2014

Relaxation Adaptive Memory Programming For The Resource Constrained Project Scheduling Problem

Robert Christopher-Lee Riley
University of Mississippi

RELAXATION ADAPTIVE MEMORY PROGRAMMING FOR THE RESOURCE CONSTRAINED PROJECT SCHEDULING PROBLEM

A Dissertation
presented in partial fulfillment of requirements
for the degree of Doctor of Philosophy
in the School of Business Administration
The University of Mississippi

by

ROBERT CHRISTOPHER-LEE RILEY

December 2014

ABSTRACT

The resource constrained project scheduling problem (RCPSP) is one of the most intractable problems in operations research; it is NP-hard in the strong sense. Due to the hardness of the problem, exact solution methods can only tackle instances of relatively small size. For larger instances commonly found in real applications heuristic solution methods are necessary to find near-optimal solutions within acceptable computation time limits.

In this study algorithms based on the relaxation adaptive memory programming (RAMP) method (Rego, 2005) are developed for the purpose of solving the RCPSP. The RAMP algorithms developed here combine mathematical relaxation, including Lagrangian relaxation and surrogate constraint relaxation, with tabu search and genetic algorithms. Computational tests are performed on an extensive set of benchmark instances. The results demonstrate the capability of the proposed approaches to the solution of RCPSPs of different sizes and characteristics and provide meaningful insights to the potential application of these approaches to other more complex resource-constrained scheduling problems.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

This study investigates the application of the relaxation adaptive memory programming (RAMP) method (Rego, 2005) to the resource constrained project scheduling problem (RCPSP). The objective is to test the method on a difficult scheduling problem of significant theoretical and practical relevance.

1.1 Project Scheduling

Projects are extremely prolific in almost all aspects of society, including business, government, and other non-profit organizations. Preparing for an event, developing a new product, constructing a building, conducting a geological survey, starting up a new plant, shutting down an existing plant, and information technology projects such as rolling out the latest version of software across an enterprise are just a few examples of various projects.

Projects have a distinct beginning and end and consist of various activities or tasks that must be completed. Activities that compose a project are usually interrelated in some way. The most common relationship is a simple precedence relation where one activity cannot start until one or more other activities are completed. Other relationships are also possible. For instance, an activity may be required to start within a certain amount of time before or after another activity starts or finishes (i.e. maximum and/or minimum time lags), or perhaps an activity must be performed during a specified time interval (i.e. time windows).

Often the objective is to complete a project in the shortest amount of time possible. Other possible objectives include completing a project by a set deadline, within a certain budget, for the minimum cost, or to maximize the project's net present value. In order to achieve any of

these objectives, a schedule that indicates when the various activities are planned to start and/or finish is necessary. For projects that must only consider simple precedence relations, a shortest duration, or minimum makespan, schedule can be generated elementarily by the critical path method (CPM) (Kelley & Walker, 1959). However, most projects require various types of resources, such as materials, equipment, and skilled workers. Usually the availability of these resources is limited. If the amount of resources available is not sufficient to satisfy the resource requirements of the CPM schedule, then the project is said to be resource constrained. The resulting resource constrained project scheduling problem (RCPSP) is the focus of this study.

As a generalization of the well-known job shop scheduling problem the RCPSP is strongly NP-hard (Blazewicz, Lenstra, & Kan, 1983). Due to the hardness of the problem, exact solution methods can only tackle instances of relatively small size. For larger instances commonly found in real applications heuristic solution methods are necessary to find near-optimal solutions within acceptable computation time limits.

A comprehensive discussion and computational analysis of heuristic methods for the RCPSP originates in two successive surveys by Hartmann and Kolisch (2000) and Kolisch and Hartmann (2006). State-of-the-art heuristics include metaheuristics, such as genetic algorithms, scatter search and path-relinking, simulated annealing, and tabu search, which are typically used in combination with some classical constructive methods and priority rules, and Lagrangian heuristics based on decomposition and optimization.

In addition to its immediate application to real-world projects, the RCPSP is often used as the fundamental building block for modeling more complex real-world projects. Many different extensions to the RCPSP have been proposed. Several of these extensions have been motivated by various real-world situations. Typically, these extensions modify or generalize the constraints

of the RCPSP. In some cases the heuristics designed for the RCPSP can be applied to the extended problem with little modification. In other cases, heuristics designed for the RCPSP provide a starting point for other types of resource constrained project scheduling problems.

1.2 Relaxation Adaptive Memory Programming

In this research the combination of fundamental principles of mathematical relaxation with fundamentals of adaptive memory programming as prescribed in the RAMP method is explored. There are two forms of RAMP—the basic RAMP and the primal-dual RAMP (PD-RAMP). The basic RAMP combines surrogate and Lagrangian relaxations with tabu search and path-relinking. The PD-RAMP extends the basic RAMP approach with scatter search and other evolutionary methods. The fundamental premise of the RAMP approach is that information gained solving a dual problem resulting from the mathematical relaxation of the original, or primal, problem can offer relevant insight for appropriate adaptive memory structures that cannot be obtained by considering the primal problem alone.

The basic RAMP approach is primarily concerned with exploring the solution space of the dual problem. The dual problem is usually obtained by applying a relaxation to the original problem. The solution to the dual problem may not be feasible for the original primal problem. Whenever the dual solution is not primal-feasible, it must be projected onto the primal-feasible solution space. An attempt is made to improve the primal-feasible solution in the primal solution space. A simple tabu search can be used, or other improvement methods may be used. The projection and improvement methods may also be combined into a single method. The results from the primal improvement method are used to generate a new relaxation problem (e.g. updating surrogate weights and Lagrangian multipliers).

The PD-RAMP approach extends the basic RAMP by including a significant primal local search method. In the basic RAMP the improvement method is characteristically a very basic method; however, in the PD-RAMP the primal-dual relationships are more thoroughly evaluated. This can be achieved by integrating scatter search, path-relinking, and other evolutionary methods.

1.3 Organization of the Dissertation

The dissertation is organized as follows. Chapter 2 provides background information and considerations for implementing heuristics for the RCSPSP. Chapter 3 discusses the methodology by which methods for solving the RCPSP are presently evaluated in the literature and provides results for several existing methodologies. Chapter 4 briefly reviews the existing literature that is immediately relevant to this study. Chapter 5 discusses some of the relevant issues in applying tabu search to the RCSP, describes the tabu search algorithms, and provides computational analysis. Chapter 6 describes the RAMP and PD-RAMP algorithms developed in this study for the RCPSP and provides computational analysis. Finally, Chapter 7 discusses conclusions and prospects.

2 CONSIDERATIONS FOR IMPLEMENTING HEURISTICS FOR THE RCPSP

When developing a method to solve the RCPSP a design decision must be made as to whether the method will work directly with a schedule or some indirect schedule representation. Relatively few heuristic methods work directly with a schedule. A likely explanation for this is the complexity of non-trivially modifying a schedule in such a way that the modified schedule is feasible. An infeasible schedule can be repaired; however, various repair methods often resort to a mechanism that either uses, or is equivalent to using, an indirect schedule representation. Utilization of an indirect schedule representation can avoid some of the complexities involved in manipulating a schedule directly. Several different representations have been proposed, but the representations most widely used in the literature seem to be based on either a priority vector or a priority list.

For any particular RCPSP instance, if at least one feasible schedule exists, then an infinite number of feasible schedules exist; though, the vast majority of these schedules are trivial. Given any feasible schedule, infinite trivial schedules can be created by merely incrementing the start time of the activity with the latest completion time. If the solution space of interest is limited to only schedules where there are no delays between activities on the critical path(s), several "equivalent alternate schedules" can be constructed by adjusting the start times of non-critical activities using the available slack time. Consider all schedules that yield the same objective function value to be "equivalent schedules." Several equivalent schedules may also exist for any given value of the objective function that are not just equivalent alternate schedules that vary only in the start times of the non-critical activities.

Equivalent schedules and equivalent alternate schedules pose a particular problem to local search techniques. Local search techniques generally search some neighborhood of solutions, that are “nearby” or “close” in some sense, in the hope of eventually finding a better solution. Some local search techniques suffer from the inability to escape from local optima. The RCPSP, in particular, poses a significant challenge to such local search techniques due to the fact that a locally optimal schedule may be surrounded by very similar schedules with the same makespan (i.e. equivalent schedules). This can be true whether a schedule or an indirect schedule representation is used to define the search space.

2.1 Schedule Representations and Schedule Generation

Indirect schedule representations generally take the form of some listing or other means of ordering the activities. A schedule generation scheme (SGS) uses this listing or ordering to prioritize activities for scheduling. As such, many SGS are list scheduling methods.

2.1.1 Schedule Generation

When an indirect schedule representation is used, some SGS is required to obtain an actual schedule. Kolisch (1996b) and Kolisch and Hartmann (1998) describe both serial SGS and parallel SGS. In the serial SGS activities are considered one at a time for scheduling. An activity is chosen from among the available activities, activities for which all predecessors have already been scheduled, and is scheduled at the earliest time possible that satisfies both precedence and resource constraints. The serial SGS generates active schedules where no activity can be started earlier without delaying another activity (Kolisch, 1996b).

In the parallel SGS, available activities are chosen to be scheduled at the current scheduling time until no other activities can be scheduled without exceeding available resources. Then the current schedule time is incremented to the smallest finish time of the currently active

activities. The process continues until all activities are scheduled. The parallel SGS generates non-delay schedules where no activity can be started earlier, even if preemption is allowed, without delaying another activity (Kolisch, 1996b).

Both the SGS and parallel SGS generate only feasible schedules, assuming that a precedence feasible schedule exists. It is known that the set of active schedules, generated by the serial SGS, will contain an optimal schedule when a regular performance measure, such as makespan minimization, is used (Sprecher, Kolisch, & Drexl, 1995). It is also known that the subset of active schedules generated by the parallel SGS, the non-delay schedules, may not necessarily contain an optimal schedule (Kolisch, 1996b).

2.1.2 Schedule Representations

Both the serial SGS and parallel SGS contain a decision step where an activity is chosen to be scheduled next. Indirect schedule representations provide the answer to that decision by indicating which activity should be chosen next. In the case of the parallel SGS, it is also possible to use a decision rule where a subset of the available activities are chosen together rather than choosing only one activity at a time. In this regard, almost all of the various schedule representations from the literature are different forms of activity prioritization.

When a SGS is applied to a schedule representation either the representation must specify only one activity each time an activity is to be chosen (i.e. no two activities can be assigned equal priority) or the SGS must include a tie-break decision rule to decide which activity to schedule next. In the former case an explicit tie-break rule may be applied to the schedule representation before applying the SGS. Alternatively, the operations applied to a schedule representation (e.g. neighborhood moves) may handle any tie-break decisions. Furthermore, it is possible to eliminate the possibility of assigning equal priorities to multiple activities as with, for

example, the priority list representation described below. A tie-break decision rule incorporated into a SGS can utilize information from the schedule generated up to that point. This information is not available to tie-break decisions which are applied directly to the schedule representation before a SGS is applied. It should be noted that a random or pseudorandom decision rule in a SGS is generally undesirable since, in that case, a single indirect schedule representation could potentially generate multiple different schedules.

2.1.2.1 Priority List

A priority list is simply an ordering of the activities. Often the activities are labeled by the set of integers $\{0, \dots, n, n+1\}$, hence the typical priority list is some permutation of those integer labels. When choosing the next activity to schedule, the SGS chooses the first activity in the list that is available for scheduling (i.e. all predecessors have already been scheduled). The SGS is responsible for ensuring precedence and resource feasibility as the schedule is generated. Noting that activities 0 and $n+1$ are artificial project start and end activities, respectively, for any particular RCPSP instance there are $n!$ possible priority lists. Although the two artificial activities could be allowed to appear anywhere in the priority list, the same schedule would result as when the artificial start and end activities are fixed to the beginning and end of the list, respectively.

The number of priority lists that are actually of interest can be reduced by considering the precedence constraints for a particular instance. Consider an activity i with x predecessors and y successors. Activity i should not be positioned any earlier in the list than $x+1$ or any later in the list than $n-y$ since at least the x predecessors should appear earlier in the list and the y successors should appear later in the list than activity i. Of course activity i can appear anywhere in the priority list and the SGS will create a feasible schedule; however, if activity i is not

positioned in the range defined, a priority list will exist where activity i is within this range for which the SGS will generate the same schedule.

It should be noted, however, that sometimes it is desirable to allow any activity to occupy any position in the list. This may be desirable, for example, in a method such as tabu search where moves are applied to the priority list in order to obtain other priority lists. The method may arrive at an improved solution with a fewer number of moves if activities may occupy any position in the list.

Multiple priority lists may map to the same schedule, but each priority list maps to exactly one schedule. The priority list provides an absolute ranking; no tie-break rules are necessary.

2.1.2.2 Activity List

An activity list is a priority list which is also precedence feasible—no activity can appear earlier in the list than any of its predecessors. A special case of the serial SGS can be used for activity lists where the decision step to choose the next activity to schedule simply chooses the next activity in the list. The SGS for activity lists does not check for precedence feasibility. Instead, any operation on an activity list must result in a precedence feasible ordering of the activities; otherwise, an activity list representation is not actually being used.

The activity list, by definition, is a topological ordering of the activities. The number of possible topological orderings of the activities decreases as the number of precedence constraints increases; however, it is impractical to simply generate all of the topological orderings and apply the SGS for activity lists.

As a special case priority list, multiple activity lists may map to the same schedule, but each activity list maps to exactly one schedule. The activity list also provides an absolute ranking; no tie-break rules are necessary.

2.1.2.3 Priority Vector

A priority vector simply assigns some priority value, either integer or real-valued, to each activity; however, there are several possible variations. The priority values are generally interpreted as ordinal rankings by a SGS; although, the priority values may be interpreted as interval or ratio measures when the priority vector is manipulated (e.g. by a neighborhood move), depending on how the schedule representation is defined. A parallel SGS that selects a set of activities to schedule at a decision point, rather than one activity at a time, may also interpret the priority values as interval or ratio measures.

One of the key properties of a specific priority vector implementation is whether activities may have equal priority values or whether activities must have unique priority values. If activities are allowed to have equal priority, the SGS used must include some type of tie-break rule when choosing which activity to schedule next. A potential benefit of allowing the SGS to break ties is that the tie-break decision rule can potentially utilize information from the schedule generated up to that point; this type of tie-break decision rule will be referred to as a "scheduling tie-break decision rule."

In general, a priority vector representation is not dependent on any parameter of the RCPSP instance aside from the number of activities. A SGS can be used to generate a feasible schedule from any priority vector. This property may be desirable when designing heuristics since a priority vector can be manipulated in any imaginable way that assigns a real or integer priority value to each activity. Unfortunately this property also means that every active schedule

can be represented by an infinite number of priority vectors. If all tie-break rules in the SGS are deterministic, each priority vector generates only one active schedule. This many-to-one relationship is not unique to the priority vector representation; the priority list and activity list representations described above both have this many-to-one relationship. A particular active schedule can be represented by a finite number of different priority lists, but the same schedule can be represented by an infinite number of priority vectors.

For all practical purposes, it does not matter whether the smallest or greatest priority value is considered to have a highest priority. The distinction must be made clear for implementation purposes, though. Unless stated otherwise, the smallest priority value will be considered to have the highest or most urgent priority.

2.2 Solution Representation and Neighborhood Structures

2.2.1 Moves for the Priority List Representation

For a list of integers where position is important, such as the priority list schedule representation, there is really only one fundamental move—eject an integer from its current position and insert it into another position and shift the other integers, preserving the existing ordering, to make room. This is the basic eject/insert move.

Define the move $PLEjPosInsPos(x, y)$ which ejects the activity in position x and inserts it into position y. Let the positions in the list be numbered $0,1, \dots, n, n+1$. If $x < y$ then each activity in positions $x + 1$ through y is shifted one position to the left in order to allow the activity originally in position x to be inserted into position y. Similarly, if $x > y$ then each activity in positions y through $x - 1$ is shifted one position to the right. If the eject and insert positions are consecutive, then the moves $PLEjPosInsPos(x, y)$ and $PLEjPosInsPos(y, x)$ will result in the same priority list. In the example below, the moves $PLEjPosInsPos(1,2)$ and

$PLEjPosInsPos(2,1)$ are shown to produce the same priority list. However, when the eject and insert positions are not consecutive the moves $PLEjPosInsPos(x, y)$ and $PLEjPosInsPos(y, x)$ will produce different priority lists. In the example below, the moves $PLEjPosInsPos(1,3)$ and $PLEjPosInsPos(3,1)$ are shown to produce different priority lists. Keeping the artificial start and end activities fixed at positions 0 and $n + 1$, there are $(n - 1)^2$ possible eject/insert moves that may be applied to a priority list.

Priority List PL								
Activity	0	2	3	4	1	6	5	7
Index/Position	0	1	2	3	4	5	6	7

Priority List PL after $PLEjPosInsPos(1,2)$ or $PLEjPosInsPos(2,1)$								
Activity	0	3	2	4	1	6	5	7
Index/Position	0	1	2	3	4	5	6	7

Priority List PL after $PLEjPosInsPos(1,3)$								
Activity	0	3	4	2	1	6	5	7
Index/Position	0	1	2	3	4	5	6	7

Priority List PL after $PLEjPosInsPos(3,1)$								
Activity	0	4	2	3	1	6	5	7
Index/Position	0	1	2	3	4	5	6	7

In addition to basing moves on positions in the list, moves can also be based on activities. Define the move $PLEjActInsBeforeAct(i, j)$ which ejects activity i and inserts it before activity j. This move can be viewed as a variation on the move $PLEjPosInsPos(x, y)$ where x is the position of activity i and y is the position immediately before the position of activity j. The move $PLEjActInsAfterAct(i, j)$ can be similarly defined. By allowing the artificial start activity 0 to serve as a possible i in insert after moves and the artificial end activity $n + 1$ to serve as a possible j in insert before moves, both the move $PLEjActInsBeforeAct(i, j)$ and the move $PLEjActInsAfterAct(i, j)$ generate the same set of resulting priority lists.

For the sake of completeness the following moves may also be defined: $PLEjPosInsBeforeAct(x, j)$, $PLEjPosInsAfterAct(x, j)$, and $PLEjActInsPos(i, y)$. It should be noted that if all possible combinations are considered for each of the moves, they all generate the same set of priority lists. The utility of defining the moves as shown here comes with the

addition of candidate lists and other strategies where only a subset of the move neighborhood is explored.

Another basic move for a list of integers is the swap move where two integers are selected and their positions are swapped. The swap move can be decomposed into two eject/insert moves; however, the swap move is usually considered to be a separate move since only a specific subset of all possible moves resulting from two subsequent eject/insert moves are actually swaps. Keeping the artificial start and end activities fixed at positions 0 and $n+1$, there are $n(n-1)/2$ possible swap moves.

Define the move $PLSwapPos(x, y)$ where the activities in positions x and y are swapped. Note that the move $PLSwapPos(a, b)$ results in the same priority list as the move $PLSwapPos(b, a)$, thus a common convention is to specify $x < y$.

Also define the move $PLSwapAct(i, j)$ where the positions of activities i and j are swapped. As with $PLSwapPos(x, y)$, the move $PLSwapAct(a, b)$ results in the same priority list as the move $PLSwapAct(b, a)$. Thus a common convention is to specify that activity i appears earlier in the priority list than activity j. For every possible move $PLSwapAct(i, j)$ there is an equivalent move $PLSwapPos(x, y)$. In the example below, $PLActSwap(4,6)$ is equivalent to $PLSwapPos(3,5)$. The neighborhoods defined by $PLSwapPos(x, y)$ and $PLSwapAct(i, j)$ will generate the same set of schedules.

Priority List PL								
Activity	0	2	3	4	1	6	5	7
Index/Position	0	1	2	3	4	5	6	7

Priority List PL after $PLSwapAct(4,6)$ or $PLSwapPos(3,5)$								
Activity	0	2	3	6	1	4	5	7
Index/Position	0	1	2	3	4	5	6	7

2.2.2 Moves for the Activity List Representation

An activity list is a precedence feasible priority list. As with priority lists, eject/insert and swap moves are the basic types of moves available for activity lists; however, by definition all activity lists must be precedence feasible.

There are two basic strategies to ensure that move operations on activity lists result in precedence feasible activity lists. The first strategy is an "elimination strategy." When using an elimination strategy, the same moves that are applied to priority lists are also applied to activity lists; however, any list resulting from a move operation that is not precedence feasible is eliminated from consideration or ignored. The second strategy is a "shift strategy" where move operations are defined that produce only precedence feasible activity lists. Typically, for a $move(x, y)$ the shift strategy shifts predecessor and successor activities of x and y, preserving their original relative order, as necessary in order to maintain precedence feasibility.

2.2.3 Moves for the Priority Vector Representation

Basic move operations on a priority vector include swapping priority values, increasing (or decreasing) a priority value by some amount, and setting a priority value to just below or just above some other priority value.

2.2.4 The Priority Vector Representation vs. the Priority List Representation

Using the priority vector representation is often equivalent to using the priority list representation. There are two notable cases where properties unique to the priority vector representation make it a preferable choice. A priority vector representation is used when it is desirable to have a scheduling tie-break decision rule used in the SGS. This type of rule cannot be used with priority lists since, by definition, a priority list cannot assign equal priority to multiple activities. A priority vector representation is also used when the property that all priority

vectors generate feasible schedules regardless of how the priority vector is manipulated is deemed beneficial. For example, this property can be useful in scatter search implementations.

This section discusses how the use of the priority vector representation is equivalent to the priority list representation in the absence of a scheduling tie-break decision rule and when not utilizing the property that all priority vectors generate feasible schedules regardless of how the priority vector is manipulated. The following discussion assumes that these two cases do not apply.

A unique priority list can be generated from a priority vector by listing the activities in order of decreasing priority (i.e. increasing priority values). In the case of a tie, simply apply a tie-break rule. Since both representations give the same ordering or prioritization of the activity, then the same schedule will be generated if the same SGS is applied. In the following consider a priority vector PV and a corresponding priority list PL shown below.

Priority Vector PV								
Priority Value	0	6	2	3	4	8	7	10
Index/Activity	0	1	2	3	4	5	6	7

Priority List PL								
Activity	0	2	3	4	1	6	5	7
Index/Position	0	1	2	3	4	5	6	7

Swapping the priority values of two activities in a priority vector is equivalent to swapping the positions of the same two activities in a priority list. Define the move $PVSwapAct(i,j)$ where the priority values of activities i and j are swapped. Also define the move $PLSwapAct(i,j)$ where the positions of activities i and j are swapped. Without loss of generality, assume that activity i has a smaller priority value (higher priority) in PV than activity j. Thus, activity i also appears earlier in PL than activity j. Both of these moves can be motivated by a desire to increase the priority of activity j relative to activity i. For example, the result of applying the moves $PVActSwap(4,6)$ and $PLActSwap(4,6)$ is shown below.

Priority Vector PV after $PVSwapAct(4,6)$								
Priority Value	0	6	2	3	7	8	4	10
Index/Activity	0	1	2	3	4	5	6	7

Priority List PL after $PLSwapAct(4,6)$								
Activity	0	2	3	6	1	4	5	7
Index/Position	0	1	2	3	4	5	6	7

The neighborhoods defined by these swap moves will generate the same set of schedules. It should be noted that the implementation of $PVSwapAct(i,j)$ is simpler than that of $PLSwapAct(i,j)$. For the move $PLSwapAct(i,j)$ the locations of activities i and j must first be determined before they can be swapped. When a priority vector is used the SGS must find the activity with the highest priority from among the available activities; however, when a priority list is used the SGS must determine the next unscheduled available activity in the list.

Consider the move $PLSwapPos(x,y)$ where the activities in positions x and y are swapped with $x < y$. This move can be motivated by a desire to increase the priority of the activity in position y relative to the activity in position x. For every possible move $PLSwapAct(i,j)$ there is an equivalent move $PLSwapPos(x,y)$. In the example, $PLActSwap(4,6)$ is equivalent to $PLSwapPos(3,5)$. The neighborhood defined by $PLSwapPos(x,y)$ will generate the same set of schedules generated by $PLSwapAct(i,j)$ and $PVSwapPos(i,j)$.

A move equivalent to $PLSwapPos(x,y)$ could be defined for priority vectors that swapped the xth highest priority value with the yth highest priority value. Such a move would require knowing the relative order of the rankings. One way to accomplish this is to sort the priority values, find the values in the xth and yth positions, find those values in the priority vectors, and swap those two positions. Another way to accomplish this is to sort the activities by decreasing priority (increasing priority values), identify the activities in the xth and yth positions, and swap the priority values of those two activities in the priority vector. Note that when the activities are sorted by decreasing priority, the priority list corresponding to the priority

vector is created. In this case it should be clear that an implementation of the priority vector move would require more computational effort than an implementation of the priority list move $PLSwapPos(x, y)$.

3 EVALUATION OF SOLUTION METHODS FOR THE RCPSP

3.1 RCPSP Test Problems

Kolisch and Sprecher (1996) and Kolisch, Schwindt, and Sprecher (1999) describe the PSPLIB library of project scheduling benchmark problems. These problems serve as the current standard test problems in the literature used to evaluate algorithms for solving the RCPSP. The problem instances in the library were generated by the project generator ProGen which is described in detail in Kolisch, Sprecher, and Drexl (1995). The instances were created by varying three parameters in a systematic fashion. The first parameter is the "network complexity" (NC) which describes the average number of direct successors for an activity. The second parameter is the "resource factor" (RF) which reflects the average number of resource types required by an activity. The third parameter is the "resource strength" (RS) which is a scaling factor related to the scarcity of the resources. If RS = 0 then the resource availability of each resource is specified to be the smallest amount that allows resource feasibility. If RS = 1 then resource availability of each resource is specified to be the maximum peak per-period usage of that resource in the CPM early start schedule. Thus, for instances where RS = 1, the optimal minimum makespan will be equal to the critical path lower bound; however, makespan minimization is not always the desired objective.

The library currently contains RCPSP instance sets with 30, 60, 90, and 120 activities commonly referred to as the j30, j60, j90, and j120 instances, respectively. The j30, j60, and j90 instance sets include 480 instances that represent 10 instances for every combination of the parameters NC = 1.50, 1.80, and 2.10; RF = 0.25, 0.50, 0.75, and 1.00; and RS = 0.20, 0.50,

0.70, and 1.00 (Kolisch et al., 1999; Kolisch & Sprecher, 1996). The j120 instance set includes 600 instances that represent 10 instances for every combination of the parameters NC = 1.50, 1.80, and 2.10; RF = 0.25, 0.50, 0.75, and 1.00; and RS = 0.1, 0.20, 0.3, 0.4, and 0.50 (Kolisch et al., 1999).

For the makespan minimization objective, the optimal solution is known for each instance in the j30 instance set. For the other instance sets, the best known lower and upper bounds are available as part of the library.

Prior to the development of the PSPLIB instances, popular instances sets were those of Patterson (1984), Alvarez-Valdes and Tamarit (1989), and Boctor (1993). The Patterson (1984) set is composed of 110 instances that represent all of the readily available multi-resource problems that existed in the literature at that time. Kolisch et al. (1995) present arguments against continued use of the Patterson (1984) set of problems.

3.2 Evaluation and Comparison Methodology

In the literature, results are typically reported as averages across all instances in a particular PSPLIB instance set as opposed to reporting results for each individual instance. The ideal benchmark is the average percent deviation from the known optimal makespan. Since the optimal makespan is not known for all instances in the j60, j90, and j120 PSPLIB instance sets, some authors prefer to report the average percent deviation from the current best known upper bounds. However, since the best known upper bounds are improved upon from time-to-time it becomes difficult to compare results since different papers use different benchmarks. In the absence of another benchmark this may be acceptable; however, in the case of the RCPSP the critical path lower bound is a preferable benchmark. The critical path lower bound is a well-defined lower bound that can be calculated easily for any precedence-feasible RCPSP and is not

subject to changing over time. The optimal makespan will only equal the critical path lower bound when the project is not resource constrained (i.e. RS = 1). Thus the average percent deviation from the critical path lower bound can only approach zero for these RCPSP instances. When comparing methods, a lower average deviation from the critical path lower bound indicates better results.

Hartmann and Kolisch (2000) presents an experimental evaluation of heuristics for the RCPSP that is later updated by Kolisch and Hartmann (2006). Much of the literature since these surveys follows the same experimental design and present results in the same format. The PSPLIB j30, j60, and j120 instance sets are used. The objective is makespan minimization. Algorithms are compared on the basis of the average percent deviation from a specified benchmark. For the j30 PSPLIB instance set, the benchmark is the known optimal makespan. For the j60 and j120 instances, the benchmark is the critical path lower bound. In Hartmann and Kolisch (2000) the lowest makespan found by any of the tested heuristic was also used as a second benchmark; however, this benchmark was not included in Kolisch and Hartmann (2006).

In order to form a basis for comparison, stopping criteria is defined that limits the number of schedules generated or partially generated. Stopping criteria is defined as limits of 1,000 and 5,000 generated schedules in Hartmann and Kolisch (2000). In addition Kolisch and Hartmann (2006) present results for 50,000 generated schedules. It is assumed that the computational effort required for constructing a schedule is similar across different heuristics and algorithms. If this assumption is accepted, then results can be readily compared even if they are obtained on different computer architectures and operating systems. Kolisch and Hartmann (2006) points out that this test method is also independent of compilers and implementation skills therefore

heuristic concepts are evaluated rather than program codes. However, it is noted in both surveys that the stopping criteria cannot be easily applied to all heuristics.

3.3 Results from the Extant Literature

Kolisch and Hartmann (2006) and Gonçalves, Resende, and Mendes (2011) provide summary results of competitive heuristics for the RCPSP. Table 3.1 through Table 3.3 below include the union of the results summarized from these two sources. The experimental design for obtaining results and the format of the results follow Hartmann and Kolisch (2000) and Kolisch and Hartmann (2006). Results from other selected references are also included. Where appropriate, original source references to working papers and technical reports that have been since published in peer-reviewed journals have been updated to cite the peer-reviewed journal.

The source references listed in the tables below are primarily references for the methods. Some of the results in the tables below do not come directly from its source reference but rather from appropriate additional results provided by the authors of the source references for inclusion in Hartmann and Kolisch (2000) and Kolisch and Hartmann (2006).

The sorting of the results in the tables below follows the criteria of Kolisch and Hartmann (2006). Methods are sorted based upon the results for 50,000 generated schedules. Ties are broken by the results for 5,000 generated schedules.

For the j30 PSPLIB instances, results are listed as the average percent deviation from the known optimal makespans. For the j60 and j120 PSPLIB instances, results are listed as the average percent deviation from the well-known critical path lower bounds.

Table 3.1 – Results from the extant literature – PSPLIB j30 instances – average percent deviations from optimal makespan

Source Reference	Schedule Limits				
	1,000	5,000	50,000	100,000	500,000
Kochetov and Stolyar (2003)	0.10	0.04	0.00	-	-
Mendes, Gonçalves, and Resende (2009)	0.06	0.02	0.01	0.01	0.01
Gonçalves et al. (2011)	0.32	0.02	0.01	0.01	0.01
Debels, De Reyck, Leus, and Vanhoucke (2006)	0.27	0.11	0.01	0.01	0.01
Debels and Vanhoucke (2007)	0.15	0.04	0.02	-	-
Valls, Ballestín, and Quintanilla (2008)	0.27	0.06	0.02	-	-
Valls, Ballestín, and Quintanilla (2005) (GA)	0.34	0.20	0.02	-	-
Alcaraz, Maroto, and Ruiz (2004)	0.25	0.06	0.03	-	-
Alcaraz and Maroto (2001)	0.33	0.12	-	-	-
Tormos and Lova (2003b)	0.25	0.13	0.05	-	-
Nonobe and Ibaraki (2002)	0.46	0.16	0.05	-	-
Tormos and Lova (2001)	0.30	0.16	0.07	-	-
Hartmann (2002)	0.38	0.22	0.08	-	-
Hartmann (1998) (GA activity list)	0.54	0.25	0.08	-	-
Tormos and Lova (2003a)	0.30	0.17	0.09	-	-
Klein (2000)	0.42	0.17	-	-	-
Valls et al. (2005) (Sampling)	0.46	0.28	0.11	-	-
Bouleimen and Lecocq (2003)	0.38	0.23	-	-	-
Coelho and Tavares (2003) (GA)	0.74	0.33	0.16	-	-
Schirmer (2000)	0.65	0.44	-	-	-
Baar, Brucker, and Knust (1998)	0.86	0.44	-	-	-
Kolisch and Drexl (1996)	0.74	0.52	-	-	-
Hartmann (1998) (GA random key)	1.03	0.56	0.23	-	-
Kolisch (1996b) (Sampling LFT, serial SGS)	0.83	0.53	0.27	-	-
Coelho and Tavares (2003) (Sampling)	0.81	0.54	0.28	-	-
Kolisch (1995) (Sampling random, serial SGS)	1.44	1.00	0.51	-	-
Hartmann (1998) (GA priority rule)	1.38	1.12	0.88	-	-
Kolisch (1996a, 1996b) (Sampling WCS)	1.40	1.28	-	-	-
Kolisch (1996b) (Sampling LFT, parallel SGS)	1.40	1.29	1.13	-	-
Kolisch (1995) (Sampling random, parallel SGS)	1.77	1.48	1.22	-	-
Leon and Balakrishnan (1995)	2.08	1.59	-	-	-

Table 3.2 – Results from the extant literature – PSPLIB j60 instances – average percent deviations from critical path lower bound

Source Reference	Schedule Limits				
	1,000	5,000	50,000	100,000	500,000
Gonçalves et al. (2011)	-	11.56	10.57	10.51	10.49
Mendes et al. (2009)	11.72	11.04	10.67	10.67	10.67
Debels and Vanhoucke (2005)	11.45	10.95	10.68	-	-
Debels et al. (2006)	11.73	11.10	10.71	-	10.53
Valls et al. (2008)	11.56	11.10	10.73	-	-
Kochetov and Stolyar (2003)	11.71	11.17	10.74	-	-
Valls et al. (2005) (GA)	12.21	11.27	10.74	-	-
Alcaraz et al. (2004)	11.89	11.19	10.84	-	-
Hartmann (2002)	12.21	11.70	11.21	-	-
Hartmann (1998) (GA activity list)	12.68	11.89	11.23	-	-
Tormos and Lova (2003)	11.88	11.62	11.36	-	-
Tormos and Lova (2003a)	12.14	11.82	11.47	-	-
Alcaraz and Maroto (2001)	12.57	11.86	-	-	-
Tormos and Lova (2001)	12.18	11.87	11.54	-	-
Bouleimen and Lecocq (2003)	12.75	11.90	-	-	-
Klein (2000)	12.77	12.03	-	-	-
Nonobe and Ibaraki (2002)	12.97	12.18	11.58	-	-
Valls et al. (2005) (Sampling)	12.73	12.35	11.94	-	-
Schirmer (2000)	12.94	12.58	-	-	-
Coelho and Tavares (2003) (GA)	13.28	12.63	11.94	-	-
Hartmann (1998) (GA random key)	14.68	13.32	12.25	-	-
Hartmann (1998) (GA priority rule)	13.30	12.74	12.26	-	-
Kolisch and Drexl (1996)	13.51	13.06	-	-	-
Kolisch (1996a, 1996b) (Sampling WCS)	13.66	13.21	-	-	-
Coelho and Tavares (2003) (Sampling)	13.80	13.31	12.83	-	-
Kolisch (1996b) (Sampling LFT, parallel SGS)	13.59	13.23	12.85	-	-
Baar et al. (1998)	13.80	13.48	-	-	-
Leon and Balakrishnan (1995)	14.33	13.49	-	-	-
Kolisch (1996b) (Sampling LFT, serial SGS)	13.96	13.53	12.97	-	-
Kolisch (1995) (Sampling random, parallel SGS)	14.89	14.30	13.66	-	-
Kolisch (1995) (Sampling random, serial SGS)	15.94	15.17	14.22	-	-

Table 3.3 – Results from the extant literature – PSPLIB j120 instances – average percent deviations from critical path lower bound

Source Reference	Schedule Limits				
	1,000	5,000	50,000	100,000	500,000
Debels and Vanhoucke (2005)	34.19	32.34	30.82	-	-
Valls et al. (2008)	34.07	32.54	31.24	-	-
Mendes et al. (2009)	35.87	33.03	31.44	31.32	31.2
Gonçalves et al. (2011)	-	35.94	32.76	31.63	30.08
Alcaraz et al. (2004)	36.53	33.91	31.49	-	-
Debels et al. (2006)	35.22	33.10	31.57	-	30.48
Valls et al. (2005) (GA)	35.39	33.24	31.58	-	-
Kochetov and Stolyar (2003)	34.74	33.36	32.06	-	-
Valls et al. (2005) (Population based)	35.18	34.02	32.81	-	-
Hartmann (2002)	37.19	35.39	33.21	-	-
Tormos and Lova (2003b)	35.01	34.41	33.71	-	-
Merkle, Middendorf, and Schmeck (2002)	-	35.43	-	-	-
Hartmann (1998) (GA activity list)	39.37	36.74	34.03	-	-
Tormos and Lova (2003a)	36.24	35.56	34.77	-	-
Tormos and Lova (2001)	36.49	35.81	35.01	-	-
Alcaraz and Maroto (2001)	39.36	36.57	-	-	-
Nonobe and Ibaraki (2002)	40.86	37.88	35.85	-	-
Coelho and Tavares (2003) (GA)	39.97	38.41	36.44	-	-
Valls et al. (2005) (Sampling)	38.21	37.47	36.46	-	-
Bouleimen and Lecocq (2003)	42.81	37.68	-	-	-
Hartmann (1998) (GA priority rule)	39.93	38.49	36.51	-	-
Schirmer (2000)	39.85	38.70	-	-	-
Kolisch (1996b) (Sampling LFT, parallel SGS)	39.60	38.75	37.74	-	-
Kolisch (1996a, 1996b) (Sampling WCS)	39.65	38.77	-	-	-
Hartmann (1998) (GA random key)	45.82	42.25	38.83	-	-
Kolisch and Drexl (1996)	41.37	40.45	-	-	-
Coelho and Tavares (2003) (Sampling)	41.36	40.46	39.41	-	-
Leon and Balakrishnan (1995)	42.91	40.69	-	-	-
Kolisch (1996b) (Sampling LFT, serial SGS)	42.84	41.84	40.63	-	-
Kolisch (1995) (Sampling random, parallel SGS)	44.46	43.05	41.44	-	-
Kolisch (1995) (Sampling random, serial SGS)	49.25	47.61	45.60	-	-

4 RELATED LITRATURE

The resource constrained scheduling literature is vast. Only literature that has served as a basis or source of inspiration for this study is included.

4.1 Relaxation Methods

A relaxation method modifies or eliminates constraints of a combinatorial optimization problem in order to create a less constrained problem that is easier to solve. Lagrangian relaxation and surrogate constraint relaxation are two examples of relaxation methods. In Lagrangian relaxation a subset of the constraints are relaxed by replacing them with a penalty term in the objective function that reflects the degree of violation of the relaxed constraints. In surrogate constraint relaxation, a surrogate constraint is a non-negative linear combination of a subset of the constraints that replaces those constraints.

Christofides, Alvarez-Valdes, and Tamarit (1987) examine four different lower bounds for the RCPSP. One of these bounds is the Lagrangian relaxation of the resource constraints in a time-indexed integer programming formulation of the RCPSP. It is pointed out that the Lagrangian relaxation problem can be viewed as a generalization of a longest path computation where both the completion time and the costs of starting an activity at a particular time (i.e. start-time dependent costs) must both be minimized. These costs correspond to the coefficients of the time-indexed variables due to the Lagrangian multipliers in the objective function.

Möhring, Schulz, Stork, and Uetz (1999, 2003) describe a method of solving the project scheduling problem with start-time dependent costs by transforming it into a minimum cut problem (cf. Goldberg & Tarjan, 1988). The objective of the project scheduling problem with

start-time dependent costs is to minimize the sum of the start-time dependent costs. Resource constraints are not considered in this particular project scheduling problem. The transformation is based upon a time-indexed integer programming formulation (Christofides et al., 1987; Pritsker, Watters, & Wolfe, 1969). There is a time-indexed variable for each possible start time of an activity associated with a predetermined maximum time horizon upper bound. Each time-indexed variable is represented by a directed arc, referred to as an assignment arc, with a capacity equal to the start-time dependent cost represented by the time-indexed variables. The temporal constraints are incorporated as infinite capacity directed temporal arcs. Once the transformation to a minimum cut problem is complete, a maximum flow/minimum cut algorithm can be applied to determine a schedule that minimizes the sum of the start-time dependent costs.

Möhring et al. (1999, 2003) also demonstrate how lower bounds for the RCPSP can be obtained by transforming it to a minimum cut problem. Lagrangian relaxation is used to relax the resource constraints of the RCPSP (Christofides et al., 1987), and suitable weights are introduced that take the place of the start-time dependent costs. Note that the resulting minimum cut problem does not solve the RCPSP directly but a relaxation of the RCPSP; however, the optimal solution obtained for the minimum cut problem is a valid lower bound for the RCPSP.

4.2 Tabu Search

Tabu search (Glover, 1989, 1990) is a metaheuristic technique that attempts to guide a local search beyond a local optimum while preventing a return to that same local optimum. Tabu search modifies a current working solution by applying moves that manipulates attributes of the solution. As the search progresses tabu restrictions are imposed that classify certain moves as forbidden or tabu. The short term memory component typically consists of a tabu list and a tabu tenure but may also include aspiration criteria and candidate lists (Glover & Laguna, 1997). The

tabu tenure, or tabu list size, indicates how long a move, or a particular attribute of a move, will remain tabu. The tabu list is composed attributes of moves that are currently tabu. Aspiration criteria can be used to allow a tabu move to be accepted under certain circumstances. A common aspiration criterion is accepting a tabu move if it results in a solution that is better than any solution previously found. Sometimes it is impractical to evaluate every possible move. In these cases a candidate list of moves can be created that consists of only a subset of the possible moves. Either general strategies or strategies based on the context of the problem can be utilized to create candidate lists (Glover & Laguna, 1997). Long-term memory can also be utilized in order to diversify the search by maintaining frequency-based information for solutions previously visited and guiding the search into previously unexplored regions (Glover, 1989).

Both Icmeli and Erenguc (1994) and Thomas and Salhi (1998) apply tabu search to the RCPSP by directly modifying schedules. Icmeli and Erenguc (1994) actually consider an extension of the RCPSP—the resource constrained project scheduling problem with discounted cash flows (RCPSPDC) where the objective is to maximize the net present value of the cash flows associated with each activity while satisfying the resource and precedence restrictions. Although the RCPSPDC is an extension of the RCPSP, their tabu search methodology is applicable to the RCPSP. Rather than viewing the RCPSPDC as an extension of the RCPSP, the RCPSPDC can instead be considered as a RCPSP with a specific objective function instead of the most common objective of makespan minimization. Icmeli and Erenguc (1994) modify a schedule by either incrementing or decrementing an activity's current completion time. This type of modification may result in an infeasible schedule. Infeasibility is dealt with by imposing a penalty term to the objective function and allowing the search to continue. The penalty term is

sufficiently large to allow any feasible schedule to be favored over an infeasible schedule. An infeasible schedule is only selected when no non-tabu move results in a feasible schedule.

Thomas and Salhi (1998) employ a candidate list of moves that maintains a schedule's precedence feasibility but may violate resource constraints. They use the SHIFT heuristic (Thomas & Salhi, 1995) to repair schedules for resource feasibility before allowing the search to continue. As described in Thomas and Salhi (1998) the SHIFT heuristic creates a time window at an infeasible point by shifting parts of the schedule forward. Activities involved in resource overutilization are shifted into the window, but these activities may also be shifted earlier in the schedule as long precedence and resource constraints are satisfied. However, the method still allows "small" infeasibilities within the time windows. Thus the "repaired" schedule may still be infeasible. In addition the authors note that the SHIFT heuristic can result in a substantial restructuring of the schedule. Continuing the search with such a substantially restructured schedule (compared to the infeasible schedule that resulted from the move operation) might be better considered as a restart rather than selecting the best "neighborhood" schedule as the next working schedule.

Icmeli and Erenguc (1994) presents two tabu search procedures for solving the RCPSP with discounted cash flows (RCPSPDC). The first procedure, TABU-S, utilizes only short term memory. The second procedure, TABU-L, modifies TABU-S by incorporating long term memory. A parallel SGS is used to obtain an initial solution. The tabu search TABU-S operates directly on the current working schedule. A move is defined as completing an activity either one time unit earlier or one time unit later than in the current working schedule. This provides a neighborhood size of $2n$, where n is the number of activities in the project. Since these moves may result in an infeasible solution, an evaluation function is calculated for each move that

includes the objective function, the net present value of the cash flows, plus a penalty term. The penalty term for a particular candidate solution is the total units of violation of the precedence and resource constraints multiplied by a sufficiently large negative constant, where the constraints are represented by linear equations from a time-indexed integer program. No penalty is associated with a feasible solution. The non-tabu move with the largest evaluation function value is selected as the best move. The evaluation function value of the best move is recorded in the tabu list, which is of size 10. Any move that results in an evaluation function value currently on the tabu list is a tabu move. The second procedure, TABU-L, includes four stages, or multi-starts, of 100 iterations each. A second tabu list, L-list, is added that records the starting solution values of each stage in order to prevent revisiting a previous stage's starting solution. The first parallel SGS used in TABU-S provides the starting solution for the first stage of TABU-L. In subsequent stages, the feasible solution found in the previous stage that is "most distant" from the best solution found in that stage is used as the starting solution for the next stage. Distance between two solutions is calculated as the Euclidian distance between the vectors of activity completion times. The evaluation function value of the best solution at each stage is recorded permanently in the original tabu list. These values cause the length of original tabu list to be extended by one each iteration. Both TABU-S and TABU-L are limited to a maximum of 400 iterations.

Pinson, Prins, and Rullier (1994) presents one of the earliest applications of tabu search to the RCPSP. Five tabu search variations are provided, all of which utilize precedence feasible activity lists as an indirect schedule representation. Carlier's strict order algorithm, which is a serial SGS, (Carlier, Moukrim, & Xu, 2010) is used to construct precedence and resource feasible schedules. Three neighborhoods are defined: V1, V2, and V3. Neighborhood V1

consists of the $n-1$ lists obtained by swapping two adjacent activities in the current working activity list L. Neighborhood V2 consists of the $n(n-1)/2$ lists possible from swaps of any two activities in L. The paper states that neighborhood V3 consists of the $n(n-1)/2$ possible lists obtained from moving any activity to a different position in L, but the neighborhood described should have $(n-1)^2$ possible moves. Activity lists that violate precedence constraints are immediately discarded. The tabu list stores, for the past 15 moves, the pairs of swapped activities, or the single activity in V3, along with their positions before the move. Any move that restores one of these configurations is considered tabu. The aspiration criteria of improving the best solution found overrides tabu status. The move that provides the schedule with the smallest makespan is chosen. In the case of ties either the move closest to the end of L or the move closest to the beginning of L is chosen. Five different tabu search procedures are designated as TABU1 to TABU5. These tabu searches differ in neighborhood choice and choice of tiebreaking rule; only TABU4 breaks ties by choosing the move closest to the beginning of L. TABU1 and TABU2 use neighborhoods V1 and V2, respectively. TABU3 first applies TABU1 and then switches to TABU2 starting with the best solution from TABU1. TABU4 is the same as TABU3 except for the tiebreaking rule. TABU5 first applies TABU1 and then switches neighborhoods to V3 starting with the best solution from TABU1. It is noted that using only neighborhood V3 gives poor results. Each different neighborhood is allowed 100 iterations. So TABU1 and TABU2 perform 100 iterations, and TABU3, TABU4, and TABU5 perform 200 iterations.

Lee and Kim (1996) develop a tabu search procedure which utilizes a priority list as an indirect schedule representation. Schedules are generated with a parallel SGS. The starting priority values are generated randomly from a [0,1] uniform distribution. The procedure requires that the activities in the project network to be labeled, assigning a numerical index to each

activity, such that an ordering of activities by increasing index number is a topological ordering. A swap, or "interchange," move is defined whereby two activities are selected and their priority values are swapped. The first activity is selected randomly. The second activity is selected randomly from activities whose indices are between $i_1 - 4m$ and $i_1 + 4m$, where i_1 is the index of the first activity and m is the maximum number of predecessors or successors among all of the activities. The authors point out that, due to the manner in which the activities are indexed, activities with large difference in indices are unlikely to compete for resources. The swap move neighborhood size is $n!/(n-2)!$. Neighborhood reduction is employed; only $2n$ neighbors are randomly generated each iteration. A move swapping the priorities of activities i and j is tabu if i and j have been swapped recently. A static tabu list of size six is used. This tabu search procedure also utilizes long-term frequency-based memory. Every time a swap of activities i and j occurs, f_{ij} is incremented. Each iteration a non-tabu move is selected with the smallest value of $Makespan + \beta f_{ij}$, where $\beta = 10$ if $Makespan$ is worse than the current working solution's makespan and $\beta = 0$ otherwise. However, the aspiration criteria of improving the best solution found so far overrides tabu status. This procedure does not include diversification or intensification phases. This study included a simulated annealing procedure, a tabu search procedure, and a genetic algorithm procedure. Parameter values were selected experimentally first for the simulated annealing procedure. Parameter values for the tabu search and genetic algorithm procedures were selected experimentally to provide the best results with computation times comparable to the simulated annealing procedure for problems of the same size.

Two different tabu search algorithms are presented by Baar et al. (1998). Both algorithms utilize dynamic tabu list sizes, invoke an immediate selection procedure, utilize two restarts, and initially start from the best solution found by applying several different priority rules to a serial

SGS. The first algorithm is based on so-called critical arcs and utilizes a precedence feasible activity list solution representation from which an active schedule is obtained by applying a serial SGS. A directed graph is created from the current working solution where each activity is represented by a single node and an arc from one node/activity to another exists only if the second activity starts immediately after the first activity finishes (i.e. no delay). A critical path in this graph is defined as a simple path from the artificial start node to the artificial end node. An arc (i, j) on the critical path is a critical arc if (i, j) is not in the set of precedence constraints, which includes the precedence constraints prescribed by the problem instance and additional precedence constraints induced by transitivity and immediate selection. Three different move operators are used: a shift-operator, a backshift-operator, and a frontshift-operator. The shift-operator is defined for a critical arc (i, j) where i appears before j in L. This operator moves i and all successors of i appearing in L before j to immediately after L. The backshift-operator is defined for a critical arc (i, j) where j appears before i in L. The first activity in L appearing after i, that is not a successor of i, is moved to the position immediately before i. Symmetrically, the frontshift-operator is defined for a critical arc (i, j) where j appears before i in L. The last activity in L appearing before j, that is not a predecessor of j, is moved to the position immediately after j. The motivation for the backshift- and frontshift-operators is that the distance between i and j is increased.

The second tabu search algorithm presented by Baar et al. (1998) is based on parallelity and utilizes schedule schemes (Brucker, Knust, Schoo, & Thiele, 1998). This algorithm is also described in Brucker and Knust (1999). A schedule scheme (C,D,N,F) is composed of four disjoint relations C, D, N, and F that represent conjunctions, disjunctions, parallelity relations, and flexibility relations, respectively. A schedule scheme (C,D,N,F) represents the set

S(C,D,N,F) of schedules. A schedule scheme may represent both feasible and infeasible schedules due to the resource constraints. The heuristic used to construct a feasible schedule from a schedule scheme does not necessarily construct active schedules. The authors define a parallel critical path as a sequence of activities $i_o, i_1, \ldots, i_l$, with i_o representing the artificial start activity and i_l representing the artificial end activity, where for i_v, i_{v+1} either i_{v+1} starts when i_v ends or i_v starts before i_{v+1} and both activities are processed in parallel for at least one time unit. Four different types of operators are introduced that operate on a schedule scheme; the first three of these operators are restricted to activities on a parallel critical path.

Several candidate list strategies are suggested by Rangaswamy, Jain, and Glover (1998); however, only one of the strategies is implemented. A precedence feasible activity list is employed as an indirect solution representation. It appears that a serial SGS is used to obtain schedules. A lexicographical ordering of the activities, which happens to also be a topological ordering for the problem instances considered, is used to obtain the starting solution. Simple $(eject\ i, insert\ j)$ moves are used where activity i is ejected from its current position and inserted after activity j. A successive filtration strategy is proposed, which is the strategy that is actually implemented, as a bi-level candidate list where the top level of the list is all delayed activities on all critical paths of the current working schedule. A bottom level list is created for each activity in the top level list. The list for each activity in the top level list includes all activities that appear earlier in the working activity list and are in progress just before the current start time of the top level activity. Also, moves that insert each activity of the top level list as early as possible in the current working activity list are considered. A static tabu list of size 8 is used. In addition, various alternative implementations of a bi-level candidate list are suggested. One such suggestion is a pause and project strategy for the bottom level list, which becomes a

type of sequential fan strategy. A separate aspiration plus strategy is suggested where moves are considered until some aspiration criteria is met and continues for some specified number of moves. Also, a minimum and maximum number of moves to consider are specified. An example is shown where the aspiration criteria is the first move that either produces a better makespan, creates a positive influence on the search trajectory (e.g. reduces the maximum delay in a partial schedule), or improves a secondary objective subject to the constraints.

Thomas and Salhi (1998) introduces a tabu search that operates directly on schedules. Three different types of moves are used: swap, type 1 insertion, and type 2 insertion. All three moves involve two activities i and j where j is not a predecessor of i. If the current working schedule does not start both activities at the same time, then the swap move swaps their start times. Again if the activities are not currently scheduled to start at the same time, a type 1 insertion (i, j) will set the start time of i to the current start time of j. If the current working schedule starts both activities at the same time, a type 2 insertion (i, j) will set the start time of i to the finish current finish time of j. The moves are constructed such that precedence feasibility is maintained; however, the schedules resulting from these moves may be infeasible with regards to the resource constraints. A SHIFT heuristic routine developed by Thomas and Salhi (1995) is used to repair schedules for resource feasibility.

Tsai and D. Gemmill (1998) present a tabu search for the RCPSP that is also applied to the RCPSP with stochastic activity durations. A precedence feasible activity list is used as an indirect schedule representation, and a serial SGS is used to generate schedules. Swap, or interchange, moves are utilized. Two tabu lists, TabuListC and TabuListNC, that consist of critical and non-critical activities respectively, where an activity is considered critical if it is on a critical path of the schedule generated by CPM/PERT when resource constraints are disregarded,

are utilized. The tabu tenure for both lists is $\sqrt{n}/2$. The aspiration criteria of improving the best makespan found overrides tabu status. A candidate list of moves is created by selecting two activities randomly and swapping their positions in the activity list. If the generated sequence is not (precedence) feasible, the move is discarded. Moves are added to the list until $\sqrt{n}$ moves are on the candidate list. The moves on the candidate list are evaluated and the best admissible move, a non-tabu move or a move which satisfies the aspiration criteria, is accepted. However, based on the algorithm description and the variable NotFindAdmissible, the candidate list may not contain an admissible move in which case it appears that a new candidate list is created. This tabu search is also applied to the RCPSP with stochastic activity durations. Expected activity durations are calculated according to PERT assuming that activity durations are β distributed. For a particular candidate activity list an activity duration is randomly drawn from the β distribution for each activity and the resulting schedule is generated. This is repeated, for each candidate activity list, 100 times in order to determine the average project duration resulting from the candidate activity list. This expected project duration is used in the determination of the best admissible move. Two stopping criteria are used. The first is based on the number of candidate lists created since the best schedule found so far. The second is based on the number of candidate lists created since an admissible move was found. Several values for the maximum numbers of candidate lists created, MaxTryOnBetter and MaxTryonAdmissible respectively, are considered. However, MaxTryOnAdmissible is always 10 times larger than MaxTryOnBetter in the results presented.

Brucker and Knust (1999) represents the tabu search based on parallelity from Baar et al. (1998). Minor tuning is apparent from the slightly improved results.

Klein (2000) presents a reactive tabu search for the generalized resource constrained project scheduling problem or GRCPSP. The GRCPSP allows resource availabilities to vary over time, non-negative minimum time lags, and activity specific release and due dates. A precedence feasible activity list is used as an indirect schedule representation, and a serial SGS is used to obtain schedules. However, a schedule created by this SGS may violate release and due dates. When such a time window is violated, a penalty equal to the sum of all job durations is added to the schedule makespan for the purpose of evaluating moves. Swap moves $swap(h, j)$ are used where the positions of h and j are interchanged, successors of h appearing in the list before j are inserted immediately after h preserving their relative sequence, and predecessors of j appearing in the list after h are similarly inserted immediately before j preserving their relative sequence. A candidate list of moves is created by first considering all possible swap moves. Swaps of activities that have the same start time in the current working schedule are removed from the candidate list. Also, swaps of activities (h, j) where h is positioned in front of j and j is scheduled to start before h in the current working schedule are removed from the candidate list. Only a subset of the candidate list is evaluated. Moves from the candidate list are randomly selected with equal probability. If the last move accepted was not a deteriorating move then $2n$ moves are evaluated; otherwise, n moves are evaluated. A hash value is calculated for each schedule. The tabu list is based on these hash values. A move that results in a schedule with a hash value that has been visited recently is tabu. The tabu tenure is reactive to the search state. When a solution is revisited the tabu tenure is increased. However, when the number of iterations since the solution was revisited is greater than the moving average of cycle lengths the tabu tenure is decreased. When three solutions have been visited twice, the tabu search is restarted with a different initial feasible solution and resetting the tabu tenure to one.

Franck, Neumann, and Schwindt (2001) present a tabu search heuristic for the RCPSP with both minimal and maximal time lags. A precedence feasible activity list is used as an indirect schedule representation, and schedules are generated by a serial SGS adapted to also handle maximal time lags. However, here the heuristic is described only in terms of how it is applied to the standard RCPSP (i.e. minimal time lags only). A preprocessing phase introduces additional temporal constraints to resolve resource conflicts due to two-element forbidden sets. Four different move operators are used in the heuristic. Each operates on an activity pair (i, j). The shift operator removes activity i and inserts it behind activity j. Any successors of i appearing in the list before j in the list are positioned immediately after i, preserving their relative order. The swap operator swaps the positions of i and j. Any successors of i appearing in the list before j in the list are positioned immediately after i, preserving their relative order. Similarly, any predecessors of j appearing in the list before i in the list are positioned immediately before j, preserving their relative order. The back-shift operator places the next non-successor of activity i immediately before i. The front-shift operator places the previous non-predecessor of activity j immediately after j. Three different tabu lists are utilized. The first tabu list is simply a list of the activity lists selected in the last several iterations. The second tabu list is for shift and swap moves, and the third tabu list is for front-shift and back-shift moves. Depending on the type of move chosen, each iteration the activity pairs (i, j) and (j, i) are added to the appropriate list and the oldest two entries are removed from that list. Each iteration, the oldest activity list is removed from the first tabu list and the current activity list is added. A candidate list of activity pairs (i, j) is constructed as follows. Activities j are selected that are currently scheduled later than the latest finish time of their predecessors in the current schedule. Activities i are determined for each activity j that have a finish time equal to the start time of j in

the current schedule. If i appears in the list before j then the shift or swap operator is applied; otherwise, the back-shift or front-shift operator is applied. After a certain number of non-improving iterations diversification is employed such that activities i are determined for each activity j that have a finish time less than or equal to the start time of j. If this extended neighborhood fails to improve the project duration the schedule is diversified further. This is accomplished by the generation of a certain number of activity lists that are created by randomly applying the shift or swap operators to the current activity list. The list that results in the best schedule is selected as the next working schedule. After a certain number of iterations or a certain number of schedules have been created an intensification phase is used. The tabu lists are erased, and three iterations of the tabu search are applied to the best schedule found. The tabu search algorithm terminates after the intensification phase.

Nonobe and Ibaraki (2002) develop a tabu search based heuristic for an extended multi-mode RCPSP. Specifically, they allow for time variant renewable resource availability, non-renewable resources, minimum and maximum time lags, and immediate precedence requirements (i.e. zero time lag between activities). In addition, soft constraints, constraints that can be violated with an objective function penalty, can be specified. Precedence constraints are considered to be hard constraints that cannot be violated. Renewable resource constraints can be specified as either soft or hard. Additional soft constraints may also be specified. A weighted penalty measure is associated with each soft constraint. A precedence feasible activity list is used as an indirect solution representation. This activity list is designated b π where activity $j = \pi(i)$ where i is the position in the list. A SGS referred to as CONSTRUCT, is specified which constructs a schedule that observes all hard constraints and immediate precedence constraints. In general, the CONSTRUCT algorithm does not necessarily create active schedules. However,

here the heuristic is described only in terms of how it is applied to the standard RCPSP (i.e. single-mode, only renewable resources with fixed quantities, constant activity resource requirements, and only minimal time lag constraints) with makespan minimization as the objective. In this case, the CONSTRUCT algorithm does not backtrack and creates active schedules. Three different moves are defined. The first move, change_mod, is only applicable to the multi-mode RCPSP. The move $shift\_aft(j_1, j_2)$, where activity j_1 appears earlier in the activity list than j_2 and j_1 is not a predecessor of j_2, activity j_1 and all of its successors appearing in the list before j_2 are positioned immediately after j_2, preserving their relative positions. The move $shift\_bef(j_1, j_2)$, where activity j_1 appears earlier in the activity list than j_2 and j_1 is not a predecessor of j_2, activity j_2 and all of its predecessors appearing in the list after j_1 are positioned immediately before j_1, preserving their relative positions. Only the move $shift\_bef(j_1, j_2)$ is utilized by the heuristic when solving a standard RCPSP. The full neighborhood defined by these moves is not explored; instead the neighborhood is reduced as follows. A graph $G(J, A(\pi))$ in constructed where J is the node set of all activities and $A(\pi)$ is an arc set based on the current activity list π. (Note that it appears that the activity list may be reordered after CONSTRUCT. The activity list used in CONSTRUCT is a listing of the activities in the order that CONSTRUCT should consider them. However, the bottom of page 568 and the top of page 570 might imply that the activity list is reordered after CONSTRUCT such that all activities are listed in order of non-decreasing start times.) An arc (j_1, j_2) is in $A(\pi)$ if j_1 is an obstacle to j_2 starting earlier either because j_1 is an immediate predecessor of j_2 (immediate in the normal sense that j_1 must be completed before j_2 begins) or because j_1 was using resources also needed by j_2 at a time j_2 could have otherwise been scheduled earlier. It seems that only either the former or the latter situation would apply to a given activity j_2; however, there could

be multiple activities j_1 that imposes the situation that applies. The maximal simple directed path from the artificial start activity to the artificial end activity in graph $G(J, A(\pi))$ is determined where any ties are broken arbitrarily. The move $shift\_bef(j_1, j_2)$ is a candidate move if (j_1, j_2) is an arc in the maximal path, j_1 appears earlier in the activity list than j_2, and j_1 is not a predecessor of j_2. However, the reduced neighborhood defined by these moves may still be considered too large and the neighborhood may be further reduced by randomly choosing a subset of these moves. It does not seem clear whether such random reductions were made when applying the heuristic to the standard RCPSP. For the tabu list, the attribute for the move $shift\_bef(j_1, j_2)$ is $(shift, j_2)$. The attribute for the move $shift\_aft(j_1, j_2)$, which is not a move that is actually used when solving a standard RCPSP, is $(shift, j_1)$. However, the heuristics prohibits all moves that possess an attribute in the tabu list. So it appears that if an activity x is serves as in the role of j_2 in $shift\_bef(j_1, j_2)$ in one iteration, then activity x is prevented from serving the in role of either j_1 or j_2 until the tabu tenure for $(shift, x)$ has expired. The tabu tenure is controlled adaptively using the method the authors introduced in Nonobe and Ibaraki (1998); although, some of the details of how the method applies to this heuristic may not be immediately clear. For instance, aspiration criteria is not specified in this heuristic, but the cited tabu tenure control method does specify aspiration criteria. The heuristic is allowed to run for 5,000 iterations; no early termination conditions are specified.

Gagnon, Boctor, and d'Avignon (2004) present a tabu search algorithm which uses a precedence feasible activity list indirect solution representation. Schedules are generated using a "priority list scheduling procedure where the priority of activity j corresponds to its position" in the precedence feasible activity list. It is not stated whether a serial or parallel SGS is used. An insert move is used where an activity is selected for insertion to a new position between the

maximum position of all the activity's predecessors, $L(x,j)$, and the minimum position of all of its successors, $H(x,j)$. Thus the precedence feasibility of the activity list is maintained. Only a subset of the neighborhood is evaluated; only $\sqrt{n}$ moves are evaluated. An activity in the working activity list is randomly selected for insertion to a random position in the interval, $(L(x,j),\ \mathrm{H}(x,j))$. Activities are shifted to the left or right depending on whether the activity selected for insertion is inserted to the right or left, respectively, of its current position. Moves that result in duplicate activity lists are discarded without replacement, resulting in neighborhood samples of variable sized. If the activity selected for insertion has the same starting time of adjacent activities to its new position, then the activity "is not selected." It is tabu to select an activity to generate a move if it has been selected for insertion as the best move in the last $\sqrt{n}$ iterations, resulting in a static tabu list of size $\sqrt{n}$. Since a move will not be generated based on a tabu activity, aspiration is not possible. The search is terminated if a schedule with a makespan equal to the critical path lower bound is found or a maximum number of activity lists are evaluated. Results for both a maximum of 1,000 and 5,000 are presented. The initial working solution is obtained using a serial SGS the minimum latest finish time priority rule. Various strategies to decrease the overall computation time are employed when evaluating moves.

4.3 Evolutionary Methods

Many studies have applied evolutionary, or population based, methods such as scatter search and genetic algorithms to the RCPSP. Some of the notable studies include Alcaraz and Maroto (2001); Alcaraz et al. (2004); Coelho and Tavares (2003); Debels et al. (2006); Debels and Vanhoucke (2007); Gonçalves et al. (2011); Hartmann (1998, 2002); Kochetov and Stolyar (2003); Leon and Balakrishnan (1995); Mendes et al. (2009); Tseng and Chen (2006); Valls, Ballestín, and Quintanilla (2004); Valls et al. (2005, 2008).

Hartmann (1998) presents several genetic algorithms for the RCPSP. The most competitive genetic algorithm presented is the permutation based, or activity list based, genetic algorithm with a two-point crossover operator. The two-point crossover operator used takes precedence relations into account in order to ensure precedence feasibility of the activity lists generated.

Valls et al. (2005) demonstrates how the activity list based genetic algorithm of Hartmann (1998) with the two-point crossover operator can be improved by simply adding double justification. Double justification is applied to an existing schedule by first right justifying the schedule by increasing the start time of each activity, in order of non-increasing finish, times to be as large as possible without increasing the project makespan. Then the right justified schedule is left justified by decreasing the start time of each activity, in order of non-decreasing start times, to be as early as possible. Double justification is applied to each solution in the initial population and to each offspring generated after possible mutation.

Valls et al. (2008) present a hybrid genetic algorithm (HGA) for the RCPSP. Their HGA uses a peak crossover operator to identify and combine good parts of solutions, which they identify by periods of relatively high resource utilization. A two-phase methodology is employed. The initial phase is a general search; however, in the second phase, a new population is generated by using biased random sampling of the neighborhood of the best solution found in the initial phase. As in Valls et al. (2005) double justification is applied to each solution in the initial population and to each offspring generated after possible mutation.

5 TABU SEARCH FOR THE RCPSP

5.1 Considerations for Applying Tabu Search to the RCPSP

The solution to a RCPSP, or any scheduling problem, requires a schedule. In many cases the schedule is considered to be the solution. However, using a schedule as the working solution of a neighborhood-based search procedure can be quite problematic. The largest issue is the size of the solution space, which will also be the search space. In general, the solution space for a general scheduling problem is infinite. This can be shown trivially by simply considering that parts of a schedule (for example the last job or activity) can be delayed indefinitely. By imposing a maximum makespan constraint (an upper bound, planning horizon, or due date) the solution space can be made finite, but still quite large. Further, defining a move can be problematic. Shall a move allow the schedule to become infeasible? If so how will the infeasibility be dealt with? If infeasible solutions are projected into the feasible solution space, or repaired to be feasible, then the repair mechanism might have more impact on the schedules obtained than the actual search procedure.

The majority of previous applications of tabu search to the RCPSP seem to utilize a precedence feasible activity list as an indirect solution representation. With indirect solution representations, the search space and solution space are not identical. The solution space is the set of schedules and the search space, in this case, is the set of precedence feasible activity lists, which is also the set of topological orderings of the project's precedence network. When using an activity list indirect solution representation, a choice must be made to either only allow simple moves and eliminate moves that result in infeasible lists or to use complex moves that modify the

resulting activity list to be precedence feasible, typically by shifting other activities to allow the move to be made while maintaining precedence feasibility. In the latter case a move can modify the current working activity list to a great extent. Furthermore, it is difficult to predict the affect of such a move on the activity list until after the move is actually made. It is also difficult to determine the reverse move.

Consider an activity list AL1, shown below, where activity i appears earlier in the list than activity j. Let a move be applied to AL1 that swaps the positions of i and j that requires the positions of other activities in the list to be shifted in order to maintain precedence feasibility. To be more precise, let some successors of i, labeled Si in AL1 below, appear in the list before the position of activity j and assume no predecessors of activity j appears between i and j. Then, in order for precedence feasibility to be maintained, the successors of i that appear before j in the list must be shifted as well. This move results in an activity list AL2. Note that due to the required shifting to maintain precedence feasibility, it is not possible for activity i to occupy the position previously occupied by activity j. Let the typical "reverse" move be applied that again swaps activities i and j. It would normally be expected that the immediate application of the reverse move to AL2 would result in AL1; however, this will not be the case in this example because swapping activities i and j in AL2 does not necessarily require the previously shifted successors of i to be shifted. Even if it is required to shift them, perhaps because successors of j must now be shifted, it is in no case reasonable to assume that these activities will be shifted to the positions they held in AL1, thus some other activity list AL3 is obtained. As evident from this example, defining what constitutes a move reversal, that prevents the same activity list from being revisited, is not straightforward.

Activity List AL1								
Activity	0	x	i	Si	Si	y	j	7
Index/Position	0	1	2	3	4	5	6	7

Activity List AL2 resulting from swapping positions of i and j in AL1								
Activity	0	x	j	y	i	Si	Si	7
Index/Position	0	1	2	3	4	5	6	7

Activity List AL3 resulting from swapping positions of i and j in AL2								
Activity	0	x	j	y	i	Si	Si	7
Index/Position	0	1	2	3	4	5	6	7

In the context of tabu search, even though a reverse move cannot be defined easily, it is possible to prohibit an activity from returning to its prior position by checking each position possibly affected by a move against a list of activities that cannot occupy the position. In the example above, positions 2 through 6 in AL2 would have to be checked for the initial swap move and positions 1 through 4 in AL3 would have to be checked in the subsequent swap move. Alternatively, instead of preventing an activity from returning to its prior position due to any move, an activity can instead be prohibited to be returned to its prior position by a move that involves the activity directly. In this case, only the two positions identified by the move would require a tabu check.

Consider a priority list indirect schedule representation for the RCPSP and the use of a serial SGS which will produce only active solutions. Further assume that all priorities assigned must be unique among the activities so that no two activities may be assigned the same priority. Without loss of generality, priority values may be required to be integer. Further, since the priority list is simply a ranking, then the possible priority values may be restricted to the set of $\{0,1,\dots,n,n+1\}$ where n is the number of activities, excluding artificial project start and end activities 0 and $n+1$, respectively. Further assume that the artificial start activities 0 and $n+1$ will have fixed priorities of 0 and $n+1$, respectively. This allows the priority list to be

represented by a list of the activity labels where the positions of the activities indicate their relative priorities. The solution space is the set of active schedules, but the search space is the set of all possible permutations of the set of integers $\{1, \dots, n\}$. There are, of course, $n!$ possible such permutations; however, the set of possible priority lists is finite, and the size of the search space is known. With such a priority list representation, moves can be defined that can be reversed in a straightforward manner. Further, all possible priority lists are feasible, so no move that maintains the nature of the priority list representation described will result in an infeasible schedule. Each priority list will generate exactly one schedule. The most significant drawback of using this representation is the fact that multiple priority lists may generate the same schedule; however, this drawback is also shared by the activity list representation.

The search space defined by the priority list representation described above is all possible permutations of the set of integers $\{1, \dots, n\}$. This particular search space is not unique and for example, shared by the single machine scheduling problem. Thus, it is reasonable to consider applications of tabu search to the single machine scheduling problem when developing a tabu search method for the RCPSP. In fact, when using an indirect solution representation, the problem of solving the RCPSP can be viewed as having certain similarities to solving a single machine scheduling problem. The SGS can be viewed as a "machine" or rather "processor" which processes each activity in a specified order or priority. Although, the objective function is defined somewhat differently.

5.2 Tabu Search Algorithms for the RCPSP

Let S be a schedule of start times for each activity. Let PL be a priority list of the activities. Let $SGS(PL)$ be some schedule generation scheme, either serial or parallel, for priority lists that generates a schedule S satisfying all temporal (e.g. precedence) and resource

constraints. Let $f(S)$ be an evaluation function defined as $f(S) \stackrel{\text{def}}{=} s_{n+1}$ where s_{n+1} is the start time of the artificial project end activity. Recall that the artificial project start and end activities have zero duration, so the start time of the artificial project end activity is the same as its finish time and thus the completion time of the project. Hence $\min f(S)$ is equivalent to $\min s_{n+1}$ which is the usual makespan minimization objective. Thus $\min s_{n+1} = \min f(S) = \min f\big(SGS(PL)\big)$. $f\big(SGS(PL)\big)$ may be used as the evaluation function for a tabu search. The problem that will be solved by the tabu search can be described as follows.

$$\text{minimize} \quad f\big(SGS(PL)\big) = \min f(S) = \min s_{n+1} \qquad (1)$$

$$\text{subject to}$$

$$\text{PL is a permutation of the integers } \{0,1,\dots,n,n+1\} \qquad (2)$$

Laguna, Barnes, and Glover (1991) present several tabu search strategies for a single machine scheduling problem. The problem solved can be described as follows.

$$\text{minimize} \quad F(\Pi) \qquad (3)$$

$$\text{subject to}$$

$$\Pi \text{ is a permutation of the integers } \{0,1,\dots,n,n+1\} \qquad (4)$$

In this problem $F(\Pi)$ is an evaluation function that determines the sum of the set-up costs and linear delay penalties. The permutation of integers Π is the schedule, or order, that N jobs will be processed on the single machine.

The only difference between these two problems is the evaluation function; however, this difference has an important implication. For the single machine scheduling problem Laguna et al. (1991) can calculate a move value for each possible swap or insert move based on the current schedule and store it in a matrix. Once calculated, these move values can be easily scanned to determine the best move. Further, only one swap move and two insert moves necessitate a recalculation of the entire move value matrix. All other moves only require a partial recalculation of the move value matrix, and the components that require an update are easily identified.

Unfortunately, a move value cannot be easily calculated for the RCPSP. However, since the search space is identical, and the same types of moves are available, then the same tabu attributes and restrictions can be employed.

Laguna et al. (1991) list possible attributes and tabu restrictions for a move that swaps the positions of two jobs. This swap move is the same as $PLSwapPos(i,j)$ described in section 2.2.1 with the condition $i < j$. Let $\pi(i)$ indicate the activity label of the activity in position i of a priority list. Table 5.1 lists the attributes and tabu restrictions described in Laguna et al. (1991). Restriction R0 has been added to represent no tabu restrictions. Laguna et al. (1991) also point out that restrictions similar to R3 through R6 can also be created for $\pi(j)$; these have been added as restrictions R8 through R11. In the case of R3 through R6, the intention is to prevent an activity that has been placed later in the list from returning to earlier positions in the list. In the case of R8 through R11, the intention is to prevent an activity that has been placed earlier in the list, and thus given a higher priority, from returning to later positions in the list. These same attributes can also be applied to the move $PLEjPosInsPos(i,j)$ described in section 2.2.1.

Table 5.1 – Tabu restrictions and attributes

	Attribute	Tabu restriction
R0	not applicable	No tabu restriction
R1	$(\pi(i), \pi(j), i, j)$	Prevent any move that will result in a priority list where activity $\pi(i)$ occupies position i and activity $\pi(j)$ occupies position j
R2	$(\pi(i), \pi(j), i, j)$	Prevent any move that will result in a priority list where activity $\pi(i)$ occupies position i or activity $\pi(j)$ occupies position j
R3	$(\pi(i), i)$	Prevent activity $\pi(i)$ from returning to position i
R4	$(\pi(i), i)$	Prevent activity $\pi(i)$ from moving to position k where $k \leq i$
R5	$(\pi(i), j)$	Prevent activity $\pi(i)$ from moving to position k where $k \leq j$, i.e. keep activity $\pi(i)$ from moving any earlier than its current position
R6	$\pi(i)$	Prevent activity $\pi(i)$ from moving at all
R7	$(\pi(i), \pi(j))$	Prevent activity $\pi(i)$ and activity $\pi(j)$ from moving at all.
R8	$(\pi(j), j)$	Prevent activity $\pi(j)$ from returning to position j
R9	$(\pi(j), j)$	Prevent activity $\pi(j)$ from moving to position k where $k \geq j$
R10	$(\pi(j), i)$	Prevent activity $\pi(j)$ from moving to position k where $k \geq i$
R11	$\pi(j)$	Prevent activity $\pi(j)$ from moving at all.

Klein (2000) employs a different tabu strategy. Instead of basing tabu status on move attributes, a hash value (Carlton & Barnes, 1996; Woodruff & Zemel, 1993) is computed for each schedule generated using the following hash function (Klein, 2000).

$$\varphi(S) = \left(\sum_{j=1}^{n} z_j \cdot S_j \right) mod \Phi \qquad (5)$$

S_j is the start time of activity j. z_j are pseudo-random integers drawn from $[0, Z]$. Φ is a large prime number. Moves are designated tabu if they generate a schedule with the same hash value as a previously visited schedule. Klein (2000) varies the associated tabu tenure in a reactive manner (Battiti & Tecchiolli, 1994). The initial tabu tenure is $v = 1$, but each time a solution is revisited the tabu tenure is recalculated as follows (Klein, 2000).

$$v = min\{max\{v + 1, \lfloor 1.2v \rfloor\}, 2n\} \qquad ir - lv > ma \quad (6)$$
$$v = max\{min\{v - 1, \lceil 0.8v \rceil\}, 1\} \qquad ir - lv \leq ma \quad (7)$$

In the formulas above, ir is the current iteration, lv is the last iteration the solution was visited, and ma is a moving average of cycle lengths calculated as follows (Klein, 2000).

$$ma = 0.9ma + 0.1(ir - lv) \qquad (8)$$

5.2.1 Tabu Search Algorithms for the RCPSP Evaluating All Possible Moves

The tabu search algorithms described in this section evaluate all possible moves. In order to avoid duplicate moves, the condition $i < j$ is imposed on move $PLSwapPos(i, j)$. Similarly, since move $PLEjPosInsPos(i, i + 1)$ and move $PLEjPosInsPos(i + 1, i)$ result in the same priority list, any move $PLEjPosInsPos(i, j)$ where $i - j = 1$ is omitted in order to avoid duplicate moves.

The different algorithms are designated as TS0 *move* R# where *move* is either PLSwap or PLEjIns indicating the moves $PLSwapPos(i, j)$ and $PLEjPosInsPos(i, j)$, respectively. R# represents a tabu restriction from Table 5.1. A priority list is used as an indirect schedule

representation, and a serial SGS is used to generate schedules. For an initial priority list, the activities are listed in order of increasing activity label.

In these tabu search algorithms short-term adaptive memory is composed of a tabu list created in accordance with the different tabu restrictions in Table 5.1 and a simple aspiration criterion that overrides tabu status if a move results in the best schedule found. The tabu strategy of Klein (2000) is not used in these algorithms.

Moves are evaluated by computing all possible values of j for each i. For example, the move $(i, j) = (1,10)$ will always be evaluated before the move $(12,18)$ since j is incremented before i. In the case of multiple neighborhood schedules with the best neighborhood makespan, the first move that results in a schedule with the best neighborhood makespan is chosen. This means that moves with a smaller value of i will be preferred over moves with a larger value of i.

Table 5.2 shows the number of schedules generated each iteration for every move type and problem size combination. Table 5.2 also shows the maximum possible number of iterations before reaching a schedule limit. Note that for the j60 and j120 instance sets the 1,000 schedules generated limit is reached before the first iteration is completed. For the j120 instance set, the 5,000 schedules generated limit is reached before the first iteration is completed. Thus, the results for j60 and j120 instance sets are likely extremely sensitive to the initial priority list.

For a particular move type, moves are evaluated in the same order and start from the same starting priority list. The same best neighborhood move will be selected at the end of the first iteration regardless of the tabu restriction employed because no moves are tabu in the first iteration. Thus, for a particular move type, the results will be the same across tabu restriction types for j60 instances at 1,000 schedules generated, for j120 instances at 1,000 schedules

generated, and j120 instances at 5,000 schedules generated since these limits are reached before the first iteration is completed.

Table 5.2 – Number of neighborhood moves and maximum iterations before reaching schedule limits

Move Type	Instance Set	# Neighborhood Moves or Schedules Generated Each Iteration	Maximum Iterations Before Schedule Limit				
			1,000	5,000	50,000	100,000	500,000
Swap	j30	435	2.30	11.49	114.94	229.89	1,149.43
	j60	1,770	0.56	2.82	28.25	56.50	282.49
	j120	7,140	0.14	0.70	7.00	14.01	70.03
Eject/Insert	j30	841	1.19	5.95	59.45	118.91	594.53
	j60	3,481	0.29	1.44	14.36	28.73	143.64
	j120	14,161	0.07	0.35	3.53	7.06	35.31

Considering that these tabu search algorithms for the RCPSP might be sensitive to the starting solution, the tabu search algorithms are also tested by starting them from a schedule obtained as follows. A serial SGS is used to generate the schedule for the priority list where all activities are listed in order of increasing activity label. The following priority rules are also used with a parallel SGS to obtain additional schedules: smallest activity label first, smallest processing time first, largest processing time first, latest finish time first, latest start time first, minimum slack time first. A priority list for the schedule with the smallest makespan was used as the initial priority list for the tabu search algorithms. The tabu search algorithms that employ this method to obtain an improved starting solution are designated as TS0 IS *move* R#.

5.2.2 Tabu Search Algorithms for the RCPSP Employing Candidate Lists of Moves

As shown in Table 5.2, a large number of schedules are generated in just a few iterations. Recognizing that in order to obtain competitive results fewer schedules must be generated in each iteration in order to allow an increased number of iterations to be performed, several

candidate list strategies are considered. A candidate list strategy specifies a subset of all possible moves to consider.

The first candidate list strategies considered are very simple random candidate lists. For both the $PLEjPosInsPos(i,j)$ and the $PLSwapPos(i,j)$ moves, i and j are chosen randomly. For the $PLSwapPos(i,j)$ move, i and j are swapped if $i > j$. Following the algorithm designations introduced in section 5.2.1, tabu search TS1-*X move* R# randomly constructs a candidate list of *X* unique moves each iteration. There is no restriction on allowing the same move appearing in the candidate list in consecutive iterations. This candidate list is tested with *X* set to 10 and 100. The tradeoff is that a very limited portion of the neighborhood space is searched while allowing a larger number of moves/iterations to be performed before reaching the limits on the number of schedules generated. By allowing more iterations to be performed, it is more likely that a larger portion of the solution space will be searched. This candidate list strategy is not expected to perform particularly well in a standalone tabu search; however, this strategy may be beneficial when included as an improvement method in the RAMP algorithms presented in later.

Rangaswamy et al. (1998) propose several different tabu search candidate list strategies for the RCPSP. One such strategy is the use of bi-level candidate lists. In terms of $PLEjIns(i,j)$ and $PLSwap(i,j)$ moves, a bi-level candidate list strategy picks a value for either i or j and then chooses a value for the other move attribute based on the value of the first. The specific bi-level candidate list implemented by Rangaswamy et al. (1998) is described in section 4.2.

Several bi-level candidate list strategies are considered here. Similar to Rangaswamy et al. (1998), the higher level candidate list is composed of delayed activities or delayed critical activities. A delayed activity is any activity that is started later than its CPM early start time in

the current working schedule. A delayed critical activity is a delayed activity that is on a critical path of the current working schedule.

The concept of a resource competitor is used in some of the candidate list strategies. Two activities are resource competitors if they both require some amount of the same resource and cannot be processed simultaneously due to a precedence or temporal relationship between the two activities.

Some of the candidate list strategies utilize the per time period resource utilization ratio (RUR) which is a measure of the proportion of resources used in each time period and can be calculated as follows (Valls et al., 2008).

$$RUR(t) = \frac{1}{K} \sum_{j \in A_t} \sum_{k=1}^{K} \frac{r_{j,k}}{R_k} \qquad (9)$$

K is the number of renewable resources. R_k is the capacity of resource k. $r_{j,k}$ is the amount of resource k required by activity j each time period. A_t is the set of activities active, or in process, at time t.

The following candidate list strategies all exclude moves that involve two activities that are scheduled to start at the same time in the current working schedule. Moves where activity i appears earlier in the current working priority list than activity j but activity i is scheduled to start later than activity j in the current working schedule are also excluded.

The tabu search algorithm TS2 PLEjIns R# uses the move $PLEjActInsPos(i, y)$. The higher level candidate list is composed of delayed critical activities. The corresponding lower level list for activity i in the higher level list is composed of the positions $x + 1$ to $z - 1$. The position x is the position of the direct predecessor of i that appear latest in the current working

priority list. Similarly, the position z is the position of the direct successor of i that appears earliest in the current working list.

The tabu search algorithm TS3 PLEjIns R# is the same as TS2 PLEjIns R# with the exception that the higher level candidate list is composed of all delayed activities instead of only delayed critical activities.

The tabu search algorithm TS4 PLEjIns R# uses the move $PLEjActInsAfterAct(j,i)$. The higher level candidate list is composed of delayed critical activities that are not active during a period of high resource utilization. The threshold for high resource utilization is set to $\max\{0.1,\ number\ of\ nonimproving\ iterations/10\}$. Activities that are in process at any time t where $RUR(t) \geq threshold$ are designated as active during a period of high resource utilization. As the number of non-improving iterations increases more activities will be included in the higher level candidate list. After 10 non-improving iterations, all delayed critical activities will be included in the higher level list. By adaptively modifying the threshold in this way, the recent history of the search is taken into account. The corresponding lower level list for activity i in the higher level list is composed of activities j that are resource competitors of i that appear earlier in the list than activity i. After 10 non-improving iterations the resource competitor requirement is relaxed until the next improving iteration. As in Rangaswamy et al. (1998), one additional move is added to the candidate list for each activity in the higher level list—the move that ejects the higher level list activity and inserts it immediately after its direct predecessor that appears latest in the list

The tabu search algorithm TS5 PLEjIns R# is the same as TS4 PLEjIns R# with the exception that the higher level candidate list is composed of delayed activities that are not active

during a period of high resource utilization (i.e. activities are not required to be on a critical path).

The tabu search algorithm TS6 PLEjIns R# is the same as TS5 PLEjIns R# with the exception that the higher level candidate list is composed of all delayed activities without any consideration of the RUR.

In these tabu search algorithms short-term adaptive memory is composed of a tabu list created in accordance with the different tabu restrictions in Table 5.1, a tabu list created in accordance with the tabu strategy of Klein (2000), and a simple aspiration criterion that overrides tabu status if a move results in the best schedule found. In addition some of the candidate list strategies (TS4 – TS6) are reactive to the recent history of the search..

5.3 Computational Analysis

All of the tabu search algorithms for the RCPSP were implemented using the C++ programming language. The PSPLIB RCPSP instances with $n = 30$, 60, and 120 activities were used to evaluate the performance of the tabu search algorithms. The C++ code was compiled for 64-bit machines and all tests were conducted on machines with Intel Core i7 870 2.93 GHz CPUs and 8GB RAM running a 64-bit operating system.

Unless specified otherwise, the following parameters and conditions apply. The tabu tenure was set to $n/4$. An aspiration criterion was defined whereby a tabu move would be accepted if it resulted in a schedule with a makespan smaller than the search had already encountered. The tabu search was terminated after $n/2$ non-improving iterations or after 500,000 schedules were generated.

For the j30 instances, the average percent deviation from the known optimal makespan is given as the primary result. For the j60 and j120 instances, the average percent deviation from the critical path lower bound is given as the primary result.

Table 5.3 shows the results of applying the tabu search algorithms without candidate lists to the PSPLIB j30, j60, and j120 RCPSP instance sets. Note that the tabu strategy of Klein (2000) is not used in these algorithms. The maximum number of schedules generated in any single j30 instance was 28,595 schedules. The maximum number of schedules generated in any single j60 instance was 257,595. The 50,000, 100,000, and 500,000 schedules generated columns for the j30 instances are identical for this reason. It may be interesting to note that, for the j30 instances, if the best makespan found is exactly one time unit greater than the known optimal makespan, then the average percent deviation would be 1.78%.

Table 5.3 – Tabu search algorithms without candidate lists

Tabu Search	Schedule Limits					Avg # S	Avg # S to Best
	1,000	5,000	50,000	100,000	500,000		
j30 Instances*							
TS0 PLSwap R7	1.69	1.15	1.10	1.10	1.10	5,135.81	602.89
TS0 PLEjIns R7	2.99	1.82	1.68	1.68	1.68	10,583.60	945.11
TS0 IS PLSwap R7	1.06	0.90	0.86	0.86	0.86	2,269.91	354.38
TS0 IS PLEjIns R6	1.52	1.16	1.07	1.07	1.07	4,471.31	563.69
j60 Instances							
TS0 PLSwap R7	16.98	14.63	13.73	13.66	13.66	49,007.9	5,238.08
TS0 PLEjIns R7	17.81	16.00	14.50	14.37	14.35	94,981.5	7,615.32
TS0 IS PLSwap R5	14.28	13.29	12.90	12.87	12.87	26,704.5	3,485.90
TS0 IS PLEjIns R7	14.70	13.90	13.32	13.21	13.21	53,563.6	5,725.31
j120 Instances							
TS0 PLSwap R5	50.06	48.31	41.71	41.49	40.78	422,296	85,986.0
TS0 PLEjIns R6	51.22	49.34	43.96	42.85	42.37	469,627	77,872.6
TS0 IS PLSwap R5	41.38	40.42	38.13	38.01	37.51	390,791	75,230.0
TS0 IS PLEjIns R7	41.69	40.92	39.00	38.76	38.40	441,099	71,500.1
* Average percent deviations from optimal makespan							

The results for the tabu search algorithms using swap moves clearly dominate the results for the tabu search algorithms using eject/insert moves. The tabu search algorithms using eject/insert moves generate almost twice as many schedules each iteration than the tabu search algorithms using swap moves; however, comparing the eject/insert move results at 100,000 schedules generated with the swap move results at 50,000 schedules generated, the swap move tabu search algorithms still dominate the eject/insert move tabu searches.

The most restrictive tabu restrictions, R6 and R7, produce the best results most often. Restriction R6 prevents activity $\pi(i)$ from moving at all. Restriction R7 prevents activity $\pi(i)$ and activity $\pi(j)$ from moving at all. As shown above, these restrictions restrict a greater percentage of the neighborhood than any of the other tabu restrictions. The first non-tabu move that provides a schedule with the neighborhood best makespan is chosen, whether it is an improving or non-improving move. A tabu move will only be chosen over a non-tabu move if the tabu move results in a schedule with a makespan that is strictly less than the smallest makespan found so far across all iterations performed. Since these restrictions designate more of the neighborhood search space as tabu, these restrictions may be promoting increased diversification in the working priority list by encouraging moves involving activities other than those involved in a recent move.

The tabu search algorithms without candidate lists started from an improved starting solution yielded better results. Results from this point forward in this section are from algorithms that incorporated the method to obtain an improved starting solution described in section 5.2.1.

Table 5.4 shows the results of applying the tabu search algorithms with candidate lists to the PSPLIB j30, j60, and j120 RCPSP instance sets. These tabu search algorithms were limited to generating only 50,000 schedules. In addition to using the tabu restrictions of Laguna et al.

(1991) the tabu search algorithms with candidate lists also use the tabu strategy of Klein (2000) with the parameter values $Z = 2^{15}$ and $\Phi = 99991$, which are the same values used by Klein (2000). Although nine different candidate list strategies and 12 different tabu restrictions were tested, the results presented below only include the best performing tabu restriction for each candidate list strategy.

The TS1-10 tabu search algorithms did not perform very well; however, the TS1-100 algorithms performed particularly well compared to the all of the other tabu search algorithms considered. Candidate lists TS5 and TS6 performed best for the j30 and j60 instances. The best performing tabu restrictions are most often R6, R7, and R11 which are the most restrictive restrictions. Note that although R0 does not impose any tabu restrictions based on move attributes, the hash function tabu strategy is applied.

Table 5.4 – Tabu search algorithms with candidate lists

Tabu Search Algorithm Used	Schedule Limits			Avg. # of Schedules	Avg. # of Schedules to Best
	1,000	5,000	50,000		
j30 Instances*					
TS1-10 PLEjIns R8	1.43	1.43	1.43	80.97	25.28
TS1-10 PLSwap R0	1.25	1.25	1.25	75.79	25.05
TS1-100 PLEjIns R3	0.78	0.65	0.65	540.78	196.87
TS1-100 PLSwap R11	0.74	0.57	0.57	476.32	191.86
TS2 PLEjIns R6	1.09	0.93	0.93	532.78	177.88
TS3 PLEjIns R6	1.21	0.95	0.95	871.79	289.88
TS4 PLEjIns R4	0.93	0.86	0.86	475.84	132.47
TS5 PLEjIns R7	0.93	0.53	0.52	1,125.54	391.95
TS6 PLEjIns R7	0.93	0.53	0.52	1,125.54	391.95
j60 Instances					
TS1-10 PLEjIns R1	14.04	14.04	14.04	164.38	40.45
TS1-10 PLSwap R2	13.90	13.90	13.90	157.63	44.89
TS1-100 PLEjIns R1	13.20	12.46	12.42	1,598.18	655.38
TS1-100 PLSwap R0	13.12	12.45	12.40	1,578.92	665.89
TS2 PLEjIns R6	14.10	13.22	12.42	6,937.19	3,007.22
TS3 PLEjIns R6	14.24	13.36	12.40	9,374.03	3,936.53
TS4 PLEjIns R11	13.46	13.00	12.59	8,905.15	2,368.43
TS5 PLEjIns R11	14.07	13.13	12.20	14,524.90	5,584.80
TS6 PLEjIns R11	14.07	13.13	12.20	14,524.90	5,584.80
j120 Instances					
TS1-10 PLEjIns R3	40.49	40.48	40.48	610.28	94.03
TS1-10 PLSwap R0	40.49	40.49	40.49	584.54	83.41
TS1-100 PLEjIns R0	39.26	37.44	36.76	7,992.59	3,455.83
TS1-100 PLSwap R1	39.23	37.57	36.94	7,664.93	3,210.71
TS2 PLEjIns R7	41.53	39.91	37.78	42,288.00	19,976.90
TS3 PLEjIns R7	41.62	40.11	38.07	43,040.70	18,934.40
TS4 PLEjIns R11	39.78	38.85	37.98	39,660.10	9,127.59
TS5 PLEjIns R7	41.24	40.00	37.95	40,593.00	16,380.40
TS6 PLEjIns R7	41.24	40.00	37.95	40,710.60	14,857.60
* Average percent deviations from optimal makespan					

Table 5.5 shows the results obtained by tabu search algorithms from the literature where results based upon the number of schedules generated are reported. Klein (2000) implements a candidate list; however, only a subset of the candidate list is randomly evaluated. Nonobe and

Ibaraki (2002) implement neighborhood reduction, the results of which can be viewed as a candidate list; however, the reduced neighborhood may still be considered too large and the neighborhood may be further reduced by randomly choosing a subset of these moves.

Table 5.5 – Tabu search results from external references

Reference	Schedule Limits		
	1,000	5,000	50,000
j30 Instances*			
Baar et al. (1998)	0.86	0.44	-
Klein (2000)	0.42	0.17	-
Nonobe and Ibaraki (2002)	0.46	0.16	0.05
j60 Instances			
Klein (2000)	12.77	12.03	-
Nonobe and Ibaraki (2002)	12.97	12.18	11.58
j120 Instances			
Nonobe and Ibaraki (2002)	40.86	37.88	35.85
* Average percent deviations from optimal makespan			

The results obtained from the tabu search algorithms of this section are not competitive with the current best performing methods for solving the RCPSP. Even among other implementations of tabu search these results are not competitive. The exceptions are TS1-100 and TS4 when limited to 1,000 schedules for the j120 instances and TS-100 when limited to the 5,000 schedules for the j120 instances. However, the aim here is not to necessarily develop a best performing stand-alone tabu search for the RCPSP but to develop an understanding of how the application of tabu search principles interact with the RCPSP in order to identify aspects that will lead to the development of an efficient tabu search procedure to be used as a component for RAMP and PD-RAMP algorithms for the RCPSP.

6 RAMP FOR THE RCPSP

The RAMP algorithms developed in this study combine mathematical relaxation, tabu search, and evolutionary solution methods. The mathematical relaxations are based on a 0-1 integer programming formulation for the RCPSP due to Pritsker et al. (1969) and Christofides et al. (1987). The tabu search components are those discussed in 5.2.2. The evolutionary method components are based on the hybrid genetic algorithm for the RCPSP of Valls et al. (2008).

6.1 Integer Programming Definition of the RCPSP

The RCPSP is a combinatorial optimization problem where the objective is to minimize the project completion time (makespan) subject to both temporal (precedence) and resource constraints. Precedence constraints are typically associated with some technological requirement and specify a fixed processing order between pairs of activities. Resource constraints model the resource demand of activities in a scheduling environment with scarce resource supply. Let $J = \{0, \dots, n, n+1\}$ be a set of activities with integral, non-negative processing times $p_j, j \in J$. Jobs 0 and $n+1$ are defined as artificial activities, with zero processing times, that indicate project start and project completion, respectively. In the RCPSP activities may not be interrupted while in progress (i.e. no preemption is allowed). A schedule $S = \{S_0, S_1, \dots S_{n+1}\}$ is an assignment of start times for all activities. Let $L \subseteq J \times J$ be the set of all given precedence constraints (i, j) where activity i is required to finish before activity j is allowed to start. Without loss of generality, it is assumed that the temporal constraints always refer to the start times of the jobs and that T is some given upper bound on the project makespan. Any time feasible schedule S must satisfy $S_j \geq S_i + p_i$ for all $(i, j) \in L$. In addition to temporal requirements, an activity j

requires an amount r_{jk} of one or several renewable resources $k \in R$, where R denotes the set of all renewable resources. A renewable resource k is available in the constant amount of R_k during each time period. While an activity is in process, the required resource units are exclusively assigned to it and are not available for other jobs.

The RCPSP can be formulated as a time-indexed integer linear program as follows (Pritsker et al., 1969):

$$\text{minimize} \quad \sum_t t x_{(n+1)t} \tag{10}$$

subject to

$$\sum_t x_{jt} = 1 \qquad j \in J \tag{11}$$

$$\sum_t t\left(x_{jt} - x_{it}\right) \geq p_i \qquad (i,j) \in L \tag{12}$$

$$\sum_j r_{jk}\left(\sum_{s=t-p_j+1}^{t} x_{js}\right) \leq R_k \qquad k \in R, t = 0, \ldots, T \tag{13}$$

$$x_{jt} \geq 0 \qquad j \in J, t = 0, \ldots, T \tag{14}$$

$$x_{jt} \text{ integer} \qquad j \in J, t = 0, \ldots, T \tag{15}$$

where $x_{jt} = 1$ if activity j starts at time t and $x_{jt} = 0$ otherwise. Constraints (11) ensure each job is started exactly once. Inequalities (12) represent the temporal constraints imposed by the precedence constraints L. Inequalities (13) represent the resource requirements of each job.

Christofides et al. (1987) propose temporal constraints (16) that together with constraints (11) imply the inequalities (12) even if the time-indexed variables are allowed to be fractional. The RCPSP can thus also be formulated as a time-indexed 0-1 integer linear program as follows (Christofides et al., 1987; Pritsker et al., 1969).

minimize $\sum_t t x_{(n+1)t}$ (10)

subject to

$$\sum_t x_{jt} = 1 \qquad j \in J \qquad (11)$$

$$\sum_{s=t}^{T} x_{is} + \sum_{s=0}^{t+p_i-1} x_{js} \leq 1 \qquad (i,j) \in L, t = 0, \dots, T \qquad (16)$$

$$\sum_j r_{jk} \left(\sum_{s=t-p_j+1}^{t} x_{js} \right) \leq R_k \qquad k \in R, t = 0, \dots, T \qquad (13)$$

$$x_{jt} \geq 0 \qquad j \in J, t = 0, \dots, T \qquad (14)$$

x_{jt} integer $\qquad j \in J, t = 0, \dots, T \qquad (15)$

Christofides et al. (1987) also propose strengthened resource constraints (17) to prevent the artificial project end activity from being scheduled before all of the other activities are completed.

minimize $\sum_t t x_{(n+1)t}$ (10)

subject to

$$\sum_t x_{jt} = 1 \qquad j \in J \qquad (11)$$

$$\sum_{s=t}^{T} x_{is} + \sum_{s=0}^{t+p_i-1} x_{js} \leq 1 \qquad (i,j) \in L,\ t = 0, \dots, T \qquad (16)$$

$$\sum_{j \neq n+1} r_{jk} \left(\sum_{s=t-p_j+1}^{t} x_{js} \right) \leq R_k \left(1 - \sum_{s=0}^{t} x_{(n+1)s} \right) \qquad k \in R, t = 0, \dots, T \qquad (17)$$

$$x_{jt} \geq 0 \qquad j \in J, t = 0, \dots, T \qquad (14)$$

x_{jt} integer $\qquad j \in J, t = 0, \dots, T \qquad (15)$

6.2 Evolutionary Methodology

The evolutionary methods used in the RAMP and PD-RAMP algorithms are based on the hybrid genetic algorithm (HGA) of Valls et al. (2008). Their HGA is composed of two phases. In the first phase they generate an initial population using the regret based biased random sampling

method (Drexl, 1991). In the second phase they generate a so-called neighbor's population by applying the β-biased random sampling method (Valls et al., 2008; Valls, Quintanilla, & Ballestín, 2003) to the activity list representation of the best schedule found during the first phase. The peak crossover operator (Valls et al., 2008) is used to combine the best solution currently in the population with another random member of the population in order to create two new combined solutions. A mutation operator swaps two consecutive activities in the activity list with probability p if there is not a precedence relationship between the two activities. Double justification (Valls et al., 2005) is applied to each schedule generated. The reader is referred to Valls et al. (2008) for further details.

In the evolutionary method the population is the primary adaptive memory structure. In the RAMP algorithms the initial population for the first evolutionary phase (EP1) is composed of solutions generated by earlier stages of the RAMP methodology. The details of generating the initial population and maintaining the population are given in section 6.3.5 where the various RAMP algorithms are described. The second evolutionary phase (EP2) in the RAMP algorithms creates a neighbor's population using the β-biased random sampling method for the activity list representation of the best schedule found so far by any component of the RAMP algorithm.

6.3 RAMP Model and Algorithms for the RCPSP – Lagrangian Relaxation

The RAMP algorithms developed in this section combine Lagrangian relaxation, tabu search, and evolutionary solution methods. The Lagrangian relaxation is based on a 0-1 integer programming formulation for the RCPSP due to Pritsker et al. (1969) and Christofides et al. (1987).

6.3.1 Adaptive Memory Relaxation Method

Christofides et al. (1987) dualizes the resource constraints (13) and introduces non-negative Lagrangian multipliers λ_{tk}, $t \in \{0, \dots, T\}$, $k \in R$ to obtain the following Lagrangian relaxation problem.

minimize

$$\sum_t t x_{(n+1)t} + \sum_t \sum_k \lambda_{tk} \left[\sum_j r_{jk} \left(\sum_{s=t-p_j+1}^{t} x_{js} \right) - R_k \right] \tag{18}$$

subject to

$$\sum_t x_{jt} = 1 \qquad j \in J \tag{11}$$

$$\sum_t t\left(x_{jt} - x_{it}\right) \geq p_i \qquad (i,j) \in L \tag{12}$$

$$x_{jt} \geq 0 \qquad j \in J, t = 0, \dots, T \tag{14}$$

$$x_{jt} \text{ integer} \qquad j \in J, t = 0, \dots, T \tag{15}$$

They solve this Lagrangian relaxation problem using a branch and bound procedure. It is notable that they do not incorporate the strengthened temporal constraints (16) or the strengthened resource constraints (17) they introduce. Christofides et al. (1987) conclude "the computational results that we have obtained seem to indicate that the Lagrangian relaxation is not a useful technique for this problem."

Möhring et al. (2003) includes the strengthened constraints (16) and (17). They dualize the resource constraints (17) and also introduce non-negative Lagrangian multipliers λ_{tk}, $t \in \{0, \dots, T\}$, $k \in R$ to obtain the following Lagrangian relaxation problem.

minimize $$\sum_t t x_{(n+1)t} + \sum_t \sum_k \lambda_{tk} \left[\sum_j r_{jk} \left(\sum_{s=t-p_j+1}^{t} x_{js} \right) - R_k \left(1 - \sum_{s=0}^{t} x_{(n+1)s} \right) \right] \quad (19)$$

subject to

$$\sum_t x_{jt} = 1 \qquad j \in J \quad (11)$$

$$\sum_{s=t}^{T} x_{is} + \sum_{s=0}^{t+p_i-1} x_{js} \leq 1 \qquad (i,j) \in L,\ t = 0, \ldots, T \quad (16)$$

$$x_{jt} \geq 0 \qquad j \in J,\ t = 0, \ldots, T \quad (14)$$

x_{jt} integer $\qquad j \in J,\ t = 0, \ldots, T \quad (15)$

Möhring et al. (2003) introduce the non-negative weights

$$w_{jt} = \begin{cases} \sum_{k \in R} r_{jk} \left(\sum_{s=t}^{t+p_j-1} \lambda_{sk} \right) & if\ j \neq n+1 \\ t + \sum_{s=t}^{T} \sum_{k \in R} \lambda_{sk} R_k & if\ j = n+1 \end{cases} \quad (20)$$

and the Lagrangian problem is rewritten as follows.

minimize $$\sum_j \sum_t w_{jt} x_{jt} - \sum_t \sum_{k \in R} \lambda_{tk} R_k \quad (21)$$

subject to

$$\sum_t x_{jt} = 1 \qquad j \in J \quad (11)$$

$$\sum_{s=t}^{T} x_{is} + \sum_{s=0}^{t+p_i-1} x_{js} \leq 1 \qquad (i,j) \in L,\ t = 0, \ldots, T \quad (16)$$

$$x_{jt} \geq 0 \qquad j \in J,\ t = 0, \ldots, T \quad (14)$$

x_{jt} integer $\qquad j \in J,\ t = 0, \ldots, T \quad (15)$

Christofides et al. (1987) and Möhring et al. (1999, 2003) point out that this is a project scheduling problem with start-time dependent costs if the constant term in the objective function is neglected.

For any non-negative vector of Lagrangian multipliers λ, the optimal solution value of the Lagrangian problem is a lower bound on the optimal value of the original RCPSP due to weak Lagrangian duality (Geoffrion, 1971). If the solution of the Lagrangian subproblem for a fixed vector of Lagrangian Multipliers λ is feasible for the RCPSP and if the corresponding objective function values of both problems coincide, the solution is also optimal for the RCPSP due to strong Lagrangian duality (Geoffrion, 1971).

6.3.2 Solving the Lagrangian Relaxation Integer Programming Problem

The Lagrangian relaxation problem can be solved by any method capable of solving an integer programming problem. Several commercial solvers are available. However, here the techniques of Möhring et al. (1999, 2003) are used in order to solve the Lagrangian relaxation problem.

Möhring et al. (1999, 2003) describe a method of solving the project scheduling problem with start-time dependent costs by transforming it into a minimum cut/maximum flow problem in a directed graph and solved using a maximum flow/minimum cut algorithm (Goldberg & Tarjan, 1988). The Möhring et al. (1999, 2003) transformation is obtained by constructing a flow network as follows. A node v_{jt} is created for each activity's possible start time and for the time following each activity's latest start time. In addition a source and sink nodes, a and b, respectively, are created for the flow network. Note that the source and sink nodes a and b do not represent and are not related to the artificial project start and project end activities. The latest start time is determined based upon a pre-determined maximum time horizon upper bound T.

Each time-indexed variable x_{jt} is represented by a directed assignment arc $(v_{jt}, v_{j,t+1})$ with a capacity equal to its coefficient in (10). Let $ES(i)$ represent the earliest feasible start time and let $LS(i) \leq T - p_i$ represent the latest feasible start time for activity i. The precedence constraints $(i,j) \in L$ are represented by directed temporal arcs $(v_{it}, v_{j,t+d_i})$ for all t satisfying both $ES(i) \leq t \leq LS(i)$ and $ES(j) \leq t + p_i \leq LS(j)$. In addition there are infinite capacity auxiliary arcs $(a, v_{i,ES(i)})$ and $(v_{i,LS(i)+1}, b)$, for each activity i, connecting the source node a and sink node b to the rest of the graph. The reader is referred to Möhring et al. (1999, 2003) for further details and proofs related to their transformation.

The resulting flow network is solved using Goldberg's maximum flow algorithm (Goldberg & Tarjan, 1988). A finite minimum cut will only contain assignment arcs, and a minimum cut that contains exactly one assignment arc for each activity is referred to as an n-cut (Möhring et al., 2003). The assignment arcs in an n-cut specify a single start time for each activity, thus a schedule is obtained. Although, this schedule will be feasible for the Lagrangian relaxation problem, it will not necessarily be feasible for the primal RCPSP.

6.3.3 Adaptive Weighting Update Method

Following Möhring et al. (2003) the subgradient method (Polyak, 1969) with a modified gradient step direction proposed by Camerini, Fratta, and Maffioli (1975) is used to generate the vector of Lagrangian multipliers. For a set of constraints (17) that have been relaxed, a gradient vector is computed as follows.

$$\mathrm{g}^{\mathrm{i}}_{\mathrm{tk}} = \sum_{j} r_{jk} \left(\sum_{s=t-p_j+1}^{t} x^{i}_{js} \right) - R_k \left(1 - \sum_{s=0}^{t} x^{i}_{(n+1)s} \right) \quad k \in R, t = 0, \ldots, T \quad (22)$$

The step size δ^i is given by

$$\delta^{\mathrm{i}} = \pi \left(UB^{\lambda} - LB^{\lambda}\right) \Big/ \left(\sum_{k}\sum_{t}\left(g_{\mathrm{tk}}^{\mathrm{i}}\right)^2\right) \qquad (23)$$

where UB^{λ} is the upper bound for the optimal value of the Lagrangian dual (which in the algorithm corresponds to the best primal feasible solution found), LB^{λ} is the current lower bound (associated with the solution of the previous Lagrangian relaxation problem), and π is a user-defined (step size) parameter initialized at a certain value (e.g. 2) and reduced when a certain number of successive iterations of the subgradient search does not improve the lower-bound. Hence, if at iteration i the vector of Lagrangian multipliers λ^i is used, the vector for the next iteration $(i+1)$ is determined as

$$\lambda_{tk}^{(i+1)} = \left[\lambda_{tk}^{i} + \delta^{i}\left(\mathrm{g}_{\mathrm{tk}}^{\mathrm{i}} + \beta \mathrm{g}_{\mathrm{tk}}^{\mathrm{i}-1}\right)\right]^{+} \quad k \in R, t = 0, \ldots, T \quad (24)$$

where $[.]^{+}$ denotes the non-negative part of the vector. The scalar β is computed as proposed in Camerini et al. (1975) to modify the gradient step direction.

6.3.4 Adaptive Memory Projection Method

Since solutions for a relaxation problem are usually not feasible for the corresponding original problem, a projection method is necessary to transform a relaxation dual solution into a primal feasible solution. In the RAMP method, the objective of projecting dual solutions onto the primal feasible space serves two main purposes: (1) seeking for an improved feasible solution for the original problem and (2) guiding the search process by providing new starting points for primal and dual searches. In general, a feasible solution obtained by a projection method is subjected to local search for possible improvement. On the other hand, the new projected and eventually improved solution may be chosen to replace the upper bound in the subgradient search, which is one component in the creation of the new relaxation problem.

Although the schedule obtained from the minimum cut problem will not necessarily be resource feasible, it will be precedence feasible. This precedence feasible schedule can be used to generate precedence feasible activity lists. Möhring et al. (2003) describe a method of generating several different precedence feasible activity lists from a precedence feasible schedule S using so-called α-completion times defined as $C_j(\alpha) \coloneqq S_j + \alpha p_j$ where $0 \leq \alpha \leq 1$. A serial SGS, a parallel SGS, and a serial SGS with limited look ahead are applied to each unique activity list. The serial SGS with limited look ahead is implemented as described by Möhring et al. (2003) where, given a look ahead parameter l, the activity with the earliest possible start time among the next l available activities is chosen to be scheduled at each decision step in the serial SGS.

6.3.5 The RAMP Algorithms

Several RAMP and PD-RAMP algorithm variations are described in this section. The basic RAMP algorithms combine the Lagrangian relaxation methodology described above with one of the tabu searches described in section 5.2.2. The tabu search is used to improve the best solution obtained from the adaptive memory projection method. After the tabu search is completed, the best makespan so far is used as UB^{λ} in the adaptive weighting update method.

PD-RAMP algorithms designated PD-RAMP–TS–EP1–EP2 incorporate the evolutionary methodology described in section 6.2. In these algorithms, the initial population for EP1of the evolutionary method is composed of all solutions generated in the adaptive memory projection method and all neighborhood best solutions from the tabu search. After EP1 completes the EP1 population is composed of the EP1POPsize best solutions. The population for EP2 of the evolutionary method is generated using the best solution found so far. After completion of EP2, the solutions in the EP1 population are replaced by the EP1POPsize best solutions from the current EP1 and EP2 populations. In subsequent iterations, the solutions generated in the

adaptive memory projection method and the neighborhood best solutions from the tabu search are added to the current EP1 population in order to introduce more diversity into the population.

PD-RAMP algorithms designated PD-RAMP–EP1–TS–EP2 apply the tabu search algorithm to the best solution found during EP1 of the evolutionary method. In these algorithms, the initial population for EP1 is composed of all solutions generated in the adaptive memory projection method. After EP1 completes the EP1 population is composed of the EP1POPsize best solutions. The tabu search algorithm is applied to the best solution in the EP1 population. The population for EP2 of the evolutionary method is generated using the best solution found so far. After completion of EP2, the solutions in the EP1 population are replaced by the EP1POPsize best solutions from the current EP1 and EP2 populations. In subsequent iterations, the solutions generated in the adaptive memory projection method are added to the current EP1 population in order to introduce more diversity into the population.

PD-RAMP algorithms designated PD-RAMP–EP1–EP2–TS apply the tabu search algorithm to the best solution found after EP2 of the evolutionary method. In these algorithms, the initial population for EP1of the evolutionary method is composed of all solutions generated in the adaptive memory projection method. After EP1 completes the population for EP2 of the evolutionary method is generated using the best solution found so far. After completion of EP2, the solutions in the EP1 population are replaced by the EP1POPsize best solutions from the current EP1 and EP2 populations. The tabu search algorithm is started from the best solution found so far. The neighborhood best solutions found by the tabu search are added to the current EP1 population. In subsequent iterations, the solutions generated in the adaptive memory projection method are also added to the current EP1 population.

6.4 RAMP Model and Algorithms for the RCPSP – Cross-Parametric Relaxations

In this section RAMP algorithms using cross-parametric relaxations (Rego, 2005) that combine Lagrangian and surrogate constraint relaxation are considered. As in the Lagrangian relaxation discussed in section 6.3, the surrogate and Lagrangian relaxations are based on a 0-1 integer programming formulation for the RCPSP due to Pritsker et al. (1969) and Christofides et al. (1987).

6.5 Adaptive Memory Relaxation Methods

Three obvious surrogate relaxations of the resource constraints (13) or (17) include creating a surrogate resource constraint to take the place of all resource constraints, creating a surrogate resource constraint for each resource type, and creating a surrogate resource constraint for each time period.

Relaxing all of the resource constraints (13) into a single surrogate resource constraint and introducing the non-negative surrogate weights $v = v_{tk}, t \in \{0, \dots, T\}, k \in R$ yields the following constraint.

$$\sum_{t} \sum_{k \in R} v_{tk} \left[\sum_{j} r_{jk} \left(\sum_{s=t-p_j+1}^{t} x_{js} \right) \right] \leq \sum_{t} \sum_{k \in R} v_{tk} R_k \qquad (25)$$

Creating a surrogate resource constraint for each resource type $k \in R$ yields the following set of constraints.

$$\sum_{t} v_{tk} \left[\sum_{j} r_{jk} \left(\sum_{s=t-p_j+1}^{t} x_{js} \right) \right] \leq \sum_{t} v_{tk} R_k \quad k \in R \quad (26)$$

In one respect, these relaxations transform the renewable resource type into non-renewable resources with a total availability equal to the right-hand-side of the constraints. Due to the interaction of the surrogate weights with the activity resource demand on the left-hand-side, the

resource demands can be viewed as time-dependent. In both surrogate relaxations (25) and (26) the right-hand-side term can be very large relative to the left-hand-side unless per-time-period resource utilization is consistently high compared to resource availability. Since the test problems in the PSPLIB are designed to encompass a wide variety of problems, such consistently high per-time-period resource utilization is not typical among all problem instances. Preliminary testing indicated that these relaxations are not particularly promising.

Creating a surrogate resource constraint for each time period yields the following set of constraints.

$$\sum_{k\in R} v_{tk}\left[\sum_{j} r_{jk}\left(\sum_{s=t-p_j+1}^{t} x_{js}\right)\right] \leq \sum_{k\in R} v_{tk}R_k \quad t \in \{0,\dots,T\} \quad (27)$$

This relaxation essentially transforms the different renewable resource types into a single type of renewable resource. Relaxing the stronger resource constraints (17), the surrogate problem is obtained as follows.

minimize $\sum_{t} t x_{(n+1)t}$ (10)

subject to

$\sum_{t} x_{jt} = 1$ $j \in J$ (11)

$\sum_{s=t}^{T} x_{is} + \sum_{s=0}^{t+p_i-1} x_{js} \leq 1$ $(i,j) \in L$, $t = 0,\dots,T$ (16)

$\sum_{k\in R} v_{tk}\left[\sum_{j\neq n+1} r_{jk}\left(\sum_{s=t-p_j+1}^{t} x_{js}\right) - R_k\left(1 - \sum_{s=0}^{t} x_{(n+1)s}\right)\right] \leq 0$ $t = 0,\dots,T$ (28)

$x_{jt} \geq 0$ $j \in J$, $t = 0,\dots,T$ (14)

x_{jt} integer $j \in J$, $t = 0,\dots,T$ (15)

For any vector of non-negative surrogate multipliers υ, the optimal solution value of the surrogate subproblem is a lower bound on the optimal solution value of the original RCPSP due to surrogate duality (Glover, 1975). If the optimal solution for a fixed vector of multipliers ω is feasible for the original RCPSP, the solution is also optimal for the original RCPSP due to surrogate duality (Glover, 1975).

The associated cross-parametric relaxation is obtained by dualizing the surrogate constraint (28) using a non-negative vector of Lagrangian multipliers λ_t. The cross-parametric relaxation is then obtained as follows.

minimize
$$\sum_t t x_{(n+1)t} + \sum_t \lambda_t \sum_{k \in R} \upsilon_{tk} \left[\sum_j r_{jk} \left(\sum_{s=t-p_j+1}^{t} x_{js} \right) - R_k \left(1 - \sum_{s=0}^{t} x_{(n+1)s} \right) \right] \tag{29}$$

subject to

$$\sum_t x_{jt} = 1 \qquad j \in J \tag{11}$$

$$\sum_{s=t}^{T} x_{is} + \sum_{s=0}^{t+p_i-1} x_{js} \leq 1 \qquad \begin{matrix} (i,j) \in L \\ t = 0, \ldots, T \end{matrix} \tag{16}$$

$$x_{jt} \geq 0 \qquad \begin{matrix} j \in J \\ t = 0, \ldots, T \end{matrix} \tag{14}$$

$$x_{jt} \text{ integer} \qquad \begin{matrix} j \in J \\ t = 0, \ldots, T \end{matrix} \tag{15}$$

Introducing the weights

$$w_{jt} = \begin{cases} \sum_{k \in R} r_{jk} \left(\sum_{s=t}^{t+p_j-1} \lambda_s \upsilon_{sk} \right) & if\ j \neq n+1 \\ t + \sum_{s=t}^{T} \lambda_s \sum_{k \in R} \upsilon_{sk} R_k & if\ j = n+1 \end{cases} \tag{30}$$

the Lagrangian subproblem can be written as follows.

minimize $$\sum_{j}\sum_{t} w_{jt}x_{jt} - \sum_{t}\lambda_t \sum_{k\in R} v_{tk}R_k \quad (31)$$

subject to

$$\sum_{t} x_{jt} = 1 \qquad j \in J \quad (11)$$

$$\sum_{s=t}^{T} x_{is} + \sum_{s=0}^{t+p_i-1} x_{js} \leq 1 \qquad (i,j) \in L,\ t = 0,\dots,T \quad (16)$$

$$x_{jt} \geq 0 \qquad j \in J, t = 0,\dots,T \quad (14)$$

x_{jt} integer $\qquad j \in J, t = 0,\dots,T \quad (15)$

For any non-negative vector of Lagrangian multipliers λ_t, the optimal solution value of the Lagrangian subproblem is a lower bound on the optimal value of the surrogate problem due to weak Lagrangian duality (Geoffrion, 1971). If the solution of the Lagrangian subproblem for a fixed λ is feasible for the surrogate problem and if the corresponding objective function values of both problems coincide, the solution is also optimal for the surrogate problem due to strong Lagrangian duality (Geoffrion, 1971).

6.5.1 Solving the Cross-Parametric Relaxation Integer Programming Problem

The cross-parametric relaxation integer programming problem can be solved in the same manner as the Lagrangian relaxation integer programming problem as discussed in section 6.3.2.

6.5.2 Adaptive Weighting Update Method

A fundamental feature of the cross-parametric relaxation is the ability to generate parametric subgradients, as defined in Glover (1975). In this implementation parametric subgradients are conceived by using the subgradient method (Polyak, 1969). The subgradient method is used to generate the vector of surrogate multipliers for the relaxed constraints of the primal RCPSP to create the corresponding surrogate problem. For a set of constraints (17) that have been relaxed, a gradient vector is computed as follows.

$$G_{kt} = \sum_{j} r_{jk} \left(\sum_{s=t-p_j+1}^{t} x_{js} \right) - R_k \left(1 - \sum_{s=0}^{t} x_{(n+1)s} \right) \quad k \in R, t = 0, \dots, T \quad (32)$$

The step size δ^{υ} is given by

$$\delta^{\upsilon} = \pi^{\upsilon} \left(UB^{\upsilon} - LB^{\upsilon} \right) \Big/ \left(\sum_{k} \sum_{t} G_{kt}^2 \right) \quad (33)$$

where UB^{υ} is the upper bound for the optimal value of the surrogate dual (which in the algorithm corresponds to the best primal feasible solution found), LB^{υ} is the current lower bound (associated with the solution of the previous surrogate relaxation problem), and π^{υ} is a user-defined (step size) parameter initialized at a certain value (e.g. 2) and reduced when a certain number of successive iterations of the subgradient search do not improve the lower-bound. Hence, if at iteration i, the vector of surrogate weights υ^i is used, the vector for the next iteration $(i+1)$ is determined as

$$\upsilon_{kt}^{(i+1)} = \left[\upsilon_{kt}^{i} + \delta^{\upsilon} G_{kt} \right]^{+} \quad k \in R, t = 0, \dots, T \quad (34)$$

where $[.]^{+}$ denotes the non-negative part of the vector. This new vector is then used to create the corresponding surrogate problem. Because the surrogate problem can be difficult to solve, the problem is relaxed again by dualizing the surrogate constraint using Lagrangian relaxation. As a result, a parametric subgradient is created by using the current surrogate vector as a parameter in a subgradient search carried out on the Lagrangian relaxation of the surrogate problem aimed at determining a surrogate dual solution. As for surrogate dual solutions, cross-parametric dual solutions are obtained by subgradient optimization. The subgradient is computed as follows.

$$G_t = \sum_{k \in R} \upsilon_{tk} \left[\sum_{j} r_{jk} \left(\sum_{s=t-p_j+1}^{t} x_{js} \right) - R_k \left(1 - \sum_{s=0}^{t} x_{(n+1)s} \right) \right] \quad t = 0, \dots, T \quad (35)$$

Similarly, a step size δ^{λ} is given by

$$\delta^{\lambda} = \pi^{\lambda}\left(UB^{\lambda} - LB^{\lambda}\right) \Big/ \sum_{t} G_t^2 \qquad (36)$$

where UB^{λ} is the upper bound for the optimal value of the surrogate problem (which corresponds to the best surrogate feasible solution found) and LB^{λ} is the lower bound given by an associated Lagrangian dual solution. As above, the computation of a Lagrangian multiplier (scalar) for a new subgradient iteration is given as follows.

$$\lambda_{\mathrm{t}}^{(i+1)} = \left[\lambda_{\mathrm{t}}^{i} + \delta^{\lambda} G_t\right]^{+} \qquad (37)$$

6.5.3 Adaptive Memory Projection Method

The adaptive memory projection method used here is the same as described in section 6.3.4.

6.5.4 The RAMP Algorithms

Aside from the relaxation method used, the RAMP algorithms for the cross-parametric relaxation are identical to the RAMP algorithms described in section 6.3.5.

6.6 Computational Analysis

The RAMP algorithms developed for the RCPSP were implemented using the C++ programming language. The code was compiled for 64-bit machines and all tests were conducted on machines with Intel Core i7 870 2.93 GHz CPUs and 8GB RAM running a 64-bit operating system. The PSPLIB RCPSP instances with $N = 30$, 60, and 120 activities were used to test the performance of the RAMP algorithms. An initial upper bound, or planning time horizon, for the RCPSP was obtained by taking the best makespan obtained after applying a parallel SGS using six different priority rules (e.g. smallest duration first, largest duration first, minimum slack, etc.). The RAMP algorithms were terminated early if a schedule equal to a known optimal value was found.

The adaptive memory projection method was used to generate up to 55 schedules. Up to eleven precedence feasible activity lists were generated using α-completion times where the parameter α was set from 0.0 to 1.0 in 0.1 increments. Duplicate activity lists were discarded. The serial SGS with limited look-ahead was used with parameter $l = 2$, 4, and 8. Thus, for each unique activity list, three schedules were generated with the serial SGS with limited look-ahead, one schedule with the parallel SGS, and one schedule with the usual serial SGS, which results in five schedules generated for each unique activity list. All schedules generated by the adaptive memory projection method were added to the EP1 population; duplicate schedules were not discarded.

The tabu search components in each RAMP algorithm were limited to generating no more than 5,000 schedules each time they were invoked. The other parameters and termination criteria for the tabu search components were set to the same values as used in section 5.3. The results presented in this section only include the best performing tabu restriction for each candidate list strategy.

The evolutionary components were limited to no more than 5,000 schedules generated for both phases, EP1 and EP2, combined with an approximately equal number of schedules allocated to each phase. Following Valls et al. (2008) the probability of mutation was set to 0.05 and the lower and upper thresholds for the peak crossover operator was set to 0.75 and 0.95, respectively. The values for the parameters EP1POPsize and π were also set as in Valls et al. (2008) and are summarized in Table 6.1. The population size for EP2 was set to EP1POPsize/2. Each generation $\pi \times POPsize/2$ couples are selected.

Table 6.1 – Parameter values for POPsize and π

	EP1POPsize	π
j30	100	0.8
j60	50	0.7
j120	50	0.4

6.6.1 Lagrangian RAMP Algorithms

Noting that the Lagrangian adaptive memory relaxation method, the adaptive weighting update method, and the adaptive memory projection method without any of tabu search or evolutionary method components implement the Möhring et al. (2003) Lagrangian-based heuristic, their results are shown below in Table 6.2 for comparison purposes. As usual, the average percent deviation from the critical path lower bound is given for the PSPLIB j60 and j120 instances. Results for the PSPLIB j30 instances are not available for the Möhring et al. (2003) Lagrangian-based heuristic.

Table 6.2 – Results from the Möhring et al.(2003) Lagrangian-based heuristic

	Avg. % Dev. from CPLB
j60	15.60
j120	36.00

The Lagrangian RAMP algorithms were terminated when the objective function values for the RCPSP and the Lagrangian relaxation problem were equal or when the lower bound for the RCPSP did not improve after 10 iterations.

Results for the basic RAMP algorithms are presented in Table 6.3. Comparing these results to the results presented in Table 5.4 for the tabu search algorithms with candidate lists (which are the tabu search algorithms incorporated in the RAMP algorithms), the RAMP algorithms obtain better results when limited to only 5,000 schedules generated than the tabu search algorithms alone when allowed a maximum of 50,000 schedules. When the RAMP algorithms and tabu search algorithms are allowed to generate up to 50,000 schedules, the

RAMP algorithms clearly dominate the tabu search algorithms. Comparing the tabu search algorithms and RAMP algorithms by candidate list strategy (i.e. row by row), the RAMP algorithm performs better than its tabu search counterpart in all cases.

For the j60 and j120 PSPLIB instances, even when limited to only 1,000 schedules generated, the RAMP algorithms clearly obtain better results than the Lagrangian-based heuristic of Möhring et al. (2003) alone. Together, these observations indicate that the performance of the RAMP algorithm cannot be attributed to either the tabu search component or the Lagrangian relaxation component alone. This suggests that the RAMP methodology is allowing useful information obtained from the dual to be exploited by the primal and vice versa. Comparing these RAMP results to other tabu search algorithms for the RCPSP (see Table 5.5), the RAMP algorithms obtain better results for the larger j120 instances. For the smaller j60 instances, only some of the RAMP algorithms perform as well or better than existing tabu search algorithms.

Table 6.3 – RAMP results

Tabu Search Algorithm Used	Schedule Limits			Avg. # of Schedules	Avg. # of Schedules to Best
	1,000	5,000	50,000		
j30 Instances*					
TS1-10 PLEjIns R3	0.60	0.32	0.09	4,034.07	1,967.28
TS1-10 PLSwap R0	0.57	0.29	0.09	4,132.83	2,097.56
TS1-100 PLEjIns R8	0.74	0.25	0.04	3,379.67	2,337.00
TS1-100 PLSwap R0	0.61	0.23	0.04	3,085.70	2,108.66
TS2 PLEjIns R6	0.70	0.32	0.05	4,022.62	2,968.26
TS3 PLEjIns R7	0.79	0.45	0.08	4,851.91	3,025.05
TS4 PLEjIns R11	0.77	0.34	0.09	4,301.42	2,473.79
TS5 PLEjIns R6	0.78	0.36	0.08	4,012.65	2,262.42
TS6 PLEjIns R6	0.78	0.36	0.08	4,012.65	2,262.42
j60 Instances					
TS1-10 PLEjIns R8	12.68	12.15	11.65	12,453.0	4,245.45
TS1-10 PLSwap R4	12.72	12.18	11.64	12,380.7	4,903.09
TS1-100 PLEjIns R2	12.67	12.12	11.39	12,473.6	5,388.06
TS1-100 PLSwap R3	12.66	12.11	11.38	12,136.8	5,382.90
TS2 PLEjIns R7	12.82	12.41	11.65	13,181.6	5,408.65
TS3 PLEjIns R7	12.90	12.51	11.72	13,293.6	5,534.26
TS4 PLEjIns R11	12.82	12.56	11.76	13,310.9	5,183.91
TS5 PLEjIns R4	12.92	12.56	11.72	13,135.0	5,780.26
TS6 PLEjIns R4	12.92	12.56	11.72	13,135.0	5,780.26
j120 Instances					
TS1-10 PLEjIns R7	37.06	36.08	34.90	33,801.1	11,213.4
TS1-10 PLSwap R3	37.07	36.07	34.95	33,747.3	10,956.3
TS1-100 PLEjIns R3	36.84	36.06	34.61	34,178.2	13,868.2
TS1-100 PLSwap R0	36.89	36.18	34.63	34,012.3	15,079.2
TS2 PLEjIns R0	37.27	36.77	35.27	34,748.5	14,059.0
TS3 PLEjIns R0	37.32	36.84	35.33	34,818.6	14,147.3
TS4 PLEjIns R11	37.19	36.86	35.28	34,628.2	13,934.9
TS5 PLEjIns R0	37.31	36.94	35.40	34,394.8	13,910.4
TS6 PLEjIns R0	37.31	36.94	35.40	34,394.8	13,910.4
* Average percent deviations from optimal makespan					

Results for the PD-RAMP–TS–EP1–EP2 algorithms are presented in Table 6.4. The PD-RAMP–TS–EP1–EP2 algorithms outperform the basic RAMP algorithms at 50,000 schedules generated, and these results are obtained with a lower average number of schedules generated. In

addition, for the j30 and j60 instances, the best schedules obtained by the PD-RAMP–TS–EP1–EP2 algorithms are found with fewer schedules generated.

Table 6.4 – PD-RAMP–TS–EP1–EP2 results

Tabu Search Algorithm Used	Schedule Limits			Avg. # of Schedules	Avg. # of Schedules to Best
	1,000	5,000	50,000		
j30 Instances*					
TS1-10 PLEjIns R7	0.55	0.13	0.02	2,099.17	1,459.67
TS1-10 PLSwap R3	0.52	0.17	0.02	2,318.61	1,330.15
TS1-100 PLEjIns R11	0.69	0.15	0.01	2,068.43	1,652.12
TS1-100 PLSwap R5	0.53	0.15	0.02	2,008.78	1,417.06
TS2 PLEjIns R2	0.76	0.26	0.02	2,499.04	1,886.00
TS3 PLEjIns R7	0.79	0.24	0.03	2,605.76	1,780.68
TS4 PLEjIns R3	0.74	0.18	0.02	2,144.61	1,567.94
TS5 PLEjIns R11	0.82	0.38	0.02	2,461.73	1,889.23
TS6 PLEjIns R1	0.84	0.47	0.03	2,898.54	2,243.8
j60 Instances					
TS1-10 PLEjIns R8	12.04	11.28	10.89	10,506.5	3,413.76
TS1-10 PLSwap R5	12.09	11.27	10.88	11,037.3	3,692.70
TS1-100 PLEjIns R5	12.72	11.91	10.91	11,458.1	4,180.09
TS1-100 PLSwap R7	12.63	11.72	10.91	11,878.7	4,897.00
TS2 PLEjIns R10	12.83	12.52	10.93	12,147.9	5,080.46
TS3 PLEjIns R7	12.90	12.51	10.93	12,114.6	5,338.54
TS4 PLEjIns R2	12.89	12.71	10.90	11,510.8	4,993.45
TS5 PLEjIns R0	12.94	12.62	10.95	12,172.6	4,866.10
TS6 PLEjIns R7	12.94	12.56	10.93	11,822.2	4,710.47
j120 Instances					
TS1-10 PLEjIns R0	35.97	33.15	31.96	31,474.5	10,946.5
TS1-10 PLSwap R5	36.05	33.09	31.91	31,779.7	11,128.9
TS1-100 PLEjIns R7	36.90	36.41	32.04	32,101.3	14,431.0
TS1-100 PLSwap R2	36.87	36.14	31.99	32,129.6	14,384.9
TS2 PLEjIns R0	37.27	36.77	31.95	32,144.0	14,139.7
TS3 PLEjIns R7	37.32	36.84	31.93	32,125.9	14,327.2
TS4 PLEjIns R8	37.20	36.92	31.98	32,114.0	14,336.9
TS5 PLEjIns R0	37.31	36.96	31.96	31,882.3	13,296.2
TS6 PLEjIns R0	37.31	36.96	31.94	32,222.8	14,517.0
* Average percent deviations from optimal makespan					

Results for the PD-RAMP–EP1–TS–EP2 algorithms are presented in Table 6.5. The PD-RAMP–EP1–TS–EP2 algorithms outperform the basic RAMP algorithms for the j60 and j120 instances at all schedule limits and for the j30 instances at 50,000 schedules generated, and these results are obtained with a lower average number of schedules generated. In addition, for the j30 and j60 instances, the best schedules obtained by the PD-RAMP–EP1–TS–EP2 algorithms are found with fewer schedules generated. When limited to 1,000 or 5,000 generated, the PD-RAMP–EP1–TS–EP2 algorithms typically provide better results than the PD-RAMP–TS–EP1–EP2 algorithms for the j60 and j120 instances. However, the PD-RAMP–TS–EP1–EP2 algorithms typically provide better results at 50,000 generated schedules.

Table 6.5 – PD-RAMP–EP1–TS–EP2 results

Tabu Search Algorithm Used	Schedule Limits			Avg. # of Schedules	Avg. # of Schedules to Best
	1,000	5,000	50,000		
j30 Instances*					
TS1-10 PLEjIns R5	0.86	0.14	0.02	2,207.30	1,575.95
TS1-10 PLSwap R11	0.82	0.16	0.02	2,315.57	1,660.55
TS1-100 PLEjIns R8	0.83	0.14	0.02	2,213.71	1,388.01
TS1-100 PLSwap R11	0.87	0.14	0.02	2,395.30	1,660.42
TS2 PLEjIns R2	0.85	0.18	0.03	2,260.05	1,436.37
TS3 PLEjIns R11	0.87	0.25	0.02	2,621.88	1,792.95
TS4 PLEjIns R8	0.86	0.19	0.02	2,574.61	1,847.82
TS5 PLEjIns R7	0.84	0.38	0.02	2,697.93	2,165.13
TS6 PLEjIns R8	0.81	0.43	0.03	2,789.74	2,093.47
j60 Instances					
TS1-10 PLEjIns R4	12.18	11.29	10.92	11,588.8	4,304.20
TS1-10 PLSwap R6	12.20	11.30	10.91	11,324.0	3,486.18
TS1-100 PLEjIns R3	12.16	11.64	10.91	11,544.1	4,705.05
TS1-100 PLSwap R11	12.20	11.63	10.89	11,467.8	4,765.18
TS2 PLEjIns R6	12.18	11.65	10.94	11,629.3	4,373.07
TS3 PLEjIns R9	12.19	11.67	10.98	11,999.4	5,207.65
TS4 PLEjIns R8	12.20	11.68	10.96	11,835.5	5,042.10
TS5 PLEjIns R5	12.51	11.71	10.95	11,573.2	5,328.39
TS6 PLEjIns R4	12.18	11.66	10.96	11,851.9	5,011.27
j120 Instances					
TS1-10 PLEjIns R5	35.01	33.10	31.91	32,817.0	12,097.8
TS1-10 PLSwap R8	35.06	33.06	31.85	32,402.1	11,688.2
TS1-100 PLEjIns R8	35.03	33.67	32.04	32,988.6	14,023.5
TS1-100 PLSwap R3	35.03	33.73	32.04	32,909.7	14,108.6
TS2 PLEjIns R8	35.01	33.76	32.06	33,256.2	14,445.5
TS3 PLEjIns R2	35.01	33.68	32.09	33,162.4	13,758.3
TS4 PLEjIns R7	35.00	33.72	32.10	33,036.3	13,685.5
TS5 PLEjIns R5	35.01	33.78	32.09	33,227.4	14,346.8
TS6 PLEjIns R8	35.05	33.77	32.04	33,002.3	14,522.1
* Average percent deviations from optimal makespan					

Results for the PD-RAMP–EP1–EP2–TS algorithms are presented Table 6.6. The PD-RAMP–EP1–EP2–TS algorithms outperform the basic RAMP algorithms at all schedules limits except for the j30 instances with only 1,000 generated schedules. The best schedules found by

the PD-RAMP–EP1–EP2–TS algorithms are obtained in a lower number of average schedules compared the basic RAMP algorithms. The results for the PD-RAMP–EP1–EP2–TS and PD-RAMP–EP1–TS–EP2 algorithms are comparable. Neither strategy seems to consistently obtain better results than the other. When limited to 1,000 or 5,000 generated, the PD-RAMP–EP1–EP2–TS algorithms typically provide better results than the PD-RAMP–TS–EP1–EP2 algorithms for the j60 and j120 instances. However, the PD-RAMP–TS–EP1–EP2 algorithms typically provide better results at 50,000 generated schedules.

Table 6.6 – PDRAMP–EP1–EP2–TS results

Tabu Search Algorithm Used	Schedule Limits			Avg. # of Schedules	Avg. # of Schedules to Best
	1,000	5,000	50,000		
j30 Instances*					
TS1-10 PLEjIns R7	0.81	0.16	0.02	2,353.66	1,837.32
TS1-10 PLSwap R5	0.85	0.14	0.03	2,180.36	1256.24
TS1-100 PLEjIns R4	0.84	0.15	0.02	2,224.48	1,643.19
TS1-100 PLSwap R1	0.83	0.12	0.02	1,872.49	1,149.91
TS2 PLEjIns R6	0.85	0.15	0.03	2,187.88	1,183.80
TS3 PLEjIns R5	0.85	0.15	0.04	2,605.04	1,454.42
TS4 PLEjIns R5	0.86	0.15	0.02	2,246.10	1,533.71
TS5 PLEjIns R8	0.85	0.14	0.03	2,251.19	1,356.61
TS6 PLEjIns R6	0.87	0.18	0.03	2,351.18	1,651.53
j60 Instances					
TS1-10 PLEjIns R9	12.18	11.36	10.91	11,837.0	4,088.35
TS1-10 PLSwap R5	12.18	11.33	10.91	11,653.4	3,688.04
TS1-100 PLEjIns R10	12.20	11.38	10.91	11,018.7	3,878.32
TS1-100 PLSwap R8	12.21	11.34	10.90	11,080.7	3,685.57
TS2 PLEjIns R11	12.19	11.37	10.96	11,777.6	4,347.80
TS3 PLEjIns R7	12.17	11.34	10.97	11,680.2	4,243.52
TS4 PLEjIns R8	12.18	11.38	10.93	12,028.9	4,394.68
TS5 PLEjIns R9	12.13	11.31	10.97	11,783.0	3,780.27
TS6 PLEjIns R4	12.17	11.26	10.96	11,140.7	4,060.47
j120 Instances					
TS1-10 PLEjIns R1	35.00	33.06	31.86	32,077.7	11,661.3
TS1-10 PLSwap R6	35.07	33.08	31.90	32,218.6	11,653.0
TS1-100 PLEjIns R1	35.03	32.97	32.03	32,358.3	11,788.1
TS1-100 PLSwap R8	34.99	33.03	32.02	32,704.7	13,119.9
TS2 PLEjIns R6	35.01	32.99	32.08	32,610.7	12,211.7
TS3 PLEjIns R5	35.01	33.09	32.08	32,461.7	12,463.0
TS4 PLEjIns R1	34.99	33.09	32.07	32,761.0	11,738.1
TS5 PLEjIns R2	35.01	33.06	32.07	32,737.4	13,054.9
TS6 PLEjIns R7	35.08	33.01	32.06	32,661.2	12,626.2
* Average percent deviations from optimal makespan					

None of the various RAMP and PD-RAMP algorithms perform better than the currently best performing heuristics for the RCPSP (see Table 3.1 through Table 3.3). However, the results for the PD-RAMP algorithms are competitive with the best performing heuristics for the larger

j60 and j120 PSPLIB instances. All of the RAMP and PD-RAMP algorithms outperform the only other relaxation based heuristic that is typically cited when comparing results of heuristics for the RCPSP. It is evident that the RAMP and PD-RAMP methodologies are capable of exploiting primal-dual relationships even in hard scheduling problems such as the RCPSP.

The maximum percent deviations obtained by the various RAMP algorithms are shown in Table 6.7 through Table 6.10. The optimal minimum makespan is known for all of the j30 instances, thus the maximum percent deviations are deviations from the optimal value. For the j60 and j120 instances the maximum percent deviation from the critical path lower bound is provided. The percent of instances where the optimal makespan was found by each algorithm is shown in Table 6.11 through Table 6.14. Note that the optimal value is known for 89.8 % of the j60 instances and 48.2% of the j120 instances.

Care must be taken when evaluating the maximum percent deviation from the critical path lower bound. The maximum deviation is not related to solution quality. For example, the maximum percentage deviation of the optimal value from the critical path lower bound among the j30 instances is 120.83%. These maximum deviations are only useful in comparing algorithms to each other. The RAMP algorithms that use tabu search to improve the best schedule obtained from the adaptive memory projection method result in smaller maximum deviations in the first 1,000 schedules. For 50,000 generated schedules, the PD-RAMP algorithms typically result in smaller maximum percent deviations than the basic RAMP algorithms. Similarly, the PD-RAMP algorithms typically find more optimal solutions than the basic RAMP algorithms.

Table 6.7 – RAMP maximum percent deviations

Tabu Search Algorithm Used	Schedule Limits		
	1,000	5,000	50,000
j30 Instances			
TS1-10 PLEjIns R3	8.62	5.17	3.45
TS1-10 PLSwap R0	6.90	5.17	3.45
TS1-100 PLEjIns R8	7.89	5.26	2.17
TS1-100 PLSwap R0	8.54	4.84	2.94
TS2 PLEjIns R6	8.06	6.90	3.45
TS3 PLEjIns R7	8.49	7.22	3.09
TS4 PLEjIns R11	10.34	5.26	2.99
TS5 PLEjIns R6	7.89	6.90	3.45
TS6 PLEjIns R6	7.89	6.90	3.45
j160 Instances			
TS1-10 PLEjIns R8	119.48	114.29	110.39
TS1-10 PLSwap R4	116.88	115.58	112.99
TS1-100 PLEjIns R2	115.58	114.29	109.09
TS1-100 PLSwap R3	119.48	110.39	107.79
TS2 PLEjIns R7	120.78	112.99	110.39
TS3 PLEjIns R7	120.78	114.29	109.09
TS4 PLEjIns R11	120.78	120.78	110.39
TS5 PLEjIns R4	119.48	114.29	111.69
TS6 PLEjIns R4	119.48	114.29	111.69
j120 Instances			
TS1-10 PLEjIns R7	222.22	217.17	212.12
TS1-10 PLSwap R3	225.25	218.18	215.51
TS1-100 PLEjIns R3	220.20	216.16	210.10
TS1-100 PLSwap R0	225.25	215.15	209.09
TS2 PLEjIns R0	224.24	220.20	211.11
TS3 PLEjIns R0	224.24	220.20	211.11
TS4 PLEjIns R11	223.23	220.20	213.13
TS5 PLEjIns R0	224.24	222.22	215.15
TS6 PLEjIns R0	224.24	222.22	215.15

Table 6.8 – PDRAMP–TS–EP1–EP2 maximum percent deviations

Tabu Search Algorithm Used	Schedule Limits		
	1,000	5,000	50,000
j30 Instances			
TS1-10 PLEjIns R7	10.34	5.17	2.06
TS1-10 PLSwap R3	9.21	5.26	2.06
TS1-100 PLEjIns R11	9.18	4.88	2.06
TS1-100 PLSwap R5	6.90	4.41	2.06
TS2 PLEjIns R2	9.72	6.90	2.06
TS3 PLEjIns R7	8.49	5.26	2.44
TS4 PLEjIns R3	8.47	4.88	2.94
TS5 PLEjIns R11	10.45	7.35	2.06
TS6 PLEjIns R1	10.45	8.49	2.94
j160 Instances			
TS1-10 PLEjIns R8	115.58	107.79	105.20
TS1-10 PLSwap R5	114.29	107.79	103.90
TS1-100 PLEjIns R5	116.88	112.99	105.20
TS1-100 PLSwap R7	116.88	112.99	106.49
TS2 PLEjIns R10	120.78	114.29	106.49
TS3 PLEjIns R7	120.78	114.29	103.90
TS4 PLEjIns R2	118.18	118.18	106.49
TS5 PLEjIns R0	119.48	115.58	105.20
TS6 PLEjIns R7	119.48	115.58	105.20
j120 Instances			
TS1-10 PLEjIns R0	217.17	206.06	201.01
TS1-10 PLSwap R5	219.19	206.06	201.01
TS1-100 PLEjIns R7	222.22	220.20	200.99
TS1-100 PLSwap R2	221.21	216.16	202.02
TS2 PLEjIns R0	224.24	220.20	202.02
TS3 PLEjIns R7	224.24	220.20	200.00
TS4 PLEjIns R8	223.23	216.16	200.00
TS5 PLEjIns R0	224.24	222.22	205.05
TS6 PLEjIns R0	224.24	222.22	198.99

Table 6.9 – PDRAMP–EP1–TS–EP2 maximum percent deviations

Tabu Search Algorithm Used	Schedule Limits		
	1,000	5,000	50,000
j30 Instances			
TS1-10 PLEjIns R5	13.16	6.58	2.44
TS1-10 PLSwap R11	13.16	5.26	2.06
TS1-100 PLEjIns R8	13.16	3.45	2.06
TS1-100 PLSwap R11	13.16	5.17	2.44
TS2 PLEjIns R2	13.16	3.90	2.44
TS3 PLEjIns R11	13.16	5.19	1.72
TS4 PLEjIns R8	13.16	5.17	2.35
TS5 PLEjIns R7	13.16	6.52	2.94
TS6 PLEjIns R8	13.16	6.90	3.64
j160 Instances			
TS1-10 PLEjIns R4	116.88	105.20	103.90
TS1-10 PLSwap R6	118.18	106.49	105.20
TS1-100 PLEjIns R3	112.99	111.69	102.60
TS1-100 PLSwap R11	116.88	112.99	103.90
TS2 PLEjIns R6	115.58	115.58	105.20
TS3 PLEjIns R9	118.18	115.58	106.49
TS4 PLEjIns R8	118.18	116.88	106.49
TS5 PLEjIns R5	115.58	115.58	103.90
TS6 PLEjIns R4	118.18	114.29	106.49
j120 Instances			
TS1-10 PLEjIns R5	216.16	207.07	202.02
TS1-10 PLSwap R8	211.88	205.94	200.00
TS1-100 PLEjIns R8	216.16	206.93	201.01
TS1-100 PLSwap R3	213.13	207.07	201.98
TS2 PLEjIns R8	218.18	209.90	200.99
TS3 PLEjIns R2	216.16	207.07	203.03
TS4 PLEjIns R7	213.13	207.07	203.03
TS5 PLEjIns R5	214.85	210.10	201.01
TS6 PLEjIns R8	214.14	209.09	201.01

Table 6.10 – PDRAMP–EP1–EP2–TS maximum percent deviations

Tabu Search Algorithm Used	Schedule Limits		
	1,000	5,000	50,000
j30 Instances			
TS1-10 PLEjIns R7	13.16	6.58	2.06
TS1-10 PLSwap R5	13.16	3.95	2.35
TS1-100 PLEjIns R4	13.16	3.45	2.06
TS1-100 PLSwap R1	13.16	5.26	2.06
TS2 PLEjIns R6	13.16	5.97	2.06
TS3 PLEjIns R5	13.16	6.58	2.94
TS4 PLEjIns R5	13.16	5.17	2.17
TS5 PLEjIns R8	13.16	5.26	3.45
TS6 PLEjIns R6	13.16	5.45	3.45
j160 Instances			
TS1-10 PLEjIns R9	116.88	109.09	107.79
TS1-10 PLSwap R5	118.18	110.39	107.79
TS1-100 PLEjIns R10	118.18	109.09	107.79
TS1-100 PLSwap R8	118.18	106.49	106.49
TS2 PLEjIns R11	118.18	107.79	105.20
TS3 PLEjIns R7	116.88	106.49	102.60
TS4 PLEjIns R8	116.88	106.49	106.49
TS5 PLEjIns R9	116.88	106.49	105.20
TS6 PLEjIns R4	116.88	110.39	106.49
j120 Instances			
TS1-10 PLEjIns R1	215.15	207.07	202.02
TS1-10 PLSwap R6	215.15	206.06	200.00
TS1-100 PLEjIns R1	215.15	203.96	198.99
TS1-100 PLSwap R8	217.17	204.04	201.01
TS2 PLEjIns R6	214.14	207.92	201.01
TS3 PLEjIns R5	213.13	207.92	199.01
TS4 PLEjIns R1	214.14	202.97	200.99
TS5 PLEjIns R2	217.17	206.06	199.01
TS6 PLEjIns R7	215.15	205.05	203.03

Table 6.11 – RAMP percent optimal

Tabu Search Algorithm Used	Schedule Limits		
	1,000	5,000	50,000
j30 Instances			
TS1-10 PLEjIns R3	79.8	86.9	94.8
TS1-10 PLSwap R0	80.4	87.1	94.8
TS1-100 PLEjIns R8	76.9	88.3	97.5
TS1-100 PLSwap R0	79.6	89.2	97.7
TS2 PLEjIns R6	78.1	86.9	96.9
TS3 PLEjIns R7	77.1	83.5	94.8
TS4 PLEjIns R11	76.0	86.5	94.8
TS5 PLEjIns R6	77.3	87.1	95.6
TS6 PLEjIns R6	77.3	87.1	95.6
j160 Instances			
TS1-10 PLEjIns R8	70.6	73.1	76.5
TS1-10 PLSwap R4	71.0	74.0	76.5
TS1-100 PLEjIns R2	69.6	72.1	77.3
TS1-100 PLSwap R3	70.0	73.1	77.5
TS2 PLEjIns R7	68.8	70.2	75.2
TS3 PLEjIns R7	68.8	69.8	75.2
TS4 PLEjIns R11	68.8	69.6	75.0
TS5 PLEjIns R4	69.0	69.6	76.0
TS6 PLEjIns R4	69.0	69.6	76.0
j120 Instances			
TS1-10 PLEjIns R7	28.3	30.7	33.5
TS1-10 PLSwap R3	28.8	31.0	33.8
TS1-100 PLEjIns R3	28.2	29.2	33.5
TS1-100 PLSwap R0	28.3	29.2	33.7
TS2 PLEjIns R0	27.7	28.2	32.5
TS3 PLEjIns R0	27.3	28.2	32.5
TS4 PLEjIns R11	28.0	28.2	32.5
TS5 PLEjIns R0	28.0	28.7	32.7
TS6 PLEjIns R0	28.0	28.7	32.7

Table 6.12 – PDRAMP–TS–EP1–EP2 percent optimal

Tabu Search Algorithm Used	Schedule Limits		
	1,000	5,000	50,000
j30 Instances			
TS1-10 PLEjIns R7	82.5	93.3	98.3
TS1-10 PLSwap R3	82.1	92.3	98.5
TS1-100 PLEjIns R11	78.8	93.1	99.0
TS1-100 PLSwap R5	80.8	91.7	98.8
TS2 PLEjIns R2	77.9	90.0	98.5
TS3 PLEjIns R7	76.9	90.6	98.1
TS4 PLEjIns R3	75.8	90.8	98.5
TS5 PLEjIns R11	77.1	86.5	98.5
TS6 PLEjIns R1	76.3	86.0	98.5
j160 Instances			
TS1-10 PLEjIns R8	72.5	75.8	81.0
TS1-10 PLSwap R5	72.3	75.4	80.0
TS1-100 PLEjIns R5	70.4	74.2	79.8
TS1-100 PLSwap R7	70.0	74.4	80.0
TS2 PLEjIns R10	68.8	70.2	79.8
TS3 PLEjIns R7	68.8	70.0	79.8
TS4 PLEjIns R2	69.4	70.2	80.6
TS5 PLEjIns R0	69.0	70.0	79.2
TS6 PLEjIns R7	69.0	69.8	79.8
j120 Instances			
TS1-10 PLEjIns R0	31.0	36.2	38.5
TS1-10 PLSwap R5	30.3	35.8	37.5
TS1-100 PLEjIns R7	28.3	28.5	38.0
TS1-100 PLSwap R2	28.5	29.5	37.3
TS2 PLEjIns R0	27.7	28.2	38.3
TS3 PLEjIns R7	27.3	28.2	38.0
TS4 PLEjIns R8	27.8	28.2	37.8
TS5 PLEjIns R0	28.0	28.8	38.5
TS6 PLEjIns R0	28.0	28.8	37.7

Table 6.13 – PDRAMP–EP1–TS–EP2 percent optimal

Tabu Search Algorithm Used	Schedule Limits		
	1,000	5,000	50,000
j30 Instances			
TS1-10 PLEjIns R5	76.3	92.9	98.5
TS1-10 PLSwap R11	77.1	92.1	98.5
TS1-100 PLEjIns R8	76.9	91.9	98.1
TS1-100 PLSwap R11	75.2	92.1	98.3
TS2 PLEjIns R2	76.5	91.7	98.1
TS3 PLEjIns R11	76.5	88.8	98.1
TS4 PLEjIns R8	75.6	91.0	98.3
TS5 PLEjIns R7	76.5	85.4	98.5
TS6 PLEjIns R8	76.5	84.8	98.3
j160 Instances			
TS1-10 PLEjIns R4	70.2	75.2	80.0
TS1-10 PLSwap R6	71.0	76.5	80.4
TS1-100 PLEjIns R3	70.8	73.8	80.0
TS1-100 PLSwap R11	70.8	74.4	79.8
TS2 PLEjIns R6	70.4	74.2	80.4
TS3 PLEjIns R9	70.8	73.5	79.8
TS4 PLEjIns R8	70.4	72.7	79.6
TS5 PLEjIns R5	71.0	74.2	80.4
TS6 PLEjIns R4	71.5	73.8	80.2
j120 Instances			
TS1-10 PLEjIns R5	31.7	36.2	38.0
TS1-10 PLSwap R8	31.3	35.0	38.0
TS1-100 PLEjIns R8	30.7	34.0	38.3
TS1-100 PLSwap R3	31.8	34.8	38.2
TS2 PLEjIns R8	31.8	33.8	37.7
TS3 PLEjIns R2	32.0	33.8	37.7
TS4 PLEjIns R7	31.2	33.0	37.8
TS5 PLEjIns R5	31.2	34.0	37.5
TS6 PLEjIns R8	31.5	34.2	38.0

Table 6.14 – PDRAMP–EP1–EP2–TS percent optimal

Tabu Search Algorithm Used	Schedule Limits		
	1,000	5,000	50,000
j30 Instances			
TS1-10 PLEjIns R7	76.5	92.1	98.5
TS1-10 PLSwap R5	76.7	92.3	98.1
TS1-100 PLEjIns R4	76.9	92.5	98.5
TS1-100 PLSwap R1	76.9	94.0	98.3
TS2 PLEjIns R6	76.5	92.3	97.9
TS3 PLEjIns R5	76.9	92.5	97.5
TS4 PLEjIns R5	76.3	92.1	98.3
TS5 PLEjIns R8	76.3	92.7	98.1
TS6 PLEjIns R6	76.0	81	98.5
j160 Instances			
TS1-10 PLEjIns R9	71.5	74.8	79.8
TS1-10 PLSwap R5	71.3	75.8	80.0
TS1-100 PLEjIns R10	70.8	76.5	79.4
TS1-100 PLSwap R8	71.3	75.4	80.6
TS2 PLEjIns R11	70.6	74.6	80.0
TS3 PLEjIns R7	70.6	75.6	80.0
TS4 PLEjIns R8	71.3	74.2	78.8
TS5 PLEjIns R9	70.8	75.4	79.6
TS6 PLEjIns R4	71.3	76.9	80.2
j120 Instances			
TS1-10 PLEjIns R1	31.7	35.7	38.7
TS1-10 PLSwap R6	31.5	35.7	38.5
TS1-100 PLEjIns R1	31.5	36.2	38.3
TS1-100 PLSwap R8	32.0	35.3	38.0
TS2 PLEjIns R6	31.7	35.2	37.8
TS3 PLEjIns R5	31.8	35.8	38.3
TS4 PLEjIns R1	31.3	35.2	37.8
TS5 PLEjIns R2	31.7	35.0	37.8
TS6 PLEjIns R7	31.3	35.3	37.5

6.6.2 Cross-Parametric RAMP Algorithms

In the cross-parametric RAMP algorithms the inner Lagrangian relaxation loop was terminated when the optimal objective function values for the surrogate and Lagrangian relaxations were equal or when the lower bound for the surrogate relaxation did not improve

after 10 iterations. The outer surrogate relaxation loop was terminated when the upper bound for the RCPSP did not improve after 10 iterations.

Results for the basic RAMP algorithms are presented in Table 6.15. Comparing these results to the results presented in Table 5.4 for the tabu search algorithms with candidate lists (which are the tabu search algorithms incorporated in the RAMP algorithms), the RAMP algorithms obtain better results in all but two cases (TS5 and TS6 for the j60 instances). For the j30 and j120 instances, the RAMP algorithms obtain better results when limited to only 5,000 schedules generated than the tabu search algorithms alone when allowed a maximum of 50,000 schedules.

For the j60 instances, the RAMP algorithms obtain better schedules within 1,000 generated schedules than the Lagrangian-based heuristic of Möhring et al. (2003). The results obtained from the Lagrangian basic RAMP shown in Table 6.3 clearly dominate the results from the cross-parametric RAMP. The Lagrangian basic RAMP generates about four times as many schedules as the cross-parametric basic RAMP.

Table 6.15 – RAMP results

Tabu Search Algorithm Used	Schedule Limits			Avg. # of Schedules	Avg. # of Schedules to Best
	1,000	5,000	50,000		
j30 Instances*					
TS1-10 PLEjIns R9	0.91	0.87	0.87	745.80	139.79
TS1-10 PLSwap R8	0.82	0.79	0.79	732.01	148.65
TS1-100 PLEjIns R10	0.77	0.43	0.41	1,079.00	398.79
TS1-100 PLSwap R1	0.71	0.40	0.36	1,054.45	424.62
TS2 PLEjIns R6	0.81	0.51	0.48	1,267.73	388.87
TS3 PLEjIns R6	0.90	0.57	0.49	1,568.64	512.46
TS4 PLEjIns R2	0.82	0.62	0.60	1,214.46	339.18
TS5 PLEjIns R6	0.87	0.51	0.40	1,806.99	739.37
TS6 PLEjIns R6	0.87	0.51	0.40	1,806.99	739.37
j60 Instances					
TS1-10 PLEjIns R1	13.10	12.93	12.93	1,032.84	211.06
TS1-10 PLSwap R2	13.11	12.86	12.86	1,027.97	239.10
TS1-100 PLEjIns R5	12.93	12.24	12.03	2,922.78	1,088.91
TS1-100 PLSwap R8	12.86	12.18	11.98	2,844.61	1,016.66
TS2 PLEjIns R6	13.15	12.60	12.34	3,464.30	1,153.56
TS3 PLEjIns R6	13.22	12.66	12.37	3,520.07	1,259.71
TS4 PLEjIns R11	13.13	12.71	12.42	3,520.13	1,173.23
TS5 PLEjIns R7	13.23	12.68	12.40	3,478.57	1,185.16
TS6 PLEjIns R7	13.23	12.68	12.40	3,478.57	1,185.16
j120 Instances					
TS1-10 PLEjIns R0	37.67	37.33	37.33	2,602.22	390.38
TS1-10 PLSwap R3	37.70	37.36	37.36	2,582.10	376.25
TS1-100 PLEjIns R4	37.29	36.32	35.85	8,047.97	2,573.74
TS1-100 PLSwap R0	37.31	36.36	35.92	8,025.86	2,548.65
TS2 PLEjIns R2	37.75	37.18	36.74	8,102.65	2,202.93
TS3 PLEjIns R6	37.78	37.24	36.80	8,122.11	2,290.15
TS4 PLEjIns R7	37.60	37.07	36.65	8,103.18	2,105.03
TS5 PLEjIns R7	37.73	37.32	36.89	8,130.62	2,074.24
TS6 PLEjIns R7	37.73	37.32	36.89	8,130.62	2,074.24
* Average percent deviations from optimal makespan					

Results for the PD-RAMP–TS–EP1–EP2 algorithms are presented in Table 6.16. The PD-RAMP–TS–EP1–EP2 algorithms outperform the basic RAMP algorithms at 50,000 schedules generated. Compared to the Lagrangian PD-RAMP–TS–EP1–EP2 algorithms, the

cross-parametric PD-RAMP–TS–EP1–EP2 algorithms generally generate more schedules yet obtain slightly inferior results.

Table 6.16 – PDRAMP–TS–EP1–EP2 results

Tabu Search Algorithm Used	Schedule Limits			Avg. # of Schedules	Avg. # of Schedules to Best
	1,000	5,000	50,000		
j30 Instances*					
TS1-10 PLEjIns R2	0.50	0.18	0.04	2,636.62	1,541.55
TS1-10 PLSwap R6	0.59	0.14	0.03	2,426.94	1,504.44
TS1-100 PLEjIns R1	0.80	0.18	0.03	2,580.03	1,595.20
TS1-100 PLSwap R7	0.67	0.15	0.02	2,344.96	1,619.74
TS2 PLEjIns R9	0.84	027	0.04	3,186.57	2,359.40
TS3 PLEjIns R4	0.92	0.33	0.04	3,397.92	2,413.70
TS4 PLEjIns R10	0.86	0.18	0.03	2,776.84	1,914.05
TS5 PLEjIns R5	0.89	0.49	0.02	3,069.12	2,401.14
TS6 PLEjIns R7	0.87	0.37	0.03	2,974.05	2,075.76
j60 Instances					
TS1-10 PLEjIns R8	12.28	11.30	10.93	11,852.3	3,933.99
TS1-10 PLSwap R10	12.23	11.27	10.90	11,799.3	4,606.57
TS1-100 PLEjIns R5	12.94	12.02	10.92	11,789.4	4,817.37
TS1-100 PLSwap R6	12.90	11.83	10.90	11,738.1	4,839.58
TS2 PLEjIns R7	13.15	12.54	10.92	12,233.9	5,219.41
TS3 PLEjIns R9	13.22	12.77	10.97	12,850.0	4,967.51
TS4 PLEjIns R4	13.14	12.73	10.88	11,965.4	4,953.26
TS5 PLEjIns R6	13.23	12.70	10.93	11,923.0	5,134.70
TS6 PLEjIns R7	13.23	12.68	10.93	11,990.1	5,582.63
j120 Instances					
TS1-10 PLEjIns R2	36.38	33.17	31.95	33,010.5	11,376.8
TS1-10 PLSwap R2	36.41	33.03	31.90	32,922.6	11,094.2
TS1-100 PLEjIns R6	37.31	36.55	32.11	32,984.4	14,849.6
TS1-100 PLSwap R7	37.27	36.73	32.05	32,780.7	14,648.7
TS2 PLEjIns R10	37.75	37.18	32.05	33,618.2	15,096.9
TS3 PLEjIns R7	37.78	37.24	32.05	33,576.2	15,062.2
TS4 PLEjIns R1	37.61	37.21	32.07	33,138.7	13,764.9
TS5 PLEjIns R6	37.73	37.32	32.03	33,291.3	14,703.7
TS6 PLEjIns R9	37.73	37.32	31.98	33,161.9	15,079.4
* Average percent deviations from optimal makespan					

Results for the PD-RAMP–EP1–TS–EP2 algorithms are presented in Table 6.17. The PD-RAMP–EP1–TS–EP2 algorithms outperform the basic RAMP algorithms in all cases except the j30 instances at only 1,000 schedules generated. For the j60 and j120 instances at 1,000 and 5,000 schedules generated, the PD-RAMP–EP1–TS–EP2 algorithms perform better than the PD-RAMP–TS–EP1–EP2 algorithms.

Table 6.17 – PDRAMP–EP1–TS–EP2 results

Tabu Search Algorithm Used	Schedule Limits			Avg. # of Schedules	Avg. # of Schedules to Best
	1,000	5,000	50,000		
j30 Instances*					
TS1-10 PLEjIns R2	0.87	0.19	0.03	2,723.73	1,732.36
TS1-10 PLSwap R11	0.87	0.19	0.04	2,576.20	1,871.14
TS1-100 PLEjIns R9	0.85	0.19	0.02	2,516.38	1,799.18
TS1-100 PLSwap R6	0.87	0.13	0.02	2,093.46	1,386.87
TS2 PLEjIns R0	0.83	0.25	0.04	3,233.09	2,457.85
TS3 PLEjIns R9	0.90	0.32	0.04	3,089.23	1,894.38
TS4 PLEjIns R8	0.91	0.22	0.02	2,998.86	2,529.50
TS5 PLEjIns R5	0.88	0.41	0.04	3,098.05	2,246.86
TS6 PLEjIns R7	0.87	0.35	0.04	3,282.11	2,454.73
j60 Instances					
TS1-10 PLEjIns R9	12.35	11.34	10.92	12,230.7	4,043.11
TS1-10 PLSwap R1	12.34	11.36	10.91	12,124.9	3,937.48
TS1-100 PLEjIns R10	12.31	11.57	10.89	10,930.4	4,794.96
TS1-100 PLSwap R1	12.35	11.53	10.91	11,459.5	4,684.08
TS2 PLEjIns R10	12.39	11.68	10.96	12,781.7	5,786.40
TS3 PLEjIns R4	12.39	11.73	10.97	13,009.6	4,805.20
TS4 PLEjIns R11	12.39	11.69	10.94	12,355.1	4,788.77
TS5 PLEjIns R1	12.33	11.65	10.95	12,236.6	5,629.56
TS6 PLEjIns R2	12.33	11.65	10.96	12,449.0	5,021.37
j120 Instances					
TS1-10 PLEjIns R8	35.28	33.12	31.97	33,357.2	11,375.5
TS1-10 PLSwap R0	35.24	33.06	31.95	33,410.5	12,126.7
TS1-100 PLEjIns R4	35.28	33.69	32.10	33,871.9	14,626.6
TS1-100 PLSwap R9	35.24	33.74	32.10	34,031.1	14,296.5
TS2 PLEjIns R5	35.22	33.78	32.14	34,452.0	13,765.8
TS3 PLEjIns R11	35.26	33.80	32.14	34,319.7	13,763.9
TS4 PLEjIns R9	35.25	33.85	32.14	34,159.1	14,458.1
TS5 PLEjIns R7	35.27	33.82	32.13	34,137.0	13,280.2
TS6 PLEjIns R5	35.28	33.86	32.11	34,017.5	14,464.5
* Average percent deviations from optimal makespan					

Results for the PD-RAMP–EP1–EP2–TS algorithms are presented in Table 6.18. The PD-RAMP–EP1–EP2–TS algorithms outperform the basic RAMP algorithms in all cases except the j30 instances at only 1,000 schedules generated. For the j60 and j120 instances at 1,000 and

5,000 schedules generated, the PD-RAMP–EP1–EP2–TS algorithms perform better than the PD-RAMP–TS–EP1–EP2 algorithms.

Table 6.18 – PDRAMP–EP1–EP2–TS results

Tabu Search Algorithm Used	Schedule Limits			Avg. # of Schedules	Avg. # of Schedules to Best
	1,000	5,000	50,000		
j30 Instances*					
TS1-10 PLEjIns R11	0.86	0.19	0.03	2,693.71	1,760.46
TS1-10 PLSwap R1	0.87	0.20	0.04	2,840.07	1,789.81
TS1-100 PLEjIns R0	0.85	0.19	0.03	2,694.36	1,944.63
TS1-100 PLSwap R8	0.88	0.16	0.03	2,476.75	1,670.65
TS2 PLEjIns R11	0.88	0.16	0.03	2,389.80	1,527.98
TS3 PLEjIns R6	0.89	0.22	0.04	2,848.80	1,862.60
TS4 PLEjIns R0	0.87	0.15	0.03	2,497.34	1,591.62
TS5 PLEjIns R11	0.89	0.18	0.04	2,705.60	1632.90
TS6 PLEjIns R4	0.85	0.22	0.04	3,075.04	2,048.93
j60 Instances					
TS1-10 PLEjIns R10	12.29	11.36	10.90	11,832.6	4,075.98
TS1-10 PLSwap R10	12.31	11.38	10.89	11,693.2	4,255.53
TS1-100 PLEjIns R1	12.34	11.33	10.89	11,495.1	4,271.89
TS1-100 PLSwap R8	12.35	11.34	10.90	11,382.7	3,822.13
TS2 PLEjIns R6	12.31	11.31	10.94	12,126.8	4,034.06
TS3 PLEjIns R7	12.35	11.35	10.98	12,359.6	4,458.78
TS4 PLEjIns R3	12.28	11.39	10.98	12,365.9	4,484.31
TS5 PLEjIns R1	12.35	11.35	10.94	11,952.4	4,328.46
TS6 PLEjIns R3	12.32	11.38	10.95	12,323.3	4,591.55
j120 Instances					
TS1-10 PLEjIns R2	35.21	33.14	31.93	33,210.4	11,120.8
TS1-10 PLSwap R2	35.28	33.05	31.92	33,075.3	10,663.6
TS1-100 PLEjIns R4	35.25	33.14	32.03	33,452.0	12,896.3
TS1-100 PLSwap R3	35.22	33.07	32.09	33,421.1	11,994.7
TS2 PLEjIns R2	35.22	33.11	32.16	33,869.2	12,564.1
TS3 PLEjIns R3	35.30	33.09	32.11	33,695.9	13,303.2
TS4 PLEjIns R8	35.26	33.07	32.06	33,601.8	12,966.5
TS5 PLEjIns R3	35.22	33.07	32.12	33,565.9	12,432.3
TS6 PLEjIns R7	35.20	33.03	32.07	33,735.2	12,314.8
* Average percent deviations from optimal makespan					

The Lagrangian RAMP algorithms generally outperform the cross-parametric algorithms in almost all cases. In the few cases where the cross-parametric RAMP algorithms perform better at 50,000 schedules generated, they do so by no more than 0.03%.

7 CONCLUSIONS

The resource constrained project scheduling problem (RCPSP) is one of the most intractable problems in combinatorial optimization; it is NP-hard in the strong sense. In this work new algorithms that make effective use of the RAMP and Primal-Dual RAMP methodologies introduced by Rego (2005) were developed for the RCPSP.

All of the tabu search and RAMP algorithms were tested using several different tabu restriction strategies. Figure 7.1 shows the frequency each tabu restriction yielded the best results for an algorithm. Restriction R7 provided the best results for 17% of the algorithms. Restriction R7 is the most restrictive among the restrictions considered (see Table 5.1). Even though restriction R7 yielded the best results most often it does not do so often enough to disregard the other tabu restrictions in favor of R7.

Figure 7.1 – Frequency tabu restrictions yielded best results - all algorithms

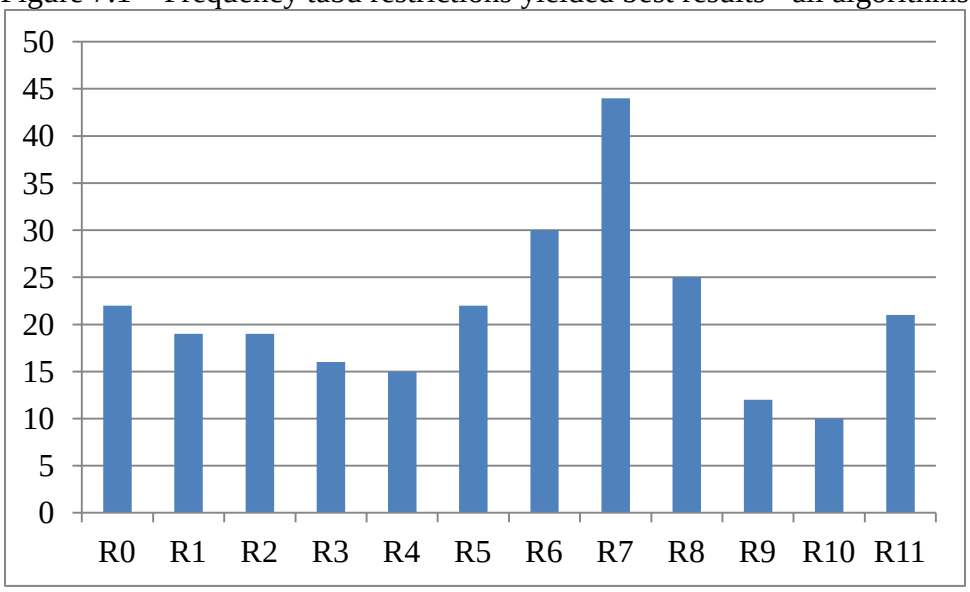
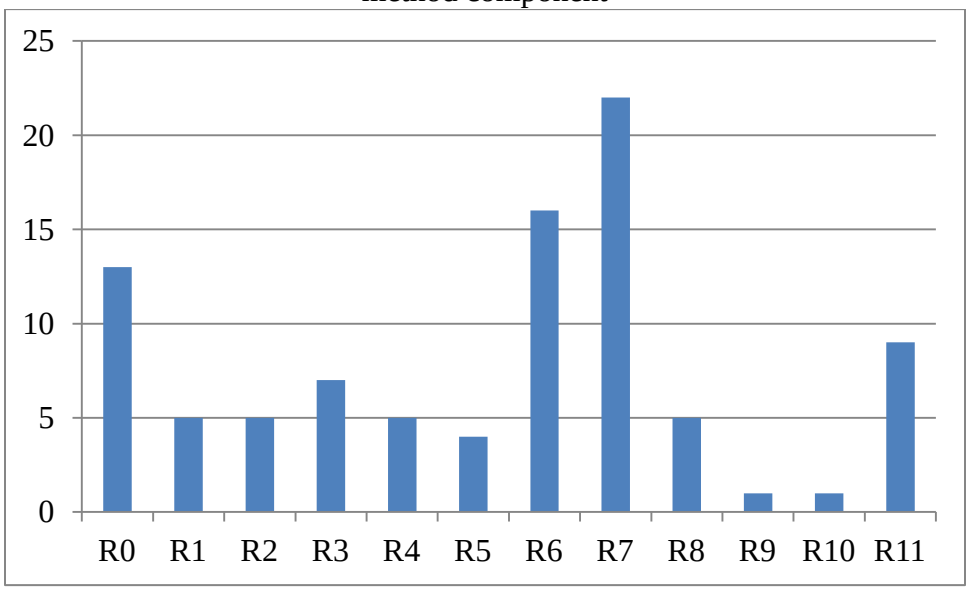


Figure 7.2 shows the frequency each tabu restriction yielded the best results for algorithms that did not include an evolutionary method (i.e. algorithms from Table 5.4, Table 6.3, and Table 6.15). The three most restrictive tabu restrictions, R6, R7, and R11, provided the best results for 51% of the algorithms without an evolutionary component. The restriction R0 provided the best results for 14% of the algorithms without an evolutionary component. In the algorithms where restriction R0 provided the best results, the hash-based tabu strategy due to Klein (2000) was also used. This suggests that the hash-based tabu strategy was a useful addition to the algorithms. The results from section 6.6, Figure 7.1, and Figure 7.2 suggest that the evolutionary method component of the PD-RAMP algorithms have a larger impact on the results than the specific tabu strategy of the tabu search component.

Figure 7.2 – Frequency tabu restrictions yielded best results - algorithms without evolutionary method component



Several tabu search candidate list strategies for the RCPSP were implemented. None of the candidate lists consistently provided the best results for the different configurations of the

PD-RAMP algorithms. As a group, the TS1 candidate lists that were composed of entirely random moves provided the best results for any particular PD-RAMP configuration in all but two cases. This suggests that the other candidate lists may be too restrictive and do not allow the tabu search to sufficiently search the local neighborhoods. Some of the candidate list strategies proposed here reacted to the state of the search to allow more of the local neighborhood to be explored, for example by dropping the resource competitor requirement, as the number of consecutive non-improving moves increased. Perhaps a better strategy would be to change to a completely different candidate list strategy after a certain number of non-improving moves.

Considering the results obtained here and results from the extant literature, tabu search metaheuristics do not seem to perform well for the RCPSP compared to population-based methods. Population-based methods such as genetic algorithms and scatter search heuristics currently dominate the best performing heuristics for solving the RCPSP. Considering the general success of evolutionary methods for the RCPSP, it is not surprising that the evolutionary components of the PD-RAMP algorithms had a large impact on the overall results.

The stand-alone tabu search algorithms using the candidate list strategies did not produce competitive results; however, the utility of the dual information provided by the RAMP approach was demonstrated by the basic RAMP algorithms based on both Lagrangian relaxation and cross-parametric relaxation. These basic RAMP algorithms outperformed the stand-alone tabu search algorithms in almost all variants.

Although the RAMP and PD-RAMP algorithms developed in this study did not outperform the current best performing heuristics for the RCPSP, they are the only relaxation based heuristics that achieve currently competitive results for the j60 and j120 PSPLIB RCPSP instances.

Improvement of the tabu search method is one area for future development. There are many possibilities for candidate list strategies that remain unexplored. One prospect that could be especially interesting is the creation of candidate lists that concentrate on the activities that violated resource constraints in the schedule obtained from the minimum cut. Another possibility is employing a tabu search strategy that combines or changes candidate list strategies based on the current state of the search.

Further work could also include improving the management of the population used by the evolutionary method. Maintaining the EP1 population as two subpopulations of best solutions and diverse solutions, as customary in scatter search, is one possibility. In addition, incorporation of other evolutionary methods in place of or in addition to the current evolutionary method is another possibility.

The current relaxations are based on a specific time-indexed formulation for the RCPSP. Investigating the use of other mathematical formulations for the RCPSP is another area for future work.

There are many variants and extensions to the RCPSP, and other problems exist that can be cast as a RCPSP. Further investigation of extending the RAMP approach presented here to these variants and extensions could be an especially promising area for future research.

LIST OF REFERENCES

Alcaraz, J., & Maroto, C. (2001). A Robust Genetic Algorithm for Resource Allocation in Project Scheduling. [Article]. *Annals of Operations Research, 102*(1-4), 83-109.

Alcaraz, J., Maroto, C., & Ruiz, R. (2004). *Improving the performance of genetic algorithms for the RCPS problem.* Paper presented at the Proceedings of the ninth international workshop on project management and scheduling, Nancy.

Alvarez-Valdes, R., & Tamarit, J. M. (1989). Heuristic algorithms for resource-constrained project scheduling: A review and an empirical analysis. *Advances in project scheduling, 134*.

Baar, T., Brucker, P., & Knust, S. (1998). Tabu Search Algorithms and Lower Bounds for the Resource-Constrained Project Scheduling Problem. In S. Voss, S. martello, I. H. Osman & C. Roucairol (Eds.), *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization* (pp. 1-18). Boston: Kluwer Academic Publishers.

Battiti, R., & Tecchiolli, G. (1994). The Reactive Tabu Search. [Article]. *ORSA Journal on Computing, 6*(2), 126.

Blazewicz, J., Lenstra, J. K., & Kan, A. H. G. R. (1983). Scheduling subject to resource constraints: classification and complexity. *Discrete Applied Mathematics, 5*(1), 11-24. doi: http://dx.doi.org/10.1016/0166-218X(83)90012-4

Boctor, F. F. (1993). Heuristics for scheduling projects with resource restrictions and several resource-duration modes. *International Journal of Production Research, 31*(11), 2547-2558. doi: 10.1080/00207549308956882

Bouleimen, K., & Lecocq, H. (2003). A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version. *European Journal of Operational Research, 149*(2), 268-281. doi: 10.1016/s0377-2217(02)00761-0

Brucker, P., & Knust, S. (1999). Solving Large-Sized Resource-Constrained Project Scheduling Problems. In J. Weglarz (Ed.), *Project Scheduling: Recent Models, Algorithms, and Applications* (pp. 27-51). Boston: Kluwer Academic Publishers.

Brucker, P., Knust, S., Schoo, A., & Thiele, O. (1998). A branch and bound algorithm for the resource-constrained project scheduling problem. *European Journal of Operational Research, 107*(2), 272-288. doi: 10.1016/s0377-2217(97)00335-4

Camerini, P. M., Fratta, L., & Maffioli, F. (1975). On improving relaxation methods by modified gradient techniques. In M. L. Balinski & P. Wolfe (Eds.), (Vol. 3, pp. 26-34): Springer Berlin Heidelberg.

Carlier, J., Moukrim, A., & Xu, H. (2010). Project Scheduling with Production and Consumption of Resources: How to Build Schedules *Resource-Constrained Project Scheduling* (pp. 161-170): ISTE.

Carlton, W. B., & Barnes, J. W. (1996). A note on hashing functions and tabu search algorithms. *European Journal of Operational Research, 95*(1), 237-239. doi: 10.1016/0377-2217(95)00249-9

Christofides, N., Alvarez-Valdes, R., & Tamarit, J. M. (1987). Project scheduling with resource constraints: A branch and bound approach. *European Journal of Operational Research, 29*(3), 262-273. doi: http://dx.doi.org/10.1016/0377-2217(87)90240-2

Coelho, J., & Tavares, L. (2003). Comparative analysis of meta-heuricstics for the resource constrained project scheduling problem (D. o. C. Engineering, Trans.): Instituto Superior Tecnico, Portugal.

Debels, D., De Reyck, B., Leus, R., & Vanhoucke, M. (Eds.). (2006) European Journal of Operational Research (Vols. 169).
Debels, D., & Vanhoucke, M. (2005). A Decomposition-Based Heuristic For The Resource-Constrained Project Scheduling Problem: Ghent University, Faculty of Economics and Business Administration.
Debels, D., & Vanhoucke, M. (2007). A Decomposition-Based Genetic Algorithm for the Resource-Constrained Project-Scheduling Problem. *Operations Research, 55*(3), 457-469. doi: 10.1287/opre.1060.0358
Drexl, A. (1991). Scheduling of Project Networks by Job Assignment. *Management Science, 37*(12), 1590-1602. doi: doi:10.1287/mnsc.37.12.1590
Franck, B., Neumann, K., & Schwindt, C. (2001). Truncated branch-and-bound, schedule-construction, and schedule-improvement procedures for resource-constrained project scheduling. *OR Spectrum, 23*(3), 297-324. doi: 10.1007/pl00013356
Gagnon, M., Boctor, F. F., & d'Avignon, G. (2004). *A Tabu Search Algorithm for the Resource-Constrained Project Scheduling Problem.* Paper presented at the ASAC 2004, Quebec, QUEBEC.
Geoffrion, A. M. (1971). Duality in Nonlinear Programming: A Simplified Applications-Oriented Development. *SIAM Review, 13*(1), 1-37. doi: 10.2307/2028848
Glover, F. (1975). Surrogate Constraint Duality in Mathematical Programming. [Article]. *Operations Research, 23*(3), 434.
Glover, F. (1989). Tabu Search--Part I. [Article]. *ORSA Journal on Computing, 1*(3), 190.
Glover, F. (1990). Tabu Search--Part II. [Article]. *ORSA Journal on Computing, 2*(1), 4.
Glover, F., & Laguna, M. (1997). *Tabu Search*. Boston: Kluwer Academic Publishers.
Goldberg, A. V., & Tarjan, R. E. (1988). A new approach to the maximum-flow problem. *J. ACM, 35*(4), 921-940. doi: 10.1145/48014.61051
Gonçalves, J., Resende, M., & Mendes, J. (2011). A biased random-key genetic algorithm with forward-backward improvement for the resource constrained project scheduling problem. *Journal of Heuristics, 17*(5), 467-486. doi: 10.1007/s10732-010-9142-2
Hartmann, S. (1998). A competitive genetic algorithm for resource-constrained project scheduling. *Naval Research Logistics (NRL), 45*(7), 733-750. doi: 10.1002/(sici)1520-6750(199810)45:7<733::aid-nav5>3.0.co;2-c
Hartmann, S. (2002). A self-adapting genetic algorithm for project scheduling under resource constraints. *Naval Research Logistics (NRL), 49*(5), 433-448. doi: 10.1002/nav.10029
Hartmann, S., & Kolisch, R. (2000). Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem. *European Journal of Operational Research, 127*(2), 394-407. doi: 10.1016/s0377-2217(99)00485-3
Icmeli, O., & Erenguc, S. S. (1994). A tabu search procedure for the resource constrained project scheduling problem with discounted cash flows. *Computers & Operations Research, 21*(8), 841-853. doi: 10.1016/0305-0548(94)90014-0
Kelley, J. E. J., & Walker, M. R. (1959). *Critical-path planning and scheduling*. Paper presented at the Papers presented at the December 1-3, 1959, eastern joint IRE-AIEE-ACM computer conference, Boston, Massachusetts. http://dl.acm.org/citation.cfm?id=1460318
Klein, R. (2000). Project scheduling with time-varying resource constraints. *International Journal of Production Research, 38*(16), 3937-3952. doi: 10.1080/00207540050176094

Kochetov, Y., & Stolyar, A. (2003). *Evolutionary Local Search with Variable Neighborhood for the Resource Constrained Project Scheduling Problem.* Paper presented at the Workshop on Computer Science and Information Technologies CSIT'2003, Ufa, Russia.

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

Kolisch, R. (1996a). Efficient priority rules for the resource-constrained project scheduling problem. *Journal of Operations Management, 14*(3), 179-192. doi: 10.1016/0272-6963(95)00032-1

Kolisch, R. (1996b). Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation. *European Journal of Operational Research, 90*(2), 320-333. doi: 10.1016/0377-2217(95)00357-6

Kolisch, R., & Drexl, A. (1996). Adaptive search for solving hard project scheduling problems. *Naval Research Logistics (NRL), 43*(1), 23-40. doi: 10.1002/(sici)1520-6750(199602)43:1<23::aid-nav2>3.0.co;2-p

Kolisch, R., & Hartmann, S. (1998). Heuristic Algorithms for the Resource-Constrained Project Scheduling Problem: Classification and Computational Analysis. In J. Weglarz (Ed.), *Project Scheduling: Recent Models, Algorithms, and Applications* (pp. 147-178). Boston: Kluwer Academic Publishers.

Kolisch, R., & Hartmann, S. (2006). Experimental investigation of heuristics for resource-constrained project scheduling: An update. *European Journal of Operational Research, 174*(1), 23-37. doi: 10.1016/j.ejor.2005.01.065

Kolisch, R., Schwindt, C., & Sprecher, A. (1999). Benchmark instances for project scheduling problems *Project Scheduling* (pp. 197-212): Springer.

Kolisch, R., & Sprecher, A. (1996). PSPLIB - A project scheduling problem library: OR Software - ORSEP Operations Research Software Exchange Program. *European Journal of Operational Research, 96*(1), 205-216. doi: 10.1016/s0377-2217(96)00170-1

Kolisch, R., Sprecher, A., & Drexl, A. (1995). Characterization and Generation of a General Class of Resource-Constrained Project Scheduling Problems. [Article]. *Management Science, 41*(10), 1693-1704.

Laguna, M., Barnes, J. W., & Glover, F. W. (1991). Tabu search methods for a single machine scheduling problem. *Journal of Intelligent Manufacturing, 2*(2), 63-73. doi: 10.1007/bf01471219

Lee, J.-K., & Kim, Y.-D. (1996). Search Heuristics for Resource Constrained Project Scheduling. *Journal of the Operational Research Society, 47*(5), 678-689.

Leon, V. J., & Balakrishnan, R. (1995). Strength and adaptability of problem-space based neighborhoods for resource-constrained scheduling. *Operations-Research-Spektrum, 17*(2-3), 173-182. doi: 10.1007/bf01719262

Mendes, J. J. M., Gonçalves, J. F., & Resende, M. G. C. (2009). A random key based genetic algorithm for the resource constrained project scheduling problem. *Computers & Operations Research, 36*(1), 92-109. doi: 10.1016/j.cor.2007.07.001

Merkle, D., Middendorf, M., & Schmeck, H. (2002). Ant colony optimization for resource-constrained project scheduling. *Evolutionary Computation, IEEE Transactions on, 6*(4), 333-346. doi: 10.1109/tevc.2002.802450

Möhring, R., Schulz, A., Stork, F., & Uetz, M. (1999). Resource-Constrained Project Scheduling:Computing Lower Bounds by Solving Minimum Cut Problems. In J. Nešetril (Ed.), (Vol. 1643, pp. 697-697): Springer Berlin / Heidelberg.

Möhring, R., Schulz, A., Stork, F., & Uetz, M. (2003). Solving Project Scheduling Problems by Minimum Cut Computations. [Article]. *Management Science, 49*(3), 330-350.
Nonobe, K., & Ibaraki, T. (1998). A tabu search approach to the constraint satisfaction problem as a general problem solver. *European Journal of Operational Research, 106*(2–3), 599-623. doi: 10.1016/s0377-2217(97)00294-4
Nonobe, K., & Ibaraki, T. (2002). Formulation and Tabu Search Algorithm for the Resource Constrained Project Scheduling Problem. In C. C. Ribeiro & P. Hansen (Eds.), *Essays and Surveys in Metaheuristics* (pp. 557-588). Boston: Kluwer Academic Publishers.
Patterson, J. H. (1984). A Comparison of Exact Approaches for Solving the Multiple Constrained Resource, Project Scheduling Problem. [Article]. *Management Science, 30*(7), 854-867.
Pinson, E., Prins, C., & Rullier, F. (1994). *Using Tabu Search for Solving the Resource-Constrained Project Scheduling Problem.* Paper presented at the 4th International Workshop on Project Management and Scheduling, Leuven, Belgium.
Polyak, B. T. (1969). Minimization of unsmooth functionals. *USSR Computational Mathematics and Mathematical Physics, 9*(3), 14-29. doi: http://dx.doi.org/10.1016/0041-5553(69)90061-5
Pritsker, A. A. B., Watters, L. J., & Wolfe, P. M. (1969). Multiproject Scheduling With Limited Resources: A Zero-One Programming Approach. [Article]. *Management Science, 16*(1), 93-108.
Rangaswamy, B., Jain, A. S., & Glover, F. (1998). Tabu Search Candidate List Strategies in Scheduling. In D. L. Woodruff (Ed.), *Advances in Computational and Stochastic Optimization, Logic Programming, and Heuristic Search: Interfaces in Computer Science and Operations Research* (pp. 215-233): Kluwer Academic Publishers. (Reprinted from: December 7, 2010).
Rego, C. (2005). RAMP: A New Metaheuristic Framework for Combinatorial Optimization. In R. Sharda, S. Voß, C. Rego & B. Alidaee (Eds.), *Metaheuristic Optimization via Memory and Evolution* (Vol. 30, pp. 441-460): Springer US.
Schirmer, A. (2000). Case-based reasoning and improved adaptive search for project scheduling. *Naval Research Logistics (NRL), 47*(3), 201-222. doi: 10.1002/(sici)1520-6750(200004)47:3<201::aid-nav2>3.0.co;2-l
Sprecher, A., Kolisch, R., & Drexl, A. (1995). Semi-active, active, and non-delay schedules for the resource-constrained project scheduling problem. *European Journal of Operational Research, 80*(1), 94-102. doi: 10.1016/0377-2217(93)e0294-8
Thomas, P. R., & Salhi, S. (1995). A Lagrangian Heuristic Approach for the Resource-Constrained Project Scheduling Problem (S. o. M. a. Statistics, Trans.). University of Birmingham.
Thomas, P. R., & Salhi, S. (1998). A Tabu Search Approach for the Resource Constrained Project Scheduling Problem. *Journal of Heuristics, 4*(2), 123-139. doi: 10.1023/a:1009673512884
Tormos, P., & Lova, A. (2001). A Competitive Heuristic Solution Technique for Resource-Constrained Project Scheduling. [Article]. *Annals of Operations Research, 102*(1-4), 65-81.
Tormos, P., & Lova, A. (2003a). An efficient multi-pass heuristic for project scheduling with constrained resources. *International Journal of Production Research, 41*(5), 1071-1086. doi: 10.1080/0020754021000033904

Tormos, P., & Lova, A. (2003b). Integrating heuristics for resource constrained project scheduling: One step forward (D. o. S. a. O. Research, Trans.) *Technical Report*: Universidad Politecnica de Valencia.

Tsai, Y.-W., & D. Gemmill, D. (1998). Using tabu search to schedule activities of stochastic resource-constrained projects. *European Journal of Operational Research, 111*(1), 129-141. doi: 10.1016/s0377-2217(97)00311-1

Tseng, L.-Y., & Chen, S.-C. (2006). A hybrid metaheuristic for the resource-constrained project scheduling problem. *European Journal of Operational Research, 175*(2), 707-721. doi: 10.1016/j.ejor.2005.06.014

Valls, V., Ballestín, F., & Quintanilla, S. (2004). A Population-Based Approach to the Resource-Constrained Project Scheduling Problem. [Article]. *Annals of Operations Research, 131*(1-4), 305-324. doi: 10.1023/B:ANOR.0000039524.09792.c9

Valls, V., Ballestín, F., & Quintanilla, S. (2005). Justification and RCPSP: A technique that pays. *European Journal of Operational Research, 165*(2), 375-386. doi: 10.1016/j.ejor.2004.04.008

Valls, V., Ballestín, F., & Quintanilla, S. (2008). A hybrid genetic algorithm for the resource-constrained project scheduling problem. *European Journal of Operational Research, 185*(2), 495-508. doi: 10.1016/j.ejor.2006.12.033

Valls, V., Quintanilla, S., & Ballestín, F. (2003). Resource-constrained project scheduling: A critical activity reordering heuristic. *European Journal of Operational Research, 149*(2), 282-301. doi: 10.1016/s0377-2217(02)00768-3

Woodruff, D. L., & Zemel, E. (1993). Hashing vectors for tabu search. [Article]. *Annals of Operations Research, 41*(1-4), 123-137.

VITA

ROBERT CHRISTOPHER-LEE RILEY

EDUCATION

B.S., Electrical Engineering, The University of Mississippi, May 2000
M.S., Engineering Science – Electrical Engineering, The University of Mississippi, December 2003

TEACHING EXPERIENCE

Graduate Instructor, Summer 2009, Fall 2010, Spring 2011
The University of Mississippi
Course: Introduction to Operations & Supply Chain Management

Graduate Instructor, Fall 2003
The University of Mississippi
Course: Programming for Engineering and Science

Additional teaching experience in Mississippi public secondary schools

PUBLICATIONS and PRESENTATIONS

Journal & Transaction Articles

- **Antenna Design and Radiation Pattern Visualization**, A. Z. Elsherbeni, M. J. Inman, and R. C.L. Riley. Applied Computational Electromagnetics Society Journal, Vol. 18, No. 4, November 2003.
- **Redundant Arrays of IDE Drives**, D. A. Sanders, L. M. Cremaldi, V. Eschenburg, C. N. Lawrence, C. Riley, D. J. Summers, and D. L. Petravick. IEEE Transactions on Nuclear Science, Vol. 49, No. 4, August 2002.

Conference Proceedings (Full Papers)

- **A Simple Dual-RAMP Algorithm for Resource Constraint Project Scheduling**, C. Riley, C. Rego, H. Li. 48th ACM Southeast Regional Conference (ACMSE 2010); Oxford, MS; April 15-17, 2010.
- **Terabyte IDE RAID-5 Disk Arrays**, D. A. Sanders, L. M. Cremaldi, V. Eschenburg, R. Godang, C. N. Lawrence, C. Riley, D. J. Summers, and D. L. Petravick. 2003 Conference

for Computing in High Energy and Nuclear Physics (CHEP 2003); La Jolla, CA; March 24-28, 2003.

- **Antenna Design and Radiation Pattern Visualization**, Atef Z. Elsherbeni, Matthew J. Inman, and R. Christopher-Lee Riley. The Applied Computational Electromagnetics Society Symposium 2003; Monterey, CA; March 24-28, 2003.
- **Tools for Electromagnetic Modeling and Visualization Using the FDTD Technique**, Atef Z. Elsherbeni, Allen W. Glisson, Chris Riley, and Charles E. Smith. The Symposium on Antenna Technology and Applied Electromagnetics (ANTEM 2000); Winnipeg, Manitoba, CANADA; July 30 - August 2, 2000.
- **A Graphical User Interface for a Finite Difference Time Domain Simulation Tool**, Atef Z. Elsherbeni, Chris L. Riley, and Charles E. Smith. IEEE Region 3 SoutheastCON 2000; Nashville, TN; April 7- 9, 2000.
- **Work with Apple's Rhapsody Operating System which Allows Simultaneous UNIX Program Development, UNIX Program Execution, and Commercial PC Application Program Execution**, Don Summers, Chris Riley, Lucien Cremaldi, and David Sanders. Computing in High Energy Physics 1998 (CHEP '98); Chicago, Illinois; August 31 - September 4, 1998.

Conference Proceedings (Abstract and Oral Presentation)

- **A RAMP Approach to the Resource Constrained Project Scheduling Problem**, C. Riley, C. Rego, H. Li. INFORMS Annual International Meeting; San Diego, CA; October 11-14, 2010.
- **Modeling and Simulation of Synthetic Aperture Radar (SAR) Imaging Systems,** R. Christopher-Lee Riley and Atef Z. Elsherbeni. MSCI 4th Annual Graduate Studies Symposium – Stennis Space Center; University, MS; April 2003.
- **Modeling and Simulation of Synthetic Aperture Radar (SAR) Imaging Systems,** R. Christopher-Lee Riley and Atef Z. Elsherbeni. UM Remote Sensing Research Forum – University of Mississippi; University, MS; April 2003.
- **Performance Analysis of a Nonuniform 3-D Finite Difference Electromagnetics Simulation Code**, R. Christopher-Lee Riley and Atef Z. Elsherbeni. Mississippi Academy of Sciences; Hattiesburg, MS; March 2003.
- **Antenna Design and Radiation Pattern Visualization**, Atef Z. Elsherbeni, Matthew J. Inman, and R. Christopher-Lee Riley. Mississippi Academy of Sciences; Hattiesburg, MS; March 2003.
- **Input Interface for a Finite Difference Time Domain Electromagnetic Simulation Tool**, Chris L. Riley, James D. Vernon, Atef Z. Elsherbeni and Charles E. Smith. Mississippi Academy of Sciences; Biloxi, MS; March 2000.

Posters (Abstract and Poster Presentation)

- **Dual-RAