NATIONAL UNIVERSITY OF SINGAPORE

New Definitions and Algorithms in Scheduling Resource-Constrained Projects

by

Fei Xiao

A Thesis Submitted
in Partial Fulfillment of the
Requirements for the Degree

DOCTOR OF PHILOSOPHY

Singapore

October, 2007

Acknowledgments

I am very thankful to my supervisors Professor Andrew Lim and Professor Tan Sun Teck who inspired me to do research in project scheduling and guided me throughout my study.

I would also like to thank Professor Brian Rodrigues for giving me invaluable suggestions to my research and helping me improve my writing skills. I always enjoy the discussions with him, which inspires me a lot.

I am very grateful to Professor Lau Hoong Chuin and Professor Ang Chuan Heng. As my thesis committee, they encouraged me to further improve my research. Without their encouragement, I could not discover the general model of Molecule Search.

Also, I would like to thank Professor Martin Henz who is willing to be my thesis committee after one of my thesis committee left for other university.

I wish to thank Professor Rainer Kolisch, who is so nice to be willing to be my external committee.

I am also indebted to Professor Ee-Chien Chang and Professor Wei Tsang Ooi who gave me invaluable suggestions in improving my writing skills.

Finally, I would like to thank my father, who always encourages me to achieve better results in my research.

Contents

Abstract

New Definitions and Algorithms in Scheduling Resource-Constrained Projects

by

Fei Xiao

In the first part of the thesis, we study the problem to interpret float and identify critical activity for projects with resource limits. The use of float and critical path are central in analyzing activity networks in project management. However, the variability in the schedules for resource-constrained projects make it difficult to calculate float and identify critical activities accurately. In this thesis, a new definition for float is proposed for projects with resource limits. With the new definition, it is possible for us to calculate float and identify critical activity without referring to a specified schedule. To measure the flexibility for more than one activity, group float is defined as the float for a set of activities. The critical set is presented as the activity set with 0 maximum group float, and float set is given as the activity set with larger than 0 maximum group float respectively. As a symmetrical complement for float, negative float and negative critical activity are also proposed.

Several algorithms are developed to calculate the maximum float. Extensive experiments are conducted to show that there are abundance of activities and activity sets with positive float and group float even the deadline of the project is already optimal. We also show that the maximum float for large size projects can be calculated approximately.

We also proposed the notion of a float graph and negative float graph to illustrate float, critical activities, float sets and critical sets, negative float, negative critical activities, negative float sets and negative critical sets. These can help project managers understand the intrinsic of flexibility between the activities for the resource-constrained projects so that to plan and manage the projects in a better way.

In the second part of the thesis, we develop a new optimization technique to solve the resource-constrained project scheduling problem (RCPSP). We based on the fact that the highly ordered structures of crystals are achieved by simultaneous movement of molecules with decreasing temperature. Simulating the process of cooling a gas into crystal, a new optimization method, Molecule Search (MS), is proposed here to tackle the RCPSP. There are two main components of MS - molecule jumping and molecule walking. Molecule jumping is used to simulate the concurrent motion of high energy molecules. Molecule walking is a local refinement procedure, which is used to simulate the motion of low energy molecules. Three different kinds of molecule jumping rules are developed in this thesis. They are randomized cooling jumping, critical activity based jumping and hidden order based jumping. We also developed the forward-backward search (FBS) algorithm as the molecule walking here. Extensive experimental results have demonstrated the power of molecule jumping and walking. The experimental results on PSPLIB also show that Molecule Search is one of the best heuristics for RCPSP so far.

In the third part of the thesis, we further develop a population based Molecule Search algorithm (Molecule Bank Algorithm) for RCPSP. A molecule bank is used to save the current generation of elite solutions and historical best solutions. Crossover, selection, molecule jumping and molecule walking are used as operators when working on the solutions retrieved from the molecule bank, and newly generated elite solutions are saved back to the molecule bank. In MBA, dynamic population, molecule aging and molecule drifting are used to balance diversification and intensification so that at the beginning of the search process, a greater proportion computational effort is put into diversification and at the end of the search process, the effort is focused on intensification. The performance of MBA is tested on PSBLIB (the standard benchmark for RCPSP). Compared with the state-of-art heuristics, MBA emerges to be one of the best heuristics so far, achieving the best results on the J60 and J120 test sets. Moreover, 3 new best results for the J60 test set and 62 new best results for the J120 test sets were found by MBA. In the fourth part of the thesis, We study an over-constrained resource-constrained project scheduling problem where constraints cannot be relaxed. This problem originates from a local defense agency where activities to be scheduled are strongly ranked in a priority scheme determined by planners ahead of time and operational real-time demands require solutions to be available almost immediately. A hybrid framework is used which is composed of two levels. A high-level component explores different orderings of activities by priorities using Tabu Search or Genetic Algorithm, while in a low-level component, constraint programming and minimal critical sets are used to resolve conflicts. Real-data used to test the algorithm show that a larger number of high priority activities are scheduled when compared to a CP-based system used currently. Further tests were performed using randomly-generated data and results compared with CPLEX.

List of Tables

List of Figures

Chapter 1 Introduction

1.1 Project scheduling and Critical Path Method

In project scheduling, a set of activities has to be scheduled so as to minimize a given objective. In a project, each activity may have some predecessors, where the activity can only be scheduled after all its predecessors are completed. These relationships between activities are called precedence constraints. The precedence constraints between the activities can be clearly illustrated in a directed graph, which is called a project network. Each activity also has a given processing time, which is the duration of the activity. As an example, a project with 6 activities is shown in Figure 1.1.

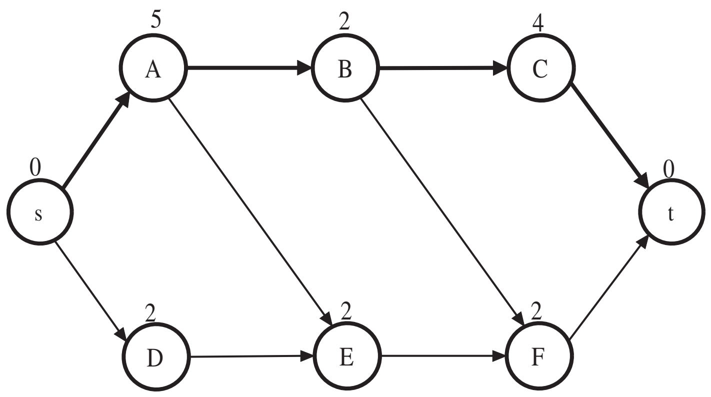


Figure 1.1 An example of project network with 6 activities

In Figure 1.1, s and t are two dummy activities marking the start and end of the project respectively. The precedence constraints are illustrated as directed edges in Figure 1.1. Since there are directed edges from A to E and from D to E, activity E should start only after activity A and activity D finish. The duration of activity A, B, C, D, E, and F is given as 5, 2, 4, 2, 2 and 2.

The most common objective in project scheduling is to minimize the makespan of the project, where the makespan can be defined as the maximum completion time of all the activities in the project. The minimum makespan of a project can be obtained using the Critical Path Method (CPM). The CPM was developed in a joint venture between DuPont Corporation and Remington Rand Corporation for managing plant maintenance projects. In CPM, a feasible sequence satisfying the precedence constraints to schedule all the activities is obtained through topological sort. After this, the earliest start time (ES) of each activity is calculated one by one according to the feasible sequence. Assuming A is the set of all the activities in a project. As $ES(s)$ is set to 0, the ES of the activity $i \in A$ is calculated as,

$$ES(i) = \max\{ES(j) + d(j), j \in predecessor(i)\} \tag{1.1}$$

where, $d(j)$ is the duration of activity j and $predecessor(i)$ is the set of direct predecessors of activity i.

The ES of the last dummy activity will be the minimum makespan of the project. For the above example, a feasible sequence to schedule all the activities can be s, A, B, C, D, E, F, t. The ES of all the activities is given in Table 1.1. In Table 1, we can

Table 1.1 CPM for example 1.1

Activity	s	A	B	C	D	E	F	t
ES	0	0	5	7	0	5	7	11
EF	0	5	7	11	2	7	9	11
LS	0	0	5	7	5	7	9	11
LF	0	5	7	11	7	9	11	11
Float	0	0	0	0	5	2	2	0

find that the minimum makespan of the project is 11 as the ES of the dummy activity t. The CPM is much more than finding the minimum makespan of the project. It can also calculate the critical path of the project, where the critical path is defined as the longest duration path in the project network. To find the critical path for a project, the latest start time of all the activities has to be calculated first. To calculate the latest start time of the activities we have to calculate the earliest finish time (EF) and latest finish time first (LF) of the activities also. The EF of activity $i \in A$ is calculated as,

$$EF(i) = ES(i) + d(i) \tag{1.2}$$

where $d(i)$ is the duration of activity i. As $LF(t)$ is set to $EF(t)$, the LF of the activity $i \in A$ is calculated as,

$$LF(i) = \min\{LF(j) - d(j), j \in successor(i)\} \tag{1.3}$$

where $successor(i)$ is the set of direct successors of activity i. The latest start time (LS) of the activity $i \in A$ can be calculated as,

$$LS(i) = LF(i) - d(i) \tag{1.4}$$

The EF, LF and LS values for the above example are given in Table 1.1 also. The difference between ES and LS of an activity is described as the slack time or float of the activity. The float of all the activities in the above example are also given in Table 1.1. Float has played central role in project scheduling, since it illustrates the flexibility of the activity, where delaying the activity within its float will not affect the completion time of the project. Float is also useful in identifying critical path. The critical path is composed of critical activities, where the critical activity is defined as the activity with 0 float. Therefore the critical activity can be identified as the activity with the same ES and LS value. For example 1.1, activities A, B and C are critical activities, since all of them have 0 float value. Hence, the critical path for example 1.1 is s-A-B-C-t, which is the bold line in Figure 1.1.

1.2 Project scheduling with resource constraints

In the traditional project scheduling, we consider that there are unlimited resources. However, in real-life projects, nearly all of the projects have limited resources, such as manpower, tools, oil and funding. Basically, there are three different resource types, renewable resource, non-renewable resource and doubly-constrained resource [9]. For renewable resource, such as man power, a pre-specified number of units of a resource is available for every period of the planning horizon. For non-renewable resource, such as funding, the total amount of the resource is limited and can not be regenerated. For doubly-constrained resource, such as oil, not only does it have limits for each unit time, but also the total amount of it is also limited. In this thesis, we only consider renewable resource, however, the definitions we proposed for float and critical activity also work for other resource types. However the definition need to be further extended for the multiple model resource constrained project scheduling problem.

With resource constraints, project scheduling becomes much harder. The classic resource-constrained project scheduling problem (RCPSP) with the objective to minimizing the makespan of the project has been proved to be NP-hard in strong sense [**?**], where there is no polynomial time or pseudo-polynomial time algorithm for RCPSP, unless $P = NP$. The CPM method no longer works for the project with resource constraints. As shown in Chapter 2, it also fails to identify float of activity, critical activity and critical path. As float and critical activity is central in project scheduling, numerous methods are developed in identifying float, critical activity for projects with resource constraints. However, as there are various optimal schedules for RCPSP, and different schedules may result in different float of activities and different critical activities, identifying float, critical activity and critical path for projects with resource constraints still remain as a great challenge in project management.

As a classic problem in project scheduling with resource constraints, RCPSP has found its applications in a lot of real-life problems. Due to the importance of RCPSP, a lot of algorithms have been proposed for it, and it still remains as one of the most important research areas in project management.

1.3 Outline of the thesis

To calculate float and to identify critical activity for projects with resource constraints, new definitions for float and critical activity are given in Chapter 2 [75]. To identify the flexibility of a set of activities, group float and critical set are also defined. As the complement for float, negative float is proposed and studied in Chapter 2. In Chapter 2, we also proposed several algorithms to calculate float and group float.

In Chapter 3, simulating the cooling process from gas to crystal, a new optimization technique, called Molecule Search (MS), is applied to solve RCPSP [78]. Molecule jumping and molecule walking are proposed as two components of MS. Three different jumping rules are designed and a new forward-backward search algorithm is proposed as molecule walking. Extensive experiments are conducted to test the performance of Molecule Search on PSPLIB (a standard benchmark of RCPSP).

In Chapter 4, a population based Molecule Search is proposed to solve RCPSP, which is called Molecule Bank Algorithm (MBA) [76]. In MBA, we store all elite solutions in a molecule bank, where 4 different kinds of operators, crossover, selection, molecule jumping and molecule walking, are used to generate new solutions from the old elite solutions in the molecule bank. Extensive experiments show that MBA is among the best algorithms for RCPSP so far, achieving the best results on the J60 and J120 test sets of PSPLIB.

In Chapter 5, a hybrid framework is proposed for a real-life over-constrained project scheduling problem [79], which is originated from a local defense agency. There are two levels of the hybrid framework. Genetic algorithm and Tabu Search are applied as the high level search algorithm. In the low level, constraint programming and minimal critical sets are used to resolve conflicts. Extensive experimental results are presented on both real-life data and random generated data.

Finally in Chapter 6, conclusions are given and possible future research is also discussed.

Chapter 2
Float and Critical Activity for Resource-constrained Projects

Float and critical path are keys to analyze activity networks in project management, when studied for the project networks with no resource limits. However, considering the limited resources, float can no longer be interpreted as the difference between the earliest start time (ES) and the latest start time (LS) of an activity and critical path is no longer viable to describe critical activities serving as the focus of the scheduling process.

The resource-constrained project scheduling problem is considered as one of the hardest problem in optimization, where it is NP-hard in strong sense [10]. Because projects are almost always resource-constrained, the RCPSP remains an important one. Key to its applications lies in using a good representation for float. This help the manager attain flexibility of activities to plan for contingency. Therefore, various approaches have been proposed to calculate float [99, 16] and identify critical activities [99, 121, 32, 120, 15, 113].

A measure for float was explored by Raz and Marshall [99] and Bowers [16], capturing the flexibility of resource-constrained projects. Raz and Marshall followed the classical definition of float and calculated it as the difference between the late schedule time and corresponding early schedule time. However, the standard float is not unique. Bowers recognized that there are different schedules with identical makespan for a resource-constrained project, and float varies according to schedules. To resolve this, he redefined three new types of float. However, floats proposed by Bowers are still calculated as the difference between the latest start time with the earliest start time of all possible schedules. On the one hand, it is prohibitive to find all the possible schedules with identical durations computationally. On the other hand, the difference between the earliest start time and latest start time can no longer capture the meaning of float for the resource-constrained project, for which a detailed example will be given in section 2.1.

Recognizing that it was misleading to apply CPM to resource-constrained projects, Wiest [32] pursued a new representation of critical activities as critical sequence in the resource-constrained projects. The critical sequence is defined as the set of activities which are determined not only by just the technological ordering, but also by resource constraints. Different approaches, such as [99, 121, 120, 15, 113] have been proposed to identify the critical sequence in resource-constrained projects. However, the critical sequence is heavily dependent on the techniques used in the scheduling of resource-constrained projects, where activities identified as critical might be quite different for different schedules.

In the book Critical Chain [51] by Goldratt, critical chain was proposed as a novel approach for project management, where critical chain is defined as a chain of activities satisfying not just precedence constraints but also resource constraints, and delaying any activities of the critical chain will result in the delay of the whole project. There is little difference between the concept of critical chain and critical sequence, both of which are schedule based, except that buffer management is used for the critical chain. Also, different feasible schedules may yield different critical chains. In other words, the activity may be critical in one schedule, but not critical in another. Detailed discussion on critical chain can be found in [98, 53].

Being aware of the lack of a suitable way to identify critical activities for the resource-constrained projects, Rivera and Duran [103] proposed the concepts of the critical sets and critical clouds, where examples were given to illustrate the inability of other current approaches to identify critical activities. However, the critical set is defined according to time. A simple example in section 2.1 will highlight that there may be other set of activities which are critical but not decided by this definition. Moreover, their algorithm is not practical to identify all the critical sets even for the medium size projects with the exact algorithm proposed by the authors, due to the fact that there may be exponential number of critical sets. In fact, to identify a single critical activity is already NP-hard.

Many methods have been developed to tackle RCPSP (resource-constrained project scheduling problem), such as Branch and Bound methods [39, 40, 21, 106, 20], constraint-propagation-based cutting planes [36], heuristics X-pass methods [30, 33, 65], Tabu Search [5, 90], Simulated Annealing [12, 14] and Genetic Algorithm [52, 62, 57, 114]. A detailed survey on the methods for resource-constrained project scheduling problem can be found in [41, 68, 67], where a standard benchmark is used to evaluate different methods [69]. Here, Branch and Bound Algorithm will be applied to calculate float for small size projects. We will also use one of the most effective but simple heuristics, Simulate Annealing, to calculate float and to identify critical activities for large size resource-constrained projects.

In the next section, we review the shortcomings of current approaches in calculating float and identifying critical activities, after which new definitions for float and critical activity are presented in section 2.2, where a natural connection are built between them. In section 2.3, group float is defined, and a more general definition for critical set is given according to group float. Three algorithms are presented in section 2.4 to calculate float. In section 2.5 the details on how to find critical sets and group sets are given. After that, we presented the float graph to illustrate float and group float for resource-constrained projects in section 2.6. we have also discussed how to manage resource-constrained projects with float graph. In section 2.7, negative float is proposed as a complement to float. Like float, we also provide definition for negative critical activity, negative group float, negative float set and negative critical set. A notion of negative float graph is also developed to illustrate negative float. We also give the definition of zero critical activity, which is critical and negative critical. We show that special attention should be paid to zero critical activity, since changing the duration of it will directly affect the completion time of the project. Extensive computational results on float calculation, critical sets and float sets are presented in section 2.8. Finally, we draw our conclusions and show some possibilities for future research in section 2.9.

2.1 Current approaches for float and critical activity

Although a lot of effort has been made to discover new ways of interpreting float for project networks with resource constraints, we still lack a proper definition for the float itself for resource-constrained projects. The difficulty in calculating float comes from the indeterminable nature of scheduling the resource-constrained projects. The resource-constrained project scheduling problem is NP-hard in strong sense, and there may be many different schedules for the identical project makespan. Therefore, float can no longer be presented in the old manner, considering a single schedule. Different new approaches have been proposed by [99, 16]. However researchers still use the difference between the earliest start time and the latest start time [99] or the difference between the earliest earliest start time and the latest latest start time of all schedules [16] to represent float of an activity. A simple example given below will illustrate that the difference between the earliest start time and the latest start time can no longer capture the meaning of the float.

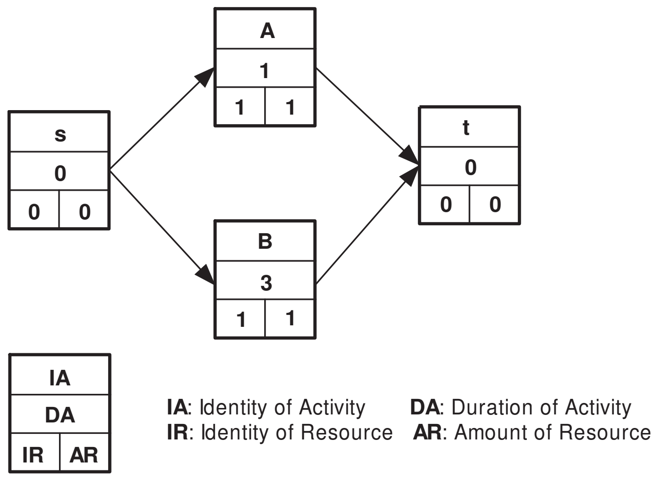


Figure 2.1 Precedence network for example 2.1

In example 2.1, given in Figure 2.1, there are only two activities, A and B, to be scheduled, where activity s and t are dummy activities illustrating the start and the end of the project. There are no precedence constraints between activity A and B, and they share the same renewable resource 1, where there is only 1 unit of resource 1 available for the plan horizon and the two activities consume 1 unit of resource 1 each. So activity A and activity B cannot be scheduled together according to the resource constraint. The two optimal schedules with project makespan 4 are shown in Figure 2.2.

In the two schedules given in Figure 2.2, the earliest earliest start time of activity A is 0, and the latest latest start time of activity A is 3. According to Bowers' calculation, the float of activity A is 3. However this has lost the meaning of float,

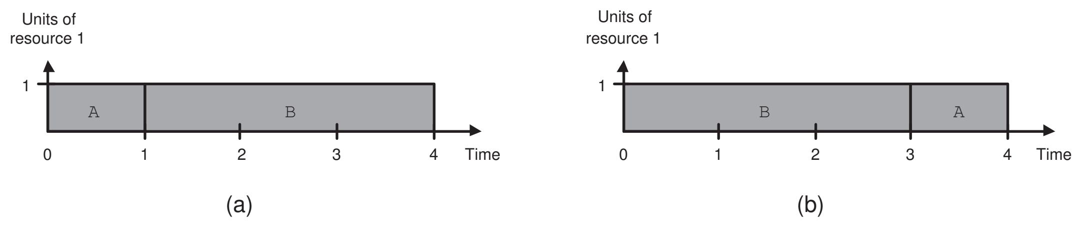


Figure 2.2 Feasible schedules for example 2.1 with project makespan 4.

which is defined as the flexibility of the activity, since activity A can only be scheduled at 0 or 3 and cannot be scheduled at time 1 or 2 without delaying the entire project, and float should be an interval within which the activity can be delayed any time. To capture the meaning of flexibility properly, a new definition of float is given in section 2.2.

Like float, the concept for critical path can no longer capture the meaning as the path of the critical activities for resource-constrained projects. In spite of the difficulties in scheduling the resource-constrained projects, a great deal of valuable research has been done to redefine the critical path considering the resource constraints, such as critical sequence [121, 32], critical chain [51] and recently published critical sets [103]. However, critical sequence and critical chain are formed according to a certain baseline schedule. The activity may be critical in one schedule but not critical in another. Rivera and Duran [103] have recognized the problem to identify critical activities with critical sequence and critical chain, and they developed a new concept as critical set, where delaying all the activities inside a critical set will cause the delay of the entire project. However critical set still faces the same problem as critical sequence and critical chain, since it is defined according to time, and different schedules may have different critical sets also. To clarify these shortcomings, the following resource-constrained project is used as example 2.2.

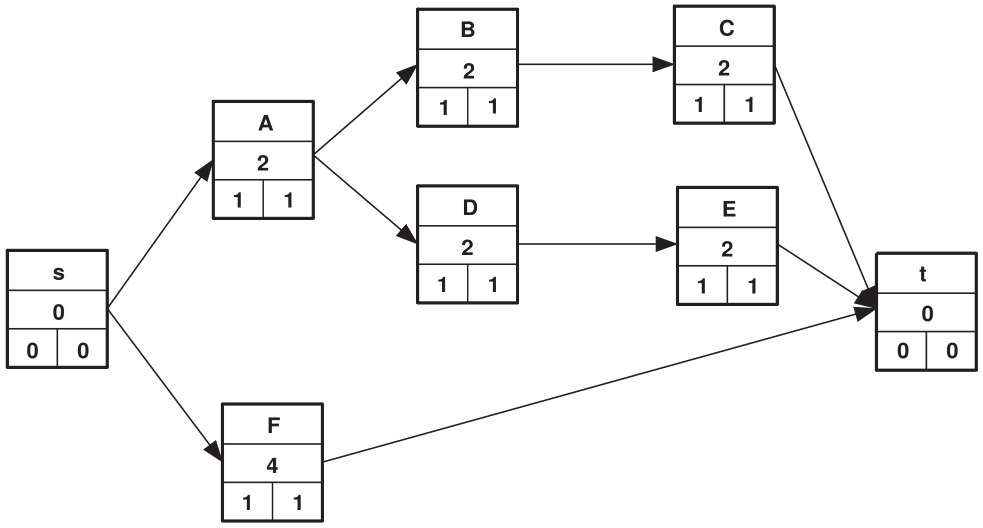


Figure 2.3 Precedence network for example 2.2.

In Figure 2.3, there are 5 activities and 1 renewable resource, where the availability of the resource is 2 and each activity consumes 1 unit of the resource. Two optimal schedules with the same duration 8 are presented in Figure 2.4, which can be verified according the symmetry between B and D, C and E. For the first schedule in Figure 2.4 (a), the critical sequence and critical chain will be identified as D-E, and the critical sets will be selected as {F,A}, {F,B} and {C,D}. While for the schedule in Figure 2.4 (b), the critical sequence and critical chain change to B-C, and the critical sets change to {F,A}, {F,D} and {B,E}. In fact, as long as all the precedence constraints and resource constraints are fixed, the critical activities should be fixed, where delaying any of the critical activities will delay the whole project. Here, in example 2.2, there is no critical activity at all, since each activity can be extended for 2 time units alone without affecting the project completion time. The critical set is a bit complicated, and we will present a more general definition for critical set in section 2.3, where delaying all the activities in the critical set will delay the entire project, and according to this definition the critical sets for example 2.2 are {A,F}, {B,D}, {B,E}, {B,F}, {C,D}, {C,E}, {D,F}, {A,B,C} and {A,D,E}. In section 2.2, a new definition for float will be introduced and it will be used to define critical activity for resource-constrained projects.

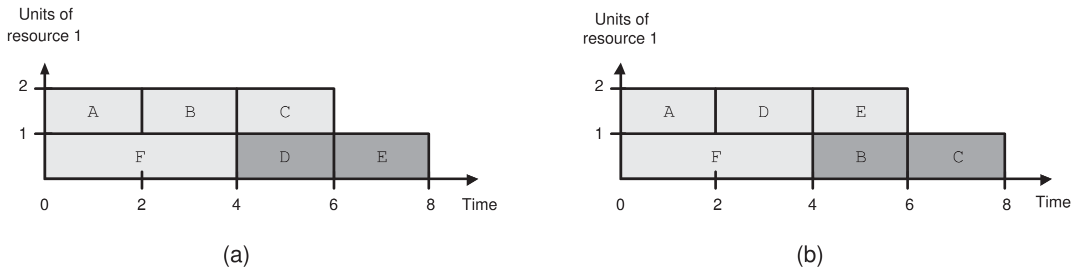


Figure 2.4 Feasible schedules for example 2.2.

2.2 New definitions for float and critical activity

Traditionally, there are two types of float to measure the scheduling flexibility associated with activities in the project networks. One type of float is total float, which is defined as the amount of time that the finish time of an activity can be delayed without affecting the completion of the entire project. Total float can be calculated as the difference between the earliest start time and the latest start time of an activity. The other type of float is free float. Free float is the amount of the time that the finish time of an activity can be delayed without affecting the finish time of any other activities. Free float can be calculated as the difference between the activity's earliest finish time and the earliest start time of all its successors. However, as discussed in the last section, we can no longer calculate float as the difference between earliest start time and latest start time for projects with resource limits, since the activity might not be scheduled within the float calculated.

A new definition for float, in fact a clarification of the old definition, will be presented here. Before this, we would like to give the formal definition for the decision form of the resource-constrained project scheduling problem(RCPSP) following the notation by Garey and Johnson [49], and review the complexity result for it.

Problem 2.2.1 (RCPSP) *Set A of activities, for each $a \in A$, a duration $d(a) \in \overline{Z^-}$, there is a partial order $\prec$ as the precedence constraints, number $r \in Z^+$ of resources, with resources bound B_i, $1 \leq i \leq r$, resource constraints $R_i(a)$, $0 \leq R_i(a) \leq B_i$, for each activity a and resource i, and an overall completion time $D \in Z^+$.*
QUESTION: Is there a schedule σ for A that meets the overall completion time D and obeys the precedence constraints and resource constraints, i.e., a one-to-one function

$\sigma : T \rightarrow Z_0^+$, with $a \prec a'$ implying $\sigma(a) + d(a) \leq \sigma(a')$, such that for all $u \geq 0$, if $S(u)$ is the set of all $a \in A$ for which $\sigma(a) \leq u < \sigma(a) + d(a)$, then for each resource i the sum of $R_i(a)$ over all $a \in S(u)$ is at most B_i?

As a generalization of the job-shop scheduling problem, the RCPSP has been proved to be NP-hard in the strong sense by Blazewicz [10].

Lemma 2.2.2 *RCPSP is NP-hard in strong sense.*

As we know, float was defined as the amount of time that the finish time of an activity can be delayed without affecting the completion of the entire project. In project management, the delay of an activity usually means the duration of the activity has been extended. The other way to interpret the delay of the finish time of an activity is the delay of the start time of the activity, without affecting the duration of activity. Without resource constraints, the two meanings of float can be calculated in the same way as the difference between the earliest start time and the latest start time of an activity. However, as discussed in section 2.1, float, the flexibility of the activity, can no longer be interpreted as the difference between the earliest start time and the latest start time and, as shown in example 2.1, the start time of the activity may not be continuous interval, which loses the meaning of float. Furthermore, to determine whether an activity can be scheduled at a certain time is NP-hard also, since RCPSP can be easily reduced to the problem with setting the start time for the dummy activity at the end of the project. Therefore we follow the first explanation for float, and the float here is defined as the amount of time the duration of the activity can be extended without affecting the completion of the entire project, where it is easy to prove that the float is a continuous interval.

Definition 2.2.3 (Float) *Float of an activity in the resource-constrained project is defined as the amount of time the duration of an activity can be extended without affecting the completion time of the entire project.*

According to the new definition for float, assuming the maximum float of a certain activity is F, it is easy to see that the duration of the activity can be extended by any $f \in Z, f \in [0, F]$, without delaying the entire project. In other words, when the duration of activity a is extended by f in the new project, if there exists a schedule for the new project satisfying the old completion time, f is one possible float for activity a. In the new float definition, to finish the new project in the old completion time, all the possible schedules are considered instead of one schedule. In [108], Sprecher et al. also considered all the possible schedules in defining the global left shift. As changing of schedules is not preferred by project managers, the float defined here is mainly to help project managers to view the flexibility of each activity, so that project managers could select a robust schedule to buffer critical activities and to minimize the risk of extending the project when uncertain events happen.

With the new definition for float, we are able to measure float as an interval again.

The decision form of the problem to calculate float of each activity for resource-constrained project scheduling (Float-RCPSP) is defined in the following.

Problem 2.2.4 (Float-RCPSP) *Set A of activities, for each $a \in A$, a duration $d(a) \in \overline{Z^-}$, there is a partial order $\prec$ as the precedence constraints, number $r \in Z^+$ of resources, with resources bound B_i, $1 \leq i \leq r$, resource constraints $R_i(a)$, $0 \leq R_i(a) \leq B_i$, for each activity a and resource i, and an overall completion time $D \in Z^+$, there is a schedule σ for A that meets the overall completion time D and obeys the precedence constraints and resource constraints, a float F for a certain activity $\mathtt{b}$, and assume $d'(a)$ is the duration function, where $d'(a) = d(a)$ for activity $a \neq \mathtt{b}$, and $d'(\mathtt{b}) = d(\mathtt{b}) + F$, assume A' is the activity set with the new duration function $d'(a)$.*
QUESTION: Is there a schedule σ' for A' that meets the overall completion time D and obeys the precedence constraints and resource constraints?

The Float-RCPSP problem is NP-hard in strong sense also.

Theorem 2.2.5 *Float-RCPSP is NP-hard in strong sense.*

PROOF. We show that the RCPSP can be Turing-reduced to Float-RCPSP. Suppose $S(A_s, d_s, P_s, R_s, D_s, b, F)$ is the subroutine for solving the Float-RCPSP, with the activity set A_s, duration function d_s, precedence constraints P_s, resource constraints R_s, overall completion time D_s, selected activity b, the Float F. The following algorithm can be used to solve the RCPSP, with activity set A, duration function d, precedence constraints P, resource constraints R, overall completion time D.

```
for each $a \in A$ do
    $d_s(a) \leftarrow 0$
end for
for each $a \in A$ do
    call $S(A, d_s, P, R, \mathtt{D}, a, d(a))$
    if the answer from $S$ is "no" then
        output "no", exit
    end if
    $d_s(a) \leftarrow d(a)$
end for
output "yes"
```

This procedure uses less than or equal to n calls of the subroutine S, where n is the total number of activities in the project. So RCPSP can be Turing-reduced to Float-RCPSP, and the subroutine has the same up-bound on the input as the RCPSP. Therefore, Float-RCPSP is NP-hard in strong sense also.

With the new definition for float, the critical activity can be defined as the activity with maximum float is 0, which means for any schedule, extending the duration of the critical activity will delay the entire project.

Definition 2.2.6 (Critical Activity) *In the resource-constrained project, an activity is defined as critical activity, if its maximum float is 0 for the specified overall completion time of the project.*

2.3 Group float, critical set and float set

We have redefined the concepts of float and critical activity for the resource-constrained projects, where the maximum float of an activity also illustrates how critical the activity is. In practice, sometimes, project managers are concerned with a group of activities, where they are interested in investigating how long the group of activities can be delayed together without affecting the project completion. There may be no critical activities inside the activity group, while simultaneously delaying in every activity inside the group would probably delay the entire project.

The critical property of a group of activities is also recognized by Rivera and Duran [103], where the novel concept of critical set and critical cloud are developed. However, in [103], the critical set is defined according to time, where only activities with common process time are possible to be considered as critical set. In section 2.1, with example 2.2, we have shown that defining critical set based on time is not enough to discover all the critical sets.

Before we give a more general definition for critical set, we would like to present the concept of group float first. The group float measures the amount of time that the duration of all the activities inside a group can be extended simultaneous without delaying the entire project.

