Integrating Iterative Crossover Capability in Orthogonal Neighborhoods for Scheduling Resource-Constrained Projects

Reza Zamani reza@uow.edu.au
Faculty of Informatics, University of Wollongong, Wollongong, 2522, Australia

Abstract

An effective hybrid evolutionary search method is presented which integrates a genetic algorithm with a local search. Whereas its genetic algorithm improves the solutions obtained by its local search, its local search component utilizes a synergy between two neighborhood schemes in diversifying the pool used by the genetic algorithm. Through the integration of these two searches, the crossover operators further enhance the solutions that are initially local optimal for both neighborhood schemes; and the employed local search provides fresh solutions for the pool whenever needed. The joint endeavor of its local search mechanism and its genetic algorithm component has made the method both robust and effective. The local search component examines unvisited regions of search space and consequently diversifies the search; and the genetic algorithm component recombines essential pieces of information existing in several high-quality solutions and intensifies the search. It is through striking such a balance between diversification and intensification that the method exploits the structure of search space and produces superb solutions. The method has been implemented as a procedure for the resource-constrained project scheduling problem. The computational experiments on 2,040 benchmark instances indicate that the procedure is very effective.

Keywords

Genetic algorithms, hybrid search, scheduling, resource-constrained projects, crossover operations, computational intelligence.

1 Introduction

The large number of multifaceted problems existing in a wide variety of fields ranging from telecommunication through logistics to production management has accelerated research on efficient search mechanisms. These mechanisms should ideally be able to effectively handle intelligent construction or generation of solutions for complex problems. A few examples of these problems can be named as locating strategic energy reserves, providing efficient telecommunication network settings, and scheduling enterprise projects under multiple resource constraints.

Having the paradigm of search as its fundamental concept, evolutionary computation is the study of computational intelligence via inspiration of multiagent systems, which integrate various information-processing systems toward providing better solution strategies in problem solving. With respect to this broad coverage, any multiagent heuristic, such as population-based strategies, which at each stage of the search manipulate a collection of solutions, can be referred to as an evolutionary method. As evolutionary computation has been involved with solving these complex problems,

the integration of population-based methods with other search techniques has become quite essential.

In the literature of search techniques, integrating local searches with population-based searches has been considered as one of the effective hybrids in search algorithms, and a marriage between a population-based global search and a heuristic local search has been referred to as a memetic algorithm (Moscato, 1989). The term memetic has been used to refer to cultural evolution, as opposed to genetic evolution, which indicates biological evolution. The analogy to cultural evolution stems from the fact that a solution in a population-based method can interact with another solution when the solution reaches a certain development through a local search.

Scatter search (Glover, 1994, 1998; Glover et al., 1995) can be considered as the other key term that refers to integrating local searches with population-based searches. In Glover et al. (1995) it was shown that despite significant differences existing between genetic algorithms and tabu search, as a metaheuristic that guides a local search, these two approaches have certain elements in common. In effect, it is through incorporating local searches into genetic algorithms that the major design of scatter search was formed. Advanced designs for scatter search were introduced in Marti et al. (2006).

Local searches are not the only major heuristic possible, because constructive methods play the same key role as local searches do. Whereas local searches operate based on the exploration of solution neighborhoods, constructive methods assemble a solution in an incremental style; and whereas local searches iteratively search for improved solutions and alter each candidate solution with one of its dominating neighbors, constructive methods use an evaluation function to iteratively select among different components. In general, constructive methods directly assemble approximate solutions through iteratively finding and fixing different components of a solution. In other words, they iteratively increase the size of a partial solution until a complete solution is obtained. In this sense, they contrast to local searches, which start from an initial complete solution and repeat altering its different components in the hope of producing a better complete solution.

1.1 Combining Biases Used in Genetic Algorithms and Local Searches

The point is not simply that genetic algorithms, local searches, and constructive heuristics conduct the search toward high-quality regions. The deeper issue is that these techniques impose certain biases toward directing the exploration of solution space. It is through combining these biases that one can expect to achieve high-quality solutions via integrating two or more effective search mechanisms.

In this paper, we present a method called crossed-over orthogonal neighborhoods explorer (CONE) which combines the biases used in genetic algorithms and local searches and through the combination of these biases directs the search from a large base of solutions to smaller bases. Whereas in its genetic algorithm component, the CONE generates further local optimal solutions when diversification is needed, in its local search component, it utilizes the synergy between two versatile neighborhood schemes and maintains necessary high-depth.

By balancing intensification versus diversification and the iterative crossover of locked solutions, the CONE is aimed at exploring the promising parts of search space. A solution is considered to be locked when the alternation of two orthogonal neighborhood schemes cannot improve it, and the neighborhood schemes employed are orthogonal in the sense that they share no single solution. The CONE also limits the

size of neighbors and improves their effectiveness through considering propagated infeasibility in preventing the formation of unfruitful swaps.

The CONE can be considered as a memetic algorithm (Moscato, 1989), which combines biases used in local search with those used in genetic algorithms. It is worth noting that, recently, the concept of memetic algorithms has shown successful applications in many different areas like continuous optimization (Molina et al., 2010), optimization with multiple criteria (Wanner et al., 2008), as well as classifying systems (Bacardit and Krasnogor, 2009). In effect, some of the most efficient procedures based on this concept are now used under the different titles of hybrid evolutionary algorithms (Deb and Sinha, 2010), cultural algorithms (Farahmand et al., 2010), and genetic local search (Fukunaga, 2008).

1.2 The Resource-Constrained Project Scheduling Problem

To show how effectively the CONE can perform, its application to the resource-constrained project scheduling problem (RCPSP) is presented. The reason why the RCPSP was selected is that solving the RCPSP is a growing challenge in many enterprise organizations, and the problem is of particular interest to systems planners, civil engineers, and project managers. In effect, the combination of its simple description and its inherent complexities has made this problem subject to extensive research in operations research, in general, and combinatorial optimization, in particular.

The single-mode RCPSP can be described as follows. In a project with J activities labeled $j = 1, 2, \ldots J$, in which the precedence relations between activities are represented by the set of immediate predecessors, P_j, the objective is to determine the starting times of activities, T_j, $j = 1, 2, \ldots, J$, subject to resource and precedence constraints, so that the project completion time is minimized.

For activity j, the set of immediate predecessors, P_j, indicates that it cannot start unless all activities in P_j have been completed. The starting and ending fictitious activities are shown by 0, and $J + 1$, respectively, and have zero duration and no resource requirements, indicating that the completion time of the project can be shown with T_{J+1}. In this setting, 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 request for resource k, respectively. There are many procedures developed to solve this problem, which are divided into two main classes of exact and heuristic methods. The detailed applications, backgrounds, and surveys on the solution methodologies of the RCPSP have been presented elsewhere (Kolisch and Padman, 2001; Kolisch and Hartmann, 2006).

