2010

An accelerating two-layer anchor search with application to the resource-constrained project scheduling problem

Reza Zamani
University of Wollongong

An accelerating two-layer anchor search with application to the resource-constrained project scheduling problem

Abstract

This paper presents a search method that combines elements from evolutionary and local search paradigms by the systematic use of crossover operations, generally used as structured exchange of genes between a series of solutions in genetic algorithms. Crossover operations here are particularly utilized as a systematic means to generate several possible solutions from two superior solutions. To test the effectiveness of the method, it has been applied to the resource-constrained project scheduling problem. The computational experiments show that the application of the method to this problem is promising.

Disciplines

Physical Sciences and Mathematics

Publication Details

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.

An Accelerating Two-Layer Anchor Search with Application to the Resource-Constrained Project Scheduling Problem

Reza Zamani

***Abstract*—This paper presents a search method that combines elements from evolutionary and local search paradigms by the systematic use of crossover operations, generally used as structured exchange of genes between a series of solutions in genetic algorithms. Crossover operations here are particularly utilized as a systematic means to generate several possible solutions from two superior solutions. To test the effectiveness of the method, it has been applied to the resource-constrained project scheduling problem. The computational experiments show that the application of the method to this problem is promising.**

***Index Terms*—Genetic algorithms, heuristics, local search, path relinking, scatter search.**

I. Introduction

SEARCH, as the key concept of artificial intelligence (AI), is involved with a wide range of practical problems from preventive maintenance to manufacturing, from transportation management to operations management, and from enterprise resource planning to the analysis of genes and proteins. In general, the enormous size of search spaces of many practical problems removes any chance for the search space to be implicitly enumerated. This makes the systematic tree search methods as an unrealistic approach to solving these problems and initiates the development of new computational paradigms, mainly based on the cooperation of agents [1].

In general, heuristic methods differ from the systematic tree search methods in the sense that not only the heuristic methods refuse to implicitly enumerate search space but they may manipulate some solutions to attain solutions with higher quality. For instance, by having a prescription for creating complete solutions, local search methods can proceed based on any representation of the neighbors of a constructed solution.

Introduced in [2], genetic algorithms (GAs) serve as an important metaheuristic strategy to solve hard optimization problems. A comprehensive introductory to GAs has been provided in [3]. Genetic algorithms, local searches, and constructive methods comprise a wide range of metaheuristics. The recent applications of metaheuristics are extremely diverse; for instance, they range from optimizing automatic word segments of some languages [4] to radio network design problems [5].

By hybridizing GAs and local search, we present an algorithm, called the accelerating two-layer anchor search (ATLAS). The algorithm aims at progressively improving a single solution and overcoming the local optimality entrapment of local search methods in the continuous improvement of a single solution. The algorithm escapes local optimality through two layers of operations while relying on the notion of anchoring in the best solution found so far. In effect, it continues to modify the best solution found in the hope of producing a sequence of improving solutions through the effective manipulation of the supportive information obtained about the search space.

The two salient features of the ATLAS are: 1) giving the highest possible priority to the best solution achieved, and 2) performing systematic crossover operations for widening the neighborhood of such a high-quality solution. Hence, the major difference of the ATLAS with ordinary GAs is that it concentrates on the continuous improvement of a single high-quality solution.

By considering the encoding of the best solution obtained as a representation of structural information about a promising area, the ATLAS conducts its search efforts in that area both to find better solutions and to update structural information to reflect more promising areas in which further search efforts should be concentrated. In effect, the ATLAS has been built upon the hypothesis that the structural information, about an area which is likely to have optimal or near optimal solutions gained up to a certain point in the search process, can be effectively incorporated into the process of conducting the rest of the search process. Operating in two entirely different layers, the ATLAS in the first layer intensifies its efforts on searching a promising area, and in the second layer attempts to change the area that the first layer was concentrating.

The rest of this paper is devoted to describing the ATLAS and its application to the resource-constrained project scheduling problem (RCPSP). In Section II, the related work is discussed, and in Section III the proposed method is presented. Section IV discusses the resource-constrained project scheduling problem to which the procedure is applied, and Section V presents the application of the method to this problem. The computational results are presented in Section VI, and the concluding remarks as well as the directions for further work are presented in Section VII.

Manuscript received August 8, 2008; revised February 12, 2009, July 8, 2009, February 26, 2010, and March 12, 2010. Date of publication August 26, 2010; date of current version November 30, 2010.

The author is with the Faculty of Informatics, University of Wollongong, New South Wales 2522, Australia (e-mail: reza@uow.edu.au).

Digital Object Identifier 10.1109/TEVC.2010.2047861

II. Related Work

Metaheuristics are general frameworks for solving a very general class of computational problems by steering some underlying heuristics aimed at effective exploitation of the problem structure. These underlying heuristics can be mainly categorized into constructive and local search methods, which create constructive and local search metaheuristics, respectively. While constructive metaheuristics [6], [7] build a solution in an incremental way, local search metaheuristics [8]–[10] operate based on the exploration of solution neighborhoods.

Moreover, while constructive metaheuristics are aimed at improving the performance of their underlying heuristics through introducing a bias in their selection criterion, local search metaheuristics are aimed at avoiding local optimality encountered in the most basic version of the local search algorithm, called iterative improvement. This iterative improvement version is specifically referred to as hill-climbing, and gradient-descent in maximization and minimization problems, respectively, and is based on terminating the search process as soon as no neighbor can offer an improvement.