Definition 2.3.1 (Group Float) *Group Float is defined as the amount of time that the duration of a set of activities can be simultaneously extended without affecting the completion of the entire project.*

The normal way to delay a group of activities is to delay the same amount of time for each activity. Alternatively, we may need to delay the activities proportional to the original duration of the activities. To clarify how long each activity inside the group can be delayed, delay vector C is introduced here. Assuming the group of activities to be measured are $U = \{a_{\mathtt{u1}}, a_{\mathtt{u2}}, ..., a_{\mathtt{un}}\}$, with duration presented as a vector $W = (d_{\mathtt{u1}}, d_{\mathtt{u2}}, ..., d_{\mathtt{un}})$, the delay vector C can be defined as, $C = (c_{\mathtt{u1}}, c_{\mathtt{u2}}, ..., c_{\mathtt{un}})$, with $c_{\mathtt{u}i} \in Z^+$ and $GCD(c_{\mathtt{u1}}, c_{\mathtt{u2}}, ..., c_{\mathtt{un}}) = 1$. With the delay vector C, group float can be presented as $k \in \overline{Z^-}$, where for the new duration vector $W' = W + kC$, there is a schedule satisfying the overall completion time. With the definition of group float, a more general form of critical set can be defined as

Definition 2.3.2 (Critical Set) *In resource-constrained projects, critical set is a set of activities, where the maximum group float of the set of activities is 0 for the specified overall completion time of the project, and there is no subset of it with the same property.*

Critical set can be identified as the set of activities with no flexibility, which is different from the minimal forbidden set [6] and minimal critical set [25]. The minimal forbidden set and minimal critical set are used to detect and resolve resource conflicts in resource-constrained-Project Scheduling Problem with Time Windows (RCPSP/max). We also define the float set as,

Definition 2.3.3 (Float Set) *In resource-constrained projects, float set is a set of activities, where the maximum group float of the set of activities is larger than 0 for the specified overall completion time of the project.*

We can easily show that the critical sets for example 2.2 are {A,F}, {B,D}, {B,E}, {B,F}, {C,D}, {C,E}, {D,F}, {A,B,C} and {A,D,E} according to the new definition, instead of only {F,A}, {F,B} and {C,D} with the old definition based on time, for Figure 2.4 (a).

It is obvious that calculating float is a sub problem of calculating group float. Therefore calculating group float is NP-hard. Similar to float, group float can also be presented as an interval. Like float, group float for a set of activities can be any integer value from 0 to the maximum group float of it.

2.4 Algorithms for calculating maximum float

Float for any activity in the project is an integer value ranging from 0 to its maximum float and in Definition 2.2.6 critical activity is also defined as the activity with maximum float 0. Therefore, in order to find the range of the float and identify critical activities in a resource-constrained project, several algorithms are developed to calculate maximum float. Since *Float-RCPSP* is NP-hard in strong sense (Theorem 2.2.5), there is no polynomial time or pseudo-polynomial time algorithm to find the maximum float, unless $P = NP$. However, with the help of an enormous amount of research done on the RCPSP, it is possible for us to accurately calculate the maximum float for relatively small size projects by branch and bound algorithm, where the accurate maximum float calculated by the exact method is named as *E-float*. Meanwhile a lot of heuristic algorithms have been proposed to tackle the large size RCPSP. With the help of Simulated Annealing, we have proposed a novel way to calculate the float statistically by Testing Hypothesis. We also develop another fast algorithm to calculate the maximum float using SA directly. We define the approximated maximum float calculated by heuristics as *H-Float*, and *H-Float* calculated by Testing Hypothesis is a good approximation for the *E-float* with the support of extensive computational results in section 2.8. In this section, we also study the way to calculate the float based on a certain schedule, where the maximum float for a certain schedule is called *S-Float*.

2.4.1 Calculating ***E-Float***

According to theorem 6, there is no polynomial time or pseudo-polynomial time algorithm to calculate float, unless $P = NP$. However, a great deal of research has been done to develop various lower bounds [39, 20, 23] for RCPSP, and there are different branch and bound algorithms proposed to find the optimal schedules with the minimum makespan for RCPSP, such as [39, 40, 21, 106, 20]. Embedding with the branch and bound algorithm *dh_procedure* proposed by Demeulemeester and Herroelen [39, 40], the following binary search algorithm *bab_float* is used to calculate float, where a is the activity selected and the its float will be calculated, D is the original completion time for the project and D' is the makespan of the project with duration of activity b extended, the *dh_rocedure* is called with the activity set A, the duration of activities d, the precedence constraints P, and the resource constraints R, and it will return the minimum makespan of the project.

Algorithm 1 calculating *E-float* with branch and bound

```
bab_float(A,P,R,D,a)
$hf \leftarrow upbound\_float(A, P, R, D, a)$, $lf \leftarrow 0$
while $lf < hf$ do
  $mf \leftarrow (hf + lf)/2 + 1$
  $d'(a) \leftarrow d(a) + mf$
  $A' \leftarrow A$
  update $d'(a)$ as the duration for activity $a$ in $A'$
  $D' \leftarrow dh\_procedure(A', d', P, R)$
  if $D' \leq D$ then
    $lf \leftarrow mf$
  else
    $hf \leftarrow mf - 1$
  end if
end while
return $lf$
```

In the above algorithm, procedure *upbound_float* calculates the upper bound of float for activity a. In this work, two upper bounds for float are proposed. The difference between the earliest start time and latest start time considering no resource constraints serves as the first upper bound. The other upper bound is derived from the the lower bound, LB3 [84] for RCPSP, where it calculates how much the duration of an activity can be extended to make the lower bound LB3 equal to the original completion time. The two upper bounds of float are presented in Lemma 2.4.1 and Lemma 2.4.2. The details of algorithm *upbound_float* are given in the following.

Lemma 2.4.1 *In a resource-constrained project P, assuming current makspan for the project is D, applying the CPM on project P without considering resource constraints, for any activity a, let $est(a)$ and $lst(a)$ be earliest start time and latest start time of activity a and E be the new makespan obtained by the CPM, the upper bound $up_1(a)$ for the float of activity a, is given as:*

$$up_1(a) = lst(a) - est(a) + \mathtt{D} - \mathtt{E} \tag{2.1}$$

Lemma 2.4.2 *In a resource-constrained project P, assuming the set of all the activities is A, supposing current completion time for the project is D, for any activity a, let $T(a)$ be the set of activities which can be scheduled together with activity a, and let activity set C be $A - \{a\} - T(a)$, assuming the lower bound LB3 for activity set C is Z, the upper bound $up_2(a)$ for the float of activity a is given as:*

$$up_2(a) = \mathtt{D} - \mathtt{Z} \tag{2.2}$$

In *upbound_float* algorithm, $est(a)$ and $lst(a)$ are the earliest and latest start time

Algorithm 2 finding upper bound of float

```
$upbound\_float(A,P,R,D,a)$
$up_1(a) \leftarrow lst(a) - est(a) + \mathtt{D} - cpm(A,P)$
$C \leftarrow A - (T(a) \bigcup \{a\})$
sort $C$ into activity list $AL$ with no decreasing $|T(c)|$ (no increasing order of $d(c)$ as a tie-breaker)
$LB3 \leftarrow 0$
$i \leftarrow 0$
**while** $C \neq \emptyset$ **do**
  **if** $AL[i] \in C$ **then**
    $c \leftarrow AL[i]$
    $LB3 \leftarrow d(c)$
    $C \leftarrow C - \{c\} - T(c)$
  **end if**
  $i \leftarrow i + 1$
**end while**
$up_2(a) \leftarrow D - LB3$
**if** $up_1(a) \leq up_2(a)$ **then**
  return $up_1(a)$
**else**
  return $up_2(a)$
**end if**
```

of activity a, considering the project without resource limits, and *cpm* procedure returns makespan for the project without resource limits by CPM.

It is obvious that group float can be calculated in the same way as float with the branch and bound algorithm. The only difference is that the duration for a set of activities are extended together instead.

Assuming the time complexity for the *dh_procedure* is $\mathcal{H}$, the complexity to calculate float is $O(\mathcal{H} \lg \mathcal{D})$. As shown in [40], the *dh_procedure* is only capable of calculating 479 out of 480 cases of the J30 test set within 1 hour running time, where the J30 test set is one of the test sets proposed by Kolisch and Sprecher [69] as a standard benchmark to evaluate algorithms for RCPSP. There are 30 activities and 4 renewable resources for the J30 test set. All the instances of J30 have been solved to optimality when more computational time allowed. None of the exact methods can solve all the test cases in the J60 test set, with 60 activities. In practice, for a simple project, there may be hundreds of activities involved. It is impossible for us to calculate float with the exact methods for such a large number of activities due to the prohibitive nature of RCPSP. Therefore new methods are needed to calculate float and detect critical activities for large size resource-constrained projects.

Although exact methods can not be used to calculate float for large size projects, there are a lot of heuristic algorithms available for RCPSP, where they are designed to find good solutions in reasonable time. A detailed survey on the heuristics for

RCPSP can be found in [41, 68, 67]. Extensive computational experiments have shown that the heuristic algorithms can find all the optimal solutions for the J30 test set in a short time, and nearly all the best upper bounds for the J60 and J120 test sets are found by heuristic algorithms also. However, there are no guarantees from heuristic algorithms and most heuristic algorithms involve random process inside, such as the Simulated Annealing and Genetic Algorithms, where different results may be produced by different runs. Therefore statistical inference is used here to calculate float based on the results given by heuristic algorithms, and we name the maximum float calculated by heuristic algorithms as *H-Float* as a comparison to *E-float*.

2.4.2 Calculating *H-Float* by Testing Hypothesis

Simulated Annealing (SA), one of the most successful heuristics for combinatorial optimization problems, is notable not only for fast convergence to high quality solutions, but also for the simplicity of implementation. The intuitive idea to apply the concepts of annealing to optimization problems was first introduced by Kirkpatrick, Gelatt and Vecchi [60] and independently by Černy [27]. With the inspiration of the physical annealing process, SA has been widely used for different kinds of NP-hard problems.

SA was first applied to RCPSP by Boctor [12], where solutions for RCPSP are presented as activity lists. The activity list is an order of all the activities satisfying the precedence constraints, which was first suggested by Kelley [58]. Computational

Results showed SA outperformed the Tabu Search and some of the best priority rules based heuristics. An original cooling scheme of SA for RCPSP was explored by Bouleimen and Lecocq [14], which not only improved the efficiency of SA but also enable SA to become one of the best heuristics for RCPSP [67].

Here we follow the SA procedure proposed by Bouleimen and Lecocq [14]. A general form of Simulated Annealing Algorithm for resource-constrained project scheduling problem is given below. In the *SA* procedure, $\mathtt{T_0}$ is the start temperature of

Algorithm 3 Simulated Annealing for RCPSP

```
sa(A,P,R)
generate initial solution S
$D \leftarrow f(S)$,$N_b \leftarrow \mathtt{N_0}$
$T \leftarrow \mathtt{T_0}$, $k \leftarrow 0$, $u \leftarrow \mathtt{u_0}$
while $k < L_{SGS}$ do
  for $i \leftarrow 1$ to $N_b$ do
    generate randomly a neighbor solution S' of S
    if $e^{(f(S)-f(S'))/k_B T} > rand(0,1)$ then
      $S \leftarrow S'$
      if $f(S) < D$ then
        $D \leftarrow f(S)$
      end if
    end if
    if $k + i \geq L_{SGS}$ then
      return $D$
    end if
  end for
  $k \leftarrow k + N_b$, $N_b \leftarrow N_b \cdot u$, $u \leftarrow u + 1$
  if $T > \mathtt{T_e}$ then
    $T \leftarrow \alpha \cdot T$
  end if
end while
return $D$
```

the annealing process, $\mathtt{T_e}$ is the ending temperature, N_b is the number of neighbors explored at each temperature, the initial value for N_b is $\mathtt{N_0}$, and N_b is increased progressively as $N_b \leftarrow N_b \cdot u$, where the initial value for u is set to $\mathtt{u_0}$, L_{SGS} is the maximum step allowed, where, since SA uses SGS (Serial Schedule Generation Scheme)[66] to evaluate solutions, L_{SGS} also represents the limit of SGS calls allowed. k_B is the Boltzmann constant, $f : S \rightarrow D$ is the function map of the solutions to its overall makespan and rand(0,1) generate a random number uniformly from $(0, 1)$.

According to Bouleimen and Lecocq [14], $\mathtt{T_0}$ is set to:

$$\mathtt{T_0} = -\frac{f(S)}{5ln(0.01)} \tag{2.3}$$

so that the initial probability of selecting the solution 20% worse than the initial solution S is 0.01. According to our preliminary experiments $\mathtt{u_0}$ is set to 4 here. The setting for $\mathtt{T_e}$ and L_{SGS} will be discussed later in the computational study.

Modeled mathematically with the theory of finite Markov chains, the solutions obtained by *SA* have been proved to converge to the Boltzmann distribution in finite steps [1]. In the study of the physical annealing process, Boltzmann distribution, the distribution of the energy of solid at a certain temperature, is used to characterize thermal equilibrium. The discrete form of Boltzmann distribution is given below.

$$P(E = \epsilon_i) = \frac{g_i e^{-\epsilon_i / k_B T}}{Z(T)} \tag{2.4}$$

where E is a random variable denoting the energy of a particle in the solid at the current temperature T. g_i is the degeneracy, or number of states having energy ϵ_i.

$Z(T)$ is the partition function, which is defined as.

$$Z(T) = \sum_j g_j e^{-\epsilon_j / k_B T} \tag{2.5}$$

For the Float-RCPSP problem, we need to consider two possible ways to obtain the original overall completion time D. In the first case, D is a fixed number selected by the project manager. In the second case, D is also obtained by heuristics, since the RCPSP is NP-hard, where we assume the heuristic used to obtain a near optimal schedule with completion time D is SA. Here we will only discuss the second case where D is obtained by SA, because the method we present in the following can also be applied to the first case.

Since the solutions obtained by SA converge to the Boltzmann distribution, the expected value of the SA solutions will be used to acquire *H-Float*. The theorem given below reveals that the difference between the expected values of the SA results for two instances of RCPSP depicts the difference between the optimal completion times of the two instances, when the temperature T approaches 0.

Theorem 2.4.3 *Suppose* $P(X = x_i) = \frac{g_i e^{-x_i / k_B T}}{\sum_j g_j e^{-x_j / k_B T}}$ *is the distribution for the solutions obtained by SA on the origin RCPSP instance, where X is a random variable on the completion time of the project, let* $P(Y = y_i) = \frac{g_i e^{-y_i / k_B T}}{\sum_j g_j e^{-y_j / k_B T}}$ *be the distribution for the solutions obtained by SA on the RCPSP instance with activity b's duration extended for F, float given in the problem Float-RCPSP, let* x_{min} *represent the optimal solution for the origin RCPSP instance and* y_{min} *be the optimal solution for the new*

RCPSP instance, we have:

$$\lim_{T\to 0}(E(X) - E(Y)) = x_{min} - y_{min} \tag{2.6}$$

PROOF.

$$\lim_{T\to 0}(E(X) - E(Y)) = \lim_{T\to 0} E(X) - \lim_{T\to 0} E(Y) \tag{2.7}$$

$$\begin{aligned}
\lim_{T\to 0} E(X) &= \lim_{T\to 0} \sum_i \frac{g_i x_i e^{-x_i/k_B T}}{\sum_j g_j e^{-x_j/k_B T}} && (2.8)\\
&= \lim_{T\to 0} \sum_i \frac{g_i x_i e^{-(x_i - x_{min})/k_B T}}{\sum_j g_j e^{-(x_j - x_{min})/k_B T}} && (2.9)\\
&= \sum_i \lim_{T\to 0} \frac{g_i x_i e^{-(x_i - x_{min})/k_B T}}{\sum_j g_j e^{-(x_j - x_{min})/k_B T}} && (2.10)\\
&= \frac{g_{min} x_{min}}{g_{min}} && (2.11)\\
&= x_{min} && (2.12)
\end{aligned}$$

In the same way, we have,

$$\lim_{T\to 0} E(Y) = y_{min} \tag{2.13}$$

Therefore,

$$\lim_{T\to 0}(E(X) - E(Y)) = x_{min} - y_{min}. \tag{2.14}$$

If there are many energy states, we can use the continuous form of the Boltzmann distribution to approximate the discrete Boltzmann distribution. Without T approaching 0, We have the same results as theorem 2.4.3.

Theorem 2.4.4 *Assume X is a random variable on the completion time of the project for the solutions obtained by SA on the original RCPSP instance, with density function* $f(x) = \frac{e^{-(x - x_{min})/k_B T}}{k_B T}$*, let Y be a random variable on the completion time of* *the project for the solutions obtained by SA on the RCPSP instance with activity b's duration extended by F given in the problem Float-RCPSP, with density function* $f(y) = \frac{e^{-(y_i - y_{min})/k_B T}}{k_B T}$*, where* y_{min} *represent the optimal solution for the origin RCPSP instance and* y_{min} *be the optimal solution for the new RCPSP instance, we have,*

$$E(X) - E(Y) = x_{min} - y_{min} \tag{2.15}$$

PROOF. The expected value $E(X)$ can be obtained as,

$$\begin{aligned} E(X) &= \int_{x_{min}}^{\infty} \frac{xe^{-(x-x_{min})/k_B T}}{k_B T} \\ &= x_{min} + k_B T \end{aligned} \tag{2.16}$$

In the same way, we have,

$$E(Y) = y_{min} + k_B T \tag{2.17}$$

Therefore, $E(X) - E(Y) = x_{min} - y_{min}$.

The variance of the continuous form of Boltzmann distribution also can be derived as,

$$Var(X) = E(X^2) - [E(X)]^2 \tag{2.18}$$

$$= \int_{x_{min}}^{\infty} \frac{x^2 e^{-(x-x_{min})/k_B T}}{k_B T} - [\int_{x_{min}}^{\infty} \frac{xe^{-(x-x_{min})/k_B T}}{k_B T}]^2 \tag{2.19}$$

$$= k_B^2 T^2 \tag{2.20}$$

With Theorem 2.4.3 and Theorem 2.4.4, whether the duration of an activity can be extended for F can be decided approximately according to the difference between the means of SA solutions for the origin instance and for the instance with extended activity. If the difference is near 0, the activity can probably be extended for F without affecting the overall completion time of the project. Otherwise, if the difference is larger than 0, there is a large chance that extending the activity for F will cause delay of the entire project. With the help of Central Limit Theory (CLT), we can use Testing Hypothesis to decide whether the original RCPSP instance and the revised instance with duration of an activity extended by F have the same mean, which also indicates the same optimal completion times. Supposing $X_1, X_2, ..., X_n$ are n independent variables and each X_i has the same Boltzmann distribution with mean μ_x and variance σ^2 for n runs of SA results for the original RCPSP instance, where the sample mean $\overline{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$, $\overline{X}$ approaches normal distribution with a large enough sample size n according to CLT.

$$\overline{X} \sim N[\mu_x, \frac{\sigma^2}{n}] \tag{2.21}$$

In the same way, let $Y_1, Y_2, ..., Y_n$ be n independent variables and each Y_i has the same Boltzmann distribution with mean μ_y and variance σ^2 for n runs of SA results for the revised RCPSP instance, where the sample mean $\overline{Y} = \frac{1}{n}\sum_{i=1}^{n} Y_i$, we have,

$$\overline{Y} \sim N[\mu_y, \frac{\sigma^2}{n}] \tag{2.22}$$

Therefore we have the $\overline{Y} - \overline{X}$ also approaches normal distribution with large enough sample size n.

$$\overline{Y} - \overline{X} \sim N[\mu_y - \mu_x, \frac{2\sigma^2}{n}] \tag{2.23}$$

we consider testing the following hypothesis:

$$\begin{aligned} H_0 &: \mu_x = \mu_y \\ H_A &: \mu_x < \mu_y \end{aligned} \tag{2.24}$$

The null hypothesis H_0 asserts that there is no difference between the means of the two distributions, and extending the duration of the activity by F will be considered as not affecting the overall completion time. Let the above test be at level α, the reject region for this test is

$$\frac{\overline{Y} - \overline{X}}{\sigma\sqrt{2/n}} > z(\alpha) \tag{2.25}$$

Since the convergence of SA strongly depends on the L_{SGS} in *sa* procedure, different size projects may need different L_{SGS} to converge. To balance the performance and running time, SA may not converge well and the sample variance may be much larger than the given variance as ${k_B}^2T^2$ or ${T_e}^2$. Therefore we will use the sample variance in computing the reject region, where the sample variances are given as

$$s_x = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \overline{X})^2 \tag{2.26}$$

$$s_y = \frac{1}{n-1}\sum_{i=1}^{n}(Y_i - \overline{Y})^2 \tag{2.27}$$

The reject region with the sample variance is

$$\frac{\overline{Y} - \overline{X}}{\sqrt{({s_x}^2 + {s_y}^2)/n}} > z(\alpha) \tag{2.28}$$

With the help of the Testing Hypothesis, the following algorithm embedded with *SA* procedure can be used to calculate *H-Float*, where b is the activity selected for calculating float, and T_e is the ending temperature of *SA* procedure.

Algorithm 4 calculating *H-Float* with Testing Hypothesis

```
*th_float(A,P,R,a)*
**for** $i \leftarrow 1$ to $n$ **do**
    $X_i \leftarrow SA(A, P, R)$
**end for**
$\overline{X} \leftarrow \frac{1}{n}\sum_{i=1}^{n} X_i$
$s_x = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \overline{X})^2$
$hf \leftarrow$ UpBound$(A, P, R, X_1, a)$ , $lf \leftarrow 0$
**while** $lf < hf$ **do**
    $mf \leftarrow (hf + lf)/2 + 1$
    $d'(b) \leftarrow d(b) + mf$
    $A' \leftarrow A$
    update $d'(a)$ as the duration for activity a in $A'$
    **for** $i \leftarrow 1$ to $n$ **do**
        $Y_i \leftarrow SA(A', P, R)$
    **end for**
    $\overline{Y} \leftarrow \frac{1}{n}\sum_{i=1}^{n} Y_i$
    $s_y = \frac{1}{n-1}\sum_{i=1}^{n}(Y_i - \overline{Y})^2$
    **if** $\frac{\overline{Y}-\overline{X}}{\sqrt{({s_x}^2+{s_y}^2)/n}} < z(\alpha)$ **then**
        $lf \leftarrow mf$
    **else**
        $hf \leftarrow mf - 1$
    **end if**
**end while**
return $lf$
```

In the computational study, we will further discuss parameter selection in calculating *H-Float* by Testing Hypothesis. Using SA, calculating group float can be done in the same way as calculating float; the only difference is that for the revised instance, the duration of a set of the activities will be extended together instead.

2.4.3 Calculating *H-Float* by Simulated Annealing

With Theorem 2.4.3 and Theorem 2.4.4, *H-Float* also can be calculated directly with Simulated Annealing on the condition that the Simulated Annealing process converges well. With the binary search algorithm, almost the same as the *bab_float* algorithm, the *sa_float* algorithm is presented in the following. Here the original

Algorithm 5 calculating *H-Float* by Simulated Annealing directly

```
sa_float(A,P,R,a)
$D \leftarrow SA(A, P, R)$
$hf \leftarrow UpBound(A, P, R, D, a)$, $lf \leftarrow 0$
while $lf < hf$ do
  $mf \leftarrow (hf + lf)/2 + 1$
  $d'(a) \leftarrow d(a) + mf$
  $A' \leftarrow A$
  update $d'(a)$ as the duration for activity a in $A'$
  $D' \leftarrow SA(A', P, R)$
  if $D' \leq D$ then
    $lf \leftarrow mf$
  else
    $hf \leftarrow mf - 1$
  end if
end while
return $lf$
```

completion time D is obtained by SA. The algorithm will not change, if D is given as a predetermined parameter.

2.4.4 Calculating schedule based *S-Float*

Although we have redefined the concept of float for RCPSP, where the exact maximum float (*E-float*) can be calculated by the Branch and Bound Algorithm and the heuristic maximum float (*H-Float*) can be calculated by Test Hypothesis or Simulated Annealing, there is still a need to provide a direct method to calculate the maximum float based on a specified schedule, which is defined as *S-Float*. *S-Float* will not be calculated as the maximum amount of time the duration of an activity can be extended without affecting the start time of other activities in a specified schedule here, since it will be too strict to be useful in reality. Instead, we are interested in how much an activity can be extended considering both the left-justified and right-justified schedule for a given activity list without increasing the project completion time. Wiest [32] introduces the left-justified schedule as a feasible schedule in which no activity can be started at an earlier date by local left shifting of the activity alone. Accordingly, right-justified schedule is defined as a feasible schedule in which no activity can be ended at a later date by local right shifting. Valls et al. [116] used the iterative left-justified right-justified procedure to find better solutions for RCPSP. To calculate *S-Float*, we first find the left-justified schedule of the given activity list. After that, the activities will be scheduled as right-justified one by one with non-increasing finish time in the left-justified schedule. During this process, the *S-Float* of the activity is calculated as the maximum amount of time the activity could be extended just before the activity is scheduled right-justified. The detailed algorithm to calculate *S-Float* is presented in the following.

Algorithm 6 calculating *S-Float*

```
cal_sfloat(A,P,R, AL,D)
Find left-justified schedule for $AL$ and save the start time of activities to $st$
Set the finish time of activities as $fn$ according to $st$
Sort activities into $AL_r$ with no increasing of finish time $fn$ (ties are broken according to the label of activity)
for $i \leftarrow 1$ to n do
    $a \leftarrow AL_r[i]$
    Set the start time of activity $a$ as st(a)
    Extend activity $a$ as late as possible and record the maximum extension as $sfloat(a)$
    Set duration of $a$ to the old value
    Schedule activity $a$ as late as possible and save the start time to $st_r(a)$
end for
Return sfloat
```

In the above algorithm, the way it finds left-justified and right-justified schedule is similar to the famous SGS algorithm [66]. Same as the SGS algorithm, the time complexity for algorithm *cal_sfloat* is $O(r|A|^2)$, where $|A|$ is the number of activities and r is the number of resources.

The following useful fact plays a central role in speeding up the *th_float* algorithm to find *H-Float* in computational study.

Lemma 2.4.5 *Let S be one of the optimal schedules for the RCPSP, S-Float for an activity of the schedule S is the lower bound of E-float for the activity in the project for the optimal makespan.*

2.5 Finding critical sets and float sets

Like the float, group float can be calculated by Branch and Bound algorithm, Test Hypothesis and Simulated Annealing in the same way. The only difference is that all the activities in a given set have to be delayed together to check whether it is possible to find a new schedule with the old completion time. Therefore, small changes can be applied to the *bab_float*, *th_float* and *sa_float* as *bab_groupfloat*, *th_groupfloat* and *sa_groupfloat* to calculate the group float for a set of activities. The parameters for the three new procedures have changed to *(A,P,R,S,C)*, where S is the activity set and C is the delay vector we discussed in section 2.3. The details for the three new procedures are given in the appendix.

It is useful to find all the float sets and critical sets for a resource-constrained project with a given completion time, Since the float sets and critical sets explore the intrinsic nature of the project by clearly illustrating the flexibilities and relationships between activities. Although the number of float sets and critical sets may be exponential, $2^{|A|}$ and $\binom{|A|}{\lfloor \frac{|A|}{2} \rfloor}$ in the worst case, it is still possible for us to calculate float sets and critical sets for small size projects or for part of activities in large size projects. In practice, float sets and critical sets not only can help project managers select a suitable schedule to provide more buffer for a certain set of high priority activities but also they can help them to reallocate resources for the float sets and the critical sets to achieve a better schedule with shorter makespan. The *find_floatset* algorithm is developed to find all the float sets and critical sets for a resource-constrained project. The basic idea of the algorithm is to explore all the sets with only one activity first. Those activity sets with float larger than 0 will be recorded as float set, while for those activity sets with 0 float will be recorded as critical sets. After that, one activity will be added to all the float sets. If the maximum group float of a new set is larger than 0, the new set will be recorded as the float set. Otherwise, all the subsets of the new set with one activity less will be tested as float set or not. If there is subset which is not float set, the new set is neither float set nor critical set. If all the subsets with one activity less are float set, the new set will be critical set. Then the same process goes on, one activity will be added to all the float sets with 2 activities to explore the sets with 3 activities. The algorithm will keep on exploring the sets with k+1 activities as expansion of the float sets with k activities until there is no float sets with k activities. Similar approach has been applied by Neumann et al.[88] to identify the minimal forbidden set. The detailed *find_floatset* algorithm is given below.

The *find_floatset* algorithm will return *FS,CS* and *groupfloat*, where *FS* is the set of all float sets, *CS* is the set of all critical sets, *gf* is the function recording the float for all the float sets, *C* is the delay vector for the group float, here all the elements of *C* are set to 1. The *groupfloat* procedure can be any of the three procedures *bab_groupfloat*, *th_groupfloat* and *sa_groupfloat* to calculate float for a set of activities. Assuming the time complexity for the *groupfloat* procedure is $\mathcal{F}$, the time complexity

Algorithm 7 Finding all the float sets and critical sets

```
*find_floatset(A,P,R)*
$FS \leftarrow \emptyset$, $CS \leftarrow \emptyset$
**for** each $a \in A$ **do**
  $gf(\{a\}) \leftarrow sa\_float(A, P, R, a)$
  **if** $gf(\{a\}) > 0$ **then**
    $FS \leftarrow FS \bigcup \{\{a\}\}$
  **else**
    $CS \leftarrow CS \bigcup \{\{a\}\}$
  **end if**
**end for**
$FS_1 \leftarrow FS$
**while** $FS_1 \neq \emptyset$ **do**
  $FS_2 \leftarrow \emptyset$
  **for** each $S \in FS_1$ **do**
    **for** each $a \in A$ **do**
      $S_1 \leftarrow S \bigcup \{a\}$
      **if** $gf(\{a\}) > 0$ and $a \notin S$ and $S_1 \notin FS_2$ **then**
        $gf(S_1) \leftarrow groupfloat(A, P, R, S_1, C)$
        **if** $gf(S_1) > 0$ **then**
          $FS_2 \leftarrow FS_2 \bigcup \{S_1\}$
        **else**
          **if** $S_2 \in FS$, (for all $S_2 \subset S_1$ and $|S_2| = |S_1| - 1$) **then**
            $CS \leftarrow CS \bigcup \{S_1\}$
          **end if**
        **end if**
      **end if**
    **end for**
  **end for**
  $FS \leftarrow FS \bigcup FS_2$, $FS_1 \leftarrow FS_2$
**end while**
return $FS, CS, gf$
```

for the *find_floatset* algorithm will be $O(|A|2^{|A|}\mathcal{F})$. Although the problem to find all float sets and critical sets can not be solved in polynomial time, the *find_floatset* algorithm is quite efficient for small size projects. It can find all the float sets and critical sets for 216 test cases of the J30 test set with 30 activities and 4 resources using the *sa_groupfloat* procedure within 10000 seconds, where details can be found in section 2.8.5.

2.6 Float graph

In the last two sections, we have presented methods to calculate maximum float for an activity and for a set of activities. To give the project managers a direct view of the flexibility of activities inside the resource-constrained project, a float graph is proposed as an undirected graph illustrating float and group float as labels for vertices and edges.

2.6.1 Definition of float graph

The formal definition of float graph is given below.

Definition 2.6.1 (Float Graph) *An float graph $G = (V, E, f_v, f_e)$ is a undirected graph to demonstrate float and group float for a resource-constrained project, each $v \in V$ represents an activity in the project, there is an edge $(u, v) \in E$, if and only if the maximum group float for activity group {u,v} is larger than 0, $f_v : V \rightarrow \overline{Z^-}$ is the label function for vertices, each $v \in V$, $f_v(v)$ is defined as the maximum float of activity v*

in resource-constrained project, $f_e : E \rightarrow [(N_1, A_1, F_1), (N_2, A_2, F_2), ..., (N_m, A_m, F_m)]$ *is the label function for edges, each* $e \in E$*,* $f_e(e)$ *is defined as a sequence of 3-tuple* (A_i, N_i, F_i)*, each 3-tuple illustrate that e is included in a certain group of activities, with* N_i *is the group id,* A_i *is the number of activities inside the group,* F_i *is the maximum float of the group.*

As an example, the float graph for example 2.2 presented in section 2.1 is given in Figure 2.5, where every component of the float vector is set to 1.

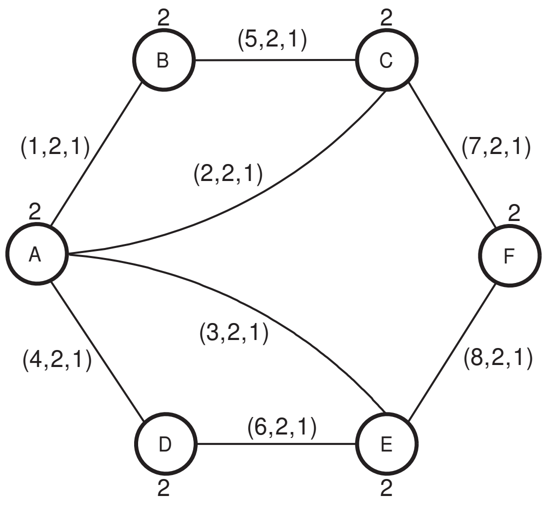


Figure 2.5 Float graph for example 2.2.

In Figure 2.5, it is clear that all the activities can be delayed by 2 units time alone without delaying the project, although the project completion time is minimum already. We also can find that with group float vector $(1, 1)$, the activity sets {A,B}, {A,C}, {A,D}, {A,E}, {B,C}, {C,F}, {D,E} and {E,F} have group float 1 each, which means both of the activities in each set can be delayed simultaneously by 1 time unit. However there are no edges connecting vertices A and F, B and D, B and E, B and F, C and D, C and E, D and F. According to the definition of float graph, the floats of activity sets {A,F}, {B,D}, {B,E}, {B,F}, {C,D}, {C,E} and {D,F} are all 0. Since none of the activities is critical, these 7 sets are critical sets. Although there are edges AB, BC and CA in float graph G, activities A,B and C can not be delayed together, as there is no 3-tuple on edge AB, BC and AC with same group id. Therefore the activity set {A,B,C} is a critical set. In the same way, {A,D,E} is also a critical set.

In all we can generalize the features of float graph G as:

- *float*

  The maximum float of the activity is illustrated as the label of the vertex in G.

- *float set*

  If a group of activities have maximum group float larger than 0, they must form a clique in G, where each edge in the clique has the same 3-tuple (N_k, A_k, F_k), with N_k as the identity of the current group, A_k as the number of activities inside the group, F_k as the float of the group.

- *critical activity*

  Critical activity is illustrated as the vertex with label 0, which is also the isolated vertex in G.

- *critical set*

  critical set is either an isolated vertex or two non-critical vertices with no edge between them or a clique in G. When it is a clique, no common 3-tuple can be found on the labels of all the edges in the clique. However, there is common 3-tuple for any sub graph of the clique.

Although there may be exponential number of float sets and critical sets ($2^{|A|}$ and $\binom{|A|}{\lfloor \frac{|A|}{2} \rfloor}$), for a resource-constrained project with fixed completion time, it is still possible for us to find all the float sets and critical sets for small size projects or for a subset of activities in large size projects. The *find_floatset* Algorithm proposed in section 2.5 is designed to find all the critical sets and float sets for a resource-constrained project, where detailed experimental results are given in section 2.8.5.

