial pool simply is filled with random genomes and is modified with a two-point crossover operator [13].

No mutation operator has been used because the randomness, discussed in separating the set C in the pseudocode, diversifies the search routinely. The employed randomness can even overcome the situations in which a local optimal solution is surrounded by several other local optimal solutions.

In effect, as was shown in the pseudocode, for the same set of priorities and the same value of the polarizer parameter, the PASS can produce slightly different schedules. Figure 2 depicts a situation in which the random component of the PASS can be beneficial in local searches.

Assuming the size of the pool as m, each genome in the pool is crossed over with a randomly selected genome from the pool, and, in this way, m new offspring genomes are created. Then among the $2m$ genomes, composed of m new offspring genomes and m parent genomes, the best m genomes, in terms of their fitness value, enter in the pool of the next generation.

Among the three mappings mentioned, the employed genetic algorithm in its current implementation does not use the third mapping. This has been decided after excessive experimentation and the reason is that the time-consuming nature of the third mapping prevents the procedure from efficient exploration of solution space, leading to low quality solutions. However, in the case of having faster

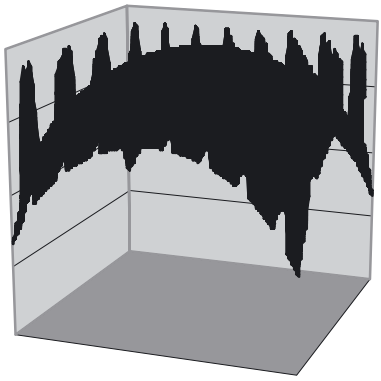

FIGURE 2. A local optimal solution surrounded by several other local optimal solutions.

computers, offspring genomes can be extended by selecting an appropriate type of a neighbourhood scheme embedded in the genomes.

Because of its capability in performing extensive changes in the performance of genomes, such a gene will play the role of a regulatory gene. With having such a regulatory gene, the genetic algorithm employed will find the chance of controlling all the three major mappings traditionally employed in solving the RCPSP.

5. Computational experiments

The computational experiments have been performed using a DELL PC (1.86 GHz) under Windows XP operating system and with 2 GB of RAM, from which never has more than 1 GB RAM been used. The procedure has been coded in C++ and its performance has been investigated on benchmark instances that all are obtainable from the library PSPLIB [23]. We have employed the standard sets $j30, j60$, $j90$ and $j120$ which have 32, 62, 92 and 122 activities, respectively. In each of the first three sets, there are 480 instances; whereas in the last set there are 600 instances. In all 2040 instances, the number of resource types is 4 and activity duration is in the interval $[10, 33]$.

For many of the instances in the sets $J60$, $J90$, and $J120$ optimal solutions are not known. Hence, for benchmarking our results, rather than using the criterion of deviation percentage from optimal solutions, we have used the CPM_dev% criterion, which measures the deviation percentage from CPM (Critical Path Method) lower bounds. For each problem instance, the CPM lower bound is simply calculated based on relaxing all resource constraints and solving the relaxed problem.

For setting the parameters of the procedure, without using the entire instances, we have fine-tuned the parameters for a small set of hard instances. The final results of our computational experiments have been compared with the results provided for LSSPER in [32].

TABLE 1. Comparing the performance of the PASS with that of LSSPER.

Set	PASS		LSSPER	
	Average CPM_dev%	Time (s)	Average CPM_dev%	Time (s)
$J30$	13.38	1	13.37	10
$J60$	10.64	3	10.81	38
$J90$	9.94	4	10.29	61
$J120$	31.22	8	32.41	207

LSSPER is a state-of-the-art procedure that for the first time has improved 14, 9, and 4 of the best solutions for the set $J60$, $J90$, and $J120$, respectively. By its developers, it has been run on a PC with 2.3 GHz speed and 1 GB RAM on the same benchmark instances to which the PASS has been applied. Table 1 compares the results. As the table indicates, with the exception of the set $J30$, for all the sets the PASS has produced solutions with higher quality in shorter time.