While local search and constructive metaheuristics rely on manipulating a single solution, the evolutionary metaheuristics, in general, and GAs [2], in particular, are based on manipulating a pool of solutions. Crossover operations can be considered as the main mechanism used by GAs to both diversify and intensify the search. The design of crossover operations is based on the rationale that good solutions tend to have a great deal of common structures, and utilizing these structures increases the quality of the survived solutions.

Relying on these effective structures, however, is not limited to genetic algorithms. Two of the most successful methodologies that innovatively utilize such effective structures are scatter search and path relinking [11].

Initially proposed as a heuristic for integer programming [12], scatter search now offers several conceptual advantages in integrating evolutionary algorithms with stochastic local search methods [13]. These integrations aim at achieving search diversification and hence increase the exploration capabilities of the search process through the combination of promising features from a number of elite solutions.

Originally suggested as an approach to balance diversification against intensification in the context of tabu search [14], the idea of path relinking now is considered as a generalization of scatter search [13]. In effect, path relinking extends the solution combination method of scatter search in the direction of generating paths between and beyond reference set solutions in neighborhood space. The nature of such paths is multifaceted and can be formed based on a variety of changes including the addition, reduction, or any modification of some solution attributes. With such a broad conception of solution combination, path relinking offers numerous alternatives for exploiting the structure of two or more solutions for the purpose of exploring different trajectories that connect these solutions.

Scatter search and path relinking make their exploration via a set of reference points that are both diverse and of good quality. In a template for scatter search and path relinking methodologies, Golver [15] outlines five fundamental components, including: 1) diversification generator; 2) improvement method; 3) reference set update method; 4) subset generation method; and 5) solution combination method. Among the five components mentioned, the second one is optional but the others are mandatory.

While the diversification generator generates a collection of diverse trial solutions, the improvement method converts the trial solutions to the improved solutions, and "the reference set update method" builds and maintains a reference set consisting b best solutions found. The value of b is typically between 20 and 40, and these b best solutions are selected among a larger number of trial improved solutions.

A key point in choosing these best b solutions is that, the selection criterion is based on both the quality and diversity of solutions. The subset generation method, as the fourth component, operates on the reference set and produces different subsets of these best b solutions. Consequently, these subsets of solutions are used by the fifth component, the solution combination method, which transforms each given subset of solutions into one or more solution vectors. These solution vectors can update the reference set and activate a new iteration.

More advanced designs for scatter search and path relinking have been offered in [13]. These advanced designs include: 1) the dynamic updating of the reference sets; 2) the creating of multitier reference sets; 3) the employing of effectual diversity controls; and 4) the effective use of memory in interactions among the components mentioned.

Using these advanced designs, we embed one and two-point crossover operations into the most basic version of local search (iterative improvement) to create a hybrid that can overcome local optimality. In deriving such a hybrid, two factors of conceptual simplicity and effectiveness have been the most important considerations.

III. ATLAS

The ATLAS starts with constructing two local optimal solutions. The encoding of one of the solutions is called "*base genome*" and that of the other is called "*transient genome.*" The base genome is alternatively called the *base* as well. The base and transient genomes are systematically crossed over and, among the offspring generated, the best one is selected. If the best offspring is not inferior to the base, it will replace the base. Regardless of the replacement of the base, another transient genome is generated, and again the base and transient genomes are systematically crossed over. The course of actions of generating transient genomes, performing systematic crossover operations, and the probable replacement of the base genome continues until a feedback mechanism terminates the process. The feedback process measures the length of the last period in which no progress has been made and as soon as this length exceeds a limit, the procedure is terminated.

Unlike ordinary crossover operations used in GAs, the systematic crossover incorporated in the ATLAS does not rely on a random combination of two genomes. On the contrary, in line with path relinking framework [13], it produces a

set of genomes that collectively can be considered as diverse combinations of their parents. In particular, the ATLAS produces several possible combinations of the two genomes subject to the rule that a number of contiguous elements of the base genome are maintained. In effect, the ATLAS can be considered as a hybridization of local search and evolutionary algorithms which aims at:

1) concentrating on improving a single solution encoded by the base genome;
2) crossing over the base and transient genomes systematically, in the sense of generating all their possible offspring;
3) replacing the base genome with the best offspring generated if it is not inferior to the base;
4) generating a high-quality transient genome repeatedly by restarting a local search process which uses different starting points.

Mechanism 1 forces the method to fine-tune a single solution (the base genome) and prevents disorganized efforts on improving a pool. Mechanism 2 widens the scope of the crossover operation and uses this operation to systematically create a set of neighbors. Mechanism 3 guarantees that the best solution obtained up to any point of the search process is encoded in the genome that is being fine-tuned for further improvements. Mechanism 4 scans different parts of the search space and at the same time keeps the quality of the transient genome as high as possible.

Four factors contribute to the quality of the offspring generated: 1) in each round the transient genome is quite independent of the base genome; 2) the base genome is the encoding of the best solution obtained so far; 3) the transient genome itself is the result of a local search process; and 4) crossover operations are performed systematically. The combination of these factors results in selecting a superior offspring among a large number of high-quality genomes and, therefore, increases the potential quality of the base.