2.6.2 Managing resource-constrained projects with float graph

Float has played a central role in managing projects without resource limits. With the new definitions of float, group float, critical activity and critical set, it is possible for us to manage the project with resource limits more effectively. With the help of float graph, four possible ways to support the management for resource-constrained project are investigated here.

- *focusing on critical activities*

There are many activities inside a project. As a project manager only has limited energy to monitor them, focusing on the right activities is essential for the project to finish on time. With the new definition of float and critical activity, it is clear that the critical activity is critical in all the schedules, for extending the critical activity, there exists no schedule finishing before the original completion time. No matter what kind of schedule the project manager selects or changes, the critical activity can always serve as the focus for the project. To finish the project on time, the project manager has to guarantee the critical activities be finished on time. With the help of float graph, the critical activities can be easily identified as the vertices with label 0, which can certainly help the project manager to monitor them.

- *selecting proper schedule for high risk activities*

  In project management, there usually exist some high risk activities, where their completion time may be affected by some random events such as weather. As we have shown in the float graph of example 2.2, even when the makespan of the schedule is optimal, many activities still have float. However, the maximum float for different activities are achieved in different schedules. As an example, activity D, E can not be extended in the schedule given in Figure 2.4 (a). While either of activity D or E can be extended for 2 time units in the schedule given in Figure 2.4 (b). If activity D and E have higher risk than activity B and C, the later schedule will be favored. As the float of all the activities and all the sets of activities are illustrated in the float graph, the project manager can easily select the schedule to give a high risk activity or a set of high risk activities more float. In this way, the chance for the whole project to be finished in time is increased.

- *selecting proper completion time to buffer activities*

  As we have discussed, random events usually affect the completion of the project. Therefore, a specified schedule needs to be selected to give high risk activities more float. However, sometimes, selecting a good schedule may not give high risk activities enough buffers. In those cases, a new completion time should be selected to buffer the high risk activities. As an example, in Figure 2.5, the float graph of example 2.2, activity set {A,B,C} is a critical set. However, assuming the project manager wants to assign float {2,1,1} to them. A new completion time can be selected using binary search. The upper bound for the new completion time is 12 (8+2+1+1). Let the delay vector C for activity set {A,B,C} be (2,1,1), and group float be 1. Instead of using binary search, set the duration of activities A,B,C as 4,3,3, Branch and Bound or SA can be applied directly to find the minimum makspan, where the new completion time is calculated as 10. In some scenarios, the project manager may want to buffer all the activities. Assuming the project manager wants to give each activity in example 2.2 50% of its duration as its buffer, the delay vector C for the activity set {A,B,C,D,E,F} can be defined as (1,1,1,1,1,2) and the group float for it is 1. The upper bound for the new completion time is 15. The new completion time can be calculated directly as 12, since all the activities extended 50% and the minimum makespan will extend 50% also.

- *planning more resources for critical activities*

  In the projects without resource limits, CPM can be used to find the critical paths including all critical activities. The critical properties of activities are the results of precedence constraints. In scheduling projects with resource limits, the minimum makespan is usually larger than lower bound given by CPM. Assuming the minimum makespan is larger than lower bound by CPM for a resource-constrained project, if we remove all the resource constraints, it is easy to see that all the activities will have the maximum float larger than 0, which means there is no critical activity. Therefore, in those cases, the critical properties of the activities are mainly caused by the resource constraints. Planning more resources for critical activities will probably reduce the number of critical activities so that it may also reduce the minimum makespan of the project. For example 2.1 in Figure 2.2, if the amount of the resource is increased to 2 units, the makespan will be reduced to 3. For example 2.2 in Figure 2.4, if the amount of the resource is increased to 3 units, the makespan will be reduced to 6. Further research can be done to investigate resource consumption of critical activities so that more corresponding resources can be planned for critical activities to reduce the number of critical activities or to reduce the makespan of the project.

2.7 Negative float

2.7.1 Negative float and negative critical activity

Float demonstrates the flexibility to extend an activity or a set of activities. A corresponding schedule can be selected to give an activity specified amount of float so that extending the duration of the activity within the float will not affect the completion time of the project. In project management, sometimes, we are more interested in knowing how to reduce the completion time of the entire project through reducing the duration of an activity or a set of activities. In the definition of float and critical activity, we studied the effect on the completion time of the project through extending the duration of an activity. As a compensation for float, negative float and negative critical activity are defined here to show the effect on the makespan of the project through reducing the duration of an activity.

Definition 2.7.1 (Negative Float) *Negative Float of an activity in a resource-constrained project is defined as $-F$, where $F \in \overline{Z^-}$, and F represents the amount of time the duration of an activity can be reduced without affecting the completion time the project*

Here, not affecting the completion time of the project means the completion time of the project can not be reduced. Float is the flexibility of an activity to extend its duration, while negative float is the flexibility of an activity to reduce its duration. When reducing 1 unit duration of an activity results in reducing the completion time of the project, we say that the activity is a negative activity. When the duration of an activity is reduced to 0 and the completion time of the project still remains the same, the negative float of the activity will be recorded as $-\infty$. It is easy to see that the negative critical activity has minimum negative float as 0. With negative float, we can define negative critical activity as

Definition 2.7.2 (Negative Critical Activity) *In the resource-constrained project, an activity is defined as negative critical activity, if its minimum negative float is 0.*

The definition for negative critical activity is also applicable to the project without resource limits. When there are two critical paths, composed of different activities, reducing the duration of any critical activity on the two critical paths alone will not reduce the completion time of the project. Therefore, an activity being critical does not mean it is negative critical. Negative critical activity is also critical activity for the project without resource limits, since the negative critical activity is on the critical path. However, negative critical activity may not be critical activity for the project with resource limits. As an example, all the activities of example 2.2 in Figure 2.3 are negative critical, where reducing the duration of any of them will reduce the completion time of the project. However, all those activities have float 2 as shown in Figure 2.5.

Considering a set of activities, we also give the definitions for negative group float, negative critical set and negative float set as follows.

Definition 2.7.3 (Negative Group Float) *Negative group float is defined as the amount of time that the duration of a set of activities can be simultaneously reduced without change the completion time of the project.*

Similar to the delay vector, reduction vector B is proposed here. Assuming the group of activities to be measured are $U = \{a_{\mathtt{u1}}, a_{\mathtt{u2}}, ..., a_{\mathtt{un}}\}$, with duration given as a vector $W = (d_{\mathtt{u1}}, d_{\mathtt{u2}}, ..., d_{\mathtt{un}})$, the reduction vector B can be defined as, $B = (b_{\mathtt{u1}}, b_{\mathtt{u2}}, ..., b_{\mathtt{un}})$, with $b_{\mathtt{u}i} \in Z^+$ and $GCD(b_{\mathtt{u1}}, b_{\mathtt{u2}}, ..., b_{\mathtt{un}}) = 1$. With the reduction vector B, negative group float can be stated as $k \in \overline{Z^+}$. For the new duration vector $W' = W + kB$, the completion time of the project can not be reduced.

Definition 2.7.4 (Negative Critical Set) *In resource-constrained projects, negative critical set is a set of activities, where the minimum negative group float of the set of activities is 0, and there is no subset of it having the same property.*

Definition 2.7.5 (Negative Float Set) *In resource-constrained projects, negative float set is a set of activities, where the minimum negative group float of the set of activities is smaller than 0.*

All the algorithms to calculate float are applicable for negative float. The *find_floatset* algorithm also can be used to find all the negative float sets and negative critical sets.

2.7.2 Negative float graph

To clearly illustrate negative float, negative float graph is proposed here. The structure of negative float graph is similar to float graph, with a detailed definition given as

Definition 2.7.6 (Negative Float Graph) *A negative float graph $G' = (V', E', f'_v, f'_e)$ is an undirected graph to illustrate negative float and negative group float for a resource-constrained project, each $v \in V'$ represents an activity in the project, there is an edge $(u, v) \in E'$, if and only if the maximum negative group float for activity set $\{u,v\}$ is smaller than 0, $f'_v : V' \to \overline{Z^+}$ is the label function for vertices, each $v \in V'$, $f'_v(v)$ is defined as the minimum negative float of activity v in resource-constrained project, $f'_e : E' \to [(N'_1, A'_1, F'_1), (N'_2, A'_2, F'_2), ..., (N'_m, A'_m, F_m)']$ is the label function for edges, each $e \in E'$, $f'_e(e)$ is defined as a sequence of 3-tuple (A'_i, N'_i, F'_i), each 3-tuple illustrate that e is included in a certain group of activities, with N'_i is the group id, A'_i is the number of activities inside the group, F'_i is the minimum negative float of the group.*

As an example, the negative float graph of example 2.2 is shown in Figure 2.6. All the activities except activity A are negative critical activity, which means reducing the duration of the activity results in reducing the completion time of the project. The negative float is -1 for activity A in Figure 2.6. It is easy to see that when the duration of activity A is reduced by 2 units, the completion time will change to 6 instead. As only activity A is not negative critical, the negative float set is only {A}. Therefore there is no edge in Figure 2.6. The negative critical sets for example 2.2 are {B},{C},{D},{E},{F}.

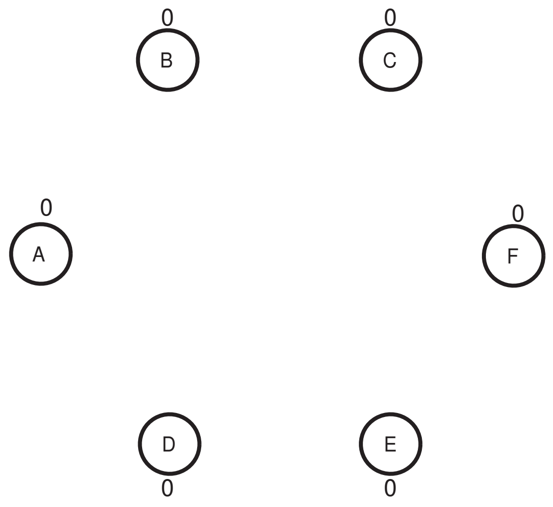


Figure 2.6 Negative float graph for example 2.2.

To demonstrate negative group float, example 2.3 is given here, where the precedence network for example 2.3 is shown in Figure 2.7.

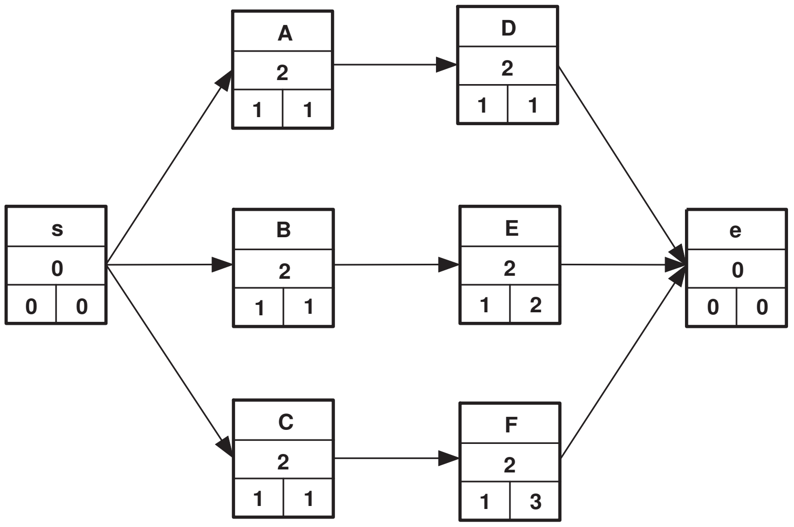


Figure 2.7 Precedence network for example 2.3.

In example 2.3, there is 1 renewable resource with the availability as 3 units. Activities A,B,C,D consume 1 unit resource each, and activity E, F consume 2 and 3 units of resource respectively. A schedule with completion time 6 is illustrated in Figure 2.8.

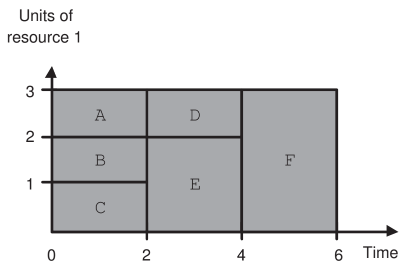


Figure 2.8 An optimal schedule for example 2.3.

Obviously, all the activities in example 2.3 are critical activities. However, only activity F is a negative critical activity. The negative float graph for example 2.3 is given in Figure 2.9.

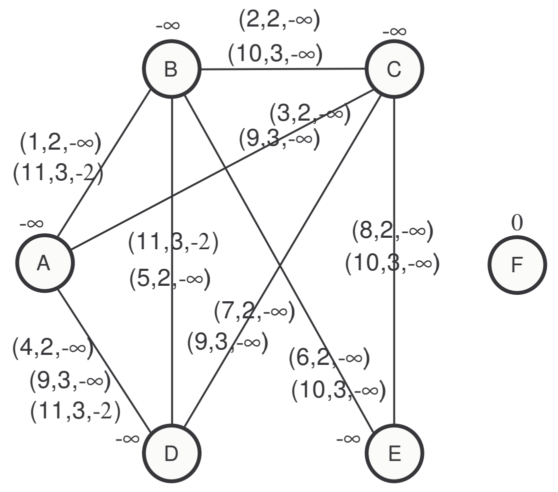


Figure 2.9 Negative float graph for example 2.3.

As shown in Figure 2.9, all the activities have $-\infty$ negative float except activity F, which means even reducing those activities duration to 0 alone can not reduce the completion time. Considering the activity sets with two activities, the activity sets {A,B}, {A,C}, {A,D}, {B,C}, {B,D}, {B,E} , {C,D} and {C,E} have negative group float as $-\infty$ also, while activity sets {A,E}, {D,E} are negative critical sets. For the activity sets with 3 activities, activity sets {A,C,D}, {B,C,E} have negative group float as $-\infty$, and activity set {A,B,D} has negative group float -2. Activity sets {A,B,C}, {B,C,D} are negative critical sets. There are no negative float sets or negative critical sets with more than 3 activities. In summary, the following rules are given to identify negative float, negative critical activity, negative float sets and negative critical sets in negative float graph.

- *negative float*

  The minimum negative float of the activity is illustrated as the label of the vertex in negative float graph G'.

- *negative float set*

  If a group of activities has minimum negative group float smaller than 0, they must form a clique in G, where each edge in the clique has the same 3-tuple (N'_k, A'_k, F'_k), with N'_k as the identity of the current group, A'_k as the number of activities inside the group, F'_k as the float of the group.

- *negative critical activity*

  Negative critical activity is illustrated as the vertex with label 0, which is also the isolated vertex in G'.

- *negative critical set*

  negative critical set is either an isolated vertex or two non-negative-critical vertex with no edge between them or a clique in G. When it is a clique, no common 3-tuple can be found on the labels of all the edges in the clique. However, there is common 3-tuple for any sub graph of the clique.

2.7.3 Zero critical activity

The minimum negative float and the maximum float form an interval for an activity. As an example, the float interval for the activities A,B,C,D,E and F in example 2.2 are [-1,2], [0,2], [0,2], [0,2], [0,2] and [0,2] respectively. The float interval demonstrates the flexibility of an activity in a project with resource limits. The float interval can also be interpreted as a sensitivity analysis of the duration of the activity under consideration the makespan of the project. Changing the duration of the activity within its float interval will not affect the completion time of the project. As we have discussed, the fact that an activity is critical does not mean it is negative critical, and an activity being negative critical does not mean it is critical also. Here, we have special interest in the activity which is critical and negative critical and we define it as zero critical activity, since the float interval for it is [0,0].

Definition 2.7.7 (Zero Critical Activity) *In resource-constrained projects, an activity is defined as zero critical activity if it is critical activity and negative critical activity, where its float interval is [0,0].*

Zero critical activity has a special property, where extending the duration of the zero critical activity will receive penalty of extending the project, reducing the duration of the activity will receive benefit of reducing the makespan of the project. Therefore special attention should be paid to the zero critical activities in project management. As an example, the activity F in example 2.3 is zero critical activity with float interval [0,0]. Increasing or decreasing the duration of activity F will affect the makespan of the project directly. Special effort should be made to make sure activity F finishes on time. The benefit of shorter makespan can be gained immediately, if the duration of activity F can be reduced.

In the next section, extensive experiments have been conducted to calculate float. As the calculation for negative float is very similar to the calculation of float, detailed experimental results will not be provided here.

2.8 Computational Study

The purpose of this research is to resolve the puzzle of float and critical activity for project networks with resource limits. With the new definitions for float, we are able to identify critical activities and critical sets for resource-constrained projects. Meanwhile the critical relationships between activities can be shown explicitly in a float graph, which will probably be a very useful tool for project planning. However, due to the NP-hard property of the Float-RCPSP problem, it is only possible for us to find the maximum float accurately (*E-float*) for small size projects. With the help of Simulated Annealing, we have also provided two algorithms to calculate float approximately (*H-Float*), which may be more useful in practice. The computational study is mainly designed to test how well the three algorithms perform in calculating float, and it is divided into 4 parts. We start by investigating the performance of the branch and bound algorithm for *E-float* on J30 test set of the PSPLIB [70], where there are 30 activities and 4 renewable resources for each test project. In section 2.8.2, we compare *E-float* with *H-Float* with extensive experiments, where parameter settings for the *th_float* approach will be discussed also. Then, a heuristic lower bound and probabilistic cut are used to improve the speed of *th_float* algorithm. After that the results from *sa_float* algorithm are presented in section 2.8.4. Finally, the *find_floatset* algorithm is used to find all the float sets and critical sets for the J30 test set in section 2.8.5.

All the experiments are carried out on Intel P4 2.8GHZ PC with 1G Memory, where the algorithms are implemented in C++, complied by GNU g++ with maximum optimization option. The PSPLIB [70], benchmark test sets for RCPSP, is used here to evaluate all the algorithms.

2.8.1 Experimental results on *E-float*

In section 2.4.1, algorithm *bab_float* embedding with the branch and bound procedure *dh_procedure* has been developed to find the exact value of maximum float (*E-float*). *dh_procedure* developed by Demeulemeester and Herroelen is still one of the best branch and bound algorithms for the RCPSP. We have implemented all the cutting techniques used in the first version of *dh_procedure* [39]. We also applied the newly developed lower bound, LB3, introduced by Mingozzi et al. [84], adopted in the second version of *dh_procedure* [40]. However, the *dh_procedure* still can not solve all the test cases in the J30 test set of PSPLIB in 1 hour CPU time, reported by Demeulemeester and Herroelen [40], although its speed had been improved a lot by the cutting techniques and lower bounds. In our test, since there are 30 activities inside the project and calculating float for each activity may result in calling the *dh_procedure* for several times, we have set the time limit for one test case as 10000 seconds. With this time limit, the floats of all the activities in 416 out of 480 test cases are calculated. We note that our *bab_float* procedure can be further speed up using the recently developed branch and bound method by Sprecher [106].

Table 2.1 *E-float* of the J30 test set

	E-Float		Crt-Act	Makespan	CPU
	case(periods)	act(periods)	(#)	(periods)	(sec)
avg	6.35	6.35	9.85	58.39	324.68
max	23.17	81	26	129	9075
min	0.37	0	0	35	0.17
dev	3.76	8.16	4.2	13.58	1200.12

Table 2.1 reports detailed results of float and critical activity on the J30 test set. The first two columns of table 1 show the case-based average *E-Float* value and activity based *E-Float* value, where case-based average is the average results of all the average float of each test case while activity based average is the average results considering floats of all the activities in all the test cases. The number of critical activities out of 30 activities, the makespan, and the CPU time are presented

in column 3-5, where the average, maximum, minimum and deviation for all those values are reported in detail. We can see that although the *E-Float* is reported as the exact maximum float for the optimal solution for RCPSP, the average *E-Float* for the activity is 6.35, more than 10 percent of the average makespan. We can also find that on average only around one third of activities are critical which means most activities have considerable flexibility in some schedules even when the solution is already optimal. Therefore, the new definition of float could help the project manager plan the project by selecting the schedule which gives some high risk activities more flexibility, so that the project will have higher chance to finish in time. The new float concept also helps us identify critical activity as the activity with 0 *E-Float* value. The number of critical activities has the average value 9.85 and ranges from 0 to 26 among 30 activities in the J30 test sets. Since the critical activities defined here are critical in any schedule, obviously, the critical activities should be among the main focus in project management, where extending the duration of critical activities will certainly result in the delay of the project.

As mentioned above, the algorithm *bab_float* is capable of finding floats for 416 out of 480 test cases for the J30 test set in 10000 seconds. Although the average CPU time is only 324.68 sec, the deviation for the CPU time is quite large, where CPU time ranged from 0.17 sec to 9075 sec. The reason is that the lower bounds and cutting techniques have different effect on different cases, which also explains why the *bab_float* could not solve all the test cases in 10000 seconds.

2.8.2 Experimental results on *H-Float* by *th_float*

In section 2.4.2, based on Theorem 2.4.3 and Theorem 2.4.4, algorithm *th_float*is proposed to find *H-Float* as an approximation for the *E-float*. The performance of the *th_float* algorithm depends on the selection of various parameters, such as ending temperature, the SGS limit for Simulated Annealing and the sample size, sample level α for the Testing Hypothesis. Extensive experiments have been done here to find a suitable parameter setting to achieve high accuracy in approximating *E-Float* with acceptable running time. Since the duration of activities and the makespan of the project are all integer value here, it is reasonable to change the hypothesis to

$$H_0 : \mu_x + 0.5 \;=\; \mu_y \tag{2.29}$$

$$H_A : \mu_x + 0.5 \;<\; \mu_y \tag{2.30}$$

Table 2.2 compares the new Testing Hypothesis to the old one on the J30 test set. In this experiment, $\mathtt{T_e}$ is set to $-1/\ln(0.0001)$, the sample size is set to 50 and the sample level α is set to 0.025. The maximum step for Simulated Annealing is set to be 5000 and 50000. The accuracy and CPU time are reported respectively in Table 2.2. The accuracy in Table 2.2 shows the percent of activities whose *H-Float* are the same as *E-Float* according to the 416 test cases reported in Table 2.1. It is clearly shown in Table 2.2 that the new Hypothesis outperforms the old one. We also can find in Table 2.2, with the new Hypothesis, 90% of activities have the same *H-Float* as *E-Float* value for 5000 SGS calls and 98% of activities have same *H-Float* as *E-float* value for 50000 SGS calls, which highlights the effect of using *H-Float* obtained by the *th_float* algorithm as an approximation for *E-Float*.

Table 2.2 New Testing Hypothesis

	5000		50000	
TH	accuracy(%)	CPU(sec)	accuracy(%)	CPU(sec)
old_TH	89.89	213.41	96.15	2311.59
new_TH	90.06	281.05	98.43	2501.28

According to Theorem 2.4.4, even without the temperature approaching 0, we can still get $E(X) - E(Y) = x_{min} - y_{min}$. Therefore the Testing Hypothesis still can be used to obtain *H-Float*. Table 2.3 reports the results for some different ending Temperature $\mathtt{T_e}$. In Table 2.3, if the ending temperature is set to be $\mathtt{T_0}$, which means the temperature does not decrease, *H-Float* still has 91% accuracy for 50000 SGS calls. This has verified Theorem 2.4.4. However, the results of setting $\mathtt{T_e}$ to $\mathtt{T_0}$ and $\mathtt{T_0/4}$ are worse than the results of setting $\mathtt{T_e}$ to a near 0 value $-1/\ln(0.0001)$. There are two reasons for this phenomenon. First, when the temperature does not change, it becomes the Metropolis Algorithm where Catoni [24] has proved that the Simulated Annealing converges faster than the Metropolis Algorithm. Secondly, Theorem 2.4.4 has given the variance for a random variable on the project makespan as $\mathtt{T_e}^2$. Since the smaller the variance, the larger the power for the Testing Hypothesis [102], Setting

$\mathtt{T_e}$ to a near 0 value makes the Testing Hypothesis more accurate.

Table 2.3 Results for different ending temperature $\mathtt{T_e}$

	5000		50000	
$\mathtt{T_e}$	accuracy(%)	CPU(sec)	accuracy(%)	CPU(sec)
$-1/\ln(0.0001)$	90.06	281.05	98.43	2501.28
T_0	85.41	295.03	91.13	2559.96
$T_0/4$	88.77	282.48	95.91	3041.51

As the power for the Testing Hypothesis is also determined by the significance level α, experiments are conducted to find a proper significance level α for the Testing Hypothesis. Three α value are considered here. The results are presented in Table 2.4. In Table 2.4, it is clear that 0.025 is the best choice for significance level α .

Table 2.4 Results for different sample level α

	5000		50000	
α	accuracy(%)	CPU(sec)	accuracy(%)	CPU(sec)
0.01	89.63	299.13	96.70	2353.02
0.025	90.06	281.05	98.43	2501.28
0.05	88.65	267.09	95.79	2696.33

2.8.3 Speed up *th_float* algorithm

The above experiments show that the *H-Float* found by the *th_float* algorithm is 98% same as the *E-Float* found by *bab_float*. Unlike the *bab_float*, *th_float* finishes running in polynomial time and it can find *H-Float* for all the test cases of the J30 test set. However, the *th_float* algorithm is still quite slow as it needs around 2500 seconds to find *H-Float* for each test case when SGS limit is set to 50000 and the sample size is set to 50. A heuristic lower bound and probabilistic cut are proposed here to improve the speed of the *th_float* algorithm. According to Lemma 2.4.5, *S-Float* for any optimal schedule can be the lower bound for the *E-float*. Assuming the Simulated Annealing converges well to the optimal solutions, in all the sample *SA* solutions, the *S-Float* of the solutions with the minimum makespan will be used as the heuristic lower bound for *H-Float*. Since the solutions obtained by *SA* have been proved to converge to the Boltzmann distribution, applying the continuous form of Boltzmann distribution, the probability that a sample solution for the revised project with the duration of some activity extended is smaller or bigger than the mean sample value is given as:

$$\begin{aligned} P(Y_i < \mu_y) &= \int_{y_{min}}^{y_{min}+k_B T} \frac{e^{-(y-y_{min})/k_B T}}{k_B T} && (2.31) \\ &= 1 - \frac{1}{e} && (2.32) \\ P(Y_i > \mu_y) &= 1 - P(y_i < \mu_y) && (2.33) \\ &= \frac{1}{e} && (2.34) \end{aligned}$$

Therefore, if the *SA* converges well, the probability for all 7 sample solutions larger than μ_y is around 0.001 and the probability for all 15 sample solutions smaller than μ_y is around 0.001 also. In other words, the probability for the smallest solution from 7 sample solutions smaller than μ_y is 0.999 and the probability for the largest solution from 15 sample solutions larger than μ_y is 0.999 also. This result is presented formally as follows:

$$Y_{min} = \min_{1 \leq i \leq 7} Y_i \tag{2.35}$$

$$Y_{max} = \max_{1 \leq i \leq 15} Y_i \tag{2.36}$$

$$P(Y_{min} < \mu_y) = 1 - (\frac{1}{e})^7 \tag{2.37}$$

$$\approx 0.999 \tag{2.38}$$

$$P(Y_{max} > \mu_y) = 1 - (1 - \frac{1}{e})^7 \tag{2.39}$$

$$\approx 0.999 \tag{2.40}$$

The above results can be used to form the probabilistic cut. Because the probability for y_{min} less than μ_y is 0.999, if $\frac{Y_{min}-\overline{X}-0.5}{\sqrt{({s_x}^2+{s_y}^2)/n}} > z(\alpha)$, we will have the chance for $\frac{\mu_y-\overline{X}-0.5}{\sqrt{({s_x}^2+{s_y}^2)/n}} > z(\alpha)$ to be larger than 0.999. In this case, the null hypothesis H_0 will be rejected, since the accuracy here is much higher than the Testing Hypothesis itself already. In the same way, $\frac{Y_{max}-\overline{X}-0.5}{\sqrt{({s_x}^2+{s_y}^2)/n}} < z(\alpha)$ leads to $\frac{\mu_y-\overline{X}-0.5}{\sqrt{({s_x}^2+{s_y}^2)/n}} < z(\alpha)$, H_0 will be accepted. The probabilistic cut is performed when there are 7 sample solutions and 15 sample solutions respectively. When *SA* converges well, there is no need to finish all the sample runs, which certainly saves a lot of computational time. Embedding with the heuristic lower bound and probabilistic cut, the details of speeding up algorithm *th_float2* are given in the appendix.

With the help of the new heuristic lower bound and probabilistic cut, the *th_float2* is about 10 times faster than the old *th_float* algorithm on the J30 test set. The result for the new *th_float2* is reported in the first row of Table 2.5, where the accuracy is improved also for 5000 SGS calls. In Table 2.5, various sample sizes are also compared. For 5000 SGS calls, the accuracy improved significantly as the sample size increased. Since *SA* does not converge well for 5000 steps, increasing the sample size has increased the power of the Testing Hypothesis. This also explains that there are only slight differences between the results of various sample sizes for 50000 SGS calls, where *SA* converges well.

Table 2.5 Results for different sample size

	5000		50000	
sample size	accuracy(%)	CPU(sec)	accuracy(%)	CPU(sec)
50	92.60	25.9	98.45	160.70
100	93.97	34.85	98.82	178.62
200	94.78	46.28	98.85	226.15

According to all the preliminary experiments, to balance the performance and computation time, the ending temperature is set to $-1/\ln(0.0001)$, the sample level α is set to 0.025 and the sample size is set to 50. The detailed results for *th_float2* algorithm on the J30 test set are given in Table 2.6, where the average of *H-Float* value, number of critical activity (Crt-Act), accuracy, makespan mean and deviation and CPU time for each test case are reported. The maximum, minimum value and the deviation for these features can also be found. Like the *E-float*, there are more than 6 periods *H-Float* for each activity on average Meanwhile, the average number of critical activities detected by *th_float2* is also around 9.

Table 2.6 Results of *H-Float* by *th_float2* for the J30 test set

No. of Step		*H-Float* (periods)	No. of Crt-Act	Accuracy (%)	Makespan avg	Makespan dev	CPU (sec)
5000	avg	6.48	9.03	92.60	58.50	0.12	25.90
	max	23.17	26	100.00	129	2.02	173.15
	min	0.37	0	0.00	35	0.00	1.57
	dev	3.65	3.82	19.84	13.70	0.30	26.33
50000	avg	6.38	9.71	98.45	58.41	0.03	160.70
	max	23.17	27	100.00	129	0.91	1037.15
	min	0.37	0	0.00	35	0.00	6.68
	dev	3.73	4.13	9.04	13.61	0.12	105.84
500000	avg	6.35	9.86	99.82	58.39	0.01	1469.19
	max	23.17	27	100.00	129	0.78	6144.47
	min	0.37	0	76.67	35	0.00	6.68
	dev	3.76	4.25	1.75	13.58	0.05	743.13

In Table 2.6, we also can find that the accuracy increases as the number of steps increases. Although the accuracy for 5000 and 50000 SGS calls are more than 92% and 98%, there are some cases in which the *SA* does not converge well resulting in the minimum accuracy being 0. Therefore, the convergence of *SA* not only depends on how many steps of *SA* there are but also depends on the test case. This is a common problem for the heuristics - they can find high quality solutions, but there is no worst case guarantee. Fortunately, how well *SA* converges can be inferred by the makespan deviation for the sample solutions. A smaller makespan deviation indicates a better *SA* convergence, where we can find when the steps of *SA* increase the average makespan deviation decreases. Although the *bab_float* can not find all the *E-Float* for the J30 test set, it is still possible for us to find the *H-Float* for larger size projects. The *H-Float* found by the *th_float2* algorithm for the J60 test set with 60 activities and 4 resources is shown in Table 2.7. Since there is no *E-float* to compare, the *H-Float* value achieved by 500000 steps of *SA* is used here as a standard. The accuracy now does not have the old meanings, since the error may be doubled using the *H-Float* itself as the standard. However, the trend is still clear that more steps of *SA* leads to less makespan deviation and more same *H-Float* value as the 500000 steps one.

In the *SA* procedure here, we used the Serial Schedule Generation Scheme (SGS)[66], and the time complexity for the *SA* algorithm presented in section 2.4.2 is $O(m|A|^2r)$, where m is the number of steps, $|A|$ is the number of activities and r is the number of resources. with the time complexity for *SA*, we can easily get the time complexity for *th_float2* as $O(nm|A|^3r)$, where n is the number of solutions sampled. Therefore, theoretically, the running time of the J60 test set should be around 8 times as much as the running of the J30 test set for the same *SA* steps. While the actual running time of the J60 test set is more than 10 times that of the J30 test set. This can be explained by the convergence of *SA* also. It is obviously that *SA* converges better for the J30 test set for same steps. Because the probabilistic cut depends on how well *SA* converges, the probabilistic cut works better on the J30 test set than on the J60 test set. The *th_float2* algorithm is still applicable to large size projects, since it has polynomial time complexity and it can be easily implemented on parallel machines to reduce the running time.

Table 2.7 Results of *H-Float* by *th_float2* for the J60 test set

No. of Step		*H-Float* (periods)	No. of Crt-Act	Accuracy (%)	Makespan avg	Makespan dev	CPU (sec)
5000	avg	11.11	8.90	73.82	80.94	0.39	490.66
	max	38.32	23.00	100.00	159.40	2.49	6656.23
	min	1.42	0.00	0.00	44.00	0.00	26.89
	dev	6.21	5.47	33.77	18.82	0.61	875.93
50000	avg	11.13	9.52	81.41	80.42	0.24	4267.62
	max	38.32	27.00	100.00	157.20	2.49	65373.23
	min	1.57	0.00	1.67	44.00	0.00	34.56
	dev	6.34	5.34	27.91	18.23	0.44	8457.95
500000	avg	11.07	10.39	100.00	80.08	0.12	31108.39
	max	38.32	35.00	100.00	156.06	1.23	455824.01
	min	1.10	0.00	100.00	44.00	0.00	2394.92
	dev	6.48	5.26	0.00	17.80	0.24	53215.40

2.8.4 Experimental results on ***H-Float*** by ***sa_float***

In Tables 2.8 and 2.9, the experimental results on *H-Float* for the J30 test set by *sa_float* are reported. Since there is only one run for *SA* to obtain the makespan, the deviation of makespan in Table 2.8 represents the deviation of the makespan to the optimal makespan and the deviation of makespan for the J60 test set is neglected. We can find in Table 2.8 and 2.9 that the accuracy of *sa_float* is worse than the accuracy of *th_float2* for same steps of *SA*, which certifies the power of the Testing Hypothesis. However, the *sa_float* is much faster than the *th_float2* due to the fact that only one sample solution is used in *sa_float*. Therefore, for large size projects, *sa_float* will be a better choice.