6. Conclusion

While examining the set of non-delay schedules is often preferable for large-sized instances, probing the set of active schedules can often perform better for small-sized instances. By parametrizing a schedule generation mechanism, the PASS can generate schedules between non-delay and active ones and therefore can adjust its behaviour for various problem instances. This ability combined with the stochastic nature of the PASS, will extend its power beyond the capabilities of the two ordinary generation schemes, and provides versatility in dealing with a wide range of problem instances.

The PASS has also been incorporated in a genetic algorithm that effectively controls two mappings to produce high-quality schedules. The employed genetic algorithm, without using any mutation operators, avoids degeneration and learns how to improve its own performance through dynamically revising both the activity list and the polarized parameter.

A key direction for future work can be involved with equipping the PASS with a flexible forward-backward passing that can continually and stochastically switch back and forth between the forward and backward passes. This can be done exactly in the same way that several stochastic switches were made between the serial and parallel schedule generation schemes. This continual and stochastic switching between the forward and backward passes has strong synergy with stochastic switches between the schedule generation schemes and seems of paramount importance in the development of a unified approach to solving the RCPSP.

References

[1] D.D. Bedworth and J.E. Bailey, *Integrated production control systems-management, analysis, design*. Wiley, New York (1982).

[2] K. Bouleimen and H. Lecocq, A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version. *Eur. J. Oper. Res.* **149** (2003) 268–281.

[3] P. Brucker *et al.*, A branch and bound algorithm for the resource-constrained project scheduling problem. *Eur. J. Oper. Res.* **107** (1998) 272–288.

[4] I. Charon and O. Hudry, The noising method: a new method for combinatorial optimization. *Oper. Res. Lett.* **14** (1993) 133–137.

[5] R.-M. Chen and S.-T. Lo, Using an enhanced ant colony system to solve resource-constrained project scheduling problem. *Int. J. Comput. Sci. Netw. Secur.* **6** (2006) 75–84.

[6] J.H. Cho and Y.D. Kim, A simulated annealing algorithm for resource constrained project scheduling problems. *Oper. Res. Soc.* **48** (1997) 736–744.

[7] P. Chrétienne and F. Sourd, PERT scheduling with convex cost functions. *Theor. Comput. Sci.* **292** (2003) 145–164.

[8] B. De Reyck and W. Herroelen, A branch-and-bound procedure for the resource-constrained project scheduling problem with generalised precedence relations. *Eur. J. Oper. Res.* **111** (1998) 152–174.

[9] E. Demeulemeester and W. Herroelen, A branch-and-bound procedure for multiple resource-constrained project scheduling problem. *Manage. Sci.* **38** (1992) 1803–1818.

[10] E. Demeulemeester and W. Herroelen, A new benchmark results for the resource-constrained project scheduling problem. *Manage. Sci.* **43** (1997) 1485–1492.

[11] U. Dorndorf, E. Pesch and T. Phan-Huy, A branch-and-bound algorithm for the resource-constrained project scheduling problem. *Math. Methods Oper. Res.* **52** (2000) 413–439.

[12] S. Hartmann, A competitive genetic algorithm for resource-constrained project scheduling. *Nav. Res. Logist.* **45** (1998) 733–750.

[13] S. Hartmann, A self-adapting genetic algorithm for project scheduling under resource constraints. *Nav. Res. Logist.* **49** (2002) 433–448.

[14] S. Hartmann and R. Kolisch, Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem. *Eur. J. Oper. Res.* **127** (2000) 394–407.

[15] W. Herroelen, B. De Reyck and E. Demeulemeester, Resource-constrained project scheduling: A survey of recent developments. *Comput. Oper. Res.* **25** (1998) 279–302.

[16] P. Jedrzejowicz and E. Ratajczak-Ropel, Agent-based approach to solving the resource constrained project scheduling problem, in *Adaptive and natural computing algorithms*. Springer LNCS **4431** (2007) 480–487.

[17] J. Kelley, *The critical-path method: resource planning and scheduling*, in *Industrial scheduling*, edited by J.F. Muth and G.L. Thompson. Prentice-Hall, New Jersey (1963) 347–365.