Fig. 1 shows a C-type pseudocode for the ATLAS. The pseudocode first generates a random solution encoding called s, at line 4. The solution encoding of each particular problem has a constant number of elements. The definition of "*element*" depends on the problem at hand and the encoding used; for instance, a solution encoding for the traveling salesman problem (TSP) can be provided by a permutation of n cities (elements). Note that unlike in the TSP where each solution encoding can be easily converted to a real solution, in most problems, like the RCPSP to which we have applied the ATLAS, an appropriate decoder is required for such conversion.

The random solution encoding, generated at line 4, is used as a seed for the local search process in line 5, which produces the base. Lines 9 through 19 are repeatedly executed until no improvement becomes possible. In line 9, a random solution encoding is generated as a seed to produce a transient genome in line 10. A nested loop starts at line 12, which attempts to perform systematic crossover operations on the transient and base genomes.

In line 12, the parameter n represents the size of each solution encoding in terms of its elements and the value of $f(n)$ is bounded by G, which will be introduced in (2) shortly. In each iteration, a genome is generated by maintaining a number (between 1 and $n-1$) of contiguous elements of the base and obtaining the remaining elements from the transient genome. If the quality of the genome produced is not inferior to that of the base, line 17 replaces the base with such a genome.

In general, there is a limited number of ways to maintain i contiguous elements of the base genome and to fill the remaining $n-i$ elements from their corresponding elements of the transient genome. The pseudocode starts with maintaining i elements from 1 to i; and then from 2 to $1+i$, and finally from $n-i+1$ to n. Therefore, if m_i represents the number of potential ways to maintain i contiguous elements of the base solution, its value can be calculated as follows:

$$m_i = n - i + 1. \tag{1}$$

This means that the number of iterations performed by line 12 of the pseudocode, *f(n)*, is bounded by G calculated as follows:

$$G = \sum_{i=1}^{n-1} m_i = \sum_{i=1}^{n-1} (n-i+1) = \frac{(n-1)(n+2)}{2}. \tag{2}$$

It should be noted that G shows the maximum number of genomes that can be produced and includes both one and two-point crossover operations. However, in cases where it is decided to generate smaller number of genomes, the choice can be limited to solely one-point crossover operations. This is simply obtained by maintaining the first i elements of the base genome and selecting the remaining ones from the transient genome. In this case, the number of produced genomes will be reduced to $n-1$.

When the base and transient genomes are systematically crossed over, any similarity between the two genomes can be exploited both to keep the quality of the resultant offspring high and to reduce the number of offspring genomes generated. To exploit some of these similarities, the ATLAS distinguishes the similar corresponding elements of the two genomes, and after fixing the positions of those elements in all offspring, focuses only on the remaining elements. In the following, we show the contribution of this operation to the functioning of the procedure.

By ignoring the similar elements, and fixing their positions in offspring, the size of each solution, n, decreases, and since G depends on n, the value of G in (2) changes. To measure the effect of fixing positions on the changes of G, let $100 \times (1-\beta)$ percent of elements have the same positions in the two genomes. This will change n to ßn and the new value of G can be computed based on

$$\hat{G} = \frac{(\beta n - 1)(\beta n + 2)}{2}. \tag{3}$$

The ratio of the new value to the previous one is shown as

$$\lim_{n\to\infty} \frac{\hat{G}}{G} = \beta^2. \tag{4}$$

Hence, the effect of the fixing of positions can be demonstrated by the quadratic relation represented in (4). For the

```
1    ATLAS()
2    {
3       Set n to the number of elements of the solution encoding.
4       Generate a random solution encoding and call it s;
5       Improve s via the local search process and call the result Base;
6       Improvement=.True.;
7       While (Improvement)
8       {
9          Generate a random solution encoding and call it s;
10         Improve s via the local search process and call the result
11                                                          Transient;
12          for (i=1;i<f(n);i++)//performing systematic crossover
13          {
14            Generate a genome by selecting the determined elements from
15                          the Base and its other elements from Transient;
16            If the quality of the genome is not inferior to that of Base
17                                       then replace Base with the genome;
18          }
19          If (the feedback process enforces halting)Improvement=.False.;
20      }
21      return the solution associated with Base
22   }
```

Fig. 1. C-type pseudocode of the ATLAS.

purpose of simplicity and in view of relatively insignificant effect which this fixing has, the pseudocode has not shown the details involved with distinguishing these similarities between the two genomes.

Since among all the solutions generated, none is of better quality than that associated with the base, in line 21, the solution associated with the base has been returned as the output of the ATLAS. The halting criterion, in line 19, is determined based on a feedback mechanism that checks the progress made in improving the solution. In general, by the passage of time, the process of improvement in the base genome can become exponentially slow. That is why the feedback mechanism, in line 19, checks the progress made in the base and as soon as for a specified period of time, τ, no improvement is detected, it terminates the operations. The larger values of τ lead to better solution qualities at the cost of requiring more computational resources.

IV. Resource-Constrained Project Scheduling Problem

The RCPSP subsumes job shop, flow shop, and assembly line balancing in scheduling [16], and occurs in a variety of applications such as packets competing to pass a network, computer programs competing for processor resources, radio-isotope production in nuclear reactors, and the control of multiproduct chemical processes.

