Publié par : Faculté des sciences de l'administration
Published by : Université Laval
Publicación de la : Québec (Québec) Canada G1K 7P4
Tél. Ph. Tel. : (418) 656-3644
Fax : (418) 656-7047

Édition électronique : Aline Guimont
Electronic publishing : Vice-décanat - Recherche et partenariats
Edición electrónica : Faculté des sciences de l'administration

Disponible sur Internet : http ://www.fsa.ulaval.ca/rd
Available on Internet rd@fsa.ulaval.ca
Disponible por Internet :

DOCUMENT DE TRAVAIL 2003-033

A LOCAL SEARCH FOR THE RESOURCE-CONSTRAINED PROJECT SCHEDULING PROBLEM

Michel Gagnon
Gilles d'Avignon
Fayez F. Boctor

Version originale : ISBN – 2-89524-183-X
Original manuscript :
Version original :

Série électronique mise à jour : 09-2003
On-line publication updated :
Seria electrónica, puesta al dia

A Local Search for the Resource-Constrained Project Scheduling Problem

Michel Gagnon[a]
[a]*Defence R&D Canada – Valcartier, 2459 Pie-XI Blvd North, Val-Bélair, QC, Canada, G3J 1X5*
Michel.Gagnon@drdc-rddc.qc.ca

Gilles d'Avignon[b] and Fayez F. Boctor[b]
[b]*Faculty of Business Administration, Laval University, Quebec, QC, Canada, G1K 7P4*
gilles.davignon@fsa.ulaval.ca fayez.boctor@fsa.ulaval.ca

Abstract

This paper presents a new and efficient Local Search procedure to solve the Resource-Constrained Project Scheduling problem with variable resource availability. It investigates the efficiency of some neighborhood structures and describes the strategies used to reduce the computation time. Numerical experimentations show that the proposed Local Search procedure provides competitive quality solutions compared to others heuristics proposed in the literature and it provide the best-known solutions to two benchmark problem sets.

Keywords: Resource-constrained project scheduling; Local Search

1. Introduction

Project scheduling is an important issue in project management. It is a process that refers to the allocation of resources over time to perform a set of planned activities in order to optimize performance criteria sought. It is a research domain for which it has been numerous papers over the last decade. This paper deals with the classical Resource-Constrained Project Scheduling (RCPS) problem. The classical RCPS problem is a scheduling problem for which limited amounts of multiple resource types are allocated in a way to minimize the project duration. Historically, this problem has been tackled by optimal and approximate methods, the later being commonly referred as heuristics. The recent papers of Herroelen et al. (1998) and Kolisch et al. (2001) survey the RCPS problem and its numerous variants.

In the optimal method category, we find branch and bound algorithms applying different dominance rules to shorten the search space. There have been numerous attempts to develop fast optimal solution procedures for the RCPS problem (cf. Stinson (1978), Demeulemeester and Herroelen (1992), Mingozzi et al. (1998)). Demeulemeester and Herroelen (1992) propose a branch and bound algorithm based on the concept of minimal delaying alternatives to resolve resource conflicts. Each node of the branch and bound represents a resource and a precedence feasible partial schedule, and resource conflicts are resolved by additional precedence constraints. The algorithm solves all test problems assembled by Patterson (1984) in a very short time. On the other hands, Mingozzi et al. (1998) introduce a new 0-1 linear programming formulation that is used to derive five new lower bounds that are tighter than the bounds proposed by Stinson et al. (1978). Experimentation shows the branch and bound algorithm based on this formulation can solve hard instances that cannot be solved by the Demeulemeester and Herroelen's algorithm (1992).

In the heuristic method category, we find mainly construction methods based on priority rules (Boctor 1990, Kolisch 1996), then local search methods and at last shortened optimal methods often referred as truncated branch-and-bound solution approaches. Construction

methods are based on a scheduling scheme and an activity selection mechanism mainly by priority rules. These methods are most often the core of commercial project scheduling software. They are considered as robust, comprehensive and, most important when they are used within an interactive software program, they require a minimal computation effort.

In the last years, researchers have introduce local search methods to solve the RCPS problem. Local search methods mainly refer to Genetic Algorithm (GA) (Hartman 1998), Simulated Annealing (SA) (Boctor 1993, 1996a, Bouleimen and Lecoq 2003) and Tabu Search (TS) (Barr et al. 1996, Pinson et al. 1994, Tsai and Gemmill 1998). The others are variants of variable neighborhood search.

Local search methods provide, in most cases, better solutions than the previous construction methods as they most often start with an initial feasible schedule generated by one or many construction methods, a feasible schedule of a RCPS problem referring to the one respecting both precedence relations and resource constraints.

Since, the RCPS problem is a generalization of the Job Shop problem and hence it belongs to the class of NP-hard problems (cf. Blazewicz et al. 1983). It may be conceivable to use optimal solution procedures only for small size projects (e.g., up to 50 activities) if it is necessary to find an optimal solution. For larger problems, one needs to use approximate methods to get a good solution within a convenient response time (cf. Lee and Kim 1996, Tsai and Gemmill 1998).

As specialized resources are scare, producing the best project schedule considering variable resource availability over time remains a challenge for many applications. Hereafter, in this paper, we also consider the problem where the resource availability of each resource type may vary over the project horizon, this variant being a generalization of the traditional RCPS problem.

To propose the best schedules for large problems and knowing the efficiency of Local Search method adaptations over construction methods (cf. Glover and Laguna (1997)), the objective of this paper is to present a Local Search procedure based on new computation strategies to provide within a fast computation very good schedules time in a real-time environment. This Local Search procedure is designed to solve RCPS problem having either fixed or variable resource availability.

The paper is organized as follows. Section 2 recalls the definition of the RCPS problem and gives a review of different procedures with results obtained to solve the RCPS problem. Section 3 describes the proposed Local Search procedure. Section 4 presents the performance results obtained with different test problem sets obtained form the open literature. Conclusions and suggestions for further research follow in Section 5.

2. Previous research works

2.1 Problem formulation and notations

The RCPS problem consists of a set A of J activities, a set P defining finish-to-start precedence constraints between the activities and K renewal resource types. Each activity j, $j = 1, ..., J$, consists of d_j periods and may require zero, one or more renewal resources, denoted by q_{jk}, $k = 1, ..., K$. The amount of resources q_{jk} used is the same during the activity duration. Under the non-preemption assumption, activities, once started, cannot be

interrupted. The set of immediate predecessors of an activity j is denoted by the set P_j, $P_j \subseteq P$. Any activity j may start only when all its immediate predecessors in the set P_j has been completed. The project planning horizon is a number of periods, denoted by T, large enough to complete all activities according to the resource availability of each resource type, denoted by Q_{kt}, $k = 1, ..., K$, $t = 1, \ldots, T$. For the traditional RCPS problem, the objective pursued is to minimize the project duration (makespan) without violating neither precedence constraints nor resource constraints.