Table 2.8 Results of *H-Float* by *sa_float* for the J30 test set

No. of Step		*H-Float* (periods)	No. of Crt-Act	Accuracy (%)	Makespan avg	Makespan dev(%)	CPU (sec)
5000	avg	6.33	9.87	88.42	58.50	0.16	3.85
	max	23.17	27.00	100.00	129.00	8.51	76.85
	min	0.10	0.00	0.00	35.00	0.00	0.59
	dev	3.82	4.53	23.71	13.67	0.70	3.82
50000	avg	6.32	9.96	97.30	58.40	0.02	39.96
	max	23.17	28.00	100.00	129.00	2.33	76.98
	min	0.23	0.00	0.00	35.00	0.00	6.68
	dev	3.79	4.38	9.88	13.60	0.17	12.18
500000	avg	6.35	9.87	99.66	58.39	0.00	372.83
	max	23.17	27.00	100.00	129.00	0.55	791.46
	min	0.37	0.00	76.67	35.00	0.00	6.68
	dev	3.76	4.26	2.17	13.58	0.03	113.42

Table 2.9 Results of *H-Float* by *sa_float* for the J60 test set

No. of Step		*H-Float* (periods)	No. of Crt-Act	Accuracy (%)	Makespan avg(periods)	CPU (sec)
5000	avg	10.69	16.18	66.97	80.96	25.10
	max	38.32	60.00	100.00	159.00	182.46
	min	0.00	0.00	0.00	44.00	4.75
	dev	6.77	11.29	39.12	18.80	17.27
50000	avg	10.83	15.30	74.42	80.38	253.16
	max	38.32	58.00	100.00	155.00	1277.00
	min	0.03	0.00	0.00	44.00	60.52
	dev	6.76	10.72	36.79	18.15	123.35
500000	avg	10.80	14.80	82.50	80.06	2549.31
	max	38.32	59.00	100.00	155.00	12507.27
	min	0.02	0.00	0.00	44.00	627.04
	dev	6.82	9.85	30.15	17.74	1215.81

In Table 2.10, the experimental results for the J30 test set by *sa_float* with fixed completion time are presented, where we use the makespan of the optimal solution as the completion time, instead of using the makespan obtained by the *SA*. The purpose of this experiment is to demonstrate that the better *SA* converges, the larger the average *H-Float* values and the smaller the number of critical activities it will find, which is demonstrated by the results in Table 2.10. In fact, there are many other heuristics which may be applied to find the *H-Float*, and the performance of the heuristics in calculating *H-Float* depends on how well they converge to the optimal solutions.

Table 2.10 *H-Float* by *sa_float* with optimal completion time for the J30 test set

No. of Step		*H-Float* (periods)	No. of Crt-Act	Accuracy (%)	Makespan avg(periods)	CPU (sec)
5000	avg	6.08	11.28	91.63	58.39	4.00
	max	23.17	30.00	100.00	129.00	76.85
	min	0.00	0.00	13.33	35.00	0.59
	dev	4.02	5.80	15.76	13.58	3.85
50000	avg	6.29	10.13	97.78	58.39	40.83
	max	23.17	27.00	100.00	129.00	77.58
	min	0.13	0.00	43.33	35.00	6.03
	dev	3.82	4.58	6.75	13.58	12.00
500000	avg	6.35	9.88	99.60	58.39	415.65
	max	23.17	27.00	100.00	129.00	787.14
	min	0.37	0.00	76.67	35.00	6.68
	dev	3.76	4.27	2.41	13.58	121.75

2.8.5 Experimental results on float sets and critical sets

With the *find_floatset* algorithm, it is possible for us to find all the float sets and critical sets for small size projects with limited resources. As the number of float sets and critical sets may be exponential, *find_floatset* algorithm is only capable of finding all float sets and critical sets for 216 out of 480 test cases for the J30 test set in 10000 seconds reported in Table 2.11, where the partial results of the unfinished cases are included also. To save computation time, the *SAGroupFloat* algorithm has been chosen to find group float, where the makespan of the optimal solution is used as the original completion time.

All the results shown in Table 2.11 are case-based. We can find that there are

Table 2.11 Results of float set and critical set for the J30 test set

	float set			critical set		CPU
	No. of Sets	Size	*H-Float*(periods)	No. of Set	Size	(sec)
avg	102330.43	5.24	2.40	174.59	2.36	6163.90
max	1233355	8.91	13.67	3022.00	7.19	10000
min	0	1	0.00	6	1	0.00
dev	113382.84	1.96	1.65	409.73	1.62	4465.28

102330 float sets on average, while there are only 174 critical sets on average. We also can find that the average group float is 2.40 and the average float set size is 5.24. All those results have demonstrated there is still a lot of flexibility even in the projects with optimal completion times. Moreover, all the flexibility and relationships between activities can be clearly shown in the float graph proposed in section 2.6.

2.9 Conclusion

In this chapter, new definitions for float and critical activity have been given. Several algorithms have also been developed to calculate float and identify the critical activities. Extending the float definition to a set of activities as group float, the float set is defined as the set of activities with larger than 0 group float, and the critical set is defined as the activity set with 0 maximum group float. Negative float is also proposed in this research to investigate the effect on the minimum makespan of the project through reducing the duration of its activity. Similarly, negative critical activity, negative group float, negative float set and negative critical set are defined and studied. With the help of the float graph and negative float graph, the flexibility and relationships between activities are clearly illustrated.

Extensive experiments have been conducted to compare the results of the three algorithms in calculating floats. The *bab_float* algorithm can find the exact value for float (*E-float*), but it needs exponential time to finish running. The *H-Float* calculated by *th_float* and *sa_float* have accurately approximated the *E-float*, and both of them can finish running in polynomial time. Although the *H-Float* from *th_float* approximates *E-float* slightly better than the *sa_float*, the *sa_float* is much faster than the *th_float*. Therefore *sa_float* is recommended for large size projects. Embedding with the *sa_float* algorithm, the *find_floatset* algorithm has been proposed to find all the float sets and critical sets for small size projects or part of the large size projects, where experiments have been performed on the J30 test set to demonstrate that there is still a great deal of flexibility between activities even when the completion time is optimal already.

The new concepts of float and critical activity will help the project manager to further understand the intrinsic of real-life resource-constrained projects in order to plan and manage the projects in a better way.

According to the experiments, the better the heuristic converges to the optimal solution, the larger the float it may detect. Future research may be conducted to investigate the performance of other heuristics on float calculation. More research may also be done to further explore the properties of the float graph or to provide more ways to apply it to project management.

Chapter 3
Molecular Search for Resource-Constrained Project Scheduling Problem

All matter of our world is made up of molecules and atoms. The different physical properties (color, smell, density, etc.) of the matter come from the different combination of molecules and atoms, the motion of them and the interaction between them. The motion of molecules and atoms and the interaction between them are highly related to the internal energy of the molecules. There are mainly two types of internal energy, kinetic energy and potential energy. Kinetic energy is reserved in the motion of the molecules, where temperature is a measure of it [118]. Potential energy is mainly reserved in the positions of the molecules.

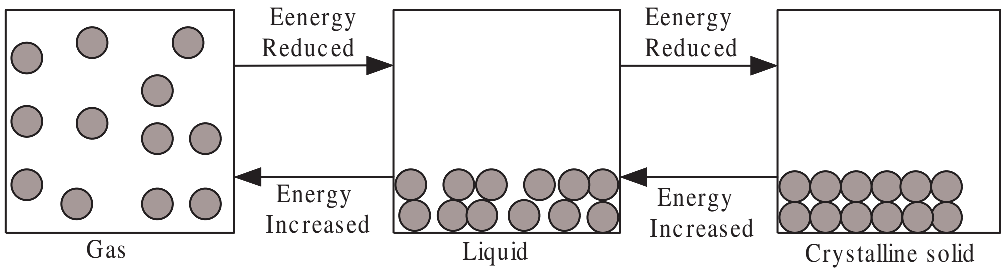


Figure 3.1 Three phases of matter

Regarding a certain range of temperature and pressure, there are three basic phases of matter - gas, liquid and solid. For the gas phase, when the temperature is high, the kinetic energy of molecules and atoms is high enough to overcome the attractive force between them, so that they can move around freely to fill the entire container as shown in Figure 3.1. Cooling the system, gas transforms into liquid, where molecules and atoms can still slide pass one another while remain loosely bound. Therefore, liquid has a definite volume but it takes the shape of the container. At still lower temperature, the average kinetic energy approaching 0, liquid transforms into solid. In the solid phase, the molecules and atoms stick together to minimize the potential energy, hence solid has a definite shape and volume. There are two types of solids, crystalline solids and amorphous solids. The crystalline solids are formed by highly ordered structure of molecules and atoms. As an example, the beautiful structures of snow crystal captured by Kenneth G. Libbrecht using a specially designed snowflake photomicroscope [74] can be found in Figure 3.2. The amorphous solids, such as glass, are usually formed by rapidly cooling, so that there is not enough time for them to crystallize. The solids referred in this research are crystals, which have highly ordered structures.

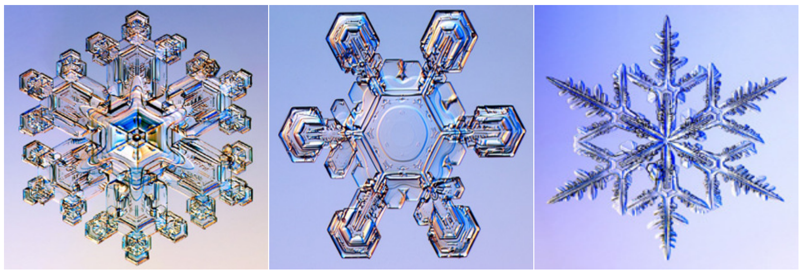


Figure 3.2 Snow crystal by Kenneth G. Libbrecht

The beautiful structures of crystals are formed by the concurrent motion of molecules and atoms to achieve the most stable structure which has the lowest global potential energy. Through cooling gas into liquid and cooling liquid into crystal, the molecules change from a highly disordered state to an highly ordered state. As a measure of disorder of the system, entropy, symbolized by S, is defined by the differential quantity of $\delta S = \delta Q/T$, where δQ is the amount of heat absorbed. When cooling the system, δQ is negative so that δS is negative also. Therefore when cooling the system or the internal energy of the system decreases, the entropy decreases also, which means the order of the system increases.

The whole cooling process can be viewed as an optimization process to achieve the lowest global potential energy, where nature solves it in an intuitive way through reducing the average kinetic energy, and letting all of the molecules move together according to the force between them until they reach the most stable position. This enlightens us to simulate the cooling process in solving combinatorial optimization problems.

Simulating the concurrent motion of molecules and atoms in the cooling process, a new optimization method named Molecular Search is proposed to solve resource-constrained project scheduling problem (RCPSP) in this chapter. The ideal model of Molecule Search for a combinatorial optimization problem is given below.

In the Ideal Molecular Search, a solution of the combinatorial optimization prob-

Algorithm 8 Ideal Molecular Search

```
1. Encode a solution of the combinatorial optimization problem as the order or positions of molecules
2. Define the force (rule) to guide the motion of molecules and set the initial value of environment temperature $T$
3. Simulate motion of the molecules according to the force (rule)
4. Decode order or positions of molecules as the solution, and recording the best solution
5. Reduce the environment temperature $T$
6. if $T > T_e$ goto 3
```

lem has to be encoded as the order or positions of molecules. Since each different optimization problem has a different objective function, the force (rule) to guide the motion of the molecules has to be defined for the specified objective function. The initial environment temperature has to be set first. After the initialization, the cooling process is simulated iteratively. In each iteration of Ideal Molecule Search, new order or positions of the molecules are calculated according to the force (rule), and they are decoded as a new solution. The new solution will be recorded as the best solution, even if it is better than the old best solution. The environment temperature decreases in each iteration. The whole simulation process ends when the environment temperature T reach T_e, or in other words, the matter has changed to crystal. We have not defined the speed or the distribution of the kinetic energy of the molecules, because the force (rule) is defined to lead the molecules to the order or positions decoded as optimal or near optimal solutions, and it may not be the physical force at all. Therefore, the motion of the molecules is decided by the force (rule) directly, instead of their speed.

3.1 Molecular Search

The key question in the Ideal Molecular Search is how to define the force (rule) to guide the molecules to the best positions representing optimal solutions. Unfortunately it is difficult to find suitable forces (rules) for different combinatorial optimization problems. The difficulties come from varieties of the objective functions. Moreover, crystal has the optimal substructure property, while the NP-hard combinatorial problems do not have such property, otherwise they can be solved in polynomial time. To achieve the global optimal solutions, the force (rule) has to guide part of molecules to bad positions for some substructures .

As discussed above, there still lacks a set of perfect rules to guide the molecules. To simplify the molecule search procedure, the motion of the molecules has been divided into two types: jumping and walking, where the motion of molecules with high kinetic energy is named as jumping and the motion of molecules with low kinetic energy is named walking.

The jumping process is simulated as the concurrent motion of all the high kinetic energy molecules according to the jumping rules. In thermodynamics, the kinetic energy of the molecules satisfies Maxwell-Boltzmann distribution [118]. According to the Maxwell-Boltzmann distribution given below, there are only a small potion of the molecules with high kinetic energy and the number of molecules with high kinetic energy decreases when average kinetic energy decreases. Therefore the number of jumping molecules usually decreases with the decreasing of temperature, except for the phase transition periods. There are two major procedures in the cooling process, one is decreasing temperature of the matter, the other is the phase transition of the matter. In the decreasing temperature process, the average kinetic energy of the molecules decreases. As a result, the number of jumping molecules decreases. In the phase transition process, only the potential energy of the molecules decreases and the kinetic energy of the molecules does not change. Therefore the number of jumping molecules does not decrease also. Instead of simulating the entire process, the jumping rule may be made to simulate one of the two process, where different jumping rules for RCPSP can be found in section 3.4.

$$f(E) = \frac{e^{-E/k_B T}}{k_B T} \tag{3.1}$$

where k_B is the Boltzmann constant and T is the temperature.

To reduce the chaos caused by concurrent motion of molecules, the walking process is simulated as a sequence moves of lower energy molecules, where the global energy is checked before each move to ensure that the move leads to an equal or better solution. The walking process can be considered as the classic local search procedure. The Molecular Search algorithm with molecule jumping and walking is shown below.

In the Molecular Search Algorithm, ML is a molecule list used to representing a solution for the combinatorial optimization problem, where molecules are ordered

Algorithm 9 Molecular Search

```
Generate initial $ML$, $T \leftarrow T_0$
Define jumping rule and walking rule
**while** $T > T_e$ **do**
    $molecule\_jumping(ML, T)$
    $molecule\_walking(ML, T)$
    **if** $f(ML) < f(ML_{best})$ **then**
        $ML_{best} \leftarrow ML$
    **end if**
    $T \leftarrow \alpha \cdot T$
**end while**
```

in a linear sequence. T_0 is the initial temperature. f is the function to evaluate the molecule list, and here we assume the objective function is to minimize $f(ML)$. The best solution so far is saved in S_{best}. The Molecule Search Algorithm first generates an initial solution presented as molecule list ML. After that the cooling process is simulated by molecule jumping and molecule walking until the environment temperature reaches T_e. The temperature decreases in each loop as $T \leftarrow \alpha \cdot T$, where $\alpha (0 < \alpha < 1)$ is a predefined constant.

Although the model of the Molecular Search is given here for the first time, the Molecular Search has already been successfully applied to Bandwidth Minimization Problem [81, 80], Minimum Linear Arrangement Problem [77], where benchmark results are achieved in a short running time.

3.2 Resource-constrained project scheduling problem

Resource-constrained project scheduling problem (RCPSP) considers scheduling a set of activities $A = 1, 2, ..., n$, where the duration of activity j is given as $d(j)$.

There are two types of constraints in RCPSP. First, the precedence constraints force an activity j start only after all its immediate predecessors ($predecessor(j)$) finished. Second, there are K different renewable resources, where the capacity for resource k is R_k. The activity j needs $r_{j,k}$ resource K for all the time units while it is being processed. The objective for the RCPSP is to find a precedence and resource feasible schedule so that the makespan of the project is minimized, where the makespan of the project can be defined as the maximum completion time of all the activities. An example of RCPSP is given as example 3.1. Assuming the finish time of activity is $f(j)$, the conceptual decision model of RCPSP is given as [28, 66]

In the above model, $S(t)$ is the set of activities which is processed at time t. The objective function (3.2) minimizes the makespan of the project, which is counted as the maximum finish time of all the activities. (3.3), (3.4) enforce precedence constraints and resource constraints respectively. Finally, (3.5) states that none of the activities can start before time 0.

A lot of algorithms have been developed to tackle RCPSP, such as Branch and Bound methods [39, 40, 21, 106, 20], constraint-propagation-based cutting planes [36], heuristics X-pass methods [30, 33, 65], Tabu Search [5, 90], Simulated Annealing [12, 14] and Genetic Algorithm [52, 62, 57, 114]. A detailed survey on the methods for resource-constrained project scheduling problem can be found in [41, 68, 67], where a standard benchmark is used to evaluate different methods [69].

For RCPSP, the molecule list is the same as the traditional activity list, where each activity is mapped to a molecule and the solution is obtained by scheduling each activity (molecule) one by one as earlier as possible according to the corresponding order of molecules in the molecule list with SGS (Serial Generation Scheme) [66].

Detailed jumping rules for the RCPSP will be discussed in section . A new forward-backward search (FBS) procedure is proposed as molecule walking in section 3.5. Simulated Annealing Algorithm simulates the annealing process, which looks like the Molecular Search. However, the two procedures are quite different. Simulated Annealing is based on energy distribution (Boltzmann Distribution) in the annealing process, where the solution values are mapped to different energy states of atoms. While Molecular Search simulates the motion of molecules under the microscopic world and solutions are encoded as order or positions of molecules instead. The other significant difference is that Simulated Annealing mainly considers local moves. On the contrary, molecule jumping in molecule search simulates the concurrent motion of molecules.

3.3 Molecule list and position vector

The traditional activity list (AL) is often used to represent a solution for RCPSP, where the activities are ordered in a linear sequence without violating the precedence constraints. Here, the molecule list (ML) is used to present the solution instead. The only difference between molecule list and activity list is that activity is named as a molecule in the molecule list. Therefore, the molecule list also has to satisfy the precedence constraints between molecules, which is same as the precedence constraints between activities.

In the molecule walking, we consider insertion operation on the molecule list without violating the precedence constraints. However, in the molecule jumping, as the selected molecules are jumping concurrently, the position vector P of molecules is introduced to present the solution alternatively. The position vector P is defined as $P = (p_1, p_2, \ldots, p_n)$, with $p_i \in R$, $(1 \leq i \leq n)$, where p_i represents the position of molecule i, and n is the number of activities in the project. The following formula is used to encode the molecule list ML to the position vector P.

$$P[ML[i]] \leftarrow i, (1 \leq i \leq n) \tag{3.2}$$

To clarify the transformation between molecule list and position vector, an example with 10 activities and 1 renewable resource is used here.

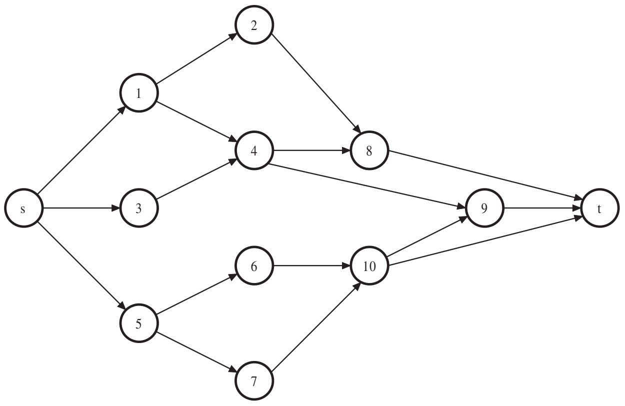


Figure 3.3 Precedence network for example 3.1.

In Figure 3.3, example 3.1, activity s and t are dummy activities, denoting the start and end of the project. There is only 1 renewable resource in example 3.1, and the availability of the resource is 10 units. The duration and resource usages of all the activities are given in Table 3.1.

Table 3.1 Duration and resource usage of activities in example 3.1

Activity	1	2	3	4	5	6	7	8	9	10
Duration	6	5	7	7	4	9	9	5	4	2
Resource usage	4	2	5	7	3	5	6	3	4	4

For example 3.1, an activity list AL_1 can be [1,3,5,6,2,7,10,4,9,8], which satisfies all the precedence constraints. Like the activity list, the molecule list ML_1 is presented as [1,3,5,6,2,7,10,4,9,8]. Encoding from ML_1, the position vector P_1 is [1,5,2,8,3,4,6,10,9,7]. To evaluate a solution with ML_1, the molecules (activities) are scheduled one by one as early as possible following their order in the molecule list with the Serial Generation Scheme (SGS) [66], where the makespan of the project is obtained as the latest finish time of all the activities. For example, to evaluate ML_1, the molecules (activities) are scheduled one by one according to the order in ML_1. The corresponding schedule is illustrated in Figure 3.4, and the makespan is calculated as 42. Since the objective function of RCPSP is to minimize the makespan, the value of the makespan is used as the quality of the molecule list and the position vector. The decoding scheme from the position vector to the molecule list is more complicated than the encoding one, since the positions of molecules can be any real number in the jumping process. A detailed decoding scheme for position vector will be given in section 3.4.

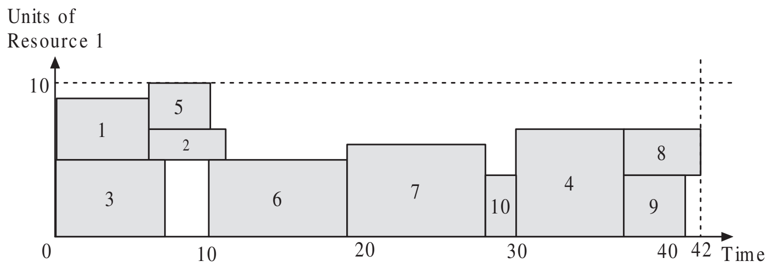


Figure 3.4 Scheduling example 3.1 according to ML_1

3.4 Molecule jumping

Molecule jumping simulates the simultaneous motion of high kinetic energy molecules. According to the Boltzmann distribution, only a small portion of the molecules have high kinetic energy. Moreover, the speed of those molecules with high energy is also high. Since the positions of the high energy molecules change very fast, the motion of the high energy molecules is simulated concurrently as molecule jumping. Three different jumping rules are proposed in this research to guide the motion of the high energy molecules. They are randomized cooling jumping, critical activity based jumping and hidden order based jumping. Randomized cooling jumping selects the high energy molecules randomly, and the new positions of the high energy molecules are also randomly chosen, where the number of higher energy molecules decreases when environment temperature decreases. In the critical activity based jumping, we consider the flexibility of an activity as the difference between its start time in the schedules with the same makespan in FBS. The activities are considered as critical heuristically here, if its flexibility is 0, which means in all the schedules of FBS with the same makespan, it has the same start time. Those critical activities form the high energy molecules and will be assigned with smaller position values compared with other molecules. Unlike the other two jumping rules, the hidden order based jumping generates an entire new position vector according to precedence information updated by the history best solutions in Molecule Search.

3.4.1 Randomized cooling jumping

In the randomized cooling jumping, a small portion of the molecules will be randomly selected and the new position values of those molecules are also randomly chosen from 1 to n, where n is the number of activities in the project. Since the average kinetic energy decreases when temperature of the matter decreases, the number of molecules selected to jump also decreases according to temperature. The detail for randomized cooling jumping is given as:

Algorithm 10 Randomized cooling jumping

```
molecule_jumping(ML, T)
$J \leftarrow \emptyset$
for $i \leftarrow 1$ to $n \cdot T$ do
    repeat
        $k \leftarrow rand(1, n)$
    until $k \notin J$
    $J \leftarrow J \cup \{k\}$
end for
$P \leftarrow encode(ML)$
for each $k \in J$ do
    $P[k] \leftarrow rand(1, n)$
end for
$ML \leftarrow decode(P)$
```

In the above randomized cooling jumping, the percent of the jumping molecules is decided by environment temperature T as $n \cdot T$. The jumping molecules are randomly selected, which form the set J. The position vector for jumping molecules will be randomly generated, while the position vector for other molecules will remain the same as before.

As we have previously mentioned the decoding scheme is not as simple as the encoding scheme. To decode the position vector to the molecule list in molecule jumping, the molecules have to be sorted in increasing order of their position values.

However, as the molecules jump randomly, the new order of the molecules (activities) may not satisfy the precedence constraints. Therefore, the following algorithm is used to decode the position vector. We note similar procedure has been described by Hartmann [52].

Algorithm 11 Decoding molecule position vector to molecule list

```
decode(P)
for $i \leftarrow 1$ to n do
    if $predecessor(i) = \emptyset$ then
        save molecule $i$ to MIN-HEAP with the key P[i]
    end if
end for
$t \leftarrow 1$
while MIN-HEAP is not empty do
    extract molecule i with minimum P[i] from the MIN-HEAP
    $ML[t] \leftarrow i$
    $t \leftarrow t + 1$
    for each $u \in successor(i)$ do
        $predecessor(u) \leftarrow predecessor(u) - \{i\}$
        if $predecessor(u) = \emptyset$ then
            save molecule $u$ to MIN-HEAP with the key P[u]
        end if
    end for
end while
return ML
```

In the decoding algorithm, a minimum heap MIN-HEAP is used to save all the molecules which can be scheduled at the current state. The position value of the molecule is used as the key for the MIN-HEAP, and ties are broken randomly. If there is a directed edge from i to j in the precedence network of the project, i is the predecessor of j and j is the successor of i. We have defined $predecessor(i)$ as the set of all the predecessors of molecule (activity) i. In the same way, $successor(i)$ is defined as the set of all the successors of molecule (activity) i. As an example, assuming the position vector is (4,2,1,3,2,8,5,7,10,7), it can be decoded as molecule list [3,5,1,2,4,7,8,6,10,9]. The time complexity for the above decoding scheme is $O(n \lg n + m)$, where n is the number of activities and m is the number of precedence constraints or the number of the edges in the precedence network of the project.

3.4.2 Critical activity based jumping

According to the new definition of float for RCPSP, critical activity is defined as the activity with 0 float value for the fixed deadline of the project. Since the critical activities have less flexibility than other activities, scheduling the critical activity first may result in an improved makepspan. For critical activity based jumping, the critical activities are selected as the high energy molecules and negative position values will be assigned to them so that they can be scheduled earlier than other activities. However, it is intractable to find all the critical activities, since it is NP-hard to find the maximum float of an activity. As *S-Float* can be taken as the approximated value for *E-Float*, *S-Float* is used here to select the critical activities. Since the forward-backward search are used as the walking process, to reduce the running time, *S-Float* of activity is calculated approximately as the difference between the start time of the activity in the left-justified schedule and right-justified schedule. The details of the critical activity based jumping are given in the following.

Algorithm 12 Critical activity based jumping

```
$molecule\_jumping(ML, T)$
$J \leftarrow 1, 2, ..., n$
for $k \leftarrow 2$ to $N_{walk}$ do
    for $i \leftarrow 1$ to n do
        if $start\_time[k][i] \neq start\_time[0][i]$ then
            $J \leftarrow J - \{i\}$
        end if
    end for
end for
$P \leftarrow encode(ML)$
for each $k \in J$ do
    $P[k] \leftarrow -rand(1, n)$
end for
$ML \leftarrow decode(P)$
```

In the critical activity based jumping, N_{walk} is the number of steps in molecule walking, start_time is the array recording the start time of the activities in all the steps of molecule walking. If all the start time of the molecules has the same value, the molecule will be selected as a jumping molecule. The position values of the jumping molecules are set as an integer value from [-n,1] randomly, so that all the critical activities can be scheduled ahead of other activities. According to the experimental results, only a small portion of activities will be selected as critical activities here.

3.4.3 Hidden order based jumping

The strength of both precedence and resource constraints mainly determines how difficult RCPSP is [70]. If there are no precedence constraints, RCPSP with one renewable resource is equal to 2D bin packing problem. If there are feasible precedence constraints between all pairs of activities, RCPSP can be simply solved by topology sort. According to preliminary experiments, the solution quality of RCPSP highly depends on the order of some pairs of molecules in the molecule list, where there are no precedence relationships between those molecules. We call the order of activities which leads to high quality solutions as hidden order. For the hidden order based jumping, the concurrent motion of the molecules is guided by the hidden order between molecules. To discover the hidden order, for a molecule list ML, a random variable $X_{i,j}$ for any two activities i and j is defined as

$$X_{i,j} = \begin{cases} 1 & \text{molecule i ahead of molecule j in } ML, \\ 0 & \text{molecule j ahead of molecule i in } ML. \end{cases} \tag{3.3}$$

Assuming $X_{i,j}^1, X_{i,j}^2, ..., X_{i,j}^k$ are k random variables for the hidden order between activity i and activity j in k molecule lists of the best solutions sampled in our program, the sample mean for those variables is given as, $\overline{X_{i,j}} = \frac{1}{k}\sum_{u=1}^{k} X_{i,j}^u$. A hidden order matrix is defined as, $H := (h_{i,j})_{n\times n}$, where $h_{i,j} = \overline{X_{i,j}}$. It is obvious that $h_{i,j}$ is 0 or 1 when there is precedence constraint between two activities. Here, we only consider the hidden order between those activities which can be scheduled together according to precedence constraints. In the hidden order based jumping, $h_{i,j}$ is used as the probability for molecule (activity) i being ahead of molecule (activity) j in the molecule lists for best solutions. To simplify the calculation, we also consider that $h_{i,j}$ is independent for different i and j. Therefore, the probability for molecule (activity) i ahead of a set of molecules (activities) $J = \{j_1, j_2, ..., j_m\}$ can be calculated as

$\prod_{k=1}^{m} h_{i,j_k}$, which is taken as the weight $w[i]$ of molecule i. In the hidden order based jumping, the molecules will be selected randomly proportional to the weight of them according to their topological order. The details for hidden order based jumping are presented in Algorithm 13.

Algorithm 13 Hidden order based jumping

```
molecule_jumping(ML, T, H)
$P \leftarrow encode(ML)$
$J \leftarrow \emptyset$, $t \leftarrow 0$
**for** $i \leftarrow 1$ to n **do**
  **if** $i = \emptyset$ **then**
    $J \leftarrow J \cup \{i\}$
  **end if**
**end for**
**while** $J \neq \emptyset$ **do**
  **for** each $k \in J$ **do**
    $w[k] \leftarrow \prod_{j \in (J - \{k\})} h_{k,j}$
  **end for**
  select molecule i from J randomly proportional to weight w[i]
  $ML[t] \leftarrow i$
  $t \leftarrow t + 1$
  $J \leftarrow J - \{i\}$
  **for** each $u \in successor(i)$ **do**
    $predecessor(u) \leftarrow predecessor(u) - \{i\}$
    **if** $predecessor(u) = \emptyset$ **then**
      $J \leftarrow J \cup \{u\}$
    **end if**
  **end for**
**end while**
```

In the hidden order based jumping, all the positions of molecules may changed concurrently. An entire new molecule list is generated according to the hidden order matrix H.

3.5 Molecule walking – forward-backward search

3.5.1 Reverse molecule list

Molecule walking is designed as the local search process to simulate the motion of the low energy molecules in a series of steps. Various local search based methods have been proposed for RCPSP, including Simulated Annealing [12, 14], Tabu Search [5, 90] and variable neighborhood search [61]. Wiest [32] introduces the left-justified schedule as a feasible schedule in which no job can be started at an earlier date by local left shifting of the job alone. Accordingly, right-justified schedule is defined as a feasible schedule in which no job can be ended at a later date by local right shifting. A detailed discussion for local and global shifting can be found in [108]. A left-justified schedule can be obtained through scheduling molecules (activities) in the molecule list one by one as early as possible, which is just the SGS procedure. A right-justified schedule can be obtained in a similar way, while the molecules (activities) are scheduled by reverse order of the molecule list as late as possible. Here the reverse order of the molecule list is named as reverse molecule list. To give a formal definition of the reverse molecule list, we would initially like to present the reverse precedence network. Assuming a precedence relationship between activity i and activity j is presented as a partial order $i \prec j$, implying activity j can only be scheduled after the completion of activity i, there is a reverse precedence relationship $j \prec i$ in the reverse precedence network if and only if there is a precedence relationship $i \prec j$ in the precedence network. The reverse precedence network for example 3.1 is illustrated in Figure 3.5.

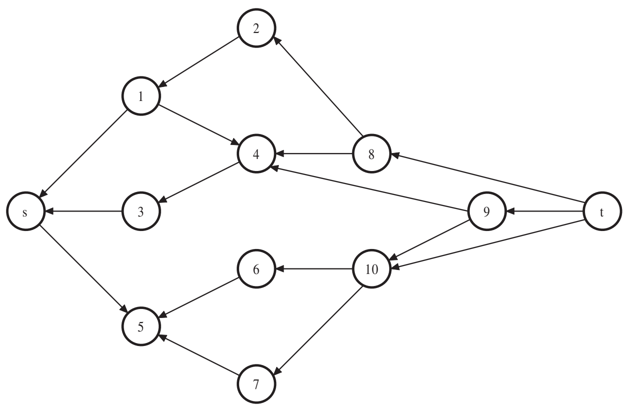


Figure 3.5 Reverse precedence network for example 3.1.

The reverse precedence network represents the reverse precedence constraints between the activities (molecules). A reverse molecule list is defined as the linear sequence representing a permutation of molecules (activities), which satisfies the reverse precedence constraints. As an example, [9,10,7,6,8,4,2,5,3,1] is a reverse molecule list for example 3.1.