The rest of the paper is as follows. In Section 2, the related work is discussed. Section 3 presents the proposed method. The computational results are presented in Section 4. Finally, Section 5 sketches several directions for further development of the method.

2 Related Work

Related work is discussed in two different subsections. The first subsection presents a brief survey on related search techniques and the second subsection introduces some major methods currently used to solve the RCPSP.

2.1 A Brief Survey on the Related Search Techniques

The term metaheuristic was first introduced in Glover (1986) where tabu search was presented. In general, this term refers to a creative strategy for guiding a heuristic, with its design being based on a wide variety of inspirations. These inspirations can be either

taken from nature or specifically devised based on a novel intuition. In effect, each metaheuristic is constructed based on a collection of creative insights that can direct a wide variety of heuristics, aiming at effective coordination of the search in achieving superb solutions.

Metaheuristics can be divided into four categories: (1) evolutionary metaheuristics, (2) local-search-based metaheuristics, (3) constructive metaheuristics, and (4) hybrids. Whereas evolutionary metaheuristics mainly include genetic algorithms, local-search-based metaheuristics, which guide local search heuristics, include (1) simulated annealing (Kirkpatrick et al., 1983), (2) tabu search (Glover, 1986), (3) guided local search (Voudouris and Tsang, 2003), and (4) iterated local search (Lourenco et al., 2003). Constructive metaheuristics, which comprise the third category and guide constructive heuristics, include greedy randomized adaptive local search (Feo and Resende, 1995), and ant colony optimization (Dorigo and Gambardella, 1997). The term, Orthogonal method (Hu et al., 2008), has been used for effective guidance of ant colony optimization techniques and is based on orthogonal design techniques widely applied in scientific research. It is worth mentioning that orthogonal neighborhood-preserving discriminant analysis has been successfully applied in the area of face recognition (Hu, 2008).

The fourth category, hybrids, can be considered as any amalgamation of the members of the first three categories. Two prominent members of the hybrids category are scatter search (Glover, 1994) and memetic algorithms (Moscato, 1989), both of which combine local searches with genetic algorithms. In general, the above metaheuristic provides a specific framework in coping with local optimality via conducting the underlying problem-specific heuristic.

Local-search-based metaheuristics operate based on manipulating complete solutions. In effect, the term local search signifies moving from one complete state to the next complete state through a local variation to the value(s) of one or more variables (Dechter, 2003). Local search differs from a constructive search in that a local search typically explores a search space whose states are complete, whereas a constructive search seeks to complete a partial solution through different stages. The reason why a metaheuristic is needed to guide local searches is that they have the limitation of getting stuck in local optima and a metaheuristic, as a general extendable framework, can assist them in escaping local optimality.

Not only with local-search metaheuristics, but with the other three categories, should the most important component of design focus on striking a balance between getting stuck in a series of high-quality solutions and taking exploration strategies for searching other parts of search space. In other words, whereas such a trade-off should ensure that attention is paid to features historically found suitable, it should not prevent the exploration of unvisited regions.

Neighborhood relations are the major ingredients of local searches; and in a k-exchange neighborhood, two candidate solutions are neighbors if and only if they differ in at most k solution components. It is true that larger neighborhoods contain potentially better candidate solutions, but they lead to generating an unmanageable number of neighbors. One of the effective methods in dealing with this dilemma is the Lin and Kernighan (1973) method which has successfully been applied to the traveling salesman problem. The success of this method has inspired the development of many effective methods, such as variable neighborhood decomposition (Hansen et al., 2001), and variable depth search (VDS; Hoos and Stutzle, 2005), all with the goal of integrating simple neighborhoods to represent the behavior of complex and large neighborhoods.

2.2 A Brief Survey of RCPSP Procedures

The procedures developed to tackle the RCPSP are divided into two major classes of exact and heuristic methods. The most efficient exact methods for the problem have been presented elsewhere (Demeulemeester and Herroelen, 1992, 1997; De Reyck and Herroelen, 1998; Brucker et al., 1998; Mingozzi et al., 1998; Nazareth et al., 1999; Sprecher, 2000; Dorndorf et al., 2000; Zamani, 2001; Demassey et al., 2005). In general, strategies used in producing optimal procedures include dynamic programming, zero-one programming, constraint propagation, and implicit enumeration with branch and bound techniques. The algorithms in this class cannot solve all of the instances with 60 activities introduced in Kolisch and Sprecher (1996).

Heuristic methods have been studied elsewhere (Kolisch, 1995, 1996a; Özdamar and Ulusoy, 1995; Kolisch and Drexl, 1996; Cho and Kim, 1997; Mori and Tseng, 1997; Hartmann, 1998, 2002; Kochetov and Stolyar, 2003; Valls et al., 2004; Debels et al., 2006; Zamani, 2010). Theoretical results on the two building blocks of heuristic methods, which are called parallel and serial schemes, were presented in Kolisch (1996b).

In Kolisch and Padman (2001), heuristic algorithms were categorized as priority-rule based, truncated branch and bound, disjunctive-arc based, and metaheuristic techniques; and also in Sprecher (2002); Zamani (2004); Debels and Vanhoucke (2007); Zamani (2011), four decomposition techniques were presented that can integrate heuristics and exact as well as near-exact and genetic algorithms to tackle the RCPSP. The performances of several important heuristics for the problem were investigated in Kolisch and Hartmann (2006), concluding that forward-backward improvement (FBI), scatter search, and path-relinking ideas can contribute highly to solution quality when incorporated in a proper framework. Two of the recent effective hybrids that were presented for this problem are a path-relinking search presented in Mobini et al. (2009), and a hybrid combing ant colony optimization, scatter search, and local search, presented in Chen et al. (2010).

2.2.1 Priority Rules and Forward/Backward Methods

Priority rules have long been considered as one of the facilitating mechanisms in providing solutions for the RCPSP. To direct a search method, these priorities can be represented either in a form of a random key representation or an activity list (Kolisch, 1996a). Whereas random key representation is a list of numbers, with each number representing the priority of its corresponding activity, an activity list represents the order of activities based on their priorities. Recently, random key representation has been modified to eliminate some of its inefficiencies (Debels et al., 2006). A key point with this modification is to employ a topologic order that uses the same order for activities with the same start time. This modified representation guarantees that each solution corresponds to a unique schedule.

Solving the RCPSP in the forward and backward directions is also one of the other facilitating mechanisms in providing effective solutions to the RCPSP. A method called iterative forward-backward scheduling has been developed in Li and Willis (1992) and because of its effectiveness in generating high-quality solutions, it has been incorporated in several effective frameworks.

One of these frameworks is called justification (Valls et al., 2005) and is based on right active schedules, which are defined as mirror shifting of left-active or simply active schedules. For justifying a given activity to the right (left), a schedule is found in which the starting times of all activities except the given activity remain the same and the starting time of the given activity increases (decreases). Except for the ending

activity, this procedure right-justifies activities one at a time in the hope that the starting times of all critical activities increase and in this way the duration of the entire project decreases. The mirror of these operations is applied for the left justification.

Similar to the justification platform, in Tormos and Lova (2001) a multipass method has been presented that uses random sampling and performs backward-forward passes. In successive passes, this method aims at obtaining project duration shorter than the incumbent solution.

The forward-backward technique has also been used in boosting diversification. The critical activity reordering algorithm (CARA; Valls et al., 2003) is a nonstandard implementation of tabu search in which diversification is provided through a strategic oscillation mechanism loosely related to the forward-backward technique. CARA employs the topological order (TO) representation of schedules, which plays the role of priority values. To improve a given feasible schedule, CARA attempts to advance the start times of its all critical activities, which are activities with zero total slack. To advance the starting times of critical activities, it advances the starting times of noncritical activities if needed.

2.2.2 Self-Adaptation, Peak Crossover, and Variable Neighborhood Search

The self-adaptation capability of some algorithms in determining specific values for their parameters, in response to each specific instance, plays a key role in their efficiency. In Hartmann (2002) a self-adapting genetic algorithm has been presented that makes use of two different decoding mechanisms as well as an additional gene that determines the decoding mechanism employed, with each decoding mechanism converting an activity list to a schedule and being based on either a parallel or a serial method. The results of the corresponding computational experiments show that this self-adaptation mechanism is very effective.

Similar to the concept of self-adaptation, a concept called peak crossover (Valls et al., 2003, 2008) has also been employed to improve the performance of the genetic algorithms specifically developed to solve the RCPSP. Peak crossover operators, rather than randomly combining selected parts of good solutions, prioritize portions with higher usage of resources. The use of a peak crossover operator is involved with using an activity list that should be compatible with the starting times of activities, that is, if an activity has started before a given activity, it should also appear before the given activity in the list. A parameter called resource utilization ratio threshold (δ) determines which periods of time should be considered as periods with a high usage of resources. The value of δ is randomly set in the range between 0.75 and 0.9. This random setting causes crossover operations operating on the same genomes to lead to different offspring.

Another effective search mechanism, a variable neighborhood search (VNS) method for the RCPSP, was presented in Fleszar and Hindi (2004), which employs the concept of lower bounding and performs precedence augmentation. It uses two types of sophisticated move strategies. In this search, a forceful move strategy, called enhanced move, was employed that shifts an activity forward (backward) in any range along with its successors (predecessors). This extends the left (right) limit of a boundary that an activity can shift, and effectively enlarges the size of the neighborhood.

3 The Crossed-Over Orthogonal Neighborhood Explorer (CONE)

As a memetic algorithm, the CONE was developed based on balancing the two features of high depth and mobility of the search. CONE achieves mobility via the alternation of two synergetic neighborhood schemes; and it attains high depth through iterative

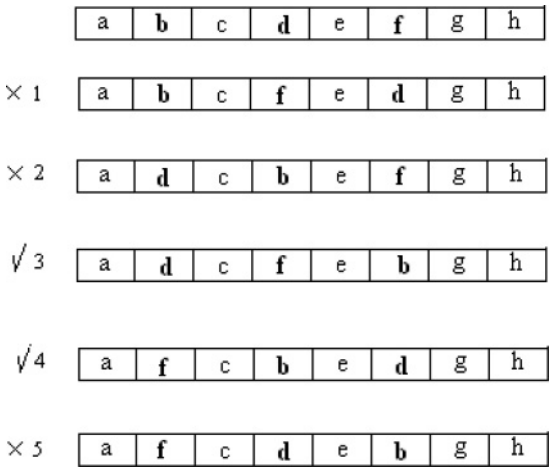

Figure 1: The triple activities interchange (TAI) scheme.

crossover of locked solutions. It is through these two features that it iteratively discovers high-quality solutions and combines these solutions to attain a solution with a higher quality.

Both neighborhood schemes employed in the CONE use activity list representation, in which, with respect to a given schedule, activities with smaller starting times appear sooner than those with greater starting times (Valls et al., 2003). Each activity list is converted into two schedules through the applications of serial and parallel methods (Kolisch, 1996b) followed by a backward forward (BF) iteration (Tormos and Lova, 2001). Among the two schedules produced, the best one is selected.

The two orthogonal neighborhood schemes employed operate on the same activity list and improve it progressively. In employing these neighborhood schemes, two factors are the key contributors in achieving high-quality solutions: (1) the set of neighbors generated by the first scheme does not have any intersection with the set generated by the second scheme, and (2) many unfruitful neighbors are prevented from being scheduled.

3.1 Triple Activities Interchange (TAI)

The first neighborhood scheme is a modified version of the typical three-exchange neighborhood. In the three-exchange neighborhood, two candidate solutions are neighbors if and only if they differ in, at most, three solution components. The difference between this new neighborhood, which we call the triple activities interchange (TAI), and the typical three-exchange neighborhood, is that in the TAI, two candidate solutions are neighbors if and only if they differ in exactly, rather than at most, three solution components. The TAI can alter the locations of every three nonadjacent activities on the list systematically and produce associated neighbors.

As is depicted in Figure 1, every three activities can at most be interchanged in five different ways, among which only two of them change the positions of all three activities. For efficiency purposes that will be discussed later, the TAI uses only these two neighbors and ignores the rest. As is seen, only the third and forth orders in Figure 1 interchange all three activities. The reason is that the first order changes the positions of only f and d, the second order changes the positions of only b and d, and the fifth

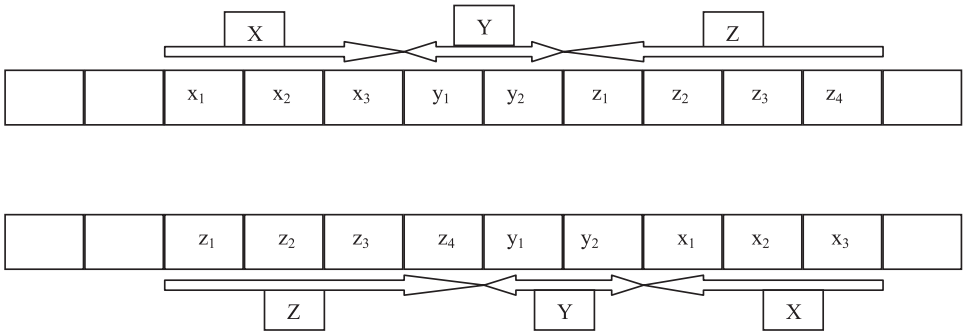

Figure 2: The double partitions interchange (DPI) scheme.

order changes the positions of only f and b. Among the two orders changing all three activities, if any of them proves to be an activity list, it is considered as a neighbor.

One of the advantages of the TAI is that for generating neighbors, not all triplets need to be considered for interchange. The reason is that the CONE avoids many unnecessary interchanges when the interchange of a triplet is found infeasible, and this has been facilitated through detecting all related infeasible interchanges. Computational efficiency gained by ignoring unfruitful triplets contributes to the effectiveness of the procedure significantly.

3.2 Double Partitions Interchange (DPI)

The second neighborhood scheme, called the double partitions interchange (DPI) is more sophisticated than the TAI because it operates based on partitions and swaps every two partitions systematically, with a partition being defined as a group of contiguous activities on the activity list. The partitions that are swapped can be either adjacent or separated by another partition. Hence, if X, Y, and Z represent three sequential partitions, the DPI converts XYZ to ZYX. Figure 2 illustrates this conversion.

As can be seen, the partitions X and Z have been separated by the partition Y, which can be a null partition. In the cases where Y is a null partition, XZ is converted to ZX. The CONE implicitly enumerates all double partitions for swapping, regardless of whether or not they have been separated by a third partition. The swaps which violate precedence constraints are ignored and only those which lead to the generation of activity lists are considered.

As is true with the TAI, the number of neighbors is limited by the consideration of propagated infeasibility, and as soon as the swapping of two partitions leads to infeasibility, the CONE detects all related infeasible swaps. This detection is toward improving efficiency by avoiding unnecessary computational expenses and has proved to be very effective.

3.3 Limiting the Moves and Converting Activity Lists to Schedules

To limit the moves performed by the DPI and TAI, we have extended the concept of peak crossover mentioned in the literature survey and have used it in the context of local search. Based on this extended concept, activities placed in portions with higher usage of resources in the activity list will never be subject to changing their position. Hence, based on any given resource utilization ratio threshold (δ), an activity list can have several contiguous parts which should remain unchanged. Each contiguous part

of the activity list includes several activities and is associated with a specific peak. The value of δ is between zero and one. It should be noted that the smaller values of this parameter lead to a smaller number of neighbors.

As mentioned, the CONE uses two methods for converting an activity list into a schedule, namely, the serial and parallel methods (Kolisch, 1996b). By assigning higher priorities to activities with lower order, and executing them one after another, the serial method can convert any activity list into a feasible schedule. The parallel method, on the contrary, at each stage, increases time based on the completion of activities and selects among eligible activities those with higher priority.

3.4 The C-type Pseudocode of the CONE

Algorithm 1 presents a C-type pseudocode describing the operations of the CONE. The pseudocode starts by generating a random activity list at line 3. This activity list is revised in two nested while loops at lines 17 and 20. The inner while loop (line 20) performs gradient descent operations, and the outer while loop (line 17) alternates the TAI and DPI neighborhood schemes. Moreover, a while loop starting at line 5 creates a diverse pool of locked solutions based on different activity lists generated at lines 3 and 43. The iterative crossover operations are performed at line 45, in which the initial pool filled by lines 3 through 44 is refined. In other words, line 45 activates a genetic algorithm for improving the results and exchanges information between high quality solutions kept in the pool. Exchanging information between high quality solutions is aimed at increasing the power of the search in achieving a solution with a better quality than that of the best solution obtained so far. Line 45 is further clarified later in this section.

In Algorithm 1, the best makespan is called QualityMakespan, and is initialized in line 4. Whenever this variable is updated (explicitly at lines 13 and 41 as well as implicitly at line 45), its associated schedule is also saved. The inner while loop (line 20) is performed until a local optimal solution is obtained for the associated neighborhood scheme. The solution obtained in this stage is further improved by another while loop (line 17), which converts it into a local optimal solution for the other scheme.

A Boolean variable called Switch, initialized at line 15 and updated in lines 19 and 30, ensures that as far as the application of one neighborhood scheme breaks the deadlock occurred by the application of the other scheme, the two schemes can, in turn, be applied and improve the solution. In the situations where a local optimal solution is surrounded by many neighboring local optimal solutions, this switching causes several of these neighboring local optimal solutions to be visited without any significant computational effort. Figure 3 depicts such a situation.

Moreover, a variable called NSwitches, initialized at line 16 and updated at line 38, guarantees that each neighborhood scheme is used at least once. This variable is useful in situations where the TAI scheme is incapable of making any improvement. This occurs only in the cases where S, generated as an initial solution at line 3, is a local optimum for the TAI scheme.

The simultaneous use of the two variables of Switch and NSwitches ensures that when the inner (line 20) and outer (line 17) nested while loops are terminated, the solution obtained is a local optimum for both the TAI and DPI schemes. The while loop presented at line 5 repeats these operations and fills a pool of high quality solutions that are local optimal for both orthogonal neighborhood schemes.

Line 45 improves the high quality pool of the locked solutions through crossover operations. For this purpose, two members of the pool are randomly selected and

Algorithm 1 The C-type pseudocode of the CONE based on the simplified assumption of separating the genetic algorithm from its local search component

```
CONE()
{
   Generate a random activity list and call it S;
   QualityMakespan = A large positive integer;
   while (the pool of locked solutions is not filled)
   {
      Decode S via the parallel method and
      call the corresponding  makespan A;
      Decode S via the serial method and
      call the corresponding  makespan B;
      Makespan = min (A,B);
      if (Makespan < QualityMakespan)
             QualityMakespan = Makespan;
     Set the current neighbourhood scheme to TAI;
     Switch=.True.;
     NSwitches=0;
     while (Switch==.True.)
     {
          if (NSwitches >1)Switch=.False.;
          For (every S', as a neighbour of S)
          {
             Decode S' via the parallel method and then apply
             the Backward/Forward method & call the makespan A;
             Decode S' via the serial method and then apply
             the Backward/Forward method & call the makespan B;
             C = min (A,B);
             if (C < Makespan)
             {
                     Makespan = C;
                     Switch = .True.;
                     S = S';
                     Break;
             }
           }
           Switch the neighbourhood scheme to the scheme
               which is not currently in use (from TAI to
               DPI or vice versa);
           NSwitches++;
       }
       if (Makespan < QualityMakespan)
            QualityMakespan = Makespan;
       Add the locked solution to the pool;
       Generate a random activity list and call it S;
   }
   Activate a genetic algorithm to improve the pool;
}//end of CONE
```

one-point and two-point crossover operations are performed on them to create two activity lists. Then, each of the resultant activity lists is converted into two schedules through the applications of serial and parallel methods (Kolisch, 1996b), each followed by a BF iteration (Tormos and Lova, 2001).

The best schedule generated can replace the schedule with the lowest quality in the pool if it improves this comparatively low quality schedule. In the cases where the two selected parents are similar to one another, the employed genetic algorithm replaces

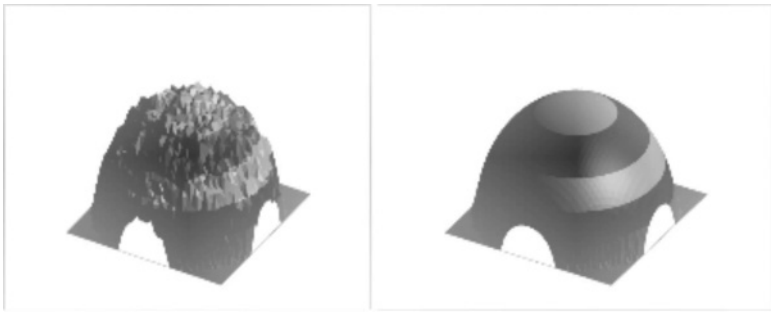

Figure 3: (a) A local optimal solution surrounded by many other local optimal solutions. (b) The corresponding easy-to-find optimal solution.

the parent which has lower quality with a fresh solution being local optimal for both orthogonal neighborhood schemes. In other words, line 45 can implicitly call lines 3 through 44 as many times as needed.

4 Computational Experiments

Computational experiments were performed using a DELL PC (1.86 GHz) with Windows XP operating system and 2 GB of RAM from which at most 1 GB of RAM was used. The algorithm was coded in C++ and the performance of the procedure was examined on 2,040 benchmark instances obtainable from the PSPLIB library (Kolisch and Sprecher, 1996).

These instances are composed of four sets, namely *j*30, *j*60, *j*90, and *j*120. Each of the first three sets includes 480 instances, whereas the last set, *j*120, includes 600 instances, $2{,}040 = 3 \times 480 + 600$. The number at the end of each set name indicates the number of activities that each instance of the corresponding set has. For example, each benchmark instance in the set *j*120 has 120 activities.

Whereas all optimal solutions for the set *j*30 have been identified in the literature, the optimal solutions for many instances of the sets *j*60, *j*90, and *j*120 are still unknown. For this reason, in the literature, instead of using the criterion of Optimal_dev%, as the percentage of deviation from an optimal solution, the criterion of CPM_dev%, as the percentage of deviation from the CPM lower bound, has been used for benchmarking the results. The CPM lower bound is obtained by removing the limits on the availability of resources and solving the simplified problem. In effect, such a simplified problem includes only precedence constraints, which determine the precedence relations between activities, and can be easily solved.

4.1 The Parameter Setting Phase

Because of the large number of benchmark instances, the parameter setting phase was performed in two separate stages. In the first stage, a small number of benchmark instances was selected, and a large number of combinations for the values of the parameters were tested on them. In the second stage, the combination of a small number of critical parameters, determined in the first stage, was tested on the entire 2,040 benchmark instances. Even keeping the number of critical parameters at two and considering

only four values for each parameter, the number of times the program was run in the second stage is 32,640, $4 \times 4 \times 2040$.

In order for the small instances selected in the first stage to be a fair representative of the hard-to-solve instances, we have considered the fact that these 2,040 benchmark instances were originally produced, by their authors, through the systematic variation of network complexity (NC), resource strength (RS), and resource factor (RF). Whereas NC determines the ratio of nonredundant precedence relations to the number of activities, RF shows the portion of resource types which an activity uses on average. The third factor, RS, shows the degree of constraints on the resources in the range between 0 and 1; the larger this value, the less constrained the resources are. Therefore, when RS is 1, the resources are so abundant that each activity can start as soon as all of its predecessors have been completed.

As the result of the first phase, we set the number of generated schedules to 50,000 and found out that for this number of generated schedules, two parameters mainly affected the performance of the procedure: the utilization ratio threshold (δ), and the population size (η). In addition, the results of experimentations in phase 1 indicated that when δ is set to a value greater than 0.85 or η is set to a value greater than 10, the results deteriorated. However, the differences could not be shown to be statistically significant and this reveals the robustness of the procedure: the CONE is not highly dependent on the values of its parameters.

As will be shown, even in phase 2, the statistical test of ANOVA (analysis of variance) has not found any significant difference among the 16 combinations considered. The result of this statistical test will be discussed in presenting the results of phase 2, in which four different values for δ and four different values for η were considered. The values considered for δ are 0.70, 0.75, 0.80, and 0.85; and the values considered for η are 4, 6, 8, and 10. This has created 16 combinations to be tested, and with each of these 16 combinations, the procedure was run on the entire 2,040 instances.

Before presenting the results of phase 2 of the parameter setting, we emphasize that in the cases in which the two chosen parents are similar, the genetic algorithm component replaces the parent which has lower quality with a fresh solution that is obtained through its local search component. This may describe why the values over 10 for η have not produced solutions with higher quality. Moreover, when the value of δ reaches beyond 0.85, δ can rarely limit the number moves, and when the δ value is below 0.75, it limits the number of moves exceedingly. This also may be a reason why in phase 1, the values outside this range could not perform better than those inside the range.

Table 1 shows the results of the second phase of parameter setting. As is seen with each combination, both CPM_dev% and Best_dev% have been reported. Note that Best_dev% shows the percentage of division from the best available solution in the literature and is not as reliable as CPM_dev%. The reason is that by the passage of time, better solutions are made available, and this makes the Best_dev% criterion invalid. As is seen, the procedure has produced the best result for set $j30$ when the combination of $\delta = 0.80$, $\eta = 4$ was used. In effect, with this combination, the CONE found the optimal solutions of all 480 instances of the set $j30$.

For the set $j60$, the combination of $\delta = 0.70$, $\eta = 8$ and for the sets $j90$ and $j120$, the combination of $\delta = 0.75$, $\eta = 10$ produced the best results. These computational experiments indicate that a fixed combination of δ and η does not work perfectly for all four sets. That is why the procedure has been used with $\delta = 0.80$, $\eta = 4$ for the set $j30$, with $\delta = 0.70$, $\eta = 8$ for set $j60$, and with $\delta = 0.75$, $\eta = 10$ for sets $j90$ and $j120$.

Table 1: The performance of the CONE measured by percentage deviation from the best available solutions and CPM lower bound for each set *j*30, *j*60, *j*90, and *j*120, based on 16, 4 × 4, combinations of threshold delta (δ) and pool size (η).

		Threshold delta (δ)							
		0.7		0.75		0.8		0.85	
Pool size (η)		%Dev CPM	%Dev BEST	%DEV CPM	%Dev BEST	%Dev CPM	%Dev BEST	%Dev CPM	%Dev BEST
4	J30	13.38	0.00	13.38	0.00	**13.37**	0.00	13.38	0.00
	J60	10.66	0.17	10.68	0.19	10.67	0.18	10.72	0.21
	J90	10.11	0.29	10.11	0.28	10.13	0.30	10.19	0.34
	J120	31.84	1.14	31.87	1.14	31.88	1.15	31.92	1.02
6	J30	13.38	0.00	13.38	0.00	13.39	0.01	13.39	0.01
	J60	10.67	0.19	10.69	0.19	10.67	0.18	10.68	0.19
	J90	10.11	0.29	10.11	0.28	10.08	0.27	10.14	0.31
	J120	31.66	1.03	31.82	1.12	31.76	1.10	31.86	1.19
8	J30	13.39	0.01	13.38	0.00	13.39	0.01	13.38	0.00
	J60	**10.64**	0.16	10.67	0.18	10.67	0.18	10.67	0.18
	J90	10.06	0.25	10.11	0.28	10.07	0.26	10.11	0.30
	J120	31.74	1.08	31.67	1.05	31.73	1.08	31.90	1.22
10	J30	13.38	0.01	13.38	0.00	13.38	0.00	13.39	0.01
	J60	10.65	0.17	10.67	0.18	10.66	0.17	10.07	0.20
	J90	10.07	0.26	**10.03**	0.23	10.08	0.27	10.14	0.32
	J120	31.65	1.04	**31.65**	1.04	31.73	1.07	31.96	1.25