A feasible solution of the problem is a schedule represented by a J-tuple giving the start time s_j of each project activity j while respecting the precedence constraints and the availability of renewable resources.

2.2 Previous local search methods

Up to now, Genetic Algorithm, Simulated Annealing and Tabu Search have mainly been proposed as local search method to solve the RCPS problem.

Genetic Algorithm (GA)

The ideas involved in GAs originated from Holland (1976). Goldberg (1989) detailed adaptation to optimization problems. These algorithms apply search techniques based on concepts of natural selection and genetic. Apply GAs have mainly been successfully restricted to production scheduling problems. But very good application results are now reported to solve RCPS problems.

Lee and Kim (1996) compare GA algorithms with those of the two others local search methods . The authors propose a coding of the solution by a priority list and they describe the application of operators. The encoding scheme of the priority list is similar to the one described in Boctor (1996b). Activity positions in the list represent the priorities of the corresponding activities. By using a priority scheduling method, the algorithm generates only feasible solutions reducing by this way the needed computation time.

Also, Hartmann (1998) evaluates the performance of three different solution representations within three corresponding GAs. The initial population is obtained by using a regret biased random sampling heuristic incorporating the LFT rule. A topological ordered list of jobs (i.e. activity priority list) is used to represent a solution. Crossover operations are applied on these lists. A population size of 40 job lists is used within the algorithm. The author concludes to the effectiveness of job list representation and the serial scheduling method. Results are obtained on the test problem sets generated with the ProGen problem generator proposed by Kolisch (1995) and they are discussed below.

Simulated Annealing (SA)

SA method has been used successfully for solving both the RCPS problem. It is a kind of iterative improvement algorithm in which the initial solution is repeatedly improved by making changes to the current solution until no more change results in a better solution. SA method attempt to avoid local optimum by accepting a current solution that deteriorates the value of the objective function. The algorithm is able to optimize different objectives functions.

Within the general model sketched for evaluating the three common local search methods, Lee and Kim (1996) use a priority list for the SA algorithm as described above for

the GA algorithm. After adjusting the parameters guiding the search, the SA algorithm succeeds to solve optimally 91 of the 110 test problems assembled by Patterson (1984) with an average deviation of 0,57 from optimal project duration.

To overcome the difficulty of finding all optimal solutions encountered with the previous algorithm, Cho and Kim (1997) propose a new version of the SA method solving the RCPS problem. In their point of view, Lee and Kim's algorithms have problems in that the priority list scheduling generates only non-delay schedules. Contrary to the majority of priority-based scheduling method applied, the search algorithm can delay some activities in order to enlarge the solution space within the active schedule set, the set of non-delay schedules being a subset of the set of active schedules. Solutions found can also be ameliorated by delaying the start time of some activity, because the non-delay schedules is not a dominant set. In fact, the set of non-delay schedules may not contain an optimal solution (Sprecher et al. 1995). Therefore, according to the authors, Lee and Kim's algorithms (1996), in which delay schedules are left out from consideration, always fail to find optimal solutions of problems for which delay schedules are optimal, even though sufficiently long computation time is let to the search algorithms. Cho and Kim's (1997)'s SA algorithm succeeds to solve optimally 103 of the 110 test problems assembled by Patterson (1984) with an average percentage increase of 0,14 from the optimal project duration within 18,4 seconds on a Pentium® processor operating at 60 MHz.

Tsai and Gemmill (1997) propose a SA algorithm for the RCPS problem applicable to a wide range of problems with resource constraints. The proposed algorithm provides good solutions for both the deterministic and the stochastic case where the duration of activities is random (e.g. PERT model). SA algorithm produces solutions better than those obtained by traditional heuristics within a minimal computation time. According to the results reported in Tsai and Gemmill (1998), the experimentation based on the Patterson's test problems obtains up to 84 optimal solutions of the Patterson (1984) 110 test problems in 10 trials. The corresponding average deviation above the optimal project duration is 0,23. The SA algorithm's performance is compared with the performance of a Tabu Search algorithm reported below.

Boctor (1996b) proposes a new adaptation of the SA method for both the RCPS and the Multiple Mode RCPS problems without preemption. Statistical experiments are made on the Patterson's test problem set for the single mode. Results show the efficiency of the proposed SA adaptation in comparison to previous Tabu Search algorithms evaluated by Pinson et al. (1994). As described earlier, the SA adaptation uses an activity list representation in which the position of the activity in the list indicates its priority. To generate a neighbor solution, it chooses at random an activity and changes its position in the list representing the current solution. To assure this new solution is feasible, the selected activity must be moved to a position in the resulting list that neither precedes any of its predecessors nor succeeds any of its successors. The initial solution is generated with a parallel method with MIN SLK evaluated in Boctor (1990) and considered able to generate 33 % among the optimal solutions on the basis of a set of 66 test problems having different degree of complexity. The proposed SA adaptation produces 109 optimal solutions of the 110 Patterson's test problems with an average percentage increase above optimum of 0,015 %. This result corresponds to an average number of 4 579 solutions generated. Experiments are conducted on a SLC2 processor running at 25 MHz.

Tabu Search (TS)

Initially proposed by Hansen (1986) and Glover (1989, 1990), the TS method is a neighborhood search improvement heuristic. The basic method progressively transforms step by step a current solution x into a new and hopefully better solution. To this aim, it examines a selected neighborhood obtained from the current solution x and it tries to find a better solution belonging to this neighborhood. The neighborhood of solution x, noted $N(x)$ is the set of feasible solutions reachable in one step from the current solution x by applying a transformation function, often called a move. The best move y^* of $N(x)$ is then accepted even if this new solution is not better than the current solution x to avoid getting trap in local optima. The adaptation of the method requires the definition of possible neighborhoods. Note that some TS adaptations consider transformation functions that may yield unfeasible solutions and even use a starting unfeasible solution to begin the search process. For a review of the latest developments in this area, we refer the reader to Glover and Laguna (1997).

One of the first adaptations of the TS method for the RCPS problem comes from Pinson et al. (1994). It is based upon the Carlier's strict order algorithm where various solutions evaluated during the search are specified as a list of J activities and the positions of activities in the list respect precedence constraints among activities.