3.5.2 Benefit reversing

As we have mentioned, a left-justified schedule can be obtained through scheduling the activities one by one following the order in the molecule list. A reverse molecule list can be obtained through sorting the activities with decreasing order of their finish time in the left-justified schedule, where ties are broken randomly. A right-justified schedule can be obtained through scheduling the activities one by one as late as possible according to the reverse molecule list. It can be proved simply by mathematical induction that the makespan of the right-justified schedule obtained in this process is less or equal to the makespan of the left-justified schedule, since each activity is able to start later or equal than its previous start time. Therefore, this process is named as benefit reversing (BR) here. A similar BR can be applied to the right-justified schedule to obtain a left-justified schedule through sorting the activity in increasing order of their start time. The BR is first introduced by Tormos and Lova [110], where notable experimental results are achieved through combining a random sampling procedure with an iterative BR. The iterative BR is named as forward-backward improvement (FBI) in the survey by Kolisch and Hartmann [67] in 2005. Further research has been conducted by Valls et al. [116] to combine FBI with existing heuristics, where significant improvements have been achieved.

3.5.3 Details of forward-backward search

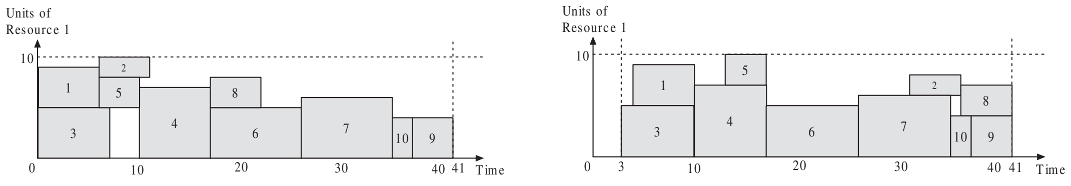
Algorithm 14 molecule walking (forward-backward search)

```
  molecule_walk(ML, T)
  for i=1 to $\frac{N_{walk}}{2}$ do
    $P \leftarrow encode(ML)$
    $k \leftarrow rand(1, n)$
    $u \leftarrow \max_{j \in predecessor(k)} P[j]$
    $j \leftarrow rand(u + 1, P[k] - 1)$
    $ML' \leftarrow insert(ML, j, P[k])$
    if $f(ML') \leq f(ML)$ then
      $ML \leftarrow ML'$
    end if
  end for
  $RML \leftarrow BR(ML)$
  for i=1 to $\frac{N_{walk}}{2}$ do
    $P' \leftarrow encode(RML)$
    $k \leftarrow rand(1, n)$
    $u \leftarrow \max_{j \in predecessor'(k)} P(j)$
    $j \leftarrow rand(u + 1, P'(k) - 1)$
    $RML' \leftarrow insert(RML, j, P[k])$
    if $f(RML') \leq f(RML)$ then
      $RML \leftarrow RML'$
    end if
  end for
  $ML \leftarrow BR(RML)$
```

The forward-backward search (FBS) proposed here is different from FBI. Instead of running BR iteratively, local search is applied to the molecule list and the corresponding reverse molecule list in forward-backward search (FBS). There are 4 basic steps for the FBS. Firstly, local search is applied to the molecule list, where it is named as the forward search. After that, at the second step, the molecule list is transformed to the reverse molecule list with BR. At the third step, local search is applied again to the reverse molecule list, where it is named as the backward search. Finally, at the fourth step, the reverse molecule list is transformed back to the molecule list with BR. The insertion operation is used in the neighborhood generation, which has also been used in the simulated annealing approach by Boctor [12] and Bouleimen, Lecocq [14]. To obtain the precedence constraints feasible solution, the molecule can only be inserted after all its predecessors and before all its successors in the molecule list. Since local search is applied to both the molecule list and the reverse molecule list, we only consider forward insertion in the local search, where forward insertion in the reverse molecule list has the same effect as the backward insertion on the molecule list. The detailed forward-backward search is presented in Algorithm 14.

In the molecule walking (forward-backward search), there are $\frac{N_{walk}}{2}$ local search steps in the forward search, where a molecule (activity) k is randomly chosen first. A new position j for molecule (activity) k is also randomly generated after the latest predecessor of molecule k and before the position of molecule k. A new molecule list ML' is obtained through inserting the molecule (activity) k before the molecule at position j in ML. If the makespan for ML' is less than or equal to ML, ML will be updated as ML'. After the forward search, the molecule list ML will be transformed to the reverse molecule list RML with BR function. Similar to the forward search, the backward search will be applied to RML. Since the precedence constraints have changed to reverse precedence constraints, the set of predecessors of a molecule (activity) k changes to predecessor'(k) instead, where it is easy to see that the set predecessor'(k) is equal to the set successor(k).

As an example of FBS (forward-backward search), the initial molecule list is given as [1,3,5,6,2,7,10,4,9,8], the schedule of which is shown in Figure 3.4. In the forward search, assuming molecule 2 is inserted at position 2 before molecule 3, the new molecule list is [1,2,3,5,6,7,10,4,9,8]. After that, molecule 4 is inserted at position 5, just before molecule 6, and the molecule list changes to [1,2,3,5,4,6,7,10,9,8],where the corresponding schedule can be found in Figure 3.6 (a).

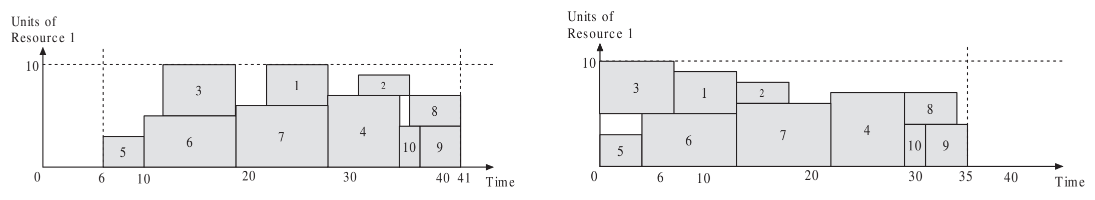


(a) Forward search (b) Benefit reversing to RML

Figure 3.6 An example of forward search

After the forward search, the BR is used to transfer the molecule list [1,2,3,5,4,6,7,10,9,8] to the reverse molecule list [9,10,7,6,8,4,2,5,3,1]. The right-justified schedule for the reverse molecule list is illustrated in Figure 3.6 (b). In Figure 3.6 (b), we can observe that the makespan of the project has been reduced to 38.

The backward search is applied to the reverse molecule list [9,10,7,6,8,4,2,5,3,1]. By inserting molecule 8 and 4 ahead of molecule 7, molecule 2 ahead of molecule



(a) Backward search (b) Benefit reversing to ML

Figure 3.7 An example of backward search

7 and molecule 1 ahead of molecule 7 in sequence, we get the reverse molecule list [9,10,8,4,2,1,7,6,5,3]. The schedule for the new reverse molecule list [9,10,8,4,2,1,7,6,5,3] can be found in Figure 3.7 (a), where the makespan for the project has been reduced to 35. Finally the reverse molecule list is transformed to the molecule list [5,6,3,7,1,4,2,10,8,9] with BR, which can be found in Figure 3.7 (b). With the molecule walking (FBS), the makespan for the project has been reduced from 42 to 35.

In FBS, if the start time of a molecule (activity) is equal to the finish time of one of its predecessors, forward insertion of the molecule (activity) will not change the schedule (start time of the activities). Therefore, without calling SGS to evaluate the new molecule list, the molecule list will be updated directly by the new molecule list, which significantly improves the speed of the forward-backward search. A checking vector C is used to record whether the start time of the molecules' (activities') equal to the latest finish time of their predecessor, where $C[k] \leftarrow 1$, if molecule k's start time equal to the latest finish time of its predecessor, otherwise $C[k] \leftarrow 0$. As an example, the checking vector C for the molecule list [1,3,5,6,2,7,10,4,9,8] is [0,1,0,0,0,1,0,1,1,1]. Since $C[2] = 1$, inserting molecule 2 at position 2 will not change the schedule. However, as $C[4] = 0$, SGS needs to be called, when molecule 4 is inserted at position 5. Since all the vertices and edges need to be visited in the precedence network for the project to update the checking vector C, the complexity for updating C is $O(n+m)$, where n is the number of activities and m is the number of precedence constraints.

3.6 Computational study on Molecule Search

Currently, the performance of different heuristics on a certain combinatorial optimization problem is usually judged by how well they converge on standard benchmark test cases within a given time limit. It is reasonable to compare heuristics in this way, since the ability for a heuristic to search for high quality solutions within time limit decides how useful the heuristic is in practice. However, the speed of the machine and the compiler used to compile the programs will also affect the results of heuristics. To more accurately measure different heuristics on RCPSP, Kolisch and Hartmann [66, 67] have proposed to use the number of SGS (Serial Generation Scheme) [66] calls as limit rather than the running time, since almost all the heuristics use SGS to assign start time of the activities. The new scheme works fairly successfully in comparing the performances of various heuristics on PSPLIB (standard benchmark test problems [70]), where detailed results are presented in the survey [67] by Kolisch and Hartmann.

Following the standard in the 2005 survey by Kolisch and Hartmann, we have tested Molecule Search on the J30, J60 and J120 test sets of PSPLIB with 1000, 5000 and 50000 SGS calls limit. The computational study is presented as 4 parts. Since the two main components of the Molecule Search are molecule walking and molecule jumping, the first and second parts of experiments are used to justify the performance of the molecule walking and molecule jumping. Detailed experimental results comparing Molecule Search with all the other state-of-the-art heuristics are reported as the third part. Finally, Molecule Search is applied to identify float for projects with resource constraints. The Molecule Search is implemented in C++ and compiled with GNU g++ with maximize optimization option. All the experiments are carried out on Intel P4 3.0GHZ PC with 1G Memory.

3.6.1 Effectiveness of molecule walking

The molecule walking is the local refinement process to simulate the motion of low energy molecules. In this research, the newly developed FBS is used as the molecule walking process for RCPSP. Here, we compare FBS with other most well-known local search algorithms for RCPSP, including forward-backward improvement (FBI), Tabu Search (TS), Simulated Annealing (SA) and Hill Climbing (HC). We also compared FBS with the well used refinement procedure FBI, although it is not a local search procedure. The average deviations from the optimal solution of the J30 test set and the average deviations from the Critical Path Method lower bounds of all those algorithms on the J60 and J120 test sets with 5000 and 50000 iterations of SGS are reported in Table 3.2. Here N_{walk} is set to be 100. The program will run the FBS algorithm iteratively until it reaches the SGS limit. There are different versions of sampling with FBI and TS. As the best of them, the sampling with FBI [112] and the TS [90] are reported here, where the initial solutions for FBI are generated by regret based sampling with latest finish time (LFT) priority rule.

Table 3.2 Effectiveness of molecule walking–FBS

	J30		J60		J120	
Algorithm	5000	50000	5000	50000	5000	50000
FBS	0.15	0.13	11.42	11.16	33.73	32.19
FBI	0.13	0.05	11.62	11.36	34.41	33.71
TS	0.16	0.05	12.18	11.58	37.88	35.85
SA	0.23	-	11.90	-	37.68	-
HC	0.37	0.24	12.34	11.88	37.41	36.48

In Table 3.2, we can find the FBS achieves the best results on the J60 and J120 test sets among all the local search algorithms. We note that the FBI is most successful for the J30 test set. We also can find in Table 3.2 that the results of FBS is much better than the results of HC, which shows that searching from forward and backward side iteratively is much more effective than searching from one side alone.

3.6.2 Effectiveness of molecule jumping

There are different kinds of molecule jumping rules, including randomized cooling jumping (RCJ), critical activity based jumping (CAJ) and hidden order based jumping (HOJ), which are discussed in section 3.4. To evaluate the effectiveness of molecule jumping, we compare three versions of Molecule Search MS_{hoj}, MS_{rcj} and MS_{caj} with FBS alone and rand_FBS, where, for rand_FBS, a randomly generated solution will be used as the initial solutions for each iteration of FBS instead of using the jumping solutions. The results for 500000 SGS are also reported here to further investigate the performance of the different jumping rules. According to preliminary experiments, in the randomized cooling jumping, the environment start temperature T_0, and ending T_e is set to be 0.1 and 0.05 respectively, and temperature decreasing rate α is set to be 0.95. When the environment temperature T reaches T_e and the number of SGS calls has not been reached, T will be set as T_0 and the loop in molecule search will restart.

The detailed experimental results are given in Table 3.3. In Table 3.3, all the Molecule Search algorithms usually perform better than the FBS and rand_FBS, especially for the J120 test set, which demonstrates that the molecule jumping is effective in guiding the FBS. We can also find MS_{hoj} achieves better results than MS_{rcj} and MS_{caj}, except MS_{rcj} obtains best results for the J60 test set with 500000 SGS iterations.

Table 3.3 Effectiveness of molecule jumping

	J30			J60			J120		
Algorithm	5000	50000	500000	5000	50000	500000	5000	50000	500000
$\texttt{MS}_{\texttt{hoj}}$	0.11	0.03	0.01	11.33	10.85	10.63	33.54	31.97	30.82
$\texttt{MS}_{\texttt{caj}}$	0.13	0.06	0.01	11.36	10.90	10.66	33.69	32.04	30.88
$\texttt{MS}_{\texttt{rcj}}$	0.12	0.05	0.00	11.38	10.86	10.56	33.75	32.06	30.97
FBS	0.15	0.13	0.08	11.42	11.16	10.97	33.73	32.19	31.49
Rand_FBS	0.12	0.01	0.00	11.37	10.89	10.63	33.65	32.43	31.59

3.6.3 Experimental results of Molecular Search

RCPSP is one of the most basic problem in project management and it has found its applications in wide range of real-life problems. Because of the NP-hard intractable nature of RCPSP, numerous heuristics have been developed to tackle it. The detailed survey for the heuristics on RCPSP can be found in [67]. In table 3.4, we compare our Molecule Search with the best 20 heuristics reported in the 2005 survey [67] by Kolisch and Hartmann.

In Table 3.4, we can find that MS ranked 5th among all the best heuristics for RCPSP, where the ranking is decided by the results of the J120 test set for 50000 SGS iterations and ties are broken according to the results of the J120 test set for 5000 SGS iterations. All the best 4 algorithms are population based algorithms and embedded with the FBI procedure. While, the operations of MS only are applied to a single solution. Moreover, The MS uses newly developed FBS as molecule walking instead of FBI.

Table 3.4 Comparing MS with other heuristics for RCPSP with schedule limit

		J30		J60		J120	
Algorithm	Reference	5,000	50,000	5000	50000	5000	50000
GA – hybrid, FBI	[114]	0.06	0.02	**11.10**	10.73	**32.54**	**31.24**
GA – forw.-backw., FBI	[2]	0.06	0.03	11.19	10.84	33.91	31.49
Scatter Search – FBI	[35]	0.11	0.01	**11.10**	**10.71**	33.10	31.57
GA – FBI	[116]	0.20	0.02	11.27	10.74	33.24	31.58
MS	Ours	0.11	0.03	11.33	10.85	33.54	31.97
GA, TS – path relinking	[61]	**0.04**	**0.00**	11.17	10.74	33.36	32.06
population-based – FBI	[116]	–	–	–	–	34.02	32.81
GA – self-adapting	[52]	0.22	0.08	11.70	11.21	35.39	33.21
sampling – LFT, FBI	[112]	0.13	0.05	11.62	11.36	34.41	33.71
ant system	[82]	–	–	–	–	35.43	–
GA – activity list	[52]	0.25	0.08	11.89	11.23	36.74	34.03
sampling – LFT, FBI	[111]	0.17	0.09	11.82	11.47	35.56	34.77
sampling – LFT, FBI	[110]	0.16	0.07	11.87	11.54	35.81	35.01
GA – forw.-backward	[57]	0.12	–	11.86	–	36.57	–
TS – activity list	[90]	0.16	0.05	12.18	11.58	37.88	35.85
GA – late join	[29]	0.33	0.16	12.63	11.94	38.41	36.44
sampling – random, FBI	[116]	0.28	0.11	12.35	11.94	37.47	36.46
SA – activity list	[14]	0.23	–	11.90	–	37.68	–
GA – priority rule	[52]	1.12	0.88	12.74	12.26	38.49	36.51
sampling – adaptive	[105]	0.44	–	12.58	–	38.70	–
sampling – LFT, parallel	[64]	1.29	1.13	13.23	12.85	38.75	37.74

The MS can keep on improving solution quality, if more calls of SGS are allowed. In Table 3.5, we compared the results of MS with 500000 iterations of SGS with all the other heuristics without schedule limit, where the results of other heuristics are from [67]We can find that the MS achieves second best results in a short running time in Table 3.5.

Table 3.5 Comparing MS with other heuristics for RCPSP without schedule limit

		J30		J60		J120	
Algorithm	Reference	Dev.	CPU	Dev.	CPU	Dev.	CPU
Scatter Search – FBI	[35]	0.01	7.16	**10.53**	19.61	**30.48**	69.57
MS	Ours	0.01	3.44	10.56	4.65	30.82	21.49
population – based	[115]	0.10	1.16	10.89	3.7	31.58	59.4
decompos. & local opt.	[92]	**0.00**	10.26	10.81	38.8	32.41	207.9
VNS – activity list	[45]	0.01	0.64	10.94	8.89	33.10	219.86
local search – critical	[117]	0.06	1.61	11.45	2.8	34.53	17.0
LR – activity list	[86]	–	–	15.60	6.9	36.00	72.9
TS – network flow	[4]	–	–	12.05	3.2	36.16	67.0
network decomposition	[107]	0.12	2.75	11.61	460.2	39.29	458.5
MP – network flow	[4]	1.74	–	14.20	–	39.34	–

Table 3.6 Detailed experimental results of MS on PSBLIB

			Deviation (%)		No. of	No. of	CPU-time (sec)	
Test set	Schedules	Sum	avg.	max.	best	improved	avg.	max.
J30	1000	28402	0.24	5.26	426(480)	-	0.01	0.02
	5000	28358	0.11	4.35	454(480)	-	0.04	0.14
	50000	28326	0.03	2.06	471(480)	-	0.26	0.89
	500000	28321	0.01	2.06	476(480)	-	3.44	13.98
J60	1000	38865	12.04	115.58	361(480)	0	0.01	0.06
	5000	38628	11.33	107.79	376(480)	1	0.06	0.28
	50000	38470	10.85	106.49	400(480)	1	0.40	1.69
	500000	38394	10.63	102.60	429(480)	1	5.30	26.42
J120	1000	77286	36.17	218.81	195(600)	0	0.07	0.12
	5000	75788	33.54	204.95	214(600)	0	0.29	0.56
	50000	74899	31.97	204.04	251(600)	0	2.35	5.56
	500000	74249	30.82	197.98	324(600)	1	21.49	54.09

In Table 3.6, detailed results for MS with 1000, 5000, 50000 and 500000 iterations of SGS on the J30, J60 and J120 test sets of PSPLIB are reported, where the best results of PSPLIB updated by August 22, 2005 are used as comparison here.

3.6.4 Calculating the float with Molecular Search

Table 3.7 *H-Float* by Molecular Search with optimal deadline for the J30 test set

No. of Step		*H-Float* (period)	No. of Crt-Act	Accuracy (%)	Makespan avg(period‘)	CPU (sec)
5000	avg	6.22	10.49	96.06	58.46	4.89
	max	23.17	28.00	100.00	129.00	18.16
	min	0.07	0.00	36.67	35.00	0.03
	dev	3.89	5.13	10.05	13.71	4.22
50000	avg	6.32	10.03	98.65	58.40	48.26
	max	23.17	28.00	100.00	129.00	201.65
	min	0.10	0.00	43.33	35.00	0.03
	dev	3.79	4.48	5.09	13.59	42.10
500000	avg	6.35	9.86	99.92	58.39	493.59
	max	23.17	26.00	100.00	129.00	1929.69
	min	0.37	0.00	90.00	35.00	0.03
	dev	3.76	4.21	0.73	13.58	428.54

In the first part of the thesis, the new definition of float for RCPSP (resource constrained project) is given, where we show that the float value can be calculated approximately by Simulated Annealing. Since the MS is more effective than the SA on RCPSP, we have applied MS in finding the float for all the test problems in the J30 test set. The results for calculating the float with MS are reported in Table 3.7. With 5000 iterations of SGS the *H-Float* calculated by MS already achieves 96.06%

accuracy, which is much better than 91.63% accuracy of the *H-Float* calculated by SA with 5000 SGS iterations. Those results also show that the MS converges much faster than the SA for the RCPSP.

3.7 Conclusion

Simulating the cooling process from gas to crystal, Molecule Search, divided as molecule jumping and molecule walking, is applied to solve RCPSP. Molecule jumping is used to simulate the concurrent motion of high energy molecules. Three different jumping rules are discussed in this research, including randomized cooling jumping, critical activity based jumping and hidden order based jumping. Molecule walking is used to simulate the motion of low energy molecules in a series of steps. A new local search procedure FBS is proposed here as molecule walking. The effect of molecule jumping and molecule walking has been supported by extensive experimental results. Compared with all the other state-of-art heuristics, molecule search emerges to be one of the best heuristics for RCPSP. As a new optimization method, Molecule Search may be applied to other combinatorial optimization problems in the future.

Chapter 4
Molecular Bank Algorithm for Resource-Constrained Project Scheduling Problem

In the last chapter, simulating the cooling process from gas to crystal, Molecule Search (MS) has been developed as a new optimization technique for combinatorial optimization problems, where MS has been successfully applied to the bandwidth minimization problem [81, 80], the minimal linear arrangement problem [77] and the resource constrained project scheduling problem [78]. A population based MS is proposed here to tackle the resource-constrained project scheduling problem. This approach is named as the Molecule Bank Algorithm (MBA). As a population based approach, the crossover operator from the genetic algorithm is used in MBA. Moreover, dynamic population, molecule aging and molecule drifting are also proposed here to balance diversification and intensification in the search process of MBA.

4.1 Molecule Bank Algorithm

The Molecule Bank Algorithm can be viewed as a combination of Genetic Algorithm and Molecule Search. The detailed flowchart for MBA can be found in Figure 4.1. A Molecule bank is used to store all the elite solutions, as shown in Figure 4.1. There are two different sets of elite solutions. One is the set of historical best solutions. The other is the set of elite solutions for current iteration of MBA. In the process of MBA, a number of molecule lists will be generated first and saved to the molecule bank, where the dashed line in Figure 4.1 shows the data exchange between the molecule bank and the operators of MBA. An iterative procedure, including crossover, selection, jumping and walking, will start. This procedure will only end until reaching the limit of the number of SGS calls or an user specified time limit. In MBA, the operations and the data are presented as different parts in

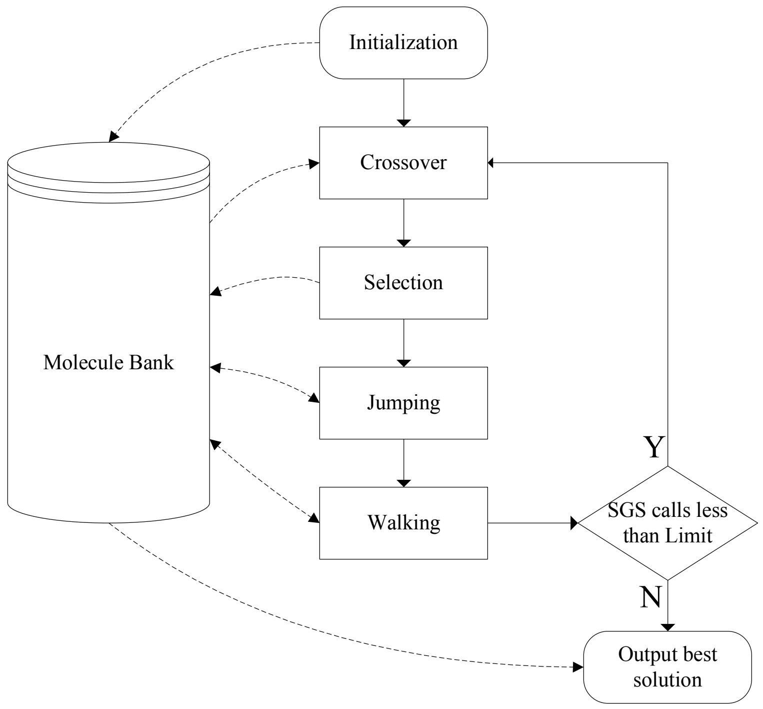


Figure 4.1 Flowchart of Molecule Bank Algorithm

Figure 4.1. The framework of the MBA is similar to knowledge-based information processing in artificial intelligence. Almost all the heuristics can be viewed in a similar way, where new solutions are generated according to the past solutions or rules and properties generalized from past solutions. In the process of searching for better solutions for combinatorial optimization problems, different heuristics have different ways of making use of previously found solutions. Therefore, how well a heuristic can use previously found solutions will probably decide how effective the heuristic will be.

4.1.1 Initialization

The molecule list is still used here to represent solutions of RCPSP, where it is the same as the traditional activity list. There is an one-to-one mapping from the molecules to the activities, and the molecules need to satisfy all the constraints of the activities also. In MBA, the population of the molecule lists for each generation is not fixed. Here we use POP_k to present population of the k^{th} generation of molecule lists. Initially, there will be POP_0 molecule lists generated randomly. The forward-backward improvement (FBI) [110] procedure will be applied on each molecule list, where the FBI is run iteratively until no more improvements can be found. After that, all the molecule lists are ranked according to their corresponding makespan. POP_1 best ranking molecule lists will be selected and saved to the molecule bank, and they form the first generation of molecule lists.

4.1.2 Crossover and selection

Introduced by Holland[56], genetic algorithms have been successfully applied for many combinatorial optimization problems. Various genetic algorithms with different encoding schemes or crossover operators have been proposed for RCPSP, where the GA approach from Hartmann [52] appears to be one of the most promising heuristics for RCPSP. Hartmann has first encoded the activity lists as chromosomes, where the performance for different combinations of crossover operators and selection methods were also compared. Since the molecule list is the same as the activity list, we encode the molecule lists directly as chromosomes here. The experimental results from Hartmann show that two points crossover outperforms the one point crossover or uniform crossover. Therefore two points crossover is also adopted in MBA. There are slight differences between the crossover operator used here compared to the one used by Hartmann. Assuming we have two parent molecule lists $U = [u_1, u_2, ..., u_n]$ and $V = [v_1, v_2, ..., v_n]$, where n is the number of activities in the project, the two points p and q will be randomly selected, where $1 < p < q \leq n$ and $q - p$ ranged from $2n/5$ to $3n/5$. Assuming the child molecule list is $W = [w_1, w_2, ..., w_n]$, the first $(1 \leq i < p)$ and the third $(q \leq i \leq n)$ part of the child molecule list is inherited from the father U directly as

$$w_i = u_i, \text{for all } 1 \leq i < p \text{ and } q \leq i \leq n. \tag{4.1}$$

The second part $(p \leq i < q)$ of the child molecule list is generated from the molecules in the second part of the father molecule list according to their ordering in the mother molecule list as

$$w_i = v_k, \text{for all } p \leq i < q, \tag{4.2}$$

where k is the lowest index such that $v_k \notin \{w_1, ..., w_{i-1}, w_q, w_{q+1}, ..., w_n\}$

For example 3.1 in section 3.3, assuming two parents molecule lists are selected as $U = [1, 3, 5, 6, 2, 7, 10, 4, 9, 8]$ and $V = [1, 5, 7, 3, 6, 4, 10, 2, 8, 9]$, where p is 4 and q is 8. The the child molecule list W produced will be $W = [1, 3, 5, 7, 6, 10, 2, 4, 9, 8]$. The two points crossover for the example is also illustrated in Figure 4.2.

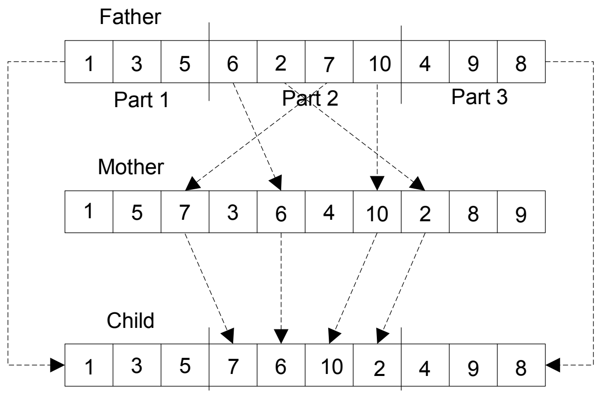


Figure 4.2 An example for two points crossover

The parent molecule lists are selected with 2-tournament selection, where 4 molecule lists ML_1, ML_2, ML_3 and ML_4 will be randomly selected from the current generation each time. The molecule list with less corresponding makespan between ML_1 and ML_2 will be selected as the father molecule list, and the molecule list with less corresponding makespan between ML_3 and ML_4 will be selected as the mother molecule list.

Considering the kth generation (k-th iteration of MBA), the population of molecule lists will be POP_k. To diversify the new molecule lists produced, two parents will only produce one child in MBA. Since $4 \cdot POP_k$ couples of parent molecule lists will be selected for crossover, $4 \cdot POP_k$ new molecule lists will be generated. Therefore, in total, there will be $5 \cdot POP_k$ molecule lists. The ranking method is used here as the selection method. Among the $5 \cdot POP_k$ molecule lists, POP_{k+1} molecule lists with the smallest corresponding makespan will be selected as the $(k+1)th$ generation of molecule lists.

4.1.3 Jumping and walking

In MBA, we treat molecule jumping and walking as separate operators, where they work on different sets of data from the molecule bank and also have different environment temperatures. As we have mentioned, there are two types of data in the molecule bank. Molecule jumping works on the historical best solutions. Molecule walking works on the current generation of elite solutions produced by the crossover operator and selected by the selection operator.

In the molecule bank, K_{best} number of history best molecule list will be stored. In the molecule jumping operation of MBA, a molecule list will be randomly selected and randomized cooling jumping will be performed on it. After the molecule jumping operation, the modified molecule list is inserted into the current generation of elite solutions in the molecule bank. For each iteration of MBA, the molecule jumping does not take place all the time. Instead, the probability for molecule jumping to take place is defined as p_{jump}. According to preliminary experiments, MBA will achieve best results when K_{best} is set to 4 and p_{jump} is set to 0.1. The detailed molecule jumping operator is presented in the following.

Algorithm 15 molecule jumping operator in MBA

```
*molecule_jumping(Molecule_Bank,$T_{jump}$,$K_{best}$,$p_{jump}$)*
$q \leftarrow rand(0,1)$
**if** $q < p_{jump}$ **then**
    retrieve history best solutions from Molecule_Bank as Best_ML
    $k \leftarrow rand(1, K_{best})$
    $ML \leftarrow Best\_ML[k]$
    $J \leftarrow \emptyset$
    **for** $i \leftarrow 1$ to n*$T_{jump}$ **do**
        **repeat**
            $k \leftarrow rand(1, n)$
        **until** $k \notin J$
        $J \leftarrow J \cup \{k\}$
    **end for**
    $P \leftarrow encode(ML)$
    **for** each $k \in J$ **do**
        $P[k] \leftarrow rand(1, n)$
    **end for**
    $ML \leftarrow decode(P)$
    $save(ML$, Molecule_Bank)
**end if**
```

In the molecule jumping procedure, T_{jump} is the environment temperature for jumping which decides how many molecules will jump. Similar to the MS, the start temperature for T_{jump} is set to 0.1 and the ending temperature is set to 0.05. For each iteration of MBA, T_{jump} will be updated by $\alpha \cdot T_{jump}$, where α is set to 0.95 here. When T_{jump} reaches 0.05, it will be reset to the start temperature. As we have stated in MS, P is a position vector, where *encode* is the procedure to encode the molecule list to a position vector and *decode* is the procedure to decode the position vector to a molecule list. The newly generated molecule list ML will be saved to molecule bank as one of the elite molecule lists for current generation.

Unlike molecule jumping, molecule walking is conducted on all the molecule lists in the current generation (current iteration of MBA). As a new local search method for RCPSP, forward-backward search (FBS) was proposed as molecule walking in MS [78]. In MBA, an annealing-like FBS is used as the molecule walking instead, which is more flexible than the old FBS procedure. The detailed molecule walking operator for MBA is given below.

In the molecule walking operator, annealing-like judgement ($p < e^{(f(ML)-f(ML'))/T_{walk}}$) is used here to decide whether the molecule list should be updated or not according to the walking temperature T_{walk}. Similar to the Simulated Annealing (SA) approach for RCPSP by Bouleimen and Lecocq [14], the start temperature for T_{walk} is defined as,

$$T^0_{walk} = -\frac{MK_{min}}{5ln(0.01)} \tag{4.3}$$

where MK_{min} is the minimum makespan obtained after the initialization proce-

Algorithm 16 molecule walking operator in MBA

```
*molecule_walk(Molecule_Bank,$T_{walk}$)*
retrieve current generation of molecule lists from Molecule_Bank as Elite_ML
**for** each ML inside Elite_ML **do**
    **for** i=1 to $\frac{N_{walk}}{2}$ **do**
        $P \leftarrow encode(ML)$
        $k \leftarrow rand(1, n)$
        $u \leftarrow \max_{j \in predecessor(k)} P(j)$
        $j \leftarrow rand(u + 1, P(k) - 1)$
        $ML' \leftarrow insert(ML, j, P(k))$ (forward insert molecule k)
        $p \leftarrow rand(0, 1)$
        **if** $p < e^{(f(ML)-f(ML'))/T_{walk}}$ **then**
            $ML \leftarrow ML'$
        **end if**
    **end for**
    $RML \leftarrow BR(ML)$
    **for** i=1 to $\frac{N_{walk}}{2}$ **do**
        $P' \leftarrow encode(RML)$
        $k \leftarrow rand(1, n)$
        $u \leftarrow \max_{j \in predecessor'(k)} P(j)$
        $j \leftarrow rand(u + 1, P'(k) - 1)$
        $RML' \leftarrow insert(RML, j, P(k))$
        $p \leftarrow rand(0, 1)$
        **if** $p < e^{(f(RML)-f(RML'))/T_{walk}}$ **then**
            $RML \leftarrow RML'$
        **end if**
    **end for**
    $ML \leftarrow BR(RML)$
**end for**
```

dure. When the new solution is 20% worse than the MK_{min}, the acceptance rate is only 0.01 according to the start temperature. The ending temperature for T_{walk} is given as,

$$T_{walk}^{e} = -\frac{1}{ln(0.001)} \tag{4.4}$$

There is only 0.001 chance to accept a solution with makespan 1 unit longer than the makespan of the current solution. According to preliminary experiments, the temperature decrease rate α_{walk} is set to 0.25. The temperature T_{walk} does not decrease according to the iterations of MBA, instead it decreases according to the number of SGS calls. Starting from T_{walk}^{0}, there will be $ln(T_{walk}^{e}/T_{walk}^{0})/ln\alpha_{walk}$ different temperature states until T_{walk} reaches T_{walk}^{e}. Here, the SGS calls are divided evenly according to all the temperature states. Defining the limit of SGS calls as L_{SGS}, the number of SGS calls for each temperature states is calculated as,

$$N_{SGS} = \frac{L_{SGS}\ ln\alpha_{walk}}{ln(T_{walk}^{e}/T_{walk}^{0})} \tag{4.5}$$

Where the SGS calls from the crossover operator and the jumping operator will both be counted as well. For each N_{SGS} calls of SGS, the walk temperature T_{walk} decreases by,

$$T_{walk} = \alpha_{walk} \cdot T_{walk} \tag{4.6}$$

4.2 Intensification and Diversification

High quality solutions to a combinatorial optimization problem usually form clusters in the search space, where inside a certain cluster the structures of the solutions are similar. However, different clusters of high quality solutions may be scattered around the search space. To explore the solution space effectively, a heuristic should have both good intensification and effective diversification properties. Intensification means the ability for the heuristic to explore good solutions within a cluster. Diversification can be interpreted as the ability for the heuristic to locate different clusters of good solutions. In MBA, the molecule walking operator gives the algorithm good intensification properties. In the molecule walking operator with annealing-like local moves, the FBS is more efficient in escaping from the local optimum. The crossover operator from GA and the molecule jumping operator from MS are used to achieve good diversification for MBA. In this section, we will further discuss three new techniques developed to balance the intensification and diversification properties of MBA, including dynamic population, molecule aging and molecule drifting.