Not only is the RCPSP strongly non-deterministic polynomial-time hard (NP-hard), but is also hard to approximate [17]. Moreover, despite being one of the most intractable problems in combinatorial optimization, it can be stated simply.

The RCPSP is involved with scheduling a project that has a number of activities. The precedence relations between activities are represented by the set of immediate predecessors, indicating that each activity cannot start unless all activities in its predecessors set have been completed. Two fictitious activities with zero duration and no resource requirements represent the starting and ending of the project, respectively.

Each activity has a specified duration and needs some resource requirements from a set of limited resources. The resources are renewable, i.e., regardless of the project length, each resource is available for every single period. Typical renewable resources are equipments, machines, and manpower. Preemption is not allowed, in the sense that once an activity starts, it should continue until it is completed. The goal is

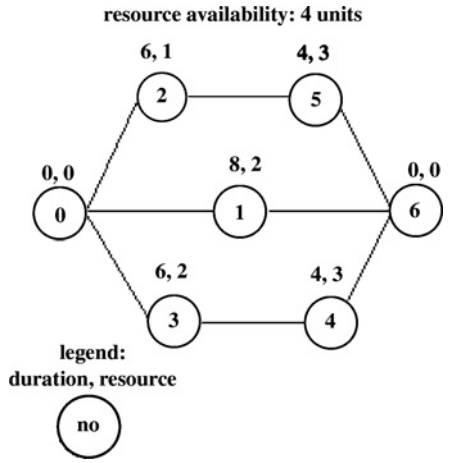

Fig. 2. Sample project with limited recourse availability.

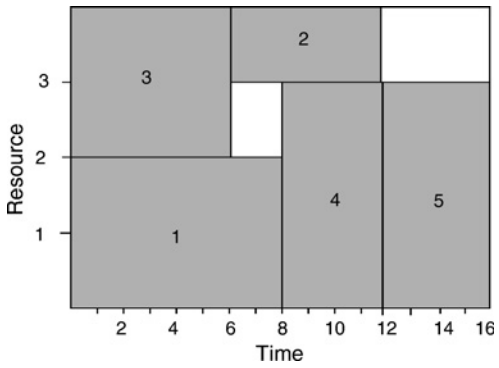

Fig. 3. Optimal schedule associated with the sample project.

to find the starting times of activities so that the project is completed in the earliest possible time. Fig. 2 illustrates a simple instance of the RCPSP, which is involved with only one resource type, and Fig. 3 presents its optimal schedule.

A solution to the RCPSP must be a feasible resolution of two types of constraints. First, there are limits on the capacity of available resources; and second, there are restrictions on the order in which the activities of a project can be performed. An extensive literature survey on heuristic solutions presented to the RCPSP can be found in [18]. Among the ideas underlying the procedures presented, we can mention those related to electromagnetism [19], ant colony optimization [20], scatter search and path relinking [19], [21]–[23], as well as rollout augmentation [24].

In most of the heuristic methods presented for this problem, two schedule generation schemes play key roles, namely serial and parallel. The serial and parallel generation schemes are the results of pioneering works of Kelly [25] and Brooks [26], respectively. These two methods operate by extending a partial schedule, in which only a subset of activities have been assigned a starting time. While the serial schedule generation scheme is activity oriented, the parallel generation scheme is time oriented. As an activity oriented scheme, the serial method consists of N stages and in each stage it schedules one activity from a precedence-feasible list at the earliest precedence and resource-feasible start time. On the other hand, as a time oriented scheme, the parallel method forwards time incrementally, and, at each point of time, it schedules a number of eligible activities based on their priorities. While the parallel scheme is superior for large sample sizes and hard instances, the serial scheme shows better results for small samples and easy instances [27]. This indicates the complementary nature of these two schemes when applied to a diverse set of benchmark instances.

Li and Willis [28] have developed an iterative forward/backward scheduling technique that iteratively applies serial forward backward scheduling to a schedule until no further improvement in the project duration can be obtained. As two extensions of forward/backward method, a method called backward forward (BF) [29], and a similar method called double justification [30] have proved very effective mechanisms to improve the schedules obtained by the serial or parallel method. Because of the strong similarity between the justification method presented in [30] and the BF procedure presented in [29], Kolisch and Hartmann [18] refer to both approaches by the notion of forward backward improvement (FBI).

FBI employs the serial schedule generation scheme and alternates the processing of forward and backward passes. Both of these passes are based on the concept of free slack. While the backward pass is based on forward free slack, the forward pass is based on backward free slack. Backward (forward)-free slack of an activity in a feasible schedule is the amount of time that the corresponding activity can shift left (right) without affecting the starting times of other activities.

In this method, while the backward pass is applied by shifting activities right to the latest feasible position into their forward-free slack, the forward pass is applied by shifting activities to the earliest feasible position into their backward free slack. The backward pass is performed based on decreasing order of finish times of activities and the forward pass is based on increasing order of the starting times.

V. Application of the ATLAS to the RCPSP

To solve the RCPSP with the ATLAS, three requirements should be addressed, namely: 1) a solution encoding; 2) a decoder for converting solution encodings to real solutions; and 3) a local search for producing the base and transient genomes. This section describes these three requirements.

For its solution encoding, the ATLAS employs a permutation based encoding called activity list [31]. Here, we represent each activity list by "A-list," both referring to the same object. In effect, each A-list represents a genome and is a list of activities subject to the rule that no activity appears in the list before any of its predecessors.

It should be noted that in combining two A-lists, through crossover operations, any repetition should be prevented; otherwise the offspring produced does not show a permutation of activities. Such prevention can be easily handled by the rule presented in [32]. Based on this rule, first the elements of the transient genome are placed in the offspring and then the remaining contiguous elements of the offspring are ordered based on their associated orders in the base genome.

The second requirement, a decoder for converting A-lists to feasible schedules, is simply obtained through the application

of both serial and parallel schedule generation schemes to the associated A-list and then the application of BF the best schedule generated.

Fig. 4 presents a pseudcode describing how the ATLAS solves the RCPSP. Comparing the general function of the ATLAS, outlined in Fig. 1, with its specific application to the RCPSP, outlined in Fig. 4, reveals how easily the ATLAS can be extended to solve similar problems. The differences between the two pseudocodes mainly point to lines 4 and 9 and are associated with generating A-lists.

For finding an initial A-list, the revised floats [33] of activities are progressively computed and based a candidate set of size 2, through the application of greedy randomized search [7], the activities are selected and ranked. Fig. 5 presents a simple pseudocode describing how an A-list is generated.

In Fig. 4, after making a request for generating an A-list in lines 4 and 9, a local search process is performed in lines 5 and 10 to improve the quality of the A-list generated. For improving an A-list, the following four operations are repeatedly performed: 1) its neighbors are generated; 2) the quality of each neighbor is calculated; 3) the best neighbor is selected; and 4) the A-list is replaced with its highest quality neighbor if the neighbor is not inferior to it.

The local search employed, as the third requirement, can generate neighbors through the extended right and left-shifting of an activity on the list within any specified diameter [34]. Whenever a shift can lead to a better schedule, the shift is performed and the A-list is improved. The local search process is terminated whenever local optimality is reached, where the shifting of no activity within the specified diameter can lead to a better schedule. This search process is responsible for generating the initial base as well as transient genomes.

While line 5 of the pseudocode generates a base genome only once, line 10 generates a new transient genome as many times as needed. The transient genome generated is systematically crossed over with the base genome and, among all offspring generated, the offspring with the highest quality replaces the base genome if not inferior to it.

VI. Computational Experiments

The procedure has been coded in C++ and the results have been obtained using a DELL PC with 1.86 GHz double processors where 1 GB RAM has been assigned to each processor. With using only one processor, and consequently 1 GB RAM, the performance of the procedure has been investigated on several standard sets obtainable from the Project Scheduling Problem Library (PSPLIB) [35]. Four standard sets namely $j30$, $j60$, $j90$, and $j120$ with 32, 62, 92, and 122 activities, respectively, have been employed. Each of the first three sets includes 480 instances and the fourth set includes 600 instances.

As a performance measure, we have used the critical path method (CPM) deviation percentage, shown by CPM_dev%, which is a key performance measure in the literature and indicates the percentage deviation of the result obtained by the procedure from the CPM lower-bound. The reason for using CPM_dev%, as a performance measure, is twofold. First, the optimal solutions of some benchmark instances are unknown and, therefore, comparing all results with their corresponding optimal solutions is impossible. Second, the CPM lower bound can be easily calculated by relaxing resource constraints and finding the optimal solution of the relaxed problem.

To fix the parameters of the procedure, we have performed some preliminary experiments with a very small number of instances. These instances have been selected so that they can be a fair representative of all 2040 instances. It should be noted that employing a large number of instances could incur a long time to find the proper values of the parameters. With respect to fixing the parameters, three points are worth noticing.

First, as mentioned, G in (2) shows the maximum number of genomes that can be produced. There are many ways to reduce the number of offspring generated. For instance, out of each ten consecutive genomes only the last can be considered and the other nine ones can be simply ignored. For consistency purposes, however, we have not set any limitation on the number of crossover operations.

The second point is that selecting large values for τ could increase solution time for some instances beyond a reasonable time and smaller values could degrade their solution quality. Therefore, we decided to deactivate the termination feedback mechanism of the procedure and instead to set two independent limits. The first limit is that the total number of schedules generated should not exceed half a million and the second limit is that the number of transient genomes generated should not exceed 80. It should be noted that the size of the neighborhood used and the two-layer feature of the procedure does not let us to examine the performance of the procedure for small number of generated schedules. Hence, no consideration has been given to tighter limits on the number of schedules.

The third point is involved with setting a diameter for the local search employed. The value determined for this diameter is 60, which, for a large number of examined benchmark instances, implies the scanning of the entire length of A-list.

We have used four different scenarios to show the effects of different schedule generation schemes on the performance of the procedure. While the first and second scenarios are involved with the serial and parallel schedule generation schemes, respectively, the third one is a mixed option and utilizes each scheme randomly with equal chances between both schemes. The fourth scenario is a cumulative one, in the sense that it applies the three other scenarios to each benchmark instance and selects the best result obtained. Obviously, the fourth scenario produces better results at the cost of extra execution time. Table I shows the performance of the ATLAS under these four scenarios. In all the cases, the ATLAS has used large neighborhood.