Baar et al. (1997) introduce two TS algorithms for the RCPS problem. The first TS algorithm relies on elimination of critical arcs and priority list scheduling techniques. The second algorithm is based on so-called schedule schemes. A schedule scheme consists of sets of four disjoint relations, which define a set of feasible schedules. These relations specify whether two activities are precedence related, may not be processed in parallel, must be processed in parallel or are not subject to any restrictions. Four types of operator are defined on a schedule scheme. The neighborhood is defined by changes of flexibility relations into parallelism relations and the inverse. To reduce the parallelism neighborhood size, the authors propose nine operators. Experimental results are reported on the three test set problems generated with the problem generator ProGen proposed by Kolisch (1995).

Tsai and Gemmill (1998) propose an adaptation of TS algorithm for the RCPS problem based on the criticality of activities. The proposed TS adaptation uses two Tabu lists, randomized short-term memory, and multiple starting schedules as a means of search diversification. Two heuristic rules, the minimum slack time first (e.g. MIN SLK) and the Composite Allocation Factor (CAF, Badiru (1993)) heuristics are used to provide a feasible starting solution for TS. The procedure is coded with Borland C++ Version 5.0 and executed on a Pentium166 MHz computer. The procedure efficiency is evaluated on the 110 test problems assembled by Patterson (1984). Experimental results are compared with a previous one conducted with Tsai and Gemmill's (1997) SA algorithm. On the basis of 110 test problems (deterministic case), the TS produces solutions with an average deviation above the optimum of only 0,19 and 0,16% with CPU time of 3,396 s and 3,364 s, by using the starting solutions produced by the two heuristic rules P-MIN SLK and CAF, respectively. Approximately 93,46% of optimal solutions where found. The number of optimal solutions found in all 10 trials is 95. The performance of SA and TS algorithms are similar for the deterministic case. For the scholastic case, the performance are also similar both the simulated annealing required two times the computation time.

3. The proposed Local Search procedure

The proposed Local Search procedure may be considered as a non-standard implementation of Tabu Search principles. What characterizes our LS procedure form other TS procedures is mainly the definition of neighborhood at a given stage of search. In the proposed LS procedure, we introduce variable sampling mechanisms.

3.1 Schedule representation

In the proposed LS procedure, a schedule is associated with a sequence x represented by a list of activities, i.e. a J-tuple $(j_{(1)},..., j_{(i)}, ..., j_{(J)})$ where $j_{(i)}$ indicates the activity at position i in the sequence x. We also use the function $p(x, j)$ to obtain the position of activity j in the sequence x. The corresponding schedule is obtained from a sequence x by applying a priority list scheduling procedure where the priority of activity j corresponds to its position in this sequence. One can demonstrate that every sequence obtained by modifying activity positions results in a feasible solution if the precedence relations are respected. So, in our LS procedure, in order to generate a feasible sequence, a selected activity must be moved to a new position that neither precedes any of its predecessors nor succeeds any of its successors. For a given sequence x, this is achieved by verifying the maximal position of all predecessors of activity j, obtained by the function $L(x, j)$, and the minimal position of all successors of j, obtained by the function $H(x, j)$.

3.2 Sampling the neighborhood

For the RCPS problem, we devise a transformation function $M(x, y)$ called a move in Tabu Search procedures. This function $M(x, y)$ is called *Insert* and it is similar to the *Shift* in Job Shop scheduling. The function $M(x, y)$ constructs a candidate sequence y by randomly selecting an activity j in the sequence x and moving it from its actual position $p(x, j)$ to a new position $p(y, j)$ selected randomly in the interval $(L(x, j), H(x, j))$. Depending if the new position $p(y, j)$ obtained is at the left or at the right of $p(x, j)$, the positions of activities between $p(x, j)$ and $p(y, j)$ are shifted to the left or the right, respectively, in the candidate sequence y. The other activities keep their position so they don't need to be moved. Also, every new sequence y obtained by applying the function $M(x, y)$ is necessarily different from its seed sequence x *i.e.*, $p(x, j) \neq p(y, j)$.

In evaluating the neighborhood of a current solution x, one can systematically evaluate all the solutions belonging to $N(x)$ at a given stage in order to try improve the current solution x (cf. Pinson et al. 1994). But, this process requires a huge amount of computation time. For this reason, the strategy adopted here restricts the search to a sample of neighborhood, designated by $N^*(x)$, where $N^*(x) \subset N(x)$. This sample must be large enough to get a better solution with a fair probability at the next stage. Here, the size of $N^*(x)$ is set equal to the square root of number of activities J. As one notes, a similar computation has been experimented by Tsai and Gemmill (1998).

As activities are selected randomly within the sequence x by the transformation function $M(x, y)$ implemented, we also introduce a check to eliminate doubles in order to don't evaluate a new sequence more than once at a given stage. By this check, one generates neighbor samples of variable size reducing then at the end the total number of sequences evaluated.

The transformation function $M(x, y)$ also verifies if an activity j' selected randomly has the same starting times in the sequence x. Then, this activity is not selected as part of the sample if all activities between $p(x, j)$ and $p(y, j')$ have also the same starting time.

3.3 Procedure description

Figure 1 gives the general structure of the LS procedure implemented. The corresponding notation is given in Table 1. Line 2 obtains a starting solution generated by a construction method and sets different parameter values among which the size of the sample $N^*(x)$ and the size of the Tabu list TL corresponding to the tabu tenure. The core of the process begins at line 4 and it continues up to the line 28. The worst completion time of the best solution y^* in the neighbourhood of x is initially set to the maximum numerical value at line 5 in order to trap the fist neighbour evaluated as the best neighbour. A sample of neighbours is then generated at line 6 along the selected transformation function. The evaluation process of each neighbour $y \in N^*(x)$ is performed at line 8. Line 9 updates the sequence number. Line 10 verifies if the current neighbour evaluated is better than the current best neighbour solution y^*. If so, line 11 verifies if this solution is equal to the CP lower bound. If yes, the procedure returns the best solution y find. Else, it is kept at line 12 as the best neighbour found. Line 14 verifies if $nMax$ is the stopping criterion. If yes, then line 15 verifies if the maximal value $nMax$ is reaches. If yes, then line 15 returns the best solution between x^* and y^*. After the evaluation of the sample $N^*(x)$ is completed, attributes of the best solution y^* are recorded in the Tabu list TL at line 18. And, the new current solution x is updated at line 19. Line 20 verifies if a new best solution has been found. If yes, it is kept at line 21 by replacing the best solution x^* found so far by the best solution y^*. And the number of iterations k performed by the procedure without improvement of the best solution x^* is reset to zero at line 22. Else, it is incremented by one at line 24. Line 25 verifies if the stopping criterion is $kMax$. If yes, the procedure returns the best solution x^* at line 26 when this stopping criterion reaches the maximal value allowed $KMax$.