4.2.1 Dynamic population

In MBA, as mentioned in section 4.1.1, POP_0 number of molecule lists will be generated first. After performing FBI, POP_1 best ranking molecule lists will remain as the first generation of elite molecule lists. The population of the elite molecule lists ML_1 decreases by a constant D_{POP} until it reaches the ending population POP_e.

In each iteration of MBA, if molecule jumping is performed, the new molecule list generated will be inserted into the current generation of elite molecule lists. To calculate the population of the molecule lists, we defined S_{jump} first as follows,

$$S_{jump} = \begin{cases} 1 & \text{when molecule jumping is performed,} \\ 0 & \text{when molecule jumping is not performed,} \end{cases} \tag{4.7}$$

The POP_k for the k-th ($k > 1$) generation can be calculated as,

$$POP_k = \max\{POP_{k-1} - D_{POP} + S_{jumping}, POP_e\} \tag{4.8}$$

With dynamic population, at the beginning of MBA, most of the computation effort is put into diversification, to search for good solution clusters. As the population keeps on decreasing, more and more computation effort is put into intensification. When the population reaches POP_e, most of the computation effort is focused on molecule walking as the intensification method. Preliminary experimental results show that the dynamic population strategy leads to better solutions for RCPSP.

4.2.2 Molecule aging

In the selection operator of MBA presented in section 4.1.2, the best POP_{k+1} molecule lists for the $(k+1)th$ generation are selected from the POP_k molecule lists of the kth generation together with the $4 \cdot POP_k$ molecule lists generated with the crossover operator. Therefore an elite molecule list may be kept for many generations. To achieve better diversification of MBA, the technique of molecule aging is used here, where each molecule list has an age property. The age value of a molecule list is the number of generations the molecule list which has been kept in the molecule bank. We also defined the L_{AGE} as the age limit. In MBA, the molecule list will not be selected as the next generation if it has reached the age limit, where the age limit L_{AGE} is set to 20 here. Although the Molecule Aging technique is proposed independently here, the similar age structure of genetic algorithm has already been studied by Kubota and Fukuba [71], where they show that even a small population size can obtain high performance with the age structure.

4.2.3 Molecule drifting

Using the ranking method as the selection operator, the molecule lists with the minimum corresponding makespan are selected for each generation. If a molecule list ML_1 has reached local optimum during the molecule walking procedure and the corresponding makespan of the molecule list is among the best of all the molecule lists, the molecule list ML_1 will be selected and it will remain in the molecule bank. However, in the next generation, since the molecule list ML_1 has already reached a local optimum, the offspring of the molecule list ML_1 usually have a longer makespan than the makespan of it. Therefore, there is very little chance for the offspring of the molecule list ML_1 to be selected into the molecule bank. Moreover, since ML_1 has already reached a local optimum, there is only a small chance to improve ML_1 in the molecule walking process. To diversify the molecule lists selected and to improve the chance of finding better solutions in the molecule walking process, the last local move of the molecule walking procedure will always be accepted even if the makespan of the new molecule list is longer than the makespan of the original molecule list. This new technique is called molecule drifting.

The best solutions found are still recorded during the molecule walking process with molecule drifting. With molecule drifting, the molecule lists have a greater chance to drift from the local optimal states. As the drifted molecule lists may have a longer makespan, there is a greater chance for their offspring to be selected for the next generation. When drifting from the local optimum, the chance to find better solutions in the molecule walking procedure for the next generation of molecule lists is increased as well.

4.3 Computational study on Molecule Bank Algorithm

Similar to MS, the J30, J60 and J120 test sets from PSPLIB are also used here as the standard benchmark to test the performance of MBA. The computational study is divided into 3 parts. Detailed experimental results for MBA on the J30, J60 and J120 test sets with the SGS calls limit as 1000, 5000, 50000 and 500000 are reported in the first part, where 62 new best results for the J120 test sets and 3 new best results for the J60 test sets are obtained. In the second part, we compare our MBA with all the other state-of-art heuristics reported in the 2005 survey [67] by Kolisch and Hartmann, including the MS proposed by us. The experimental results show that the MBA is among the best heuristics for RCPSP so far. Finally, with the help of

MBA, better results for finding the float on J30 test set are reported. The Molecule Bank Algorithm is implemented in C++ and compiled with GNU g++ with maximal compiler optimization. All the experiments are carried out on Intel P4 3.0GHZ PC with 1G Memory.

4.3.1 Experimental results of Molecule Bank Algorithm on PSBLIB

As Dynamic Population is used in MBA, we will present the initial population POP_0, the start population POP_1, the ending population POP_e and the population decreasing constant D_{POP} for 1000, 5000, 50000 and 500000 SGS calls limit first in Table 4.1.

Table 4.1 Population setting for different SGS limits

SGS limit	POP_0	POP_1	POP_e	D_{POP}
1000	50	4	4	1
5000	100	12	4	1
50000	100	55	7	1
500000	1000	1000	350	50

In Table 4.2, detailed computational results for MBA on PSBLIB are presented, which include the results on the J30, J60 and J120 test sets with 1000, 5000, 50000 and 500000 SGS limits. The average and maximum deviation from the optimal solutions of the J30 set and the average and maximum deviation from the best solutions of the J60 and J120 are given in Table 4.2, where the best results of PSPLIB updated on August 22, 2005 are used as comparison here. In Table 4.2, we also report the number of best solutions and the number improved solutions achieved by MBA. With 500000 SGS limits, MBA achieved all the optimal solutions for the J30 test set, 93% of the best solutions for the J60 test set and 81% of the best solutions for the J120 test set. In addition, MBA also obtained 3 new best results for the J60 test set and 62 new best results for the J120 test sets. The average and maximum running time of MBA is given in the last two columns. We can also find that MBA finished running in a short time, where, even for J120 test set with 500000 SGS calls, the average running time is only 30.65 seconds.

Table 4.2 Experimental results of Molecular Bank algorithm on PSBLIB

			Deviation (%)		No. of	No. of	CPU-time (sec)	
Test set	Schedules	Sum	avg.	max.	best	improved	avg.	max.
J30	1000	28389	0.21	5.26	436(480)	-	0.01	0.03
	5000	28342	0.07	3.45	460(480)	-	0.05	0.12
	50000	28318	0.01	1.25	478(480)	-	0.40	1.04
	500000	28316	0.00	0.00	480(480)	-	4.79	12.33
J60	1000	38766	11.73	115.58	363(480)	0	0.02	0.05
	5000	38570	11.14	110.39	378(480)	0	0.08	0.29
	50000	38395	10.63	102.60	421(480)	1	0.75	2.94
	500000	38353	10.51	101.30	446(480)	3	7.40	28.28
J120	1000	76588	34.95	212.87	199(600)	0	0.07	0.12
	5000	75338	32.75	204.04	228(600)	0	0.32	0.54
	50000	74188	30.71	197.98	348(600)	11	3.10	5.78
	500000	73729	29.91	194.95	484(600)	62	30.65	56.50

4.3.2 Comparing with other heuristics

In Table 4.3, we compare MBA with 20 best heuristics on RCPSP, where the results of MS are also included. All the methods are ranked according to the results on the J120 test set with 50000 SGS calls, and ties are broken according to the results on the J120 test set with 5000 SGS calls. As shown in Table 4.3, MBA achieved best results on the J120 test set and the J60 test set. MBA also achieved second best results on the J30 test set. The experimental results show that by balancing diversification and intensification during the search process, MBA converged very well to high quality solutions in a short time.

Table 4.3 Comparing MBA with other heuristics for RCPSP with schedule limit

		J30		J60		J120	
Algorithm	Reference	5,000	50,000	5000	50000	5000	50000
MBA	Ours	0.07	0.01	11.14	**10.63**	32.75	**30.71**
GA – hybrid, FBI	[114]	0.06	0.02	**11.10**	10.73	**32.54**	31.24
GA – forw.-backw., FBI	[2]	0.06	0.03	11.19	10.84	33.91	31.49
Scatter Search – FBI	[35]	0.11	0.01	**11.10**	10.71	33.10	31.57
GA – FBI	[116]	0.20	0.02	11.27	10.74	33.24	31.58
MS	Ours	0.11	0.03	11.33	10.85	33.54	31.97
GA, TS – path relinking	[61]	**0.04**	**0.00**	11.17	10.74	33.36	32.06
population-based – FBI	[116]	–	–	–	–	34.02	32.81
GA – self-adapting	[52]	0.22	0.08	11.70	11.21	35.39	33.21
sampling – LFT, FBI	[112]	0.13	0.05	11.62	11.36	34.41	33.71
ant system	[82]	–	–	–	–	35.43	–
GA – activity list	[52]	0.25	0.08	11.89	11.23	36.74	34.03
sampling – LFT, FBI	[111]	0.17	0.09	11.82	11.47	35.56	34.77
sampling – LFT, FBI	[110]	0.16	0.07	11.87	11.54	35.81	35.01
GA – forw.-backward	[57]	0.12	–	11.86	–	36.57	–
TS – activity list	[90]	0.16	0.05	12.18	11.58	37.88	35.85
GA – late join	[29]	0.33	0.16	12.63	11.94	38.41	36.44
sampling – random, FBI	[116]	0.28	0.11	12.35	11.94	37.47	36.46
SA – activity list	[14]	0.23	–	11.90	–	37.68	–
GA – priority rule	[52]	1.12	0.88	12.74	12.26	38.49	36.51
sampling – adaptive	[105]	0.44	–	12.58	–	38.70	–
sampling – LFT, parallel	[64]	1.29	1.13	13.23	12.85	38.75	37.74

We also compare MBA with other heuristics without schedule limit, and the results are reported in Table 4.4. As shown in Table 4.4, all the best results were achieved by MBA on all the test sets.

Table 4.4 Comparing heuristics for RCPSP without schedule limit

		J30		J60		J120	
Algorithm	Reference	Dev.	CPU	Dev.	CPU	Dev.	CPU
MBA	Ours	**0.00**	4.79	**10.51**	7.40	**29.91**	30.65
Scatter Search – FBI	[35]	0.01	7.16	10.53	19.61	30.48	69.57
MS	Ours	0.01	3.44	10.56	4.65	30.82	21.49
population – based	[115]	0.10	1.16	10.89	3.7	31.58	59.4
decompos. & local opt.	[92]	**0.00**	10.26	10.81	38.8	32.41	207.9
VNS – activity list	[45]	0.01	0.64	10.94	8.89	33.10	219.86
local search – critical	[117]	0.06	1.61	11.45	2.8	34.53	17.0
LR – activity list	[86]	–	–	15.60	6.9	36.00	72.9
TS – network flow	[4]	–	–	12.05	3.2	36.16	67.0
network decomposition	[107]	0.12	2.75	11.61	460.2	39.29	458.5
MP – network flow	[4]	1.74	–	14.20	–	39.34	–

4.3.3 Calculating the float with Molecular Bank Algorithm

A new definition for the float for the resource-constrained project is given in the first part of the thesis. However, calculating the float emerges to be NP-hard. Therefore, Simulated Annealing has been applied to calculate a heuristic value of the maximum float as *H-Float*. Compared with the *E-Float*, we show that using a Simulated Annealing algorithm alone can achieve 91.63% accuracy with 5000 SGS calls, 97.78% accuracy with 50000 SGS calls and 99.60% for 500000 SGS calls. As MBA converges better than SA (Simulated Annealing), we have applied MBA to calculate the float and the results are reported in Table 4.5. We can find in Table 4.5 that 97.52% accuracy is achieved for 5000 SGS calls by MBA already, which is almost the same as the results of SA with 50000 SGS calls. According to the accuracy results, MBA seems to converge 10 times as fast as SA on RCPSP.

Table 4.5 *H-Float* by Molecule Bank Algorithm with optimal deadline for J30

No. of Step		*H-Float* (period)	No. of Crt-Act	Accuracy (%)	Makespan avg(period)	CPU (sec)
5000	avg	6.27	10.23	97.52	58.41	7.43
	max	23.17	28.00	100.00	129.00	25.62
	min	0.13	0.00	46.67	35.00	0.90
	dev	3.84	4.70	7.68	13.61	5.36
50000	avg	6.34	9.92	99.42	58.39	81.55
	max	23.17	27.00	100.00	129.00	285.69
	min	0.33	0.00	66.67	35.00	7.65
	dev	3.77	4.31	3.08	13.59	57.13
500000	avg	6.35	9.87	99.86	58.39	795.28
	max	23.17	26.00	100.00	129.00	2356.38
	min	0.37	0.00	80.00	35.00	71.72
	dev	3.76	4.24	1.32	13.58	571.49

4.4 Conclusion

Molecule Bank Algorithm (MBA), a population based Molecule Search (MS), is proposed here to solve the resource-constrained project scheduling problem (RCPSP). A molecule bank is used to store elite solutions during the search process of MBA. 4 different operators, including the crossover operator, the selection operator, the jumping operator and the walking operator, are used here to generate new solutions based on the elite solutions from the molecule bank. The traditional two-point crossover method is used as the crossover operator, and the ranking method is adopted here as the selection operator. Randomized cooling jumping from MS and annealing-like FBS (Forward-backward search) are selected as the jumping and walking operator for MBA. Three new techniques, dynamic population, molecule aging and molecule drifting are proposed here to further balance diversification and intensification in the search process of MBA. Detailed experimental results on the J30, J60 and J120 test set from PSPLIB are presented, where 62 new best results are obtained for the J120 test set and 3 new best results are obtained for the J60 test set. According to the computational results, MBA emerges to be one of the best heuristics to date for RCPSP. While diversification and intensification play a central role in the searching process of heuristics, more research can be carried out in the future to further investigate how to balance diversification and intensification during the search process to achieve best performance for heuristics.

Chapter 5
A Hybrid Framework for Over-Constrained Generalized Resource-Constrained Project Scheduling Problem

In this chapter we study an over-constrained scheduling problem where constraints cannot be relaxed. This problem originates from a local defense agency where activities to be scheduled are strongly ranked in a preference or priority scheme determined by planners ahead of time and operational real-time demands require solutions to be available almost immediately. The method of solution used by the agency was to model the problem as a *Constraint Satisfaction Problem* (CSP) and to remove some activities to obtain feasible solutions. Because of the operational nature of the problem, any quick improvement in activities scheduled was beneficial to the users where the quality of resulting schedules was measured by the inclusion of high-ranked activities.

Scheduling problems are natural CSPs where conflicts can be resolved by partial satisfaction for over-constrained problems. Fox, in an early study [46], provided the concepts of constraint "relaxation" in the sense of the selection of alternatives, "preferences" among relaxations and "importance" to cope with conflicting constraints in job-shop scheduling. Scheduling finds many applications in services and manufacturing and can be deterministic or stochastic in nature. For a comprehensive survey of scheduling systems see [18][97].

In many real-life applications, conflicting constraints can arise and problems can be over-constrained with no solution. For the over-constrained case, it is necessary to relax the problem to find a good compromise in which some constraints may be violated. Over the past twenty years, many researchers have addressed constraint conflict and over-constrained CSPs. In an early study, Borning et al. [13] introduced constraint "hierarchies" for requirements and preferences which could not all be satisfied. Partial constraint satisfaction was used by Freuder and Wallace [47] for over-constrained problems. They introduced the well-known maximal constraint satisfaction problem (Max-CSP) which provides a framework to minimize the number of violated constraints. Many techniques have since been used for the Max-CSP; these include branch and bound [47], heuristics such as tabu search [48], specific filtering algorithms [95] and a range-based algorithm [96]. See also, [94], [7] and [100]. Although various frameworks derived from Max-CSP have been proposed, most can be encoded through two generic paradigms called valued CSP [104] and semiring CSP [8]. The Max-CSP, however, is not realistic enough for many real-life applications where the objective is much more complex than minimizing the number of violated constraints. In this work, the priority scheme practised by the users led us naturally to address activity priority as a primary objective in scheduling. "Priority" has featured in CSPs as soft constraints which are assigned weights (see, for example, [94],

[42]). In an extension of the Max-CSP, in the so-called *weighted* Max-CSP problem, it is preferable to violate a priority constraint with low weight than one with high weight where the goal is to minimize the sum of weights of the violated constraints. However, for the objective of prioritizing activities, using weighted constraint sets is not effective.

In the over-constrained problem we study, constraints had to be failed in sets rather than individually when constraint satisfaction techniques were used. This involved extensive computation in tracing activity-related constraints and was an underlying weakness of the technique. An alternative was to retain the benefits of constraint programming (CP) techniques but to treat activities as a whole in a hybrid approach. In our approach, we used a two-component hybrid algorithm. Heuristics are used in one component and a CP-based iterative randomized method or a Similar as Minimum Forbidden set [6], Minimal Critical Set (MCS)-based method developed by [25] is used in the other component. The use of heuristics is appropriate since the problem is NP-hard and because of the large problem sizes involved and short operational time requirements of the user. Tests of the algorithm show that significantly larger numbers of high-priority activities are scheduled when compared with the old system. Operational needs were also met and exceeded with the shorter times required to obtain final schedules. To further validate the effectiveness of the hybrid algorithms, performance was compared with solutions obtained using the CPLEX solver. The experimental results verify that for small problem sizes, the algorithm obtains near-optimal solutions in short times and for larger problems, it obtains better results in much shorter times than the branch and bound-based CPLEX solver.

The approach provided in this chapter offers a framework for problems where all constraints are treated as hard constraints and where conflict resolution is achieved only through the removal of variables rather than constraints. In this study, a priority scheme is used to achieve the latter. Although the over-constrained generalized resource-constrained project scheduling problem is addressed here, the framework can be applied to over-constrained resource-constrained project scheduling problems and over-constrained job-shop scheduling problems.

5.1 Basic Notions and Problem Description

The well-known Critical Path Method (CPM) and Program Evaluation and Review Technique (PERT) are concerned with minimizing the makespan of a project where resources are sufficient and precedence relations ensure events are completed before others are initiated. Such precedence relations have been generalized and, following [44], we distinguish between four types of generalized precedence relations (GPRs): start-start (SS); start-finish (SF); finish-start (FS) and finish-finish (FF). GPRs can be used to model a variety of problem characteristics which can be found in, for example, [31, 43, 85, 10, 87, 89, 17]. We note that precedence relations can also be described by precedence constraints [59, 122], precedence diagramming relations [85], minimal and maximal time lags [87, 89, 17] and generalized precedence constraints [119].

5.1.1 The RCPSP and resource allocation types

The resource-constrained project Scheduling problem (RCPSP) is concerned with scheduling project activities subject to precedence and resource constraints and can be described as follows: A number of activities are defined in the project network, assumed to be acyclic and represented by an activity-on-node network with nodes as activities and arcs representing finish-start precedence relations with zero time lag. Pre-emption is not allowed and there are a number of renewable resource types with constant resource requirements. The objective is to find a minimal makespan (the completion time of the last activity scheduled of the network) which provides a schedule that meets the constraints imposed by the precedence relations and resource availability. For a formulation of the RCPSP see, for example, [101].

Branch-and-bound methods have been used to solve this problem by [22, 28, 37, 93, 109] while priority rule-based heuristics have been implemented and tested experimentally by [3, 11, 34, 93]. Recently, there has been more elaborate heuristics applied to this problem including a truncated branch-and-bound heuristic by [3] and integer programming-based heuristics by [91]. To solve the RCPSP, we must satisfy temporal relationships between activities as well as resource requirements for each activity, where resource types can be renewable, non-renewable and doubly-constrained. In this work we only consider renewable resources where a pre-specified number of units of a resource is available for every period of the planning horizon.

In 1983, [10] showed that, as a generalization of the job-shop scheduling problem, the RCPSP is **NP**-hard in the strong sense. The RCPSP has many extensions and variations which include the Time-Constrained Project Scheduling Problem, the Resource-Leveling Problem, the Resource Investment Problem, the Net Present Value Problem, the Weighted Tardiness Problem and the Generalized resource-constrained Project Scheduling Problem (GRCPSP). Recent surveys on the RCPSP can been found in [19, 54].

5.1.2 The GRCPSP

Research on the RCPSP has been extensive. However, in a number of cases, assumptions made limit the model's applicability to realistic problems. An example of such a limitation is the consideration of only simple finish-to-start precedence relations which makes it impossible for the parallel processing of activities. Also, RCPSP formulations assume constant availability of resources throughout the planning horizon whereas in practice, the availability of resources fluctuates due to, for example, the planned maintenance of equipment, vacations or varying participation of workers and machines. Further to these, the release and deadline for a single activity cannot be considered within a RCPSP although it constitutes an important factor in modeling certain types of problems. These limitations are overcome in the GRCPSP where the problem assumes variable resource availability and attempts to find the makespan where precedence relations have minimal and maximal time lags and activity release dates and deadlines. A formulation of the GRCPSP can be found in [54].

5.1.3 The Over-Constrained GRCPSP

In both the RCPSP and the GRCPSP, it is common practice to minimize the makespan. However, we found that this objective was not suitable for applications sometimes found in industry and, in particular, the problem from a local defense agency. As described in the introduction, in our problem resources are atypically very limited and the consumption of resources by activities typically exceed what is available. Further, in the problem, additional activity requirements can come from operational managers who vaguely understand the presence of temporal constraints. The problem is over constrained both in resources and in temporal relations and must be relaxed. In deciding which constraints to relax or remove, the possibility of extending activities and increasing the availability of resources was ruled out by the nature of the problem at hand and the requirements of the user. The only acceptable solution then was to remove some activities and the respective constraints. As each activity had a level of importance, retaining activities with higher importance became the primary objective while attempting to minimize the loss of scheduled activities.

Priority-Based Activity Scheduling Sequence (PBASS): There are number of ways to derive an activity scheduling sequence; these include using network-based rules, critical path-based rules, resource-based rules and composite rules (see, for example, [3, 63]). In problems with over-constrained temporal relationships between activities, some activities must be removed to obtain a feasible schedule. When an activity is removed from a problem, temporal relationships can change significantly which renders scheduling sequences topologically unsuitable. An simple example can be a cycle between activity A, B and C. When removing one activity, the temporal constraints are feasible. However, the order of the sequence depends on the activity removed. To address this, we introduce a priority-based activity schedule where activities with higher priority precede activities with lower priority resulting in a sequence of priority non-increasing activities. We call such a sequence a Priority-Based Activity Scheduling Sequence (PBASS). For example, an activity sequence 1 2 3 with priorities 3 1 2 respectively is represented by the PBASS 2 3 1. Here, we have taken a lower numerical value priority value to be of higher priority. Clearly, a PBASS need not be a feasible schedule.

Once a PBASS (or any schedule) is made feasible, we call it a *Scheduled Activity Sequence* (SAS). The SAS is the activity sequence where all the activities in the sequence can be scheduled. In other words, there exists a baseline schedule for the activities inside the SAS satisfying the resource constraints and temporal constraints. In order to differentiate between the quality of SASs, we consider the priority of each activity in the SAS one by one. An SAS with more high priority activities scheduled will be the better SAS. For example, between two SASs: A: 1 2 2 3 4 and B: 1 2 3 3 4, A is a better solution since it scheduled two activities with priority 2. Priority of activities scheduled are also given importance over the total number of activities scheduled. For example, between C : 1 2 3 and D: 1 3 3 3 3, we would consider C a better solution because it scheduled an activity with priority 2 although D scheduled more activities.

Assuming that $\mathcal{S}$ is the the space of all possible activity sequences (feasible or infeasible), and P is the number of all possible priorities p_1, $p_2 ... p_P$, we define a priority value map $\pi : \mathcal{S} \rightarrow Z_+^P$ by $\pi(s) = (\#p_1 s, \#p_2 s, ..., \#p_P s)$ where s is an activity sequence, Z_+ is the set of non-negative integers and $\#p_i s$ is the number of priority p_i activities in s. As an example, suppose, we have 6 activities with 3 different priorities: activities 1 and 3 have priority 1; activities 2 and 4 have priority 2 and activities 5 and 6 have priority 3. For the schedule (an SAS say) 1 3 2 4 5 6, we then have π(1 3 4 5 6) $= (2, 2, 2)$. We now define an ordering in Z_+^P. For $a, b \in Z_+^P$

$$(a, b) = \begin{cases} a \succ b & \exists i \leq P, \text{ where } a_i > b_i \text{ and } a_j = b_j \text{ for } \forall j, j < i, \\ a \equiv b & \forall i \leq P, \; a_i = b_i, \\ a \prec b & \exists i \leq P, \text{ where } a_i < b_i \text{ and } a_j = b_j \text{ for } \forall j, j < i, \end{cases} \tag{5.1}$$

As an example, suppose $a = (1, 3, 2, 4, 0)$ and $b = (1, 3, 1, 8, 0)$, $a \succ b$, since $a_1 = b_1, a_2 = b_2$, and $a_3 > b_3$. As each SAS can be mapped under the priority value map, π, the objective for the OGRCPSP can be expressed using the given ordering on

Z_+^P. We adopt Demeulemeester et al.'s [38] model for GRCPSP, to model the **The OGRCPSP model**

$$Maximize_{s \in SAS} \pi(s) \tag{5.2}$$

subject to:

$$S_i + SS_{ij} \leq S_j \text{ for all } (i,j) \in H1 \tag{5.3}$$

$$S_i + SF_{ij} \leq F_j \text{ for all } (i,j) \in H2 \tag{5.4}$$

$$F_i + FS_{ij} \leq S_j \text{ for all } (i,j) \in H3 \tag{5.5}$$

$$F_i + FF_{ij} \leq F_j \text{ for all } (i,j) \in H4 \tag{5.6}$$

$$F_0 = 0 \tag{5.7}$$

$$S_{n+1} = Period \tag{5.8}$$

$$S_i \geq G_i \text{ for } i = 1, 2, ..., n \tag{5.9}$$

$$F_i \leq H_i \text{ for } i = 1, 2, ..., n \tag{5.10}$$

$$F_i \leq Period \text{ for } i = 1, 2, ..., n \tag{5.11}$$

$$\sum_{i \in S_t} r_{ik} \leq a_{kt} \text{ for } k = 1, 2, ..., K \text{ and } t = 1, 2, ..., F_n \tag{5.12}$$

where:

SS_{ij}	time lag of start-start relations between pairs of activities, i, j
SF_{ij}	time lag of start-finish relations between pairs of activities, i, j
FS_{ij}	time lag of finish-start relations between pairs of activities, i, j
FF_{ij}	time lag of finish-finish relations between pairs of activities, i, j
$H1$	the set of pairs of activities indicating start-start relations with a time lag of SS_{ij}
$H2$	the set of pairs of activities indication start-finish relations with a time lag of SF_{ij}
$H3$	the set of pairs activities indicating finish-start relations with a time lag of FS_{ij}
$H4$	the set of pairs of activities indicating finish-finish relations with a time lag of FF_{ij}

S_i	the start time of activity i
F_i	the finish time of activity i
G_i	the ready time of activity i
H_i	the deadline of activity i
$Period$	the given period to schedule all activities
K	the number of types of resources available
r_{ik}	the demand of renewable resource of type k for activities $i \in S_t$
a_{kt}	the availability of renewable resource type k in period $(t-1, t]$
S_t	the set of activities in process in time interval $(t-1, t]$

As described, the objective (1) is to maximize the priority value under π of all SAS's. (2) - (5) give temporal constraints between the activities. For simplification, two dummy activities are defined in (6) - (7); the first (activity 0) will finish at time 0, and the second (activity $n+1$) will start at the end time of the project. (8) - (10) ensure that start times should occur after ready times. The end time for each activity should be before its deadline and the end of the project. (11) specifies that resource utilization in any interval $(t-1, t]$ cannot exceed resource capacity.

5.2 Solution approach for the OGRCPSP - A Hybrid Framework

To solve the problem, we propose a two-level hybrid framework. In a high-level component, heuristic search methods are used to find a good PBASS and in a lower-level component, temporal and resource conflicts are resolved and suitable SAS' s are generated from PBASS' s. The framework is illustrated in Figure 5.1. As shown in

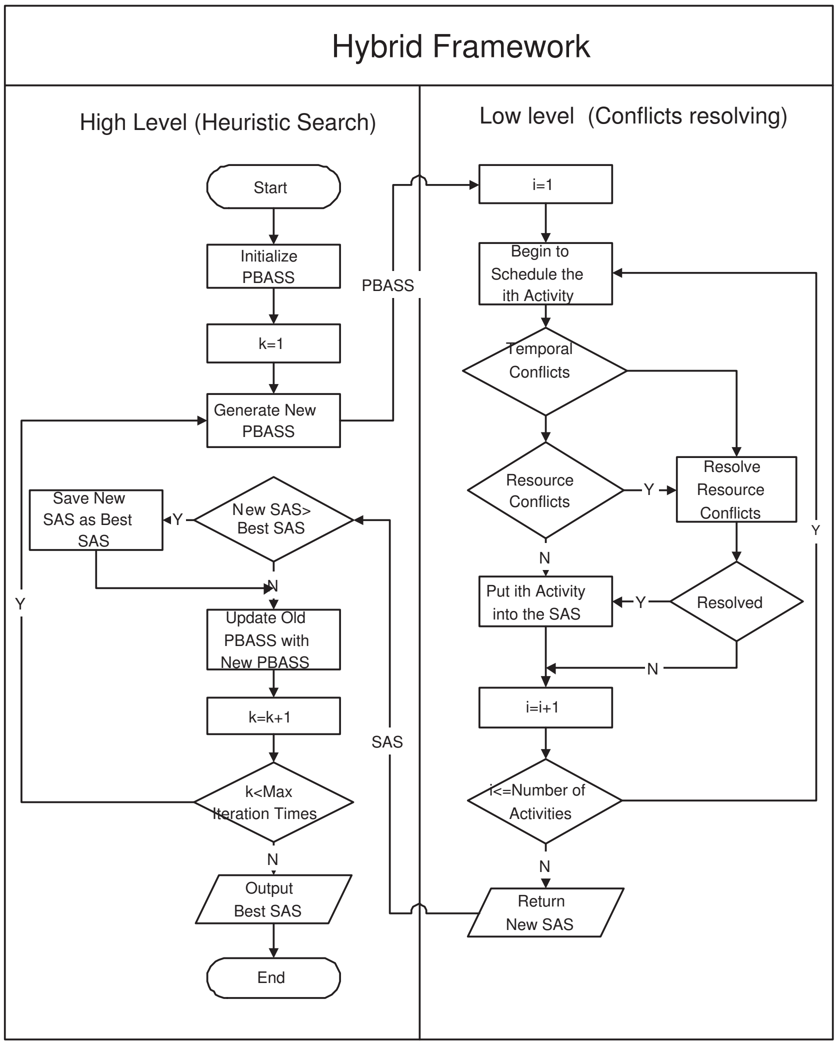


Figure 5.1 Hybrid Framework

Figure 5.1, the high-level component searches for a PBASS which it passes to the lower-level component for resource and temporal conflict resolution, if any, and a search for the best SAS. The result is returned to the high-level search component as the current schedule. We attempt to insert activities from the PBASS one by one into the SAS. When any current activity has a temporal or resource conflict with other activities inserted into the SAS before it, this current activity is dropped. That is to say, activities in the PBASS before the current activity always have priority to remain in the SAS. In the high-level component, we used Tabu Search and Genetic Algorithm methods and, in the low level component, a CP-based iterative randomized algorithm together with a Minimal Critical Set (MCS)-based method is used to resolve temporal and resource conflicts.

5.2.1 High-level Heuristic Search

A simple example 5.1 is used to illustrate the high-level Tabu Search and Genetic Algorithm and the low-level CP and MCS method. The parameters for the example are given in Table 5.1. In this example, we have six activities and one resource, and

Table 5.1 Example 5.1: activities parameters

Activity	Priority	Duration	Release Time	Deadline
1	1	5	2	9
2	2	3	3	8
3	1	6	2	10
4	3	3	1	7
5	2	5	2	10
6	3	4	6	10

all the activities will be scheduled in a period of 9 units (1-10). The lower the priority number, the more important it is. The temporal relationship between activities is given below in Table 5.2, where a minimal time lag specifies that an activity can only start (finish) when the preceding activity has already started (finished) for a certain time period and a maximal time lag specifies that an activity should be started (finished) at the latest by a number of time periods beyond the start (finish) of another activity. All relationships are represented by the start-start model In

Table 5.2 Example 5.1: activities temporal relationships

Between Activities	Minimal time lag	Maxmal time lag
(1,2)	1	4
(1,3)	-2	3
(2,4)	4	7
(3,5)	0	8

Table 5.2, the minimal time lag between activities 1 and 2 is 1, which means activity 2 should start 1 time unit after activity 1 has started. The maximal time lag between activities 1 and 2 is 4, which means activity 2 should start at most 4 time units after activity 1 starts. These relationships are illustrated in Figure 5.2.