One of the sophisticated procedures based on large neighborhood search that have produced high-quality solutions for the RCPSP is local search with subproblem exact resolution (LSSPER) [36]. LSSPER has improved the best known solutions for 14, 9, and 4 of benchmark instances in the sets j60, j90, and j120, respectively. LSSPER has been run on a PC with 1 GB RAM and 2.3 GHz speed and has scheduled all 2040 benchmark instances to which the ATLAS has been

```
1    Apply-ATLAS-TO-THE-RCPSP
2    {
3       Let G represent the number of crossover operations;
4       Generate a random A-list for activities;
5       Improve the A-list via the local search and call the result Base;
6       Improvement=.True.;
7       While (Improvement)
8       {
9          Generate a random A-list for activities;
10         Improve the A-list via the local search and call the result
11                                                          Transient;
12          for (i=1;i<G;i++)//producing A-lists via crossover
13          {
14            Generate an A-list by selecting the determined elements
15                    from Base and its other elements from Transient;
16            If the quality of the A-list is not inferior to that of
17                           Base, then replace Base with the genome;
18          }
19          If (halting criteria is reached) Improvement=.False.;
20       }
21       return the schedule associated with Base
22   }
```

Fig. 4. C-type pseudocode for applying the ATLAS to the RCPSP.

TABLE I
PERFORMANCE OF THE ATLAS UNDER DIFFERENT SCENARIOS

Set	Serial		Parallel		Mixed		Cumulative	
	Time (seconds)	Average CPM_dev %	Time (seconds)	Average CPM_dev %	Time (seconds)	Average CPM_dev %	Time (seconds)	Average CPM_dev %
30	1.3	13.41	1.2	13.49	1.3	13.38	3.8	13.37
60	12.6	11.03	12.2	11.05	12.4	10.93	36.2	10.81
90	19.7	10.73	19.1	10.58	19.3	10.53	58.1	10.34
120	37.1	33.31	36.6	33.20	36.8	32.83	110.5	32.45

TABLE II
COMPARING THE PERFORMANCE OF THE ATLAS WITH THAT OF LSSPER

Set	ATLAS		LSSPER	
	Average CPM_dev %	Time (seconds)	Average CPM_dev %	Time (seconds)
30	13.37	4	13.37	10
60	10.81	37	10.81	38
90	10.34	59	10.29	61
120	32.45	111	32.41	207

applied. Table II compares the performance of the ATLAS with that of LSSPER.

Although the performance of the procedure in its current state is comparable with the performance of LSSPER, there is an important point to note: In order to reach top performance, a metaheuristic should use problem-specific knowledge, and in its current implementation, because of not obscuring the function of the main features the ATLAS, the procedure has not used all such knowledge. In this regard, the following four key specific pieces of knowledge may further improve the performance.

First, the procedure exactly follows the same general theme of the ATLAS in systematically performing all possible one and two-point crossover operations. After all, the ATLAS is a general algorithm based on the notion of generating a large number of very similar genomes. With an activity list as an encoding solution in the RCPSP, however, most of nearly-identical genomes and even some similar genomes can lead to an identical solution and this hinders the effective scanning

```
 1:  Generate-A-List(n)
 2:  {
 3:    Set k = 0,and set the revised floats of activities to total floats;
 4:    Initialize the temporary network to the original network;
 5:    Initialize the eligible list by placing the fictitious starting
 6:    activity in it;
 7:    while ( k < n )
 8:    {
 9:      Let  m represent the number of elements in the eligible list;
10:      Using the revised floats and a random component, select i.
11:      Let S represent the i^th element of the eligible list;
12:      Assign order of k to S, and remove S out of the eligible list;
13:      k = k+1;
14 :     Update the temporary network and set the revised float of each
15:         activity to its total float computed in the temporary network;
16:      if the removing of S makes any activity (activities)
17:         precedence-feasible then add it (them) to the eligible list;
18:     }
19:     return the A-list;
20:  }
```

Fig. 5. C-type pseudocode describing the generation of a random A-list.

of the search space. Using standard random key [19] instead of activity list, for encoding solutions, may be a remedy.

Second, the decoding procedure used is time-consuming. Therefore, discarding a large percentage of unfruitful genomes before decoding them may improve the performance. One of the criteria that in this direction can be used is the idea of "peak crossover" presented in [37]. Based on an extension version of this idea, we can discard those genomes that have specific combination of activities that are expected to lead to low utilization of recourses.

Third, in the RCPSP, it is possible to enforce some new precedence relations to reduce search space. For this purpose, all pairs of activities that are not in any precedence relation are potential candidates to introduce new precedence relations. In general, enforcing such precedence augmentation can improve the performance of search through preventing many low-quality solutions from being formed. In this regard, the precedence augmentation mechanism presented in [34] seems a suitable candidate for being incorporated into the procedure.

Fourth, in the RCPSP there are a number of structural characteristics, like "network complexity" and "recourse strength" [35] that highly affect the impact of heuristics employed as well as the associated parameters. Here, in the current implementation, the parameters and the heuristics used are independent of problem instances. To circumvent this limitation, the base and transient genomes should be extended to keep the information required for self-adaptation. This will lead to an adjustment capability, which is problem-specific. With respect to integrating such capability, the self-adaptation mechanism proposed in [31] is helpful.

VII. Conclusion

Although many local search techniques appear to be only superficial variants of one another, they display significantly different problem solving capabilities. This is because of two interrelated facts that: 1) the effectiveness of a local search method is determined by highly conflicting features, and 2) successful local search methods effectively manage a trade-off between these features. Two of the features which the ATLAS attempts to balance against each other are exploitation and exploration. While improving a single solution, the ATLAS aims at preventing local optimality by making sure that the search process explores vast regions without being trapped in relatively confined ones.

In each iteration, it exploits the results obtained in previous iterations, and rather than relying on restarting and randomness, it directly aims at enhancing the solution anchored in. To emphasize on exploration, the ATLAS progressively and systematically spreads the characteristics of different locally optimal solutions found in different regions of search space in the solution anchored in.

The ATLAS has been designed based on the notions of: 1) concentrating on effective manipulation of two dynamic genomes; 2) crossing over the base and transient genomes systematically; 3) replacing the base genome with any dominant offspring generated; and 4) generating high-quality transient genomes.

Some directions for sketching future work are as follows. First, because of its concentration on improving a single solution, the ATLAS can be easily modified to be used in a parallel environment in which different threads cooperatively use the same base genome and find different transient genomes. In this case, the only information needed to be communicated between the threads is the base genome.

Second, to improve its exploitation power, the ATLAS can periodically empty different parts of the base genome from their solution components and optimally rearrange the solution components to refill the emptied parts. The process of emptying and refilling can be repeated for different overlapping contiguous parts of the base genome until a terminating criterion is satisfied [38]. Moreover, a learning capability can be embedded in the ATLAS to find out the best values of the parameters used in the procedure in dealing with a given problem instance [39].

REFERENCES

[1] M. Dorigo and L. M. Gambardella, "Ant colony system: A cooperative learning approach to the traveling salesman problem," *IEEE Trans. Evol. Comput.*, vol. 1, no. 1, pp. 53–66, Apr. 1997.

[2] J. H. Holland, *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: Univ. Michigan Press, 1975.

[3] F. Glover, J. P. Kelly, and M. Laguna, "Genetic algorithms and tabu search: Hybrids for optimization," *Comput. Oper. Res.*, vol. 22, no. 1, pp. 111–134, 1995.

[4] G. Tambouratzis, "Using an ant colony metaheuristic to optimize automatic word segmentation for ancient Greek," *IEEE Trans. Evol. Comput.*, vol. 13, no. 4, pp. 742–753, Aug. 2009.

[5] S. P. Mendes, G. Molina, M. A. Vega-Rodriguez, J. A. Gomez-Pulido, Y. Saez, G. Miranda, C. Segura, E. Alba, P. Isasi, C. Leon, and J. M. Sanchez-Perez, "Benchmarking a wide spectrum of metaheuristic techniques for the radio network design problem," *IEEE Trans. Evol. Comput.*, vol. 13, no. 5, pp. 1133–1150, Oct. 2009.

[6] M. Dorigo and T. Stützle, *Ant Colony Optimization*. Cambridge, MA: MIT Press, 2004.

[7] T. A. Feo and M. G. C. Resende, "Greedy randomized adaptive search procedures," *J. Global Optim.*, vol. 6, no. 2, pp. 109–133, 1995.

[8] F. Glover and M. Laguna, *Tabu Search*. London, U.K.: Kluwer, 1997.

[9] S. Kirkpatrick, "Optimization by simulated annealing: Quantitative studies," *J. Statist. Phys.*, vol. 34, no. 5, pp. 975–986, 1984.

[10] C. Voudouris and E. P. K. Tsang, "Guided local search," in *Handbook of Metaheuristics*, F. Glover, Ed. Norwell, MA: Kluwer, 2003, pp. 185–218.

[11] F. Glover, M. Laguna, and R. Marti, "Fundamentals of scatter search and path relinking," *Control Cybernet.*, vol. 39, no. 3, pp. 653–684, 2000.

[12] F. Glover, "Heuristics for integer programming using surrogate constraints," *Decision Sci.*, vol. 8, no. 1, pp. 156–166, 1977.

[13] R. Marti, M. Laguna, and F. Glover, "Principles of scatter search," *Eur. J. Oper. Res.*, vol. 169, no. 2, pp. 359–372, 2006.

[14] F. Glover and M. Laguna, "Tabu search," in *Modern Heuristic Techniques for Combinatorial Problems*, C. Reeves, Ed. Oxford, U.K.: Blackwell, 1993, pp. 70–141.

[15] F. Glover, "A template for scatter search and path relinking," in *Lecture Notes in Computer Science*, vol. 1363. Berlin: Springer, 1998, pp. 13–54.

[16] P. Brucker, "Scheduling and constraint propagation," *Discrete Appl. Math.*, vol. 123, nos. 1–3, pp. 227–256, 2002.

[17] R. H. Möhring, A. S. Schulz, F. Stork, and M. Uetz, "Solving project scheduling problems by minimum cut computations," *Manage. Sci.*, vol. 49, no. 3, pp. 330–350, 2003.

[18] R. Kolisch and S. Hartmann, "Experimental investigation of heuristics for resource-constrained project scheduling: An update," *Eur. J. Oper. Res.*, vol. 174, no. 1, pp. 23–37, 2006.

[19] D. Debelsa, B. De Reyckb, R. Leusc, and M. Vanhouckea, "A hybrid scatter search/electromagnetism meta-heuristic for project scheduling," *Eur. J. Oper. Res.*, vol. 169, no. 2, pp. 638–653, 2006.