Table 1 - LS procedure notation

x	Curent solution
y	Solution being evaluated
x^*	Best solution found so far
$N^*(x)$	Sample of solution constituting the neighbourhood of the current solution x
y^*	Best solution giving the best objective function value in the neighbourhood of the current solution x
f_{x^*}	Project completion time of the best solution found x^*
f_y	Project completion time of the solution y
f_{y^*}	Project completion time of the best solution y^* find in neighbourhood of $N^*(x)$
$M(x, y)$	Transformation function used to generate a feasible solution in the neighbour-hood of x
TL	Tabu List
$A(x, y)$	Information about the current solution and the new solution y generated from x
k	Number of iterations performed without finding a better solution than x^*
$kMax$	Maximum number of iterations without finding a better solution than x^*
n	Number of sequences evaluated
$nMax$	Maximum number of sequences evaluated allowed

```
1.  begin
2.      Generate the starting solution x and set Local Search  parameters
3.      x* := x, TL := ∅, k := 0, kMax, nMax
4.      do
5.          f_y* := ∞
6.          Generate N*(x) ⊂ N(x) according to the transformation function M(x, y)
7.          for each y ∈ N*(x) do
8.                  Evaluate f_y
9.                  n = n +1
10.                 if f_y < f_y* then
11.                         if f_y = CP lower bound then return the best solution y
12.                         y* := y, f_y* := f_y
13.                 end if
14.                  if the stopping criterion is nMax then
15.                         if n = nMax then return the best solution x* or y*
16.                 end if
17.         next
18.         Insert A(x, y*) in TL
19.         x := y*, f_x := f_y*
20.         if f_y* ≤ f_x* then
21.                 x* := y*, f_x* := f_y*
22.                 k := 0
23.         else
24.                 k := k + 1
25.                 if the stopping criterion is kMax then
26.                         if k = kMax then return the best solution x*
27.         end if
28.     loop
29. end
```

Figure 1 – Local Search procedure

The following section describes some features about the implementation of this procedure.

3.4 Tabu list management

Tabu Search method exploits the knowledge gained from previous solutions evaluated. This knowledge is maintained in a Tabu list used as a memory in order to avoid cycling, i.e. to repeat recent transformations applied to obtain the solutions evaluated. The literature distinguishes between the short-term memory and the long-term memory. In the present LS procedure, to save memory at the expense of greater computation time dedicated to the search, we devised only a circular list management keeping only the recency Tabu tenure. The recency Tabu tenure is recorded by keeping at each entry of a table, noted *TL*, attributes on the last sequences visited. Here, these attributes correspond to the activity number and position number occupied by the activity that is becoming Tabu when the new best sequence y^* replace the current sequence x. The size of the list denotes the time a visited solution is maintained tabu. It is set equal to the square root of number of activities J.

The transformation function $M(x, y)$ accepts to select an activity j only if this activity is not in the Tabu list *TL*.

3.5 Initial solution used

There is some dilemma concerning the choice of the initial solution used by local search methods. Should the initial solution be just good or very good? As we observed, starting with a very good solution don't let enough space to find a significant improvement. At the opposite, it may sometimes take a long computation time to improve a very bad solution. Therefore, the initial solution must be a good and feasible solution. The literature suggests the use of construction methods in order to generate rapidly a good and feasible initial solution. We can characterize construction methods according the logic used to build up a schedule. One distinguishes mainly two scheduling scheme, the so-called serial scheme and parallel scheme. Extensive researches have been conducted in order to determine which scheme and which priority rule could possibly produce the better solution (cf. Boctor 1990, Kolisch 1996). The parallel and the serial schemes provide the basis to parallel and serial methods, respectively. These methods are applied to find an initial feasible schedule on the basis of priority rule used. In this paper, we consider four construction methods depending on the scheduling scheme and the priority rule used to generate the initial solution. These methods are defined in Table 2.

Table 2 – Scheduling scheme and priority rule used of construction methods

Scheduling scheme	Priority rule	Priority name	Value selected	Priority Value
Parallel (P)	Minimum SLacK (SLK) time	P-MIN SLK	MIN	LST(j)-t
Parallel (P)	Latest Finish Time (LFT)	P-MIN LFT	MIN	LFT(j)-t
Serial (S)	Latest Finish Time (LFT)	S-MIN LFT	MIN	LFT(j)
Serial (S)	Latest Start Time (LST)	S-MIN LST	MIN	LST(j)

The P-MIN LST heuristic, as one knows, is equivalent to the P-MIN SLK heuristic. Preliminary experimentation on the Patterson's problem set shows below the P-MIN SLK parallel method obtains a greater number of optimal solutions than the S-MIN LFT serial method. However, methods based on parallel scheme produce schedules that belong to the non-delay schedule set. As shown by Kolisch (1996), this set may not contain an optimal solution. This is not a dilemma since the initial solution obtained by the parallel scheme will be modified by a priority list scheduling method that proceeds under the serial scheme. So, we evaluate either the parallel methods or the serial methods to get the initial solution and we report these results.

3.6 Optimization strategies

Many strategies have been devised in order to produce a fast and good solution. This is why the list representation described in Section 3.1 has been retained as a basis in order to take full advantage of these optimization strategies. Some of them have been mentioned in Section 3.2.

A simple strategy is to stop the search as soon as we know we prove optimality of the objective function. This is achieved by verifying bounds on the project duration. Most of the bounding techniques devised so far for the RCPS problem don't apply to the case of variable resource availability. For the RCPS problem, the critical path (CP) lower bound remains a computation time-feasible value and is used to stop the search process. Consequently, in

these situations, we can inform the project manager of the optimality of the solution found. This check also takes care of cases where the initial solution generated by the construction method yields an optimal project duration.

Different strategies aims to minimize the computation time by shortening the computation time required to evaluate the other neighbors of the sample. During each iteration of the LS procedure, a neighbor is considered as the current best neighbor y^*. The evaluation of the next neighbors generated from the sample $N^*(x)$ proceeds only if the evaluation of the project completion time of sequence y being evaluated remains less than the project completion time of the best neighbor y^* obtained so far. A first check is made when the finish time of an activity becomes greater than the latest finish time of the best neighbor y^*. This is achieved by adding the ACTIM (Activity Control Time) value to the finish time of each activity j of the solution y being evaluated and by comparing the duration obtained to the best finish time f_{y*} obtained so far. This verification is efficient since most of computation time is used checking the resource availability at every period.

Another strategy is introduced in order to substantially reduce the computation time required to search and verify the resource availability. Thus, the starting time of those activities preceding the position where an activity is inserted neither don't need to be calculated nor the resource availability needs to be re-verified.

Another strategy verifies the resource availability of a given activity at the period starting to the nearest finish time and then proceeding backward. If the required resources are unavailable, then the position and length where resources are available are preserved. The verification is restarted at the new projected finish time and again proceeding backward up to the preserved position. The procedure is aborted as soon as the computed finish time becomes greater than the latest finish time of the best neighbor y^*.

At last, attention has also been given to reduce memory space requirement. At any time, the algorithm needs only to store four solutions in memory. These are the best solution find so far, known as x^*, the current solution x, the solution y being evaluated and the best solution y^* found so far in the neighborhood of x.

3.7 Stopping criteria

Besides the verification of lower bound equality, two stopping criteria were considered in order to get the best solution: the maximal number of sequences evaluated, noted *nMax,* and the number of iterations performed without improvement of the best solution x^*, noted *kMax*. As the computation time increases with the problem size, the first criterion *nMax* gives an idea of the maximal computation time required before the procedure stops. The second criterion lets a chance to find a better solution but the required computation time is somewhat unpredictable. To overcome this uncertainty, a third criterion could be used as a combination of the two previous criteria. So, the search could then be stopped when one of the two criteria becomes true.

3.8 Intensification

Intensification strategies are mechanisms applied during or at the end of the search process in order to get easily a better final solution. Among others, we experimented the pair-wise interchange but this mechanism doesn't improve significantly the solution quality when

considering the computation time required. Since the quality of the final solution also obtained seems to leave little margin for a significant improvement.

3.9 Diversification

Diversification strategies aim at directing the search into new solution space areas not yet explored and then to restart or continue the search there. We implemented the following mechanism. As the minimal width of the interval required to insert a selected activity at a new position is three, the transformation function increases the required interval by one at each stage there is no improvement of the best solution find so far. This required interval may then increase at each iteration up to a maximum interval width. This last value is again equal to the square root of number of activities *J*. As soon as we obtain a new best solution, the required interval is reset to its minimal width.

4. Performance evaluation

In this section we present some computational results for the algorithm developed in the previous sections to solve the RCPS Problem. In Section 4.1, we describe the tested problem instances. Section 4.2 gives the computational results. The results presented in this section were obtained using a SUN/Sparc 20/801 workstation with operating system Solaris 2.5 and 64 MB general storage (80 MHz +1MB SC).

4.1 Test problems used

In order to evaluate and to compare the performance of the proposed LS procedure, different RCPS test problem sets available from the literature are used. These are the Patterson's (1984) test problem set and three sets from Kolisch (1995, 1998).

4.1.1 Patterson's test problems

We use the well-known Patterson's (1984) test problem set for which we know the optimal project duration of all problems. This test problem set consists of 110 instances. The number of activities in these test problems varies between 7 and 51. Table 3 gives the repartition of the number of activities by project instances. Among them, 103 project instances require three resource types, four project instances require two resource types and three require one resource type only.

Table 3 – The 110 Patterson's test problem repartition by activity number group

Total activities	Project instances
< 22	8
22	46
27	43
23,35,35	3
51	10
Total	**110**

4.1.2 ProGen test problems

Kolisch et al. (1995, 1998) introduce new benchmarks that many researchers have used since as a new standard. These instances are available in the project scheduling library PSPLIB from University of Kiel.

The first two sets J30 and J60 used consist of 480 instances each with four renewable resource types. The three usual parameters, network complexity (NC), resource factor (RF) and resource strength (RS) are combined together to define a full factorial experimental design. The NC defines the average number of precedence relations per activity. The RF factor gives the average percent of different resource type demand by activities. The RS measures the scarcity of resources. A RS value of zero corresponds to the minimum needs of each resource type to realize all activities while a RS value of one corresponds to the required amount of each resource type obtained from the early start time schedule. The parameter values used to built up these problem instances are: NC ∈ {1,5; 1,8; 2,1}, RF ∈ {0,25; 0,5; 0,75; 1,0} and RF ∈ {0,1; 0,2; 0,4; 0,5}. The third set J120 has 600 instances, each with four renewable resource types. Again, a full factorial design with the three problem parameters is set. The parameter values used are the same except for RF ∈ {0,1; 0,2; 0,3; 0,4; 0,5}.

We find in Hartmann and Kolisch (2000) extensive results of an experimental evaluation of competitive heuristics proposed for the RCPS problem with the three problem sets, namely J30, J60 and J120 sets. Results presented in this paper are also compared with the current results obtained with the LS procedure.

4.2 Performance criteria

For the RCPS problem, we use the average percentage deviation of project duration over the optimal project duration for the Patterson's and ProGen's J30 problem sets. For the two other activity sets J60 and J120, we give the average percentage deviation from the CP lower bound. We give the number of times the LS procedure finds the optimal solution for the set J30 and the number of times the solution obtained equals the CP lower bound. The standard-deviation is also provided.

We also report the average number of evaluated sequences and the average CPU time. One must note that average CPU time reported corresponds merely to a response time as Windows is a multi-task operating system.

The LS procedure was coded using Microsoft Visual Basic 6.0© on a Toshiba Tecra 8100 with an Intel Pentium 500 MHz processor running under the Windows 200 operating system.

Depending on the experiment, one of the two stopping criteria are applied: (1) stop the search when the number of evaluated sequences reaches a given value *nMax*, or (2) stop the search if there is no improvement of the best solution found after *nMax* iterations by the LS procedure.

Experimentation is performed with unique parameter settings for all instances. All experiments are also conducted by using the same random number sequence provided by the Visual Basic random number function.

4.3 Results and analysis

4.3.1 Patterson's problem results

Table 4 first presents the results obtain with each construction method and priority rule on the Patterson's problem set. As one notes, the parallel scheme requires more computation time. But, the construction method P-MIN SLK dominates all others and offers

the greater number of optimal solutions found on this particular problem set. On the other hand, the construction method S-MIN LFT produces the higher average deviation.