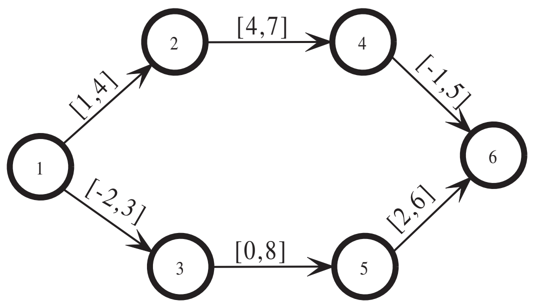


Figure 5.2 Example 5.1: activities temporal relationship

Tabu Search

Tabu search (TS) [50] involves initial solutions, tabu moves, neighborhood structures, tabu tenure, tabu memory and stopping rules. Here, an initial PBASS is generated randomly with a decreasing priority sequence while moves in the TS are defined by the swapping of a pair of activities with the same priorities in the PBASS. We take tabu tenure to be $\sqrt{\frac{n}{p}}$, where n is the number of activities and p is the number of different priorities. A short-term memory is used to prevent the search from being trapped in local optima, where current swaps are forbidden for the duration of tabu tenure. The algorithm terminates after a fixed number of iterations, set to 100 iterations for problems tested:

Algorithm 17 High Level-Tabu Search

```
Initialize(PBASS)
for $i \leftarrow 1$ to max_iterate_times do
  for $j \leftarrow 1$ to number_of_activities do
    for $k \leftarrow j+1$ to number_of_activities do
      if $activity[k]\_piority = activity[j]\_priority$ AND $pair(k,j) \leq i$ then
        swap the labeling activity[k] and the labeling of activity[j] in PBASS as the New_PBASS
        $New\_SAS \leftarrow LowLevel(New\_PBASS)$ (Pass the New_PBASS to low level conflict resolution process which returns the New_SAS)
        if $New\_SAS > Best\_New\_SAS$ then
          $Best\_New\_SAS \leftarrow New\_PBASS$,$Best\_New\_PBASS \leftarrow New\_PBASS$, $pair(k,j) \leftarrow i + tabu\_tenure$ (Save the best neighbourhood)
        end if
      end if
    end for
  end for
  if $Best\_New\_SAS > Best\_SAS$ then
    $Best\_SAS \leftarrow Best\_New\_SAS$ (Save the best solution)
  end if
  $PBASS \leftarrow Best\_New\_PBASS$ (update current PBASS with its best neighbourhood)
end for
```

. For the example given, an initial PBASS is 3 1 5 2 6 4 obtained from a quick sort. The SAS for this sequence is 3 6 with the start times for activities 3 and 6 at 2 and 6, respectively. Here, the three neighbors for the initial PBASS are: 1 3 5 2 6 4; 3 1 2 5 6 4 and 3 1 5 2 4 6, with SASs: 1 2 4, 3 6 and 3 6. The best neighbor is therefore 1 3 5 2 6 4, so the TS search will swap activity 3 and 1 in the next move to obtain 1 3 5 2 6 4.

Genetic Algorithm

We employ a genetic algorithm (GA) [55], in our search. Here, PBASS's are naturally mapped to chromosomes. In implementation, the initial population for the group is selected randomly by the rule for priority sequences which stipulates that one which has a higher priority will always be ahead of one with a lower priority. For *crossover* operations, we use two crossover methods:

A. *Priority-based crossover*: Suppose we have two parent sequences: 1 3 2 5 4 6 (father) and 3 1 5 2 6 4 (mother) and for these there are three different priorities. Activities 1, 3 are priority 1; activities 2, 5 are priority 2 and activities 4, 6 are priority 3. We have 3 checkpoints that are after the positions 2 and 4 and 6. From these three checkpoints we randomly choose two and perform a crossover. If we chose position 2 and 6, then we obtain daughter and son sequences as shown in Figure 5.3. The daughter inherit the string of genes outside the checkpoint from the father and the string between the checkpoints from mother, while the son will inherit the string between the checkpoint from the father and the string outside the checkpoints from the mother.

B. *Mid-point crossover*: We set the middle point as the checkpoint for the crossover. The daughter inherits the first string from the father and the second string from the mother, while the son inherits the first string from the mother and the second string from the father. The crossover process in shown in Figure 5.3. In our algorithm the two crossover methods are chosen randomly. There will be *PopulationSize* crossovers in each generation, where *PopulationSize* is the population of each generation. The parents for each time are randomly selected, so that there will be 2×*PopulationSize* new chromosomes in the next generation.

For *mutation* operations, we used *k*-swap mutation, where *k* is determined by *num-*

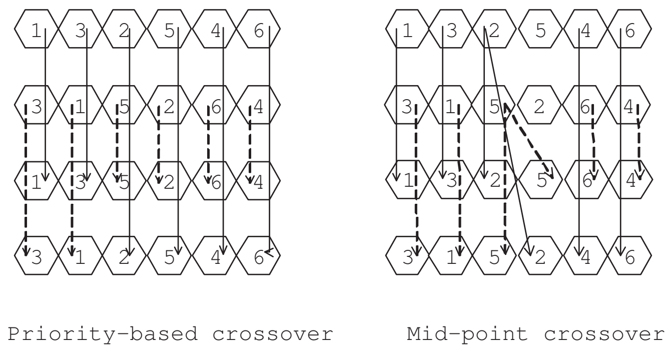


Figure 5.3 Crossover of the Genetic Algorithm

ber_of_activities × *mutation_rate*, where the later *mutation_factor* is predetermined. In each swap, we randomly pick up two activities with the same priority and swap their positions. For example, if for the chromosome 1 3 2 5 4 6, we choose to swap activity 4 and activity 6 which have the same priority, the resulting sequence will be 1 3 2 5 6 4. *Chromosome fitness:* For each PBASS, an SAS is produced from the low-level component of the algorithm. As each chromosome is mapped to a PBASS, we can define chromosome fitness using the map π. The PBASS for the chromosome

3 1 5 2 6 4 is 3 1 5 2 6 4 and after low-level scheduling we get the SAS 3 6 where the $\pi(3\ 6) = (1, 0, 1)$. However, for the chromosome 1 3 5 2 6 4, the corresponding PBASS is 1 3 5 2 6 4 with an SAS 1 5 4 6 where $\pi(1\ 5\ 4\ 6) = (1, 1, 2)$. Hence, the latter chromosome is deemed to be more fit.

Selection method: We retain the group population of the chromosome for each generation and in every generation there will be $2 \times \mathit{PopulationSize}$ new chromosomes from crossovers and mutations. We select the *PopulationSize* from the group to begin the next generation. There are selection methods such as the ranking method (see, for example, [83]), which sort chromosomes with respect to their fitness values and select the best ones while others are deleted from the population. We used the ranking method here.

An outline of the GA is given below:

Algorithm 18 High Level-Genetic Algorithm

```
Randomly generate a set of PBASS's and encode initial chromosomes
for $i \leftarrow 1$ to $PopulationSize$ do
  $SAS[i] \leftarrow LowLevel(PBASS[i])$ (SAS[i] is the fitness value for the PBASS[i])
end for
while $MaximumGeneration \geq$Generation do
  for $i = 1$ to $Population\_Size$ do
    if $rand(0,1) \leq crossoverrate$ then
      Pick up two PBASS' and perform crossover (two new PBASS' will be generated)
    end if
    mutation : perform $rand(0,1) \times number\_of\_activities$ swaps on the new PBASS
  end for
  for $i = 1$ to $Number\_of\_New\_PBASS$ do
    $New\_SAS[i] \leftarrow LowLevel(New\_PBASS[i])$
  end for
  Merge the new PBASS' with old PBASS'
  Select fittest PBASS from all the Merged_PBASS' as the next generation
end while
```

5.2.2 Low-level conflict resolution

The low-level component is used to find SASs from PBASSs by resolving temporal and resource conflicts and to return the start time for each activity scheduled. The Floyd-Warshall algorithm [73] is used detect temporal conflicts and to resolve resource conflicts, two methods are used. The first is a CP-based iterative method to reduce domains by removing temporal conflicts. This is accomplished by an iterative randomized algorithm which finds a suitable start time for each activity. The second is based on the notion of a minimal critical set (MCS)[72] which resolves resource conflicts by delaying some activities. We show how these methods work with an example in a later section.

Detecting temporal conflicts

We schedule activities one by one as in the PBASS and when any activity is found to have temporal conflicts with preceding activities that activity is discarded. This is carried out in both the CP-based and MCS methods described below.

In order to do this, we need a method to detect temporal conflicts. There are two kinds of temporal conflicts in our case. The first is when there are cycles between activities. For example, activity 1 must take place 2 time units after activity 2, activity 2 must take place 2 time units after activity 3 and activity 3 must take place 2 time units after activity 1, then activity 1 has to take place 6 time units after activity 1. Next, as the period to schedule all activities is fixed, the domain for activities can become empty in view of temporal constraints. For example, take the domain of start times of activity 1 to be {2,3}, where the duration for activity is 1 and the duration of activity 2 is 2 with scheduling period of 4. We find that the latest finish time of activity 1 is 2 and its latest start time is 1. Hence in scheduling activity 2, domain for activity 1 will become empty.

To detect cycles, we first transform all the temporal relationships into SS (Start after Start) precedence relations, using the following transformation rules given by

[?, 101].

$$S_i + SS_{ij} \quad \leq \quad S_j \rightarrow S_i + l_{ij} \leq S_j \;\; with \;\; l_{ij} = SS_{ij} \tag{5.13}$$

$$S_i + SF_{ij} \quad \leq \quad F_j \rightarrow S_i + l_{ij} \leq S_j \;\; with \;\; l_{ij} = SF_{ij} - D_j \tag{5.14}$$

$$F_i + SF_{ij} \quad \leq \quad F_j \rightarrow S_i + l_{ij} \leq S_j \;\; with \;\; l_{ij} = D_i + FF_{ij} - D_j \tag{5.15}$$

$$F_i + FS_{ij} \quad \leq \quad S_j \rightarrow S_i + l_{ij} \leq S_j \;\; with \;\; l_{ij} = D_i + FS_{ij} \tag{5.16}$$

where, D_i is the duration for activity i. By adding a dummy activity 0, with finish at time 0, we can transform temporal relationships into a graph G by making the first node a dummy and the nodes corresponding start times. The edges and the weight for the edges of the graph are given by:

$$a_{0j} \quad = \quad G_j \;\; j = 1, ..., k, \;\; where G_j \;\; is \; the \; ready \; time \; for \; activity \; j \tag{5.17}$$

$$a_{ij} \quad = \quad 0 \;\; when \;\; i = j \tag{5.18}$$

$$a_{ij} \quad = \quad l_{ij} \;\; i > 0, j > 0 \;\; and \;\; if \;\; S_i + l_{ij} < S_j \tag{5.19}$$

$$a_{ij} \quad = \quad -\infty \;\; otherwise \tag{5.20}$$

Bartusch et al. [6] have shown that by applying the Floyd-Warshall algorithm [73] to G, there must exist some $a_{ii} > 0$ if there are temporal cycles between activities. If there are no cycles, we can easily find the EST (Earliest Start Times) for activities 1...k, where EST = $(a_{01} \; a_{02} ... a_{0k})$. Similarly, we can transform temporal relationships into FF (Finish after Finish) relationships:

$$F_i + FF_{ij} \leq F_j \rightarrow F_i + l_{ij} \leq F_j \;\; with \;\; l_{ij} = FF_{ij} \tag{5.21}$$

$$F_i + FS_{ij} \leq S_j \rightarrow F_i + l_{ij} \leq F_j \;\; with \;\; l_{ij} = FS_{ij} + D_j \tag{5.22}$$

$$S_i + SF_{ij} \leq F_j \rightarrow F_i + l_{ij} \leq F_j \;\; with \;\; l_{ij} = SF_{ij} - D_j \tag{5.23}$$

$$S_i + SS_{ij} \leq S_j \rightarrow F_i + l_{ij} \leq F_j \;\; with \;\; l_{ij} = D_j + SS_{ij} - Di \tag{5.24}$$

By adding a dummy activity A_{n+1}, with the finish at time $Period$, we can transform the above relationship into another graph $G^{'}$ by making the first node A_{n+1} a dummy and other nodes to correspond to deadlines. The edges and the weight for the graph on the reverse direction as the one shown above is:

$$b_{0j} = Period - H_j \; j = 1, ..., k, H_j \; is \; the \; deadline \; for \; activity \; j \tag{5.25}$$

$$b_{ij} = 0 \;\; when \;\; i = j \tag{5.26}$$

$$b_{ji} = l_{ij} \;\; i > 0, j > 0 \;\; and \;\; if \;\; F_i + l_{ij} \leq F_j \tag{5.27}$$

$$b_{ij} = -\infty \;\; otherwise \tag{5.28}$$

Similar to the above, by applying the Floyd-Warshall algorithm to $G^{'}$, we can find the LFT (Latest Finish Times) for activities 1...k, where LFT = $(b_{01} \; b_{02}...b_{0k})$ and the LST can be computed by: LST$_i$ = LFT$_i - D_i$. If $\exists i \leq k$, such that EST$_i \geq$ LST$_i$, the start time domain for activity i is empty and this activity has to be discarded. To illustrate the above, we transform the example 5.1 from section 5.2.1 with temporal relationships shown in Figure 5.2, into a start time graph G and a finish time graph G' as shown in Figure 5.4.

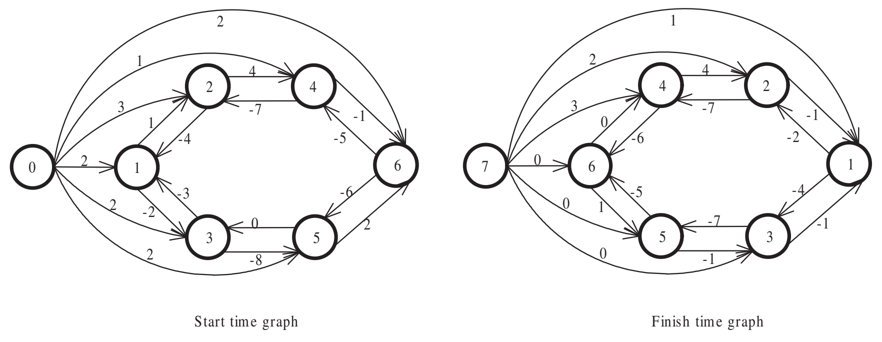


Figure 5.4 Transformation graphs

As in section 5.2.1, the priorities for the 6 activities are: 1 2 1 3 2 3. There are no positive cycles here and the EST is (2,3,2,7,2,4) and the LFT is (4,3,10,7,9,10). We can compute the LST as (-1,0,4,4,4,6). It is easy to find that activities 1, 2 and 4 have empty domains. In fact using the PBASS 3 1 5 2 6 4 to schedule all activities, we find conflict only takes place when we schedule activity 4. By removing activity 4, we can find a schedule without temporal conflict.

Using a CP-based iterative randomized algorithm

Reducing the domain for each activity makes it easier to resolve resource conflicts by applying the iterative randomized algorithm given below. Each time we add a new activity, the domain of the start time of the activities will be reduced. Finally, the activity that has been accepted can start any time in the domain of its start time. Because there are usually many activities and the domain size of the start time may be large, we cannot perform a complete search on the start time domain quickly. To determine the actual start time of each activity and to resolve the resource conflicts, we use a simple but efficient iterative randomized algorithm. A start time is picked randomly for the current activity and tested for resource conflict. If there is none, the next activity is tested. If there is, we randomly pick a start time for the current activity until we find a suitable start time or until we have tried a maximum number of times (*starttime_rand_time*), after which the current activity is removed from the scheduled sequence and the next activity attempted. Because start times for preceding activities have a large influence on the start time of the current activity, the entire process is restarted *main_rand_time* times. From this, the best SAS is recorded. This algorithm is outlined below, where the *starttime_rand_time* and the *main_rand_time* are set to be 10 and 100 respectively.

Note: A local search for this purpose is difficult since simple local moves may not resolve any resource conflicts.

Using MCS

In the MCS method, resolving the temporal conflicts is similar to the CP-based method. There are two differences, however. The first is when we add an activity into an SAS, we resolve temporal conflicts first then resolve resource conflicts. The second difference is that in the MCS method we attempt to start the activities as

Algorithm 19 Iterative Randomized Algorithm

```
for $k1 \leftarrow 1$ to $main\_rand\_time$ do
  for $i \leftarrow 1$ to $number\_of\_activity$ do
    Randomly select the start time of activity $i$ in the domain $[a_{0i},c_{0i}]$
    $k2 \leftarrow 1$
    while $k2 \leq starttime\_rand\_time$ AND exist(resource_conflict) do
      Randomly select the start time of activity $i$ in the domain $[a_{0i},c_{0i}]$
      $k2 \leftarrow k2 + 1$
    end while
    if $exist(resource\_conflict)$ then
      Put activity i into the Current_SAS
      Reduce the domain for activity $i+1$ to $n$ with $S_i$
    end if
  end for
  if $Current\_SAS < Current\_Best\_SAS$ then
    $Current\_SAS \leftarrow Current\_Best\_SAS$
  end if
end for
```

early as possible instead of recording the domain of start times. If there are resource conflicts with the current start time of the activities, we try to delay some activities to resolve these conflicts. Here it is difficult to make the right choices on delays. To overcome this problem, Laborie et al. [72] suggested the use of an MCS which specifies a set of activities that simultaneously require a resource r_k with a combined capacity requirement larger than c_k, such that the combined capacity requirement of any subset is less than or equal to c_k. The important advantage in using MCSs is when we set precedence relations between any pair of activities in the MCS, resource conflicts will be resolved. However, there may be an exponential number of MCSs in a set of activities. [25] proposed a linear sampling heuristic. In this approach, activities are considered sequentially and inserted into a queue Q until the sum of the resource requirements exceeds the resource capacity. Q is collected in a list of MCSs and the first element of Q is removed and further activities added to Q to get more MCSs. To quantify temporal flexibility, a heuristic estimator, K, suggested by [72] is used, which, given a precedence constraint set, $\{pc_1, pc_2, .., pc_k\}$, could be defined by: $\frac{1}{K(MCS)} = \sum_{i=1}^{k} \frac{1}{1+commit(pc_i)-commit(pc_{\min})}$, where, $commit(pc_i)$ has a value between 0 and 1 and estimates the loss in the temporal flexibility when posting pc_i and where $pc_{\min}$ is the precedence with the minimum value of $commit(pc_i)$. The higher the value for $K(MCS)$, the more critical the MCS set will be. The rationale here is that if the solution becomes further constrained by resolving a more temporally flexible MCS, the probability increases that less temporally-flexible MCSs will eventually reach an irresolvable state. [25, 26] defined an acceptance band, β and consider the set MCSs satisfying $K_{\max}(1-\beta) \leq K(MCS) \leq K_{\max}$, where $K_{\max}$ is the maximum K obtained from the given formula. The conflict to be resolved next then is chosen randomly from this band. By this randomization, we resolve resource conflicts using an iterative restarting the process. In the GPCPSP, the ready time and deadline for each activity are additionally available, so that it will be more difficult to determine whether a precedence pc_i can be posted or not. This can be achieved by recording the ending time relationship between activities in the temporal analysis. As shown in section 5.2.2, applying the Floyd-Warshall algorithm on G', we obtain the end time relationship between the activities, with the Latest Finish Time, LFT$_i$= $(Period - b_{0i})$. From this, latest start times can be calculated. The test of whether an activity i can be posted after activity j is given as follows: . Here, u is the maximum delay time

Algorithm 20 Test_delay(i,j)

```
$u \leftarrow -b_{ji} - D_i$
$v \leftarrow LFT_i - D_i - EST_j - D_j$
$w \leftarrow min(u, v)$
**if** $w \geq 0$ **then**
    activity $i$ can be posted after activity $j$
**else**
    activity $i$ cannot be posted after activity $j$
**end if**
```

according to the temporal relationship with j, and v is the maximum delay time with respect to the deadline for i.

5.3 Computational results

There are two levels of algorithms in our hybrid framework. In the high level, we used TS and GA to search for PBASSs. In the low level we used the CP and MCS methods to resolve conflicts. In all, we have the Tabu_CP, Tabu_MCS, GA_CP and GA_MCS hybrids.

5.3.1 Tests on real data

Data is derived from the local defense agency. We first tested our algorithms on the two sets of test data against the system in place which used a CP-based method. The first test set consists of 76 activities, which is composed of 24 activities with priority

1, 29 activities with priority 2, and 23 activities with priority 3. There are 44 different resources consumed by these activities and the whole project was to be scheduled in 100 days. The second test set consists of 165 activities, which is composed of 42 activities with priority 1, 73 activities with priority 2, 32 activities with priority 3 and 18 activities with priority 4. There are 32 different resources consumed by these activities and the whole project was to be scheduled in 100 days. The computational results for these two sets can been found in Tables 5.3 and 5.4.

Table 5.3 Scheduled Activities for the 76-activity problem

	Total	Old System	Tabu_CP	Tabu_MCS	GA_CP	GA_MCS
Priority 1	24	16	22	23	23	23
Priority 2	29	26	18	16	23	18
Priority 3	23	20	11	6	16	7
Total	76	62	51	45	62	48

Table 5.4 Scheduled Activities for the 165-activity problem

	Total	Old System	Tabu_CP	Tabu_MCS	GA_CP	GA_MCS
Priority 1	42	36	38	42	40	42
Priority 2	73	43	27	52	41	53
Priority 3	32	13	11	19	12	10
Priority 4	18	4	5	4	5	1
Total	165	96	81	117	98	106

Comparing the four methods we developed with the CP-based system used by the defense agency, we found that for the first test set the old system scheduled 16 activities with priority 1 while our best result scheduled 23 activities; for the second test set the old system scheduled 36 activities with priority 1 and our best result scheduled all 42 activities with priority 1. Generally, our methods obtained much better results. The running time was also much shorter than the old CP-based system. Comparing the four methods, if the high-level component used GA, better results are obtained than if the TS is used. In the low-level component, the MCS method performed a little better than the CP-based method for most cases. The best result obtained for the first test data set was obtained when GA_CP was used, while the best result was obtained by GA_MCS for the second test data set.

5.3.2 Tests on generated data

While the performance of the hybrid framework was found to give good results on actual data, the ability of the hybrids to achieve high quality solutions on different types of test data needed to be investigated. To test more broadly, a large number of test cases were generated where characteristics of the real data sets were simulated. Parameters used for these test cases were:

Period: *planning period*

Nact: *number of activities*

Nres: *number of different types of resources*

%temporal_constraints: *percentage of the activities which have temporal constraints, i.e., (number of precedence relations)/(total number of pairs of activities), where total* *number of pairs of activities* = **Nact**×*(***Nact**-1)/2

%resource_constraints: *percentage of resource constraints to activities-resources relationships, i.e., (number of resource constraints)/(number of pairs of activities-resources), where number of pairs of activities-resources* = **Nact**×**Nres**

range_timeslot: *the time range in which an activity must take place, i.e., (activity deadline - activity ready time)*

%duration: *activity duration over the* **range_timeslot**

range_resource: *resource capacity range*

%resource_variation: *allowable resource variation over/under resource capacity*

%activity_resource: *allowable resources consumed by activities over/under resource capacity*

Npriority: *number of different priorities*

distribution_priority: *the distribution of priorities among activities, e.g. "3 3 4 " indicates that there are 3 activities with priority 3, 3 activities with priority 2 and 4 activities with priority 1*

range_temporal: *allowed range of time lag (start-to-start) for the temporal relationships between any two activities*

To test the four methods extensively, we randomly generated 10 groups of test sets, where the **Nact** varied from 10 to 800. Each group contained three different sets of test cases, where the density of the resource constraints and temporal constraints is set to be loose, average and tight. Each test set is composed of 10 randomly generated test cases. The following parameters are fixed for all the test sets, as shown in table 5.5.

Table 5.5 Fixed parameter setting for all the test sets

Paramter	value
%resource_variation	10
Range-timeslot	[6,24]
%duration	[30,80]
Npriority	5
distribution_priority	evenly divided
range_temporal	[-20,20]
%activity_resource	[5,50]

The following parameters were set for different groups of test sets with different number of activities and resources: There were three different types of test sets in

Table 5.6 Parameter setting for different group of test sets

	1	2	3	4	5	6	7	8	9	10
Period	10	50	100	300	300	300	500	600	800	1000
Nact	10	20	50	100	150	200	300	400	500	800
Nres	5	10	20	50	50	50	50	100	100	100
range_resource	[2,20]	[2,20]	[2,20]	[2,50]	[2,50]	[2,100]	[2,100]	[2,200]	[2,20 0]	[2,200]

each test group, where the density of the resource constraints and temporal constraints is set to be loose, average and tight. The parameters for the three types of test sets are shown below: In all, we generated 10 group of different size test sets, which we named test group 1,2,... Each group test sets is composed of 3 different kinds test sets with

Table 5.7 Parameters setting for test sets with different constraint densities

	loose	average	tight
%temporal_constraint	10	40	80
%resource_constraint	10	40	80

different constraint densities representing loose, average and tight densities. Each test set is composed of 10 randomly generated test cases according to the parameters of the test sets. To further test the heuristics, we modeled our problem as an Integer Program (IP) (given in the Appendix) and used ILOG CPLEX solver solutions as a benchmark. The comparison of the five methods: ILOG CPLEX SOLVER, GA_MCS, GA_CP, Tabu_MCS, Tabu_CP, on the 30 sets of test cases is given in table 5.8,5.9 and 5.10.

Computational results

The result for the four hybrid methods under the hybrid framework compared with the ILOG CPLEX Solver are shown below. The time limit for each test case is set to be 600 seconds. In Table 5.8, the number of activities is relatively small. The

Table 5.8 Small size test sets(10,20,50)

	CPLEX	GA_MCS	GA_CP	TABU_MCS	TABU_CP
Avg Score	28856.02	28691.26	28694.32	27154.31	27565.73
Deviation	0.00%	0.57%	0.56%	5.90%	4.47%
Avg Time	9.36	3.92	5.46	1.71	2.29

score is the value as defined in the objective function in the IP model. We find in table 5.8 that CPLEX is able to determine all the optimal solutions (detailed results for Table 5.8 5.9 and 5.10 are in the Appendix). The four hybrids work well. GA_CP and GA_MCS especially obtain good solutions with results only about 0.56% worse than the optimal solution. From Table 5.9 we can find that the heuristic methods are fast even on small size problems and are faster than CPLEX. We can find from

Table 5.9 Middle size test sets(100,150,200)

	CPLEX	GA_MCS	GA_CP	Tabu_MCS	Tabu_CP
Avg Score	11145614.79	14041277.26	14044211.27	11964790.80	12026373.39
Deviation	20.92%	0.38%	0.36%	15.11%	14.68%
Avg Time	248.45	123.80	39.87	55.19	20.64

Table 5.9 that when the problem size increased CPLEX was not able to find the optimal solution in the time limit set. The upper bounds returned by CPLEX were reported here, when CPLEX could not find the optimal solution within the time limit. Here, GA_MCS and GA_CP obtained much better results than CPLEX, while tabu search did not perform well although it was faster than the GA-based methods.

In Table 5.10, we find that the best results are obtained by the GA_CP methods, which were second fastest. When the problem size increased, CPLEX was not able to find feasible solutions in the time limit. In the four methods proposed, GA_CP obtained the best solution quality for all test sets. For small-sized problems, it is 0.56% from the best solutions, 0.36% for middle-sized problems and 0.03% for large

Table 5.10 Large size test sets(300,400,500,800)

	CPLEX	GA_MCS	GA_CP	Tabu_MCS	Tabu_CP
Avg Score	1605261536	3379500385	3689765539	2830239072	2812268451
Deviation	56.51%	8.44%	0.03%	23.32%	23.81%
Avg Time	598.44	470.91	190.25	395.57	111.35

problems. Further, the speed of the GA_CP method is very good. The low-level CP-based iterative randomized algorithm is efficient as we note that GA_CP and Tabu_CP were faster than the GA_MCS and Tabu_MCS. Although the CP-based iterative randomized algorithm is simple, it is effective and outperforms the MCS-based method on large test sets. To compare CPLEX and our four hybrid methods on different density of constraints, we graphed their performance on different density constraints, where "SA" denotes small size with loose constraints (defined in Table 5.8), "SB" denotes small size with average constraints and "LC" denotes large size with tight constraints.

As shown in Figure 5.5, the performance for CPLEX deteriorated as the density of constraints increased. For the heuristics, the different density of constraints did not affect the results obtained. TS-based methods were about 25 percent worse than the GAbased methods on average-sized and large-sized sets. The MCS and the CP-based methods worked almost equally well, but the best results were obtained by the GA_CP.

5.4 Conclusion

A hybrid framework is proposed to solve the over-constrained generalized resource-constrained project scheduling problem. The problem originated in work with a local defense agency. Our hybrid framework has a two-level structure. We applied Tabu Search and Genetic Algorithm heuristic searches in a high-level component of the algorithm and a CP-based iterative randomized method and a Minimal Critical Set-based method were used to resolve temporal and resource conflicts in a low-level component. The four combinations of these - Tabu_CP, Tabu_MCS, GA_CP, GA_MCS - were tested on two sets of real test data and achieved significantly better results than the old CP-based system. We also randomly generated 30 problems with different parameters to test these techniques further. The characteristics of the real data sets were captured in the test problems generated where density variance was incorporated. We modeled the problem as an integer program and used CPLEX to compare results obtained. Results indicate that the hybrid heuristics could find near optimal solutions on small size problems and worked much better than CPLEX on middle size and large size problems. In these heuristics, Genetic Algorithm-based methods performed better than the Tabu Search-based methods. In the low-level component, CP-based iterative algorithm worked as well as the MCS-based method. In all, the GA_CP method performed best obtaining the best solution quality on small, middle and large problem sizes within short times.

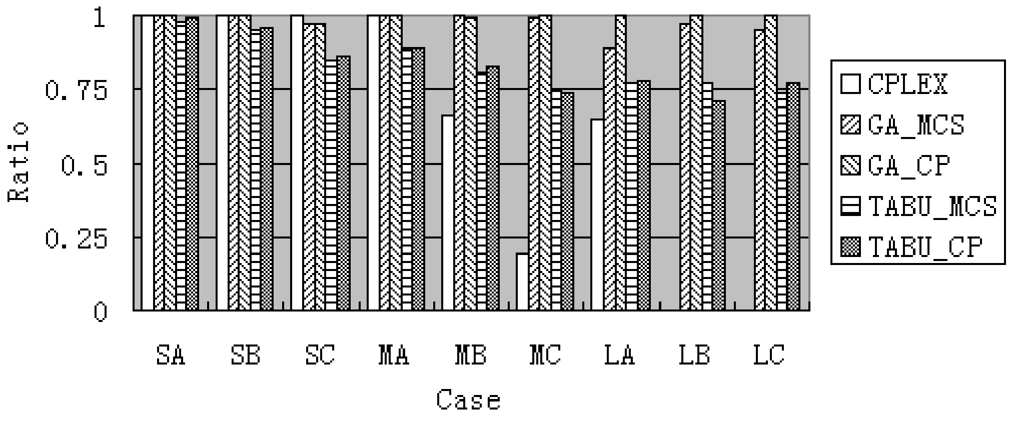


Figure 5.5 Results for different density of constraints

Chapter 6 Conclusions

This thesis focused on scheduling projects with resource constraints, and is divided into 4 parts. In the first part of the thesis, new definitions on float and critical activity are given for projects with resource constraints. We define the float as the amount of the time the activity can be extended without affecting the project makespan. The float of an activity is identified as a continuous interval from 0 to the maximum float, which captures the flexibility of activities. With the new definition of float, critical activity is defined as the activity with 0 maximum float and no longer depends on a specified schedule.

To explore the flexibility of a set of activities, group float is defined as the float for a set of activities. With group float, the float set and critical set are defined as the set of activities with positive maximum group float and 0 maximum group float respectively.

To complement float, negative float is defined as the amount of time the duration of the activity can be reduced without affecting the project deadline. Corresponding to negative float, we define negative critical activity, negative float set, negative critical set.

One of the most significant contributions of this thesis is the definition of a float graph and negative float graph. float graph and negative float graph illustrate float, critical activity, float set, critical set and negative float, negative critical activity, negative critical set, negative float set clearly in an undirected graph. With float graph, the project manager can easily discern the penalty for the completion of the entire project for delaying any activity or any set of activities, while negative float graph show the benefits to reduce the deadline of the project through reducing the duration of any activity or any set of activities. Although it is useful to apply float graph and negative float graph to project management, it is difficult to find all float sets and critical sets for project with more than 30 activities. Therefore, for the large project, it is only possible to draw a float graph for a subset of activities inside the project, which the project manager is most interested in. We can also draw the float graph or negative float graph for large projects by setting limits for the number of activities within a float set and a critical set. Further research can be done to explore float graph and negative float graph to manage resource-constrained project in a better way.

In this thesis, we also consider activity which is critical and negative critical. This kind of activity is defined as zero activity. Delaying zero activity will result in delaying the entire project, while reducing the duration of zero activity will reduce the makespan of the project. Therefore, zero activity should be central in the project management, where more attention need to be paid to it while planning and managing the project.

To determine the range of float, three different methods are presented to calculate the maximum float. As the problem to calculate maximum float is NP-hard, there is no polynomial time algorithm exists unless $P = NP$. We first adopt a well-known branch and bound procedure *dh_procedure* to find the exact value of maximum float, which is called *E-Float*. As it is impossible to calculate the *E-Float* for large projects, we proposed a way to calculate float approximately by applying Testing Hypothesis to the results obtained by a Simulated Annealing (SA) algorithm, where we call the maximum float approximation as *H-Float*. Detailed experimental results show that the *H-Float* calculated by the Testing Hypothesis achieved high accuracy compared with the *E-Float*. In a computational study, we also showed that the more samples we use, the better the accuracy achieved by Testing Hypothesis. To further increase the speed for calculating *H-Float*, we also developed an algorithm which uses Simulated Annealing to calculate *H-Float* directly. Although the accuracy for the *H-Float* calculated by SA is weaker, the speed the solution is got was greatly improved. Using our newly developed Molecule Search (MS) and Molecule Bank Algorithm (MBA) (see next paragraph), we achieved much higher accuracy for *H-Float*. This demonstrates that the better a heuristic converges to optimal solutions the higher the accuracy of *H-Float*. To further improve the accuracy of *H-Float*, we may also apply Testing Hypothesis to the results obtained by MS and MBA in the future.