[18] R. Kolisch, *Project scheduling under resource constraints – efficient heuristics for several problem classes*. Heidelberg Physica (1995).

[19] R. Kolisch, Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation. *Eur. J. Oper. Res* **90** (1996) 320–333.

[20] R. Kolisch, Efficient priority rules for the resource-constrained project scheduling problem. *J. Oper. Manage.* **14** (1996) 179–192.

[21] R. Kolisch and A. Drexl, Adaptive search for solving hard project scheduling problems. *Nav. Res. Logist.* **43** (1996) 23–40.

[22] R. Kolisch and S. Hartmann, Experimental investigation of heuristics for resource-constrained project scheduling: An update. *Eur. J. Oper. Res.* **174** (2006) 23–37.

[23] R. Kolisch and A. Sprecher, PSPLIB – A project scheduling library. *Eur. J. Oper. Res.* **96** (1996) 205–216.

[24] K. Li, and R. Willis, An iterative scheduling technique for resource-constrained project. scheduling. *Eur. J. Oper. Res.* **56** (1992) 370–379.

[25] J.J.M. Mendes, J.F. Goncalves and M.G.C. Resende, A random key based genetic algorithm for the resource constrained project scheduling problem. *Comput. Oper. Res.* **36** (2009) 92–109.
[26] A. Mingozzi *et al.*, An exact algorithm for the resource-constrained project scheduling problem based on a new mathematical formulation. *Manage. Sci.* **44** (1998) 714–729.
[27] R.H. Möhring *et al.* Solving project scheduling problems by minimum cut computations. *Manage. Sci.* **49** (2003) 330–350.
[28] M. Mori and C. Tseng, A genetic algorithm for multi-mode resource constrained project scheduling problem. *Eur. J. Oper. Res.* **100** (1997) 134–141.
[29] T. Nazareth *et al.*, The multiple resource constrained project scheduling problem: A breadth-first approach. *Eur. J. Oper. Res.* **112** (1999) 347–366.
[30] J. Orlin *et al.*, Very large scale neighborhood search. *Int. Trans. Oper. Res.* **7** (2000) 301–317.
[31] L. Özdamar and G. Ulusoy, A survey on the resource-constrained project scheduling problem. *IIE Trans.* **27** (1995) 574–586.
[32] M. Palpant, C. Artigues and P. Michelon, LSSPER: Solving the resource-constrained project scheduling problem with large neighbourhood search. *Ann. Oper. Res.* **131** (2004) 237–257.
[33] D. Panagiotakopoulos, A CPM time-cost computational algorithm for arbitrary activity cost functions. *INFOR* **15** (1977) 183–195.
[34] A. Sprecher, Scheduling resource-constrained projects competitively at modest memory requirement. *Manage. Sci.* **46** (2000) 710–723.
[35] P. Tormos and A. Lova, A competitive heuristic solution technique for resource-constrained project scheduling. *Ann. Oper. Res.* **102** (2001) 65–81.
[36] L.-Y. Tseng and S.-C. Chen, A hybrid metaheuristic for the resource-constrained project scheduling problem. *Eur. J. Oper. Res.* **175** (2006) 707–721.
[37] V. Valls, F. Ballestin, and S. Quintanilla, A population-based approach to the resource-constrained project scheduling problem. *Ann. Oper. Res.* **131** (2004) 305–324.
[38] V. Valls, F. Ballestin and S. Quintanilla, Justification and RCPSP: A technique that pays. *Eur. Oper. Res.* **165** (2005) 375–386.
[39] R. Zamani, An effective near-optimal state-space search method: an application to a scheduling problem. *Artif. Intell. Rev.* **22** (2004) 41–69.
[40] R. Zamani, An accelerating two-layer anchor search with application to the resource-constrained project scheduling problem. *IEEE Trans. Evol. Comput.* **14** (2010) 975–984.
[41] R. Zamani and S.K. Lau, Embedding learning capability in lagrangean relaxation: An application to the travelling salesman problem. *Eur. J. Oper. Res.* **201** (2010) 82–88.