Table 4 - Performance of construction method on Patterson's problems

Construction method with the priority rule used	Average deviation (%)	Standard deviation (%)	Maximum deviation (%)	Number of optimal solutions	Average CPU time (s)
P-MIN LFT	5,6842	5,2963	21,9512	30	0,00225
P-MIN SLK	4,9996	5,6813	23,6842	34	0,00202
S-MIN LFT	6,8239	5,8652	25,0000	28	0,00096
S-MIN LST	5,0915	4,9673	22,2222	33	0,00080

Table 5 presents the results of Patterson's problem set obtained with the LS procedure and the construction method selected to generate the initial solution. The main performance criteria are the average deviation of project duration over the optimal project duration, the number of times our procedure gives the optimal project duration and the average CPU time. The stopping criterion is set to 100 iterations without improvement of the best solution found.

Table 5 - Performance on Patterson's problems set with *kMax* parameter sets to 100

Construction method with the priority rule used	Number of optimal solutions	Number of solutions better than the initial solution	Average deviation (%)	Average number of sequences evaluated	Average CPU time (s)
P- MIN LFT	89	76	0,6087	468	0,033
P-MIN SLK	92	70	0,4744	463	0,035
S- MIN LFT	92	79	0,4948	483	0,032
S-MIN LST	92	71	0,4686	454	0,036

When the initial solution is generated by the construction method P-MIN SLK and the maximum number of solutions without improvement is set to 100, the average percentage deviation in project duration is 0,4744 % with an average CPU time of 0,035 seconds. In this case, the average number of sequence evaluated is 463.

Table 6 shows the number of iterations required to obtain all Paterson's optimal solutions. As example, with the construction method P-MIN SLK, if we increase the maximum number of iterations without improvement of the best solution up to 1 200 we reduce the average percentage deviation over the optimal duration to zero and we get the optimal solution of the 110 test problems. And, the required average percentage CPU time increases with an average of 0,49044 seconds by problem. It is worthwhile to know the last optimal solution is obtained after 3 453 sequences have been evaluated.

Table 6 – Optimal solutions found on Patterson's problems set varying *kMax* parameter

Construction method with the priority rule used	Number of iterations without improvement (*kMax*)	Average deviation (%)	Standard deviation (%)	Average CPU time (s)	Number of optimal solutions	Number of solutions better than the initial solution	Average number of sequences evaluated
P-MIN SLK	900	0,0404	0,3222	0,33562	108	75	3636
P-MIN SLK	1000	0,0503	0,3142	0,36942	107	76	4010
P-MIN SLK	1100	0,0382	0,2890	0,40091	108	76	4353
P-MIN SLK	1200	0	0	0,49044	110	76	4842
P- MIN LFT	900	0,0275	0,2889	0,36809	109	80	3633
P- MIN LFT	1000	0	0	0,41351	110	82	4036
S- MIN LFT	900	0,0313	0,3288	0,35043	109	82	3776
S- MIN LFT	1000	0	0	0,37278	110	82	4035
S-MIN LST	900	0,1639	0,7265	0,33231	104	76	3619
S-MIN LST	1000	0,0663	0,4022	0,37687	107	77	4052
S-MIN LST	1100	0,0120	0,1255	0,40113	109	76	4352
S-MIN LST	1200	0	0	0,43892	110	77	4764

Table 7 - Comparison of LS procedure with Boctor's (1996b) and Tsai & Gemmill's (1998) results

Boctor's (1996b) Simulated Annealing

Value of RMAX parameter	8	16	24	32	40	48	56	64
Average sequences evaluated	576	1 149	1 722	2 293	2 866	3 443	4 654	4 579
Average deviation (%)	0,669	0,307	0,196	0,135	0,075	0,107	0,024	0,015
Number of optimal solutions	87	97	102	103	106	105	108	109

Tsai and Gemmill's (1998) Tabu Search

Number of iterations without improvement	100	300	600	1 000	2 000
Average deviation (%)	1,40	0,69	0,40	0,29	0,19
Number of optimal solutions	43	62	76	87	95

Local Search procedure with the initial solution provided by the construction method P-MIN SLK

Number of iterations without improvement	100	300	600	1 000	1 100
Number of solutions found better than the initial solution	66	71	76	75	76
Average number of sequences evaluated	451	1 260	2 537	4 132	4 441
Average CPU time (s)	0,038	0,099	0,193	0,311	0,335
Number of sequences evaluated when the last optimal solution is found	529	1 711	3 976	3 780	3 453
Average deviation (%)	0,5584	0,1982	0,0624	0,0595	0,000
Number of optimal solutions	89	102	107	107	110

Table 8 compares the results obtained with the Patterson's 110 problem set with others results reported in the literature. It shows all optimal solutions are obtained. As authors propose different versions of heuristic, only the best results are reported. The average CPU time is reported but one has to note it is obtained on different processors. Sampson and Weiss (1993) report, an average of 1,98 % away from the optimal solution and 61 problems were solved to optimality. Leon and Ramamoorthy (1995) propose two GAs. The first one, GA1, uses a total of 500 solutions and solves to optimality 71 problems with an average deviation of 1,2500 % and with an average computation time of 1,97 s on an IBM RS 6000 computer. The second one, GA2, uses a total of 2 500 solutions and solves to optimality 83 problems with an average deviation of 0,7433 % and an average computation time of 7,46 s.

Table 8 - Comparison of heuristics on Patterson's problems

Author	Algorithm	Average deviation (%)	Standard deviation (%)	Average CPU time (s)	Number of optimal solutions
Gagnon et al. (2003)	Local Search	0,00	0,00	0,335[a]	110
Hartmann (1998)	Genetic Algorithm	0,00	0,00	5,0[b]	110
Boctor's (1996)	Simulated Annealing	0,14	0	?	109
Cho and Kim (1997)	Simulated Annealing	0,14	0,50	18,4[c]	103
Tsai and Gemmill (1998)	Tabu Search	0,19	?	3,396[d]	95 in 10 trials
Lee and Kim (1996)	Simulated Annealing – SA2	0,57	1,60	17,0[c]	91
Gemmill and Tsai (1997)	Simulated Annealing	0,229	101 in average	3,65[d]	84 in 10 trials
Leon and Ramamoorthy (1995)	Genetic Algorithm – GA2	0,7433	0,1357	7,46[e]	83
Pinson et al. (1994)	Tabu Search – Tabu 1-5	?	?	1,74[f]	83
Sampson and Weiss (1993)	Local Search	1,98	2,62	56[c]	61
Bell and Han (1990)	A* algorithm	2,60	3,1	28,4[g]	49

a Average CPU time range on an IBM ThinkPad 385-XD with a Pentium 233 MHz and 96 MB RAM
b Maximal CPU time on a Pentium 133 MHz with 32 MB RAM
c Average CPU time on a Pentium 80486 at 60 MHz
d Average CPU time on a Pentium 166 MHz
e Average CPU time on an IBM RS 6000
f Average CPU time on a Pentium 80486-DX2 at 66 MHz
g Average CPU time on a MacIntosh Plus

4.3.2 ProGen test problem results

We now evaluate the performance of the LS procedure with the three problem benchmarks obtained from the ProGen problem generator.