Based on these values, the performance of the procedure was categorized based on different values of NC, RF, and RC. Table 2 shows the results. As is shown in this table, the performance of the procedure for nearly all categories is satisfactory and the only two categories for which the value of %Dev-BEST is larger than 3 are those with four characteristics: (1) having 120 activities, (2) RS = 0.1, (3) NC = 1.5, and (4) RF between 0.5 and 0.75. In effect, for the majority of groups, %Dev-BEST is under 1. Note that each group includes 10 instances, and the groups of simple instances, those with RS = 1, were not included in the table because %Dev-BEST for these groups is zero.

In addition, ANOVA shows the robustness of the CONE, through reporting no significant difference among the performance of the procedure for the 16 combinations mentioned. ANOVA's results are as follows: variation between groups (49.52) with 15 DOF and variation within groups (31,589,883) with the 32,624 DOF, f-stat = 0.003, $\alpha = 0.05$, f-critical = 1.666. Note that, as stated, 32,624 is the number of times the procedure was executed in phase 2, $32{,}624 = 16 \times 2040$.

4.2 Comparing the Performance of the CONE with Other Procedures

The results of our computational experiments were compared with the results obtained for local search with subproblem exact resolution (LSSPER; Palpant et al., 2004). LSSPER is a state of the art procedure that has improved 14, 9, and 4 of the best solutions for the sets *j*60, *j*90, and *j*120, respectively. The developers ran LSSPER on a personal computer with 1 GB RAM and 2.3 GHz speed. Table 3 compares the results produced by the CONE with those produced by LSSPER.

As Table 3 indicates, for three out of four sets, the CONE produced better results in shorter times. For set *j*30, both procedures produced the same result. In effect, for this set, both the CONE and LSSPER produced the optimal solutions of all 480 instances.

Table 2: The average of percentage deviation from the best available solutions in the literature for all possible combinations of NC, RF, and RS, with each cell standing for 10 different instances which have the same NC, RF, and RS.

		NC = 1.5				NC = 1.8				NC = 2.1			
RS		RF = 0.25	RF = 0.50	RF = 0.75	RF = 1.00	RF = 0.25	RF = 0.50	RF = 0.75	RF = 1.00	RF = 0.25	RF = 0.50	RF = 0.75	RF = 1.00
Size 60	0.2	0.00	0.66	0.62	1.36	0.00	0.32	0.39	1.20	0.00	0.92	0.43	1.05
	0.5	0.00	0.00	0.00	0.44	0.00	0.00	0.14	0.12	0.00	0.00	0.00	0.22
	0.7	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Size 90	0.2	0.11	0.97	2.00	1.25	0.00	0.69	1.25	1.08	0.00	0.73	1.13	1.20
	0.5	0.00	0.00	0.00	0.00	0.00	0.00	0.23	0.00	0.00	0.00	0.23	0.22
	0.7	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Size 120	0.1	0.91	3.23	3.55	2.53	0.86	1.81	2.90	2.50	0.49	1.88	2.67	2.09
	0.2	0.33	2.00	2.56	1.68	0.39	1.25	2.27	1.88	0.54	1.20	1.76	1.85
	0.3	0.00	1.17	1.99	1.28	0.00	0.56	1.99	1.51	0.00	0.72	1.46	1.39
	0.4	0.00	0.12	0.64	0.84	0.00	0.11	0.55	0.98	0.00	0.21	1.11	1.40
	0.5	0.00	0.00	0.00	0.13	0.00	0.24	0.20	0.00	0.00	0.00	1.10	0.63

Table 3: Comparing the performance of the CONE measured by percentage deviation from the CPM lower bound from that of LSSPER for each set *j*30, *j*60, *j*90, and *j*120.

	CONE, computer speed = 1.86 GHz		LSSPER, computer speed = 2.3 GHz	
Benchmark sets	Average CPM_dev %	Time (s)	Average CPM_dev %	Time (s)
*j*30	13.37	1	13.37	10
*j*60	10.64	2	10.81	38
*j*90	10.03	4	10.29	61
*j*120	31.65	5	32.41	207

Table 4: Comparing the performance of the CONE with that of of six high performance procedures based on percentage deviation from optimal solution for the set *j*30 and percentage deviation from CPM lower bound for the sets *j*60, and *j*120.

	Benchmark set		
Reference*	*j*30	*j*60	*j*120
This paper	0.00	10.64	31.65
Chen et al., 2010	0.01	10.67	30.56
Mobini et al., 2009	0.01	10.57	31.37
Mendes, Gonçalves, et al., 2009	0.01	10.67	31.44
Valls et al., 2008	0.02	10.73	31.24
Debels et al., 2006	0.01	10.71	31.57
Valls et al., 2005	0.01	10.74	31.58

*All with the limit of generating 50,000 schedules for each instance in each set.

With respect to its performance, CONE has also been compared with six high performance procedures shown in Table 4 (Chen et al., 2010; Mobini et al., 2009; Mendes, Gonçalves, et al., 2009; Valls et al., 2008; Debels et al., 2006; Valls et al., 2005). Since for the set *j*90, these six procedures do not have any results, the comparison is only based on the sets *j*30, *j*60, and *j*120.

It is worth mentioning that CONE and these six procedures have all been run with the same limit of generating 50,000 schedules, and this limit makes the speed of the computer on which they have been run as well as their execution times inconsequential. As is seen in Table 4, an interesting point in this comparison is that none of these six procedures has been able to produce optimal solutions for all 480 instances of the set *j*30, whereas CONE has produced optimal solutions for all these instances, and this has set the percentage of deviation from optimal solutions to zero.

4.3 Remarks on Computational Experiments

At the end of this section, two points are worth mentioning. First, we deactivated the genetic algorithm part of the CONE and let the modified procedure generate the same number of schedules, 50,000. As a result, we found that for 91%, 76%, 73%, and 31% of instances in the sets *j*30, *j*60, *j*90, and *j*120, respectively, the solutions produced by the simplified procedure were the same as those produced by the CONE. This also indicated that the average ratio improvement made in the solutions by the genetic algorithm component with two significant digits were 0.00, 0.01, 0.01, and 0.03, for the sets *j*30, *j*60, *j*90, and *j*120, respectively. The statistical t-test confirms the significance of the genetic algorithm component. The values produced by the t-test are as follows: 2,039 DOF, t-stat = 24.02, $\alpha = 0.05$, t-critical = 1.96.