[20] D. Merkle, M. Middendorf, and H. Schmeck, "Ant colony optimization for recourse-constrained project scheduling," *IEEE Trans. Evol. Comput.*, vol. 6, no. 4, pp. 333–346, Aug. 2002.

[21] Y. A. Kochetov and A. A. Stolyar, "Evolutionary local search with variable neighborhood for the resource constrained scheduling problem," in *Proc. Workshop Comput. Sci. Inform. Technol.*, 2003, pp. 1–4.

[22] M. D. M. Mobini, M. Rabbani, M. S. Amalnik, J. Razmi1, and A. R. Rahimi-Vahed, "Using an enhanced scatter search algorithm for a resource-constrained project scheduling problem," *Soft Comput.-A Fusion Foundat., Methodol. Applicat.*, vol. 13, no. 6, pp. 597–610, 2009.

[23] V. Valls, F. Ballestin, and S. Quintanilla, "A population-based approach to the resource-constrained project scheduling problem," *Ann. Oper. Res.*, vol. 131, nos. 1–4, pp. 305–324, 2004.

[24] N. Xu, S. A. McKee, L. K. Nozick, and R. Ufomata, "Augmenting priority rule heuristics with justification and rollout to solve the resource-constrained project scheduling problem," *Comput. Oper. Res.*, vol. 35, no. 10, pp. 3284–3297, 2008.

[25] J. Kelley, "The critical-path method: Resource planning and scheduling," in *Industrial Scheduling*, J. F. Muth and G. L. Thompson, Eds. Englewood Cliffs, NJ: Prentice-Hall, 1963, pp. 347–365.

[26] D. D. Bedworth and J. E. Bailey, "Integrated production control systems-management," in *Analysis, Design*. New York: Wiley, 1982.

[27] R. Kolisch, "Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation," *Eur. J. Oper. Res.*, vol. 90, no. 2, pp. 320–333, 1996.

[28] K. Li and R. Willis, "An iterative scheduling technique for resource-constrained project scheduling," *Eur. J. Oper. Res.*, vol. 56, no. 3, pp. 370–379, 1992.

[29] P. Tormos and A. Lova, "A competitive heuristic solution technique for resource-constrained project scheduling," *Ann. Oper. Res.*, vol. 102, no. 1, pp. 65–81, 2001.

[30] V. Valls, F. Ballestin, and S. Quintanilla, "Justification and RCPSP: A technique that pays," *Eur. J. Oper. Res.*, vol. 165, no. 2, pp. 375–386, 2005.

[31] S. Hartmann, "A self-adapting genetic algorithm for project scheduling under resource constraints," *Naval Res. Logistics*, vol. 49, no. 5, pp. 433–448, 2002.

[32] S. Hartmann, "A competitive genetic algorithm for resource-constrained project scheduling," *Naval Res. Logistics*, vol. 45, no. 7, pp. 733–750, 1998.

[33] K. R. Baker, *Introduction to Sequencing and Scheduling*. New York: John Wiley and Sons, 1974.

[34] K. Fleszar and K. S. Hindi, "Solving the resource-constrained project scheduling problem by a variable neighborhood search," *Eur. J. Oper. Res.*, vol. 155, no. 2, pp. 402–413, 2004.

[35] R. Kolisch and A. Sprecher, "PSPLIB: A project scheduling library," *Eur. J. Oper. Res.*, vol. 96, no. 1, pp. 205–216, 1996.

[36] M. Palpant, C. Artigues, and P. Michelon, "LSSPER: Solving the resource-constrained project scheduling problem with large neighborhood search," *Ann. Oper. Res.*, vol. 131, no. 1, pp. 237–257, 2004.

[37] V. Valls, F. Ballestin, and S. Quintanilla, "A hybrid genetic algorithm for the resource-constrained project scheduling problem," *Eur. J. Oper. Res.*, vol. 185, no. 2, pp. 495–508, 2008.

[38] M. Yagiura and T. Ibaraki, "The use of dynamic programming in genetic algorithms for permutation problems," *Eur. J. Oper. Res.*, vol. 92, no. 2, pp. 387–401, 1996.

[39] R. Zamani and S. K. Lau, "Embedding learning capability in Lagrangean relaxation: An application to the traveling salesman problem," *Eur. J. Oper. Res.*, vol. 201, no. 1, pp. 82–88, 2010.

Reza Zamani received the B.S. degree in computer science and operations research, the M.S. degree in systems engineering from the Department of Industrial Engineering and Center for Systems Planning, Isfahan University of Technology, Isfahan, Iran, in 1985, and the Ph.D. degree in operations research from the University of Wollongong, New South Wales, Australia, in 1995.

He is currently a Senior Academic Staff Member with the Faculty of Informatics, University of Wollongong. For more than 25 years, he has widely worked on Lagrangean relaxation, local search, mathematical programming and the development of evolutionary algorithms. The results of his experimentations have been published in several operations research journals, such as the *European Journal of Operational Research*, *Computers and Operations Research*, *OR Spectrum*, the *Operational Research Society* as well as in several artificial intelligence journals, such as the *Artificial Intelligence Review*, *Minds and Machines*, and *Intelligent Systems*. He is currently looking for motivated Ph.D. students familiar with both mathematical programming and artificial intelligence. His current research interests include the development of intelligent search strategies with emphasis on the integration of operations research and artificial intelligence.