First, Table 9 presents the performance obtained with each construction method and priority rule for the three different test problem sets J30, J60 and J120, respectively. For the test problem set J30, we know the optimal solutions. For the two others, the number of optimal solutions is the number of times the project duration obtained is equal to the CP lower bound.

As one notes, the construction method P-MIN LFT has the best average on the first and third problem set. On the other hand, the heuristic S-MIN LST offers the greater number of optimal solutions found. On this basis, we report, in the next three tables, the experimental results obtained with the three different instance sets generated from the test problem generator ProGen.

Table 9 - Performance of construction methods on ProGen problem sets

Construction method with the priority rule used	Average deviation (%)	Standard deviation (%)	Maximum deviation (%)	Number of optimal solutions	Average CPU time (s)
		J30			
P-MIN LFT	4,3931	5,2963	24,1379	206	0,00173
P-MIN SLK	4,5283	5,9535	32,8125	213	0,00126
S-MIN LFT	5,5818	7,4614	33,8983	240	0,00086
S-MIN LST	4,9213	6,8288	33,3333	246	0,00097
		J60			
P-MIN LFT	17,461	28,792	133,766	178	0,00204
P-MIN SLK	17,121	28,655	136,364	186	0,00353
S-MIN LFT	18,131	32,536	145,455	253	0,00102
S-MIN LST	17,454	31,686	144,156	265	0,00157
		J120			
P-MIN LFT	43,8638	50,6021	243,4343	18	0,01336
P-MIN SLK	44,0395	50,9426	229,7000	20	0,01354
S-MIN LFT	48,1092	56,1976	253,5300	95	0,00317
S-MIN LST	46,7403	55,3639	255,5556	101	0,00248

Table 10 - Experimental results with ProGen problems J30

Construction method with the priority rule used	Number of sequences	Average deviation (%)	Average CPU time (s)	Number of optimal solutions	Number of solutions better	Average number of sequences evaluated
P-MIN LFT	1000	0,60	0,02188	378	265	566
P-MIN LFT	5000	0,27	0,10602	421	272	2 791
P-MIN SLK	1000	0,62	0,02186	379	258	563
P-MIN SLK	5000	0,25	0,10644	424	262	2 786
S-MIN LFT	1000	0,62	0,02179	375	233	562
S-MIN LFT	5000	0,19	0,10649	435	239	2 762
S-MIN LST	1000	0,61	0,02166	382	230	559
S-MIN LST	5000	0,19	0,10489	433	232	2 761

Table 11 - Experimental results with ProGen problems J60

Construction method with the priority rule used	Number of sequences	Average deviation (%)	Average CPU time (s)	Number of optimal solutions	Number of solutions better	Average number of sequences evaluated
P-MIN LFT	1000	12,99	0,03096	274	288	471
P-MIN LFT	5000	12,15	0,14007	285	290	2117
P-MIN SLK	1000	13,04	0,02999	276	280	452
P-MIN SLK	5000	12,11	0,13913	288	284	2086
S-MIN LFT	1000	13,22	0,02847	285	202	419
S-MIN LFT	5000	12,27	0,13486	289	204	2014
S-MIN LST	1000	13,03	0,02745	290	194	401
S-MIN LST	5000	12,23	0,13278	289	197	1996

Table 12 - Experimental results with ProGen problems J120

Construction method with the priority rule used	Number of sequences	Average deviation (%)	Average CPU time (s)	Number of optimal solutions	Number of solutions better	Average number of sequences evaluated
P-MIN LFT	1000	39,39	0,12188	71	550	916
P-MIN LFT	5000	37,34	0,56270	112	571	4233
P-MIN SLK	1000	39,53	0,12214	80	556	903
P-MIN SLK	5000	37,34	0,55927	122	573	4196
S-MIN LFT	1000	39,83	0,11511	134	496	795
S-MIN LFT	5000	37,78	0,54940	139	498	3896
S-MIN LST	1000	39,26	0,11262	134	486	784
S-MIN LST	5000	37,55	0,54115	143	488	3857

For the J30 set, the LS procedure with the serial scheme dominates the parallel one. As one can observe, the results obtained by the LS procedure with the parallel scheme and the MIN LFT rule dominates the serial scheme for the J60 and J120 sets.

Tables 13 to 15 then compare the LS performance with previous results as reported in Hartmann and Kolisch (2000). These results are updated with the new results published in Alcaraz and Maroto (2001) where the authors propose a new GA for the RCPS Problem. The scrutiny of these results clearly shows the very good performance of our LS procedure, which is constantly ranked among the best performing procedures.

Table 13 - Average deviation from optimal solutions for ProGen problems J30

Authors	Algorithm	1 000 sequences	5 000 sequences
Alcaraz and Maroto (2001)	Genetic Algorithm	0,33	0,12
Gagnon et al. (2003)	Local Search	0,61	0,19
Bouleimen and Lecocq (2003)	Simulated Annealing	0,38	0,23
Hartmann (1998)	Genetic Algorithm	0,54	0,25
Baar et al. (1997)	Tabu Search	0,86	0,44
Leon and Ramamoorthy (1995)	Genetic Algorithm	2,08	1,59

Table 14 - Average deviation from the CP lower bound for ProGen problems J60

Authors	Algorithm	1 000 sequences	5 000 sequences
Alcaraz and Maroto (2001)	Genetic Algorithm	12,57	11,86
Hartmann (1998)	Genetic Algorithm	12,68	11,89
Bouleimen and Lecocq (2003)	Simulated Annealing	12,75	11,90
Gagnon et al. (2003)	Local Search	13,04	12,11
Baar et al. (1997)	Tabu Search	13,80	13,48
Leon and Ramamoorthy (1995)	Genetic Algorithm	14,33	13,49

Table 15 - Average deviation from the CP lower bound for ProGen problems J120

Authors	Algorithm	1 000 sequences	5 000 sequences
Alcaraz and Maroto (2001)	Genetic Algorithm	39,36	36,57
Hartmann (1998)	Genetic Algorithm	39,37	36,74
Gagnon et al. (2003)	Local Search	39,53	37,34
Bouleimen and Lecocq (2003)	Simulated Annealing	42,81	37,68
Baar et al. (1997)	Tabu Search		
Leon and Ramamoorthy (1995)	Genetic Algorithm	42,91	40,69
Kolisch (1996)	Sampling LFT s	42,84	41,84

Hartmann and Kolisch (2000) observed the influence of the scheduling scheme depending and problem parameter on procedure performance. Here, we observe a strong influence of the serial scheme with the MIN LFT priority rule on the J30 set but a lead of the parallel scheme with the MIN SLK priority rule on the J60 and J120 sets.