The second point worth mentioning is that if the best solution among all the 16 combinations of (δ, η) were selected for each of the 2,040 benchmark instances, then CPM_dev% results produced by the CONE would significantly improve from 10.64, 10.03, and 31.65; to 10.49, 9.81, and 30.73 for the sets *j*60, *j*90, and *j*120, respectively. In this case, the computational times would increase from 2, 4, and 5; to 32, 64, and 80 s, for the sets *j*60, *j*90, and *j*120, respectively, indicating a 16 times increase in the computational time for each set.

5 Conclusion

The effectiveness of metaheuristics in general is based on the features of high depth and mobility, and the successful methods presented for the RCPSP can be primarily characterized by their capability in providing these two features. The CONE has performed successfully on the benchmark instances of the PSPLIB and this success can be accredited to these two features.

With respect to mobility, the CONE performs a rapid search, through altering the locations of every three activities as well as swapping the locations of every two possible partitions. Changing the locations of possible partitions is similar to the results of crossover operations in genetic algorithms, with a difference that here both partitions are selected from the same genome and change their locations. The other difference is that all the swaps are performed systematically, whereas crossover operations are performed randomly.

With respect to high depth, the CONE performs crossover operations on random pairs of high quality solutions. This increases the quality of regions in which the search is conducted. A dynamic pool, along with one-point and two-point crossover operations, facilitates achieving high depth in such promising regions. The aim of crossover operations is to exploit the knowledge existing in the set of diverse solutions, which were originally local optimal for both orthogonal neighborhood schemes. This can further explore the promising areas of the search space as well.

The capabilities of the CONE can be extended in future works toward two different directions. These directions include employing automatic parameter setting and using particle swarm optimization in combining high quality solutions kept in the pool. These two directions can be described as follows.

The values of the parameters such as population size (η) and utilization ratio threshold (δ), can influence the performance of the CONE both in terms of the quality of solutions and computation time. Hence, finding suitable problem-dependent values for them can play a key role in the performance of the CONE. Taking this into consideration, the CONE can be equipped with an additional mechanism that learns the best values of the parameters for any specific problem instance.

For instance, the current genetic algorithm can be extended to represent each of the parameters used as a gene and measure the fitness of each chromosome by evaluating the solution obtained via the associated control parameters. In this way, by comparing the quality of solutions obtained, the revised CONE will learn how to improve its own performance.

The second direction is aimed at improving solution quality at the cost of insignificant computation time through employing the technique of particle swarm optimization (Engelbrecht, 2005). Because of the diversity and high quality of the solutions kept in the initial pool, these solutions can be considered to be proper choices for the application of particle swarm optimization.

By using particle swarm optimization, one high quality schedule generated by the CONE can be flown toward the other schedule through multidimensional search space, and on its way can produce a variety of high quality schedules. In this proposed platform, the flight of one schedule toward the other is aimed at increasing mobility (Zhang et al., 2005), and is intended to improve solution quality. In the proposed platform, a learning mechanism similar to what presented in Zamani and Lau (2010) can be activated for adjusting the operations involved.

References

Bacardit, J., and Krasnogor, N. (2009). Performance and efficiency of memetic Pittsburgh learning classifier systems. *Evolutionary Computation*, 17(3):307–342.

Brucker, P., Knust, S., Schoo, A., and Thiele, O. (1998). A branch and bound algorithm for the resource-constrained project scheduling problem. *European Journal of Operational Research*, 107:272–288.

Chen, W., Shi, Y., Teng, H., Lan, X., and Hu, L. (2010). An efficient hybrid algorithm for resource-constrained project scheduling. *Information Sciences*, 180(6):1031–1039.

Cho, J., and Kim, Y. D. (1997). A simulated annealing algorithm for resource constrained project scheduling problems. *Journal of the Operational Research Society*, 48:736–744.

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

Deb, K., and Sinha, A. (2010). An efficient and accurate solution methodology for bilevel multi-objective programming problems using a hybrid evolutionary-local-search algorithm. *Evolutionary Computation*, 18(3):403–449.

Debels, D., De Reyck, B., Leus, R., and Vanhoucke, M. (2006). A hybrid scatter search/electromagnetism meta-heuristic for project scheduling. *European Journal of Operational Research*, 169(2):638–653.

Debels, D., and Vanhoucke, M. (2007). A decomposition-based genetic algorithm for the resource-constrained project-scheduling problem. *Operations Research*, 55(3):457.

Dechter, R. (2003). *Constraint processing*. San Mateo, CA: Morgan Kaufmann.

Demassey, S., Artigues, C., and Michelon, P. (2005). Constraint-propagation-based cutting planes: An application to the resource-constrained project scheduling problem. *INFORMS Journal on Computing*, 17(1):52–65.

Demeulemeester, E., and Herroelen, W. (1992). A branch-and-bound procedure for multiple resource-constrained project scheduling problem. *Management Science*, 38:1803–1818.

Demeulemeester, E., and Herroelen, W. (1997). A new benchmark results for the resource-constrained project scheduling problem. *Management Science*, 43:1485–1492.

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

Dorndorf, U., Pesch, E., and Phan-Huy, T. (2000). A branch-and-bound algorithm for the resource-constrained project scheduling problem. *Mathematical Methods of Operations Research*, 52:413–439.

Engelbrecht, A. (2005). *Fundamentals of computational swarm intelligence*. New York: Wiley.

Farahmand, A. M., Ahmadabadi, M. N., Lucas, C., and Araabi, B. N. (2010). Interaction of culture-based learning and cooperative co-evolution and its application to automatic behavior-based system design. *IEEE Transactions on Evolutionary Computation*, 14(1):23–57.

Feo, T. A., and Resende, M. G. C. (1995). Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6(2):109–133.

Fleszar, K., and Hindi, K. S. (2004). Solving the resource-constrained project scheduling problem by a variable neighbourhood search. *European Journal of Operational Research*, 155(2):402–413.

Fukunaga, A. S. (2008). Automated discovery of local search heuristics for satisfiability testing. *Evolutionary Computation*, 16(1):31–61.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13(5):533–549.

Glover, F. (1994). Genetic algorithms and scatter search: Unsuspected potentials. *Statistics and Computing*, 4(2):131–140.

Glover, F. (1998). A template for scatter search and path relinking. *Lecture notes in computer science*, Vol. 1363 (pp. 13–54). Berlin: Springer-Verlag.

Glover, F., Kelly, J. P., and Laguna, M. (1995). Genetic algorithms and tabu search: Hybrids for optimization. *Computers and Operations Research*, 22(1):111–134.

Hansen, P., Mladenović, N., and Perez-Britos, D. (2001). Variable neighborhood decomposition search. *Journal of Heuristics*, 7(4):335–350.

Hartmann, S. (1998). A competitive genetic algorithm for resource-constrained project scheduling. *Naval Research Logistics*, 45:733–750.

Hartmann, S. (2002). A self-adapting genetic algorithm for project scheduling under resource constraints. *Naval Research Logistics*, 49:433–448.

Hoos, H. H., and Stutzle, T. (2005). *Stochastic local search foundations and applications*. San Mateo, CA: Morgan Kaufmann.

Hu, H. (2008). Orthogonal neighborhood preserving discriminant analysis for face recognition. *Pattern Recognition*, 41(6):2045–2054.

Hu, X. M., Zhang, J., and Li, Y. (2008). Orthogonal methods based ant colony search for solving continuous optimization problems. *Journal of Computer Science and Technology*, 23(1):2–18.

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

Kochetov, Y. A., and Stolyar, A. A. (2003). Evolutionary local search with variable neighbourhood for the resource constrained scheduling problem. *Proceedings of the 3rd International Workshop of Computer Science and Information Technologies*.

Kolisch, R. (1995). *Project scheduling under resource constraints: Efficient heuristics for several problem classes*. Heidelberg: Physica.

Kolisch, R. (1996a). Efficient priority rules for the resource-constrained project scheduling problem. *Journal of Operations Management*, 14(3):179–192.

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

Kolisch, R., and Drexl, A. (1996). Adaptive search for solving hard project scheduling problems. *Naval Research Logistics*, 43:23–40.

Kolisch, R., and Hartmann, S. (2006). Experimental investigation of heuristics for resource-constrained project scheduling: An update. *European Journal of Operational Research*, 174(1):23–37.

Kolisch, R., and Padman, R. (2001). An integrated survey of deterministic project scheduling. *Omega: The International Journal of Management Science*, 29:249–272.

Kolisch, R., and Sprecher, A. (1996). PSPLIB; A project scheduling library. *European Journal of Operational Research*, 96:205–216.

Li, K., and Willis, R. (1992). An iterative scheduling technique for resource-constrained project scheduling. *European Journal of Operational Research*, 56:370–379.

Lin, S., and Kernighan, B. (1973). An effective heuristic algorithm for the traveling salesman problem. *Operations Research*, 21:443–452.

Lourenço, H., Martin, O., and Stützle, T. (2003). Iterated local search. In F. Glover (Ed.), *Handbook of Metaheuristics* (pp. 320–353). Dordrecht, The Netherlands: Kluwer.

Marti, R., Laguna, M., and Glover, F. (2006). Principles of scatter search. *European Journal of Operational Research*, 169(2):359–372.

Mendes, J. J. M., Gonçalves, J. F., and Resende, M. G. C. (2009). A random key based genetic algorithm for the resource constrained project scheduling problem. *Computers and Operations Research*, 36(1):92–109.

Mendes, S. P., Molina, G., Vega-Rodríguez, M. A., Gómez-Pulido, J. A., Sáez, Y., Miranda, G., Segura, C., Alba, E., Isasi, P., León, C., and Sanchez-Perez, J. M. (2009). Benchmarking a wide spectrum of metaheuristic techniques for the radio network design problem. *IEEE Transactions on Evolutionary Computation*, 13(5):1133–1150.

Mingozzi, A., Maniezzo, V., Ricciardelli, S., and Bianco, L. (1998). An exact algorithm for the resource-constrained project scheduling problem based on a new mathematical formulation. *Management Science*, 44:714–729.

Mobini, M., Rabbani, M., Amalnik, M. S., Razmi, J., and Rahimi-Vahed, A. R. (2009). Using an enhanced scatter search algorithm for a resource-constrained project scheduling problem. *Soft Computing—A Fusion of Foundations, Methodologies and Applications*, 13(6):597–610.

Molina, D., Lozano, M., García-Martínez, C., and Herrera, F. (2010). Memetic algorithms for continuous optimisation based on local search chains. *Evolutionary Computation*, 18(1):27–63.

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

Moscato, P. (1989). On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. *Caltech Concurrent Computation Program, C3P Report*, 826:1989.

Nazareth, T., Verma, S., Bhattacharya, S., and Bagchi, A. (1999). The multiple resource constrained project scheduling problem: A breadth-first approach. *European Journal of Operational Research*, 112(2):347–366.

Özdamar, L., and Ulusoy, G. (1995). A survey on the resource-constrained project scheduling problem. *IIIE Transactions*, 27:574–586.

Palpant, M., Artigues, C., and Michelon, P. (2004). LSSPER: Solving the resource-constrained project scheduling problem with large neighbourhood search. *Annals of Operations Research*, 131(1):237–257.

Sprecher, A. (2000). Scheduling resource-constrained projects competitively at modest memory requirement. *Management Science*, 46:710–723.

Sprecher, A. (2002). Network decomposition techniques for resource-constrained project scheduling. *Operational Research Society*, 53:405–414.

Tormos, P., and Lova, A. (2001). A competitive heuristic solution technique for resource-constrained project scheduling. *Annals of Operations Research*, 102(1):65–81.

Valls, V., Ballestin, F., and Quintanilla, S. (2004). A population-based approach to the resource-constrained project scheduling problem. *Annals of Operations Research*, 131(1):305–324.

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

Valls, V., Ballestin, F., and Quintanilla, S. (2008). A hybrid genetic algorithm for the resource-constrained project scheduling problem. *European Journal of Operational Research*, 185(2):495–508.

Valls, V., Quintanilla, S., and Ballestin, F. (2003). Resource-constrained project scheduling: A critical activity reordering heuristic. *European Journal of Operational Research*, 149(2):282–301.

Voudouris, C., and Tsang, E. P. K. (2003). Guided local search. In F. Glover (Ed.), *Handbook of Metaheuristics* (pp. 185–218). Dordrecht, The Netherlands: Kluwer.

Wanner, E. F., Guimarães, F. G., Takahashi, R. H. C., and Fleming, P. J. (2008). Local search with quadratic approximations into memetic algorithms for optimization with multiple criteria. *Evolutionary Computation*, 16(2):185–224.

Zamani, R. (2001). A high-performance exact method for the resource-constrained project scheduling problem. *Computers and Operations Research*, 28(14):1387–1401.

Zamani, R. (2004). An efficient time-windowing procedure for scheduling projects under multiple resource constraints. *OR Spectrum*, 26(3):423–440.

Zamani, R. (2010). An accelerating two-layer anchor search with application to the resource-constrained project scheduling problem. *IEEE Transactions on Evolutionary Computation*, 14(6):975–984.

Zamani, R. (2011). A hybrid decomposition procedure for scheduling projects under multiple resource constraints. *Operational Research*, 11(1):93–111.

Zamani, R., and Lau, S. K. (2010). Embedding learning capability in Lagrangean relaxation: An application to the travelling salesman problem. *European Journal of Operational Research*, 201(1):82–88.

Zhang, H., Li, X., Li, H., and Huang, F. (2005). Particle swarm optimization-based schemes for resource-constrained project scheduling. *Automation in Construction*, 14(3):393–404.