In the second part of the thesis, simulating the cooling process from gas to crystal, a new optimization method, Molecule Search (MS), is proposed to solve resource-constrained project scheduling problem. We presented the ideal MS model, where the motion of molecules are only guided by the predefined rules or forces. In the analogy, with cooling, when the environment temperature drops, the internal energy of the molecules drops and the molecules will gradually move to the positions which can be decoded as the optimal or near optimal solutions for combinatorial optimization problem. However, it is very hard to identify the force or rule for a specified combinatorial optimization problem. One reason is that crystals have optimal substructures, while an NP-hard combinatorial optimization problem does not possess this property. In this case, molecules have to move to some positions, which leads to bad substructure, to achieve global optimum.

Since it is difficult to apply the ideal MS model to combinatorial optimization problems, we proposed a new model which simulates the motion of high kinetic energy molecules and low kinetic energy molecules separately. As high kinetic energy molecules move very fast, we used molecule jumping to simulate their motion concurrently. Since low kinetic energy molecules move relatively slow, we used molecule walking to simulate their motion as a series of steps, where each step will lead to a position for better or equal solutions. In solving RCPSP, we proposed 3 jumping rules and developed a new local search algorithm, forward-backward search (FBS), as molecule walking. Computation study demonstrated the performance of MS as one of the best heuristics for RCPSP so far. As a general optimization method, MS can be applied to other combinatorial optimization problems in the future. Further research can also be done to study the ideal MS Model, where the rules or forces may be learned during the search process instead.

In the third part of the thesis, we have proposed a population based Molecule Search algorithm, named as Molecule Bank Algorithm (MBA), to solve RCPSP. The main contribution for MBA is the balanced diversification and intensification scheme, where the population of groups of molecules is not fixed. We have a very large population at the beginning of the search process which leads to better diversification. When the search process proceeds, the population will be decreased gradually, until the population reaches a very small number. Therefore, more and more computational effort is put into intensification at this stage. We call the changing of population size as a dynamic population. We have further developed two techniques - molecule aging and molecule drifting - for balancing diversification and intensification. With the balanced diversification and intensification scheme, MBA achieved best results on the PSPLIB. Further research can be done to investigate how to balance intensification and diversification in the search process of heuristics within a specified computational cost limit.

In the fourth part of the thesis, a hybrid framework is proposed for a real-life over-constrained resource-constrained project scheduling problem with the objective to maximize the number of high priority activities scheduled. More general temporal constraints and resource constraints are considered in this problem. The hybrid framework is composed of two levels. In the high level, Genetic Algorithm and Tabu Search are used to explore different feasible scheduling sequence, while in the low level, constraint programming and minimal critical sets are used to resolve conflicts. Experimental results on real-life data and a large amount of randomly generated data highlight the effect of the hybrid framework. Further research can be done to explore other kinds of over-constrained problems with the hybrid framework. Unlike the well-known maximal constraint satisfaction problem (Max-CSP), which provides framework to minimize the number of violated constraints, we consider removing the low priority variables to resolve conflicts instead here. In this part, we also proposed a simple iterative randomized algorithm using constraint propagation to reduce the domain of the activities each time when we fixed a start time of an activity. Extensive experimental results show that the simple CP-based iterative randomized algorithm outperforms a complicated Minimal Critical Set (MCS) based algorithm. The hybrid framework can be applied to other over-constrained problems to resolve conflicts through removing low priority variables instead of constraints in the future. Further research can also be done to apply the CP-based iterative randomized algorithm to resolve conflicts for other similar problems.

References

1. E. Aarts and J. Korst. *Simulated Annealing and Boltzmann Machines.* John Wiley & Sons, Great Britain, 1979.

2. J. Alcaraz, C. Maroto, and R. Ruiz. Improving the performance of genetic algorithms for the rcps problem. In *Proceedings of the Ninth International Workshop on Project Management and Scheduling*, pages 40–43. Nancy, 2004.

3. R. Alvarez-Valdes and J. M. Tamarit. Heurisitc algorithms for resource-constrained project sch eduling: A review and an empirical analysis. *in: R. Slowinski, J. Weglarz (Eds.), Advances in Project Scheduling, Elsevier, Amsterdam*, pages 114–134, 1989.

4. C. Artigues, P. Michelon, and S. Reusser. Insertion techniques for static and dynamic resource-constrained project scheduling. *European Journal of Operational Research*, 149:249–267, 2003.

5. T. Baar, P. Brucker, and S. Knust. Tabu-search algorithms and lower bounds for the resource-constrained project scheduling problem. In *Meta-heuristics: Advances and trends in local search paradigms for optimization*, pages 1–8. Kluwer Academic Publishers, 1998.

6. M. Bartusch, R. H. Möhring, and F. J. Radermacher. Scheduling project networks with resource constriants and time windows. *Annals of Operations Research*, 16:201–240, 1988.

7. C. Bessiere and J. C. Regin. Mac and combined heuristics: Two reasons to forsake fc (and cbj?) on hard problems. *CP*, pages 61–75, 1996.

8. S. Bistarelli, F. Montanari U Rossi, T. Schiex, H. Fargier, and G. Verfailllie. Semiring-based csps and valused csps: Frameworks, properties and comparisons. *Constraints*, 4:199–240, 1999.

9. J. Blazewicz, W. Cellary, R. Slowinski, and J. Weglarz. *Scheduling under Resource Constraints: Deterministic Models.* J. C. Baltzer, Basel, 1986.

10. J. Blazewicz, J. K. Lenstra, and A. H. G. Rinnooy Kan. Scheduling projects subject to resource con straints: Classification and complexity. *Discrete Applied Mathematics*, 5:11–24, 1983.

11. F. F. Boctor. Some efficient multi-heuristic procedures for resource-constrained project scheduling. *European Journal of Operational Research*, 49:3–13, 1990.

12. F. F. Boctor. An adaption of the simulated annealing algorithm for solving resource-constrained project scheduling problems. *International Journal of Production Research*, 34:2335–2351, 1996.

13. A. Borning, R. Duisberg, B. Freeman-Benson, A. Kramer, and M. Woolf. Constraint hierarchies. in proceedings 1987 acm conference on object-oriented programming systems. *Languages and Applications*, pages 48–60, 1987.

14. K. Bouleimen and H. Lecocq. A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple modes version. *European Journal of Operational Research*, 149:268–281, 2003.

15. JA. Bowers. Criticality in resource constrained networks. *Journal of Operations Research Society*, 46(1):80–91, 1995.

16. JA. Bowers. Interpreting float in resource constrained projects. *International Journal of Project Management*, 18(6):385–392, 2000.

17. K. Brinkmann and K. Neumann. Heuristic procedures for resource-constrained project scheduling with minimal and maximal time lags. *Journal of Decision Systems*, 5:129–156, 1996.

18. P. Brucker. *Scheduling Algorithms*. Springer-Verlag, Berlin, 1997.

19. P. Brucker, A. Drexl, R. Möhring, K. Neumann, and E. Pesch. Resource constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research*, 112:3–41, 1998b.

20. P. Brucker and S. Knust. Lower bounds for resource-constrained project scheduling problems. *European Journal of Operational Research*, 149:302–313, 2003.

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

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

23. J. Carlier and E. Neron. On linear lower bounds for the resource constrained project scheduling problem. *European Journal of Operational Research*, 149:314–324, 2003.

24. O. Catoni. Metropolis, simulated annealing, and iterated energy transformation algorithms: Theory and experiments. *Journal of Complexity*, 12:595–623, 1996.

25. A. Cesta, A. Oddi, and S. Smith. An iterative sampling procedure for resource constrained project scheduling with time windows. In *In Proceeding of the 16th International Joint Conference on Artificial Intelligence*, 1999.

26. A. Cesta, A. Oddi, and S. Smith. A constraint-based method for project scheduling with time windows. Technical report, Tech. report CMU-RI-TR-00-34, Robotics Institute, Carnegie Mellon University, February, 2000.

27. V. Černy. Thermodynamical approach to the traveling salesman problem: an efficient simulation algorithm. *Journal of Optimization Theory and applications*, 45:41–51, 1985.

28. N. Christofides, R. Alvarez-Valdes, and J. M. Tamarit. Project scheduling with resource constraints: A branch and bound approach. *European Journal of Operational Research*, 29:262–273, 1987.

29. J. Coelho and L. Tavares. Comparative analysis of meta-heuricstics for the resource constrained project scheduling problem. Technical report, Department of Civil Engineering, Instituto Superior Tecnico, Portugal, 2003.

30. D. F. Cooper. Heuristics for scheduling resource-constrained projects: an experimental investigation. *Management Science*, 22:1186–1194, 1976.

31. K. Crandall. Project planning with precedence lead-lag factors. *Project Management Quarterly*, 4(3):18–27, 1973.

32. Wiest J. D. Some properties of schedules for large projects with limited resources. *Operations Research*, 12(3):395–418, 1964.

33. E. W. Davis and J. H. Patterson. A comparison of heuristic and optimum solutions in resource-constrained project scheduling. *Management Science*, 21:944–955, 1975.

34. E. W. Davis and J. H. Patterson. A comparison of heuristic and optimum solutions in resource-constrained project scheduling. *Management Science*, 21:803–816, 1975.

35. D. Debels, B. De Reyck, R. Leus, and M. Vanhoucke. A hybrid scatter search electromagnetism meta-heuristic for project scheduling. *European Journal of Operational Research*, page to appear, 2004.

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

37. E. Demeulemeester. Minimizing resource availability costs in time-limited project networks. *Management Science*, 41:1590–1598, 1995.

38. E. Demeulemeester and W. Herroelen. A branch-and-bound procedure for the generalized resource co nstrained project scheduling problem. Technical report, Research Report No. 9206, Department of Applied Economics, Katholieke Universiteit Leuven, Belgium, 1992.

39. E. Demeulemeester and W. Herroelen. A branch-and-bound procedure for the multiple resource-constrained project scheduling problem. *Management Science*, 38(12):1803–1818, 1992.

40. E. Demeulemeester and W. Herroelen. New benchmark results for the resource-constrained project scheduling problem. *Management Science*, 43(11):1485–1492, 1997.

41. E. Demeulemeester and W. Herroelen. *Project Scheduling-A research handbook*. Kluwer Academic Publishers, Boston, 2002.

42. C. Domshlak, F. Rossi, B. Venable, and T. Walsh. Reasoning about soft constraints and conditional preferences. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*, 2003.

43. S. E. Elmaghraby. *Activity Networks: Project Planning and control by Network Models*. Wiley, New York, 1977.

44. S. E. Elmaghraby and J. Kamburowski. The analysis of activity networks under generalized precedence relations. *Management Science*, 38:1245–1263, 1992.

45. K. Fleszar and K. Hinidi. Solving the resource-constrained project scheduling problem by a variable neighbour search. *European Journal of Operational Research*, 155:402–413, 2004.

46. M. Fox. *Constraint-Directed Search: A Case Study of Job-Shop Scheduling*. Morgan Kanfmnn Publishers, CA, 1987.

47. E. Freuder and R. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58(545):21–70, 1992.

48. P. Galinier and J. K. Hao. Tabu search for maximal constraint satisfaction problems. LNCS 1330:196–208, 1997.

49. M. R. Garey and D. S. Johnson. *Computers and intractability. A Guide to the theory of NP-completeness.* W. H. Freeman and Company, New York, NY, 1979.

50. F. Glover. Heuristics for integer programming using surrogate constraints. *Decision Sciences*, 8:156–166, 1977.

51. EM. Goldratt. *Critical chain.* The North River Press, Great Barrington, MA, 1997.

52. S. Hartmann. A competitive genetic algorithm for the resource-constrained project scheduling. *Naval Research Logistics*, 456:733–750, 1998.

53. W. Herroelen and R. Leus. On the merits and pitfalls of critical chain scheduling. *Journal of Operations Management*, 19:559–577, 2001.

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

55. H. J. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, 1975.

56. H. J. Holland. *Adaption in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, 1975.

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

58. J. E. Jr. Kelley. The critical path method: resource planning and scheduling. In J. F. Muth and G. L. Thompson, editors, *Industrial Scheduling*, pages 347–365, 1963.

59. J.A. G.M. Kerbosh and H. J. Schell. Network planning by the extended metra potential method. Technical report, Report KS-1.1, University of Technology, Eindhoven, Department of Industrial Engineering, 1975.

60. S. Kirkpatrick, JR. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

61. Y. Kochetov and A. Stolyar. Evolutionary local search with variable neighborhood for the resource constrained project scheduling problem. In *Proceedings of the 3rd International Workshop of Computer Science and Information Technologies*. Russia, 2003.

62. U. Kohlmorgen, H. Schmeck, and K. Haase. Experiences with fine-grained parallel genetic algorithms. *Annals of Operations Research*, 90:203–219, 1999.

63. R. Kolisch. *Project Scheduling under Resource Constraints-Efficient Heuristics for Several Problem Classes*. Physica, Heidelberg, 1995.

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

65. R. Kolisch and A. Drexl. Adaptive search for solving hard project scheduling problems. *Naval research logistics*, 43:23–40, 1996.

66. R. Kolisch and S. Hartmann. Heuristic algorithms for solving the resource-constrained project scheduling problem: Classification and computation analysis. In *Project Scheduling: Recent Models, Algorithms and Applications*, pages 147–178, Boston, MA, 1999. Kluwer Academic Publishers.

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

68. R. Kolisch and R. Padman. An integrated survey of deterministic project scheduling. *OMEGA International Journal of Management Science*, 39(3):249–272, 2001.

69. R. Kolisch and A. Sprecher. Psplib- a project scheduling problem library. *European Journal of Operational Research*, 96:205–216, 1996.

70. R. Kolisch, A. Sprecher, and A. Drexl. Characterization and generation of a general class of resource-constrained project scheduling problems. *Management Science*, 41:1693–1703, 1995.

71. N. Kubota and T. Fukuba. Genetic algorithms with age structure. *Soft Computing*, 1(4):1432–7643, 1997.

72. P. Laborie and M. Ghallab. Planning with sharable resource constraints. In *In Proceedings of the International Joint Conference on Artificial Intelligence*, 1995.

73. E. L. Lawler. *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart and Winston, New York, 1976.

74. K. Libbrecht. http://snowcrystals.com, 1999.

75. A. Lim, B. Rodrigues, S. T. Tan, and F. Xiao. Float and critical activity for resource-constrained projects, 2007. working paper.

76. A. Lim, B. Rodrigues, S. T. Tan, and F. Xiao. Molecular bank algorithm for resource-constrained project scheduling problem, 2007. working paper.

77. A. Lim, B. Rodrigues, S. T. Tan, and F. Xiao. Molecule search for minimum linear arrangement problem, 2007. working paper.

78. A. Lim, B. Rodrigues, S. T. Tan, and F. Xiao. Molecule search for resource-constrained project scheduling problem, 2007. working paper.

79. A. Lim, B. Rodrigues, R. Thangarajoo, and F. Xiao. A hybrid framework for over-constrained generalized resource project scheduling problems. *Artificial Intelligence Review*, 22(3):211–243, 2004.

80. A. Lim, B. Rodrigues, and F. Xiao. A fast algorithm for bandwidth minimization, 2006. accepted in International Journal of Artificial Intelligence Tools.

81. A. Lim, B. Rodrigues, and F. Xiao. Heuristics for matrix bandwidth reduction. *European Journal of Operational Research*, 174:69–91, 2006.

82. D. Merkle, M. Middendorf, and H. Schmeck. Ant colony optimization for resource-constrained project scheduling. *IEEE Transactions on Evolutionary Computation*, 6:333–346, 2002.

83. Z. Michalewicz. Heuristic methods for evolutionary computation techniques. *Journal of Heuristics*, 1:177–206, 1995.

84. A. Mingozzi, V. Maniezzo, S. Ricciardelli, and L. Bianco. An exact algorithm for project scheduling with resource constraints based on a new mathematical formulation. *Management Science*, 44(5):714–729, 1998.

85. J. J. Moder, C. R. Phillips, and E. W. Davis. *Project management with CPM, PERT and Precedence Diagramming, 3ed.* Van Nostrand Reinhold, New York, 1983.

86. R. Möhring, A. Schulz, F. Stork, and M. Uetz. Solving project scheduling problems by minimum cut computations. *Management Science*, 49(3):330–350, 2003.

87. K. Neumann and C. Schwindt. Activity-on-node networks with minimal and maximal time lags and their applications to make-to-order production. *OR Spektrum*, 19:205–217, 1997.

88. K. Neumann, C. Schwindt, and J. Zimmermann. *Project Scheduling with Time Windows and Scarce Resources*. Springer, 2 edition, 2003.

89. K. Neumann and J. Zhan. Heuristics for the minimum project-duration problem with minimal and maximal time lags under fixed resources. *Journal of Intelligent Manufacturing*, 6:145–154, 1995.

90. K. Nonobe and T. Ibaraki. Formulation and tabu search algorithm for the resource constrained project scheduling problem. In *Essays and Surveys in Metaheuristics*, pages 557–588. Kluwer Academic Publishers, 2002.

91. O. Oguz and H. Bala. A comparative study of computational procedures for the resource constrai ned project scheduling problem. *European Journal of Operational Research*, 72:406–416, 1994.

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

93. J. H. Patterson, R. Slowinski, F. B. Talbot, and J. Weglarz. *An algorithm for a general class of precedence and resource constrained scheduling problems, in: R. Slowinski, J.Weglarz (Eds.), Advances in Project Scheduling*. Elsevier, Amsterdam, 1989.

94. T. Petit, J. C. Regin, and C. Bessiere. Meta constraints on violations for over constrained problems. 2000.

95. T. Petit, J. C. Regin, and C. Bessiere. Specific filtering algorithms for over-constrained problems. *CP*, pages 451–463, 2001.

96. T. Petit, J. C. Regin, and C. Bessiere. Range-based algorithm for max-csp. *CP*, pages 280–294, 2002.

97. M. Pinedo. *Scheduling - Theory, Algorithms and Systems*. Prentice-Hall, Englewood Cliffs, 1995.

98. T. Raz, R. Barnes, and D. Dvir. A critical look at critical chain project management. *Project Management Journal*, 34(4):24–32, 2003.

99. T. Raz and B. Marshall. Effect of resource constraints on float calculations in project network. *International Journal of Project Management*, 14(4):241–248, 1996.

100. J. C. Regin, T. Petit, C. Bessiere, and J. F. Puget. New lower bounds of constraint violations for over-constrained problems. *CP*, pages 332–345, 2001.

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

102. J. A. Rice. *Mathematical Statistics and Data Analysis*. Duxbury Press, California, 1995.

103. F. A. Rivera and A. Duran. Critical clouds and critical sets in resource-constrained projects. *International Journal of Project Management*, 22:489–497, 2004.

104. T. Schiex, H. Fargier, and G. Verfailllie. *Valued constraint satisfaction problems: hard and easy problems*. IJCAI, 1995.

105. A. Schirmer. Case-based reasoning and improving adaptive search for project scheduling. *Naval Research Logistics*, 47:201–222, 2000.

106. A. Sprecher. Scheduling resource-constrained projects competitively at modest memory requirements. *Management Science*, 46(5):710–723, 2000.

107. A. Sprecher. Network decomposition techniques for resource-cosntrained project scheduling. *Journal of Operational Research Society*, 53(4):405–414, 2002.

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

109. J. P. Stinson, E. W. Davis, and B. M. Khumawala. Multiple resource-constrained scheduling using branch and bound. *AIIE Transactions*, 10:252–259, 1978.

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

111. P. Tormos and A. Lova. An efficient multi-pass heuristic for project scheduling with constrained resources. *International Journal of Production Research*, 41(5):1071–1086, 2003.

112. P. Tormos and A. Lova. Integrating heuristics for resource constrained project scheduling: One step forward. Technical report, Department of Statistics and Operations Research, Universidad Politécnica de Valencia, 2003.

113. E. Uyttewaal. Take the path that is really critical. *PM Network*, 13(12):37–39, 1999.

114. V. Valls, F. Ballestin, and M. S. Quintanilla. A hybrid genetic algorithm for the rcpsp. Technical report, Department of Statistics and Operations Research, University of Valencia, 2003.

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

116. V. Valls, F. Ballestin, and M. S. Quintanilla. Justification and rcpsp: A technique that pays. *European Journal of Operational Research*, 165:375–386, 2005.

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

118. Alan J. Walton. *Three phases of matter*. Oxford University Press, Oxford, 1983.

119. E. D. Wikum, C. L. Donna, and G. L. Nemhauser. One-machine generalized precedence constrained scheduling problems. *OR Letters*, 16:87–99, 1994.

120. BM. Woodworth. Is resource-constrained project management software reliable? *Cost Engineering*, 31(7):7–11, 1989.

121. BM. Woodworth and S. Shanahan. Identifying the critical sequence in a resource constrained project. *International Journal of Project Management*, 6(2):89–96, 1988.

122. J. Zhan. Heuristics for scheduling resource-constrained projects in MPM networks. *European Journal of Operational Research*, 76:192–205, 1994.

Appendix A
Algorithms in calculating float and group float

Algorithm 21 calculating group float by branch and bound

```
bab_groupfloat(A,P,R,D,$S_1$,C)
$hf \leftarrow D$, $lf \leftarrow 0$
while $lf < hf$ do
   $mf \leftarrow (hf + lf)/2 + 1$
   for each $a \in S_1$ do
      $d'(a) \leftarrow d(a) + mf \cdot C(a)$
   end for
   $A' \leftarrow A$
   update $d'(a)$ as the duration for activity a in $A'$
   $D' \leftarrow dh\_procedure(A', d', P, R)$
   if $D' \leq D$ then
      $lf \leftarrow mf$
   else
      $hf \leftarrow mf - 1$
   end if
end while
return $lf$
```

Algorithm 22 calculating group float by Testing Hypothesis

```
$th\_groupfloat(A,P,R,S_1,C)$
for $i \leftarrow 1$ to $n$ do
    $X_i \leftarrow SA(A,P,R)$
end for
$\overline{X} \leftarrow \frac{1}{n}\sum_{i=1}^{n} X_i$
$s_x = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \overline{X})^2$
$hf \leftarrow$ upbound_float$(A,P,R,X_1,a)$ , $lf \leftarrow 0$
while $lf < hf$ do
    $mf \leftarrow (hf + lf)/2 + 1$
    for each $a \in S_1$ do
        $d'(a) \leftarrow d(a) + mf \cdot C(a)$
    end for
    $A' \leftarrow A$
    update $d'(a)$ as the duration for activity a in $A'$
    for $i \leftarrow 1$ to $n$ do
        $Y_i \leftarrow SA(A',P,R)$
    end for
    $\overline{Y} \leftarrow \frac{1}{n}\sum_{i=1}^{n} Y_i$
    $s_y = \frac{1}{n-1}\sum_{i=1}^{n}(Y_i - \overline{Y})^2$
    if $\frac{\overline{Y}-\overline{X}}{\sqrt{({s_x}^2+{s_y}^2)/n}} < z(\alpha)$ then
        $lf \leftarrow mf$
    else
        $hf \leftarrow mf - 1$
    end if
end while
return $lf$
```

Algorithm 23 calculating group float by SA directly

```
*sa_groupfloat(A,P,R,$S_1$,C)*
$D \leftarrow SA(A, P, R)$
$hf \leftarrow upbound\_float(A, P, R, D, a)$, $lf \leftarrow 0$
**while** $lf < hf$ **do**
  $mf \leftarrow (hf + lf)/2 + 1$
  **for** each $a \in S_1$ **do**
    $d'(a) \leftarrow d(a) + mf \cdot delayvector(a)$
  **end for**
  $A' \leftarrow A$
  update $d'(a)$ as the duration for activity a in $A'$
  $D' \leftarrow SA(A', P, R)$
  **if** $D' \leq D$ **then**
    $lf \leftarrow mf$
  **else**
    $hf \leftarrow mf - 1$
  **end if**
**end while**
return $lf$
```

Algorithm 24 calculating *H-Float* with speed up Testing Hypothesis

```
th_float2(A,P,R,a)
for $i \leftarrow 1$ to $n$ do
  $X_i \leftarrow SA(A, P, R)$
end for
$X_{min} = \min_{1 \leq i \leq n} X_i$
$\mathtt{lb_1}(a) \leftarrow 0$
for $i \leftarrow 1$ to $n$ do
  sfloat $\leftarrow cal\_sfloat(A, P, R, st_i)$
  if $\mathtt{sfloat}(a) > \mathtt{lb_1}(a)$ then
    $\mathtt{lb_1}(a) \leftarrow \mathtt{sfloat}(a)$
  end if
end for
$\overline{X} \leftarrow \frac{1}{n} \sum_{i=1}^{n} X_i$
$s_x = \frac{1}{n-1} \sum_{i=1}^{n} (X_i - \overline{X})^2$
$hf \leftarrow$ upbound_float$(A, P, R, X_1, a)$ , $lf \leftarrow lb_1(a)$
while $lf < hf$ do
  $mf \leftarrow (hf + lf)/2 + 1$
  $d'(a) \leftarrow d(a) + mf$
  $A' \leftarrow A$
  update $d'(a)$ as the duration for activity a in $A'$
  for $i \leftarrow 1$ to 7 do
    $Y_i \leftarrow SA(A', P, R)$
  end for
  $s_y \leftarrow s_x$
  $Y_{min} = \min_{1 \leq i \leq 7} Y_i$
  if $\frac{Y_{min} - \overline{X} - 0.5}{\sqrt{({s_x}^2 + {s_y}^2)/n}} > z(\alpha)$ then
    $hf \leftarrow mf - 1$
  else
    for $i \leftarrow 8$ to 15 do
      $Y_i \leftarrow SA(A', P, R)$
    end for
    $Y_{max} = \min_{1 \leq i \leq 15} Y_i$
    if $\frac{Y_{max} - \overline{X} - 0.5}{\sqrt{({s_x}^2 + {s_y}^2)/n}} < z(\alpha)$ then
      $lf \leftarrow mf$
    else
      for $i \leftarrow 16$ to $n$ do
        $Y_i \leftarrow SA(A', P, R)$
      end for
      $\overline{Y} \leftarrow \frac{1}{n} \sum_{i=1}^{n} Y_i$
      $s_y = \frac{1}{n-1} \sum_{i=1}^{n} (Y_i - \overline{Y})^2$
      if $\frac{\overline{Y} - \overline{X} - 0.5}{\sqrt{({s_x}^2 + {s_y}^2)/n}} < z(\alpha)$ then
        $lf \leftarrow mf$
      else
        $hf \leftarrow mf - 1$
      end if
    end if
  end if
end while
```

Appendix B
OGRCPSP IP model and experimental results

The OGRCPSP IP model

Input:

$Period$ = planning period

A = set of activities

R = set of resources

G_i = release time of activity i

H_i = deadline of activity i

$T_{ij} = 1$ if a temporal constraint exists between activity i and j, 0 otherwise

D_i = duration of activity i

SS_{ij} = Minimum time lag between activities i and j

SM_{ij} = Maximum time lag between activities i and j ($SM_{ij} = -SS_{ji}$) r_{ij} = capacity of resource i at time j

a_{ij} = amount of resource j consumed by activity i

M = a very large number

Variables:

$$x_i = \begin{cases} 1 & \text{activity } i \text{ is scheduled,} \\ 0 & \text{otherwise,} \end{cases} \tag{B.1}$$

$$st_{ij} = \begin{cases} 1 & \text{activity } i \text{ starts at time } j, \\ 0 & \text{otherwise,} \end{cases} \tag{B.2}$$

The objective is to maximize the mapped vector $\pi(SAS)$. Factors a_i are used in the following way: Suppose the number of activities with priority $1, 2, ..., m$ is p_1, p_2,...,p_m, respectively. The factor a_i should be larger than $\sum_{i<j\leq m} a_j p_j$ where:

$$a_m \leftarrow p_m \tag{B.3}$$

$$a_i \leftarrow \sum_{i<j\leq m} a_j p_j + 1 \;\; for \;\; 1 \leq i \leq m-1 \tag{B.4}$$

The IP model can then be written as follows:

$$Maximize \sum_{i\in A} a_i x_i \tag{B.5}$$

subject to:

Temporal Constraints

$$(1-x_i)M+(1-x_j)M+\sum_{G_i\leq u\leq H_i} st_{i(u-G_i)}u-\sum_{G_j\leq u\leq H_j} st_{j(u-G_j)}u \geq SS_{ij} \qquad for\ all\ i,j\in A, T_{ij}=1 \tag{B.6}$$

$$(1-x_i)M+(1-x_j)M+\sum_{G_i\leq u\leq H_i} st_{i(u-G_i)}u-\sum_{G_j\leq u\leq H_j} st_{j(u-G_j)}u \leq SM_{ij} \qquad for\ all\ i,j\in A, T_{ij}=1 \tag{B.7}$$

Resource Constraints

$$\sum_{i\in A}\sum_{s\leq k\leq j} st_{ik} ar_{iu} \leq r_{uj} s = min(j-D_i, G_i), for\ all\ u\in R, 1\leq j\leq Period \tag{B.8}$$

Variable Constraints

$$\sum_{G_i \leq j \leq H_i} st_{ij} \leq x_i \textit{for all} \quad i \in A \tag{B.9}$$

$$st_{ij}, x_i \in \{0,1\} \quad \textit{for all} \quad i \in A, G_i \leq j \leq H_i \tag{B.10}$$

Table B.1 Small size test sets

	CPLEX	GA_MCS	GA_CP	TABU_MCS	TABU_CP
1A					
score	**226.2**	**226.2**	**226.2**	**226.2**	**226.2**
Deviation	0.00%	0.00%	0.00%	0.00%	0.00%
1B					
score	**211.0**	**211.0**	210.9	**211.0**	210.9
Deviation	0.00%	0.00%	0.05%	0.00%	0.05%
1C					
score	**191.9**	191.7	191.7	191.7	191.7
Deviation	0.00%	0.10%	0.10%	0.10%	0.10%
2A					
score	**2780.5**	**2780.5**	**2780.5**	**2780.5**	**278 0.5**
Deviation	0.00%	0.00%	0.00%	0.00%	0.00%
2B					
score	**2359.1**	2357.2	**2359.1**	2357.2	**2359.1**
Deviation	0.00%	0.08%	0.00%	0.08%	0.00%
2C					
score	**1717.1**	1688.4	1714.2	1688.4	1714.2
Deviation	0.00%	1.67%	0.17%	1.67%	0.17%
3A					
score	**123108.4**	**123108.4**	**123108.4**	120046.1	121619.0
Deviation	0.00%	0.00%	0.00%	2.49%	1.21%
3B					
score	**78092.8**	**78092.8**	**78092.8**	73710.3	75163.4
Deviation	0.00%	0.00%	0.00%	5.61%	3.75%
3C					
score	**51017.2**	49565.1	49565.1	43177.4	43826.6
Deviation	0.00%	2.85%	2.85%	15.37%	14.09%

Table B.2 Middle size test sets

	CPLEX	GA_MCS	GA_CP	TABU_MCS	TABU_CP
4A					
score	**2377350.8**	2377312.8	2377350.7	2247060.9	2243233.4
Deviation	0.00%	0.00%	0.00%	5.48%	5.64%
4B					
score	**1279301.6**	1278375.5	**1279301.6**	1046810.5	1068152.8
Deviation	0.00%	0.07%	0.00%	18.17%	16.51%
4C					
score	508572.1	**727083.4**	727083.2	666930.8	646468.4
Deviation	30.05%	0.00%	0.00%	8.27%	11.09%
5A					
score	**15526524.0**	15444295.2	15441505.2	14305726.7	14124776.4
Deviation	0.00%	0.53%	0.55%	7.86%	9.03%
5B					
score	4776538.5	7260842.0	**7350311.1**	6140514.8	5959081.1
Deviation	35.02%	1.22%	0.00%	16.46%	18.93%
5C					
score	2971315.6	**4081559.3**	3992090.1	2511970.4	2604121.0
Deviation	27.20%	0.00%	2.19%	38.46%	36.20%
6A					
score	**56403681.6**	56383854.0	56398134.2	49844100.0	50106519.8
Deviation	0.00%	0.04%	0.01%	11.63%	11.16%
6B					
score	16467248.9	**25515743.6**	25240567.9	20416262.0	21263670.5
Deviation	35.46%	0.00%	1.08%	19.99%	16.66%
6C					
score	0.0	13302429.5	**13591557.4**	10503741.1	10221337.1
Deviation	100.00%	2.13%	0.00%	22.72%	24.80%

Table B.3 Large size test sets

	CPLEX	GA_MCS	GA_CP	TABU_MCS	TABU_CP
7A					
score	329360318.6	332127678.1	**332287303.7**	282355612.6	282311686.7
Deviation	0.88%	0.05%	0.00%	15.03%	15.04%
7B					
score	0.0	138754620.3	**140094192.4**	108230531.3	102714155.4
Deviation	100.00%	0.96%	0.00%	22.74%	26.68%
7C					
score	0.0	**69251921.6**	67891157.8	49914262.7	51344230.7
Deviation	100.00%	0.00%	1.96%	27.92%	25.86%
8A					
score	885978858.0	1199318175.0	**1208460215.0**	963681254.2	967831069.9
Deviation	26.69%	0.76%	0.00%	20.26%	19.91%
8B					
score	0.0	**465439988.9**	452790397.1	362448043.6	358195883.7
Deviation	100.00%	0.00%	2.72%	22.13%	23.04%
8C					
score	0.0	**219645229.6**	219593397.6	159539268.7	150930564.3
Deviation	100.00%	0.00%	0.02%	27.37%	31.28%
9A					
score	2162546908.0	3064557050.0	**3210660802.0**	2588061271.0	2618765201.0
Deviation	32.64%	4.55%	0.00%	19.39%	18.44%
9B					
score	0.0	1125195803.0	**1135185672.0**	906977091.6	864944840.8
Deviation	100.00%	0.88%	0.00%	20.10%	23.81%
9C					
score	0.0	520818185.7	**531122225.9**	416963835.0	427060794.8
Deviation	100.00%	1.94%	0.00%	21.49%	19.59%
10A					
score	15885252349.0	21924827229.0	**25015535712.0**	19046110988.0	19312773292.0
Deviation	36.50%	12.36%	0.00%	23.86%	22.80%
10B					
score	0.0	8132903022.0	**8401244958.0**	6456918605.0	5854721869.0
Deviation	100.00%	3.19%	0.00%	23.14%	30.31%
10C					
score	0.0	3361165718.0	**3562320438.0**	2621668101.0	2755627829.0
Deviation	100.00%	5.65%	0.00%	26.41%	22.65%