The approaches based on activity list representation such as Harman (1998) and Bouleimen and Lecocq (2003) as the LS procedure give consistently very good results.

By changing the seed of the random number and using the best solution of a multi-pas, the LS Procedure obtains results just below 37,00 for the J120 set. The strength of the LS procedure comes from the minimal CPU time required.

5. Conclusion

In this paper we propose a Local Search procedure to solve the RCPS problem with variable resource availability over the project horizon and with the project duration as objective function. To this hand, we evaluate the efficiency of different transformation functions used to generate a sample of the neighbors of the current solution.

Based on the number of schedules evaluated, the heuristics that perform best for solving the RCPS problem are the SA of Bouleimen and Lecocq (2003), the GA of Hartmann (1998) and the proposed LS procedure. Bouleimen and Lecocq's SA performs best on the ProGen's J30 problem set. Hartmann's GA perform better on the J60 ProGen's J30 problem set. But, on the larger J120 problem set, it is still the Hartmann's GA followed by the actual LS procedure and then the Bouleimen and Lecocq's SA.

As the proposed Local Search procedure should be used in an interactive real-time environment together with commercial software, design choices were made and special attention was given to reduce the memory and computation time required. Therefore, by only sampling the neighborhood at each step and limiting the number of concurrent solutions in memory, the proposed procedure reduces memory space and computation time to its minimum and its efficiency is still comparable to experimentation results published in the literature.

More importantly, the proposed Local Search procedure reaches close-to-optimal solutions consistently in every experiment. At last, the proposed approach is easily adaptable to different objective functions.

6. References

Alcaraz, J. Maroto, C., 2001, A robust genetic algorithm for resource allocation in project scheduling, *Annals of Operations Research*, 102, 83–109.

Baar, T., Brucker, P. and Knust, S., 1997, Tabu search algorithms for resource-constrained project scheduling problems, in: S. Voss, S. Martello, I. Osman, C. Roucairol (Eds.), *Metaheuristics: Advances and Trends in Local Search Paradigms for Optimisation*, Kluwer, pp. 1–18.

Boctor, F. F., 1993, Heuristics for Scheduling Projects with Resource Restrictions and Several Resource-Duration Modes, *International Journal of Production Research*, 31(11): 2547-2558.

Boctor, F. F., 1996a, A New and Efficient Heuristic for Scheduling Projects with Resource Restrictions and Multiple Execution Modes, *European Journal of Operational Research*, 90: 349-361.

Boctor, F. F., 1996b, Resource-Constrained Project Scheduling by Simulated Annealing, *International Journal of Production Research*, 34(8): 2335-2351.

Bouleimen, K., Lecocq, H., 2003, A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version, *European Journal of Operational Research*, Article in press.

Fleszar, K., Hindi, K. S., 2003, Solving the resource-constrained project scheduling problem by a variable neighbourhood search, *European Journal of Operational Research*, Article in press.

Glover, F., 1989, Tabu Search - Part I, *ORSA Journal of Computing*, 1: 190-206.

Glover, F., 1990, Tabu Search - Part II, *ORSA Journal of Computing*, 2: 4-32.

Glover, F. and Laguna, M., 1997, *Tabu Search*, Kluwer Academic Publishers.

Hansen, P., 1986, The Steepest Ascent Mildest Descent Heuristic for Combinatorial Programming, *Numerical methods in combinatorial programming conference*, Capri, Italy.

Hartmann, S., 1998, A Competitive Genetic Algorithm for Resource-Constrained Project Scheduling, *Naval Research Logistics*, 45, 733-750.

Hartmann, S., Kolisch, R., 2000, Experimental evaluation of the state-of-the-art heuristics for the resource-constrained project scheduling problem, *European Journal of Operational Research*, 127, 394-407.

Herroelen, W., Demeulemeester, E., De Reyck, B., 1998, Resource-constrained project scheduling - A survey of recent developments, *Computers and Operations Research*, 25(4), 279–302.

Kolisch, R., 1996, Efficient priority rule for the resource-constrained project scheduling problem, *Journal of Operational Management*, 14,179–192.

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

Kolisch, R., Sprecher, A., 1996, PSPLIB – A Project Scheduling Problem Library, *European Journal of Operational Research*, 96, 205-216.

Kolish, R., Sprecher, A., Drexl, A., 1995. Characterization and generation of a general class of resource-constrained project scheduling problems, *Management Science*, 41 (10), 1693–1703.

Kolisch, R., Padman, R., 2001, An integrated survey of project deterministic scheduling, OMEGA *International Journal of Management Science* 29 (3), 249–272.

Lee, J.-K., Kim, Y.-D., 1996, Search Heuristics for Resource-Constrained Project Scheduling, *Journal of Operational Research Society*, 47, 678-689.

Patterson, J. H., 1984, A Comparison of Exact Approaches for Solving the Multiple Constrained Resource, Project Scheduling Problem, *Management Science*, 30(7): 854-867.

Pinson, E., Prins, C. and Rullier, F. 1994, Using Tabu Search for Solving the Resource-Constrained Project Scheduling Problem, Proceedings of the 4th international workshop on project management and scheduling, Leuven, Belgium, pp-102-106.

Sprecher, A. Kolisch, R., Drexl, A., 1995, Semi-active, active and non-delay schedules for the resource-constrained project scheduling problem, *European Journal of Operational Research,* 80, 94–102.

Tsai, Y. and Gemmil, D. D., 1997, Using a Simulated Annealing Algorithm to Schedule Activities of Resource-Constrained Projects, *Project Mnagement*.

Tsai, Y. and Gemmil, D. D., 1998, A Tabu Search to Schedule Activities of Stochastic Resource-Constrained Projects, *European Journal of Operational Research*, 111: 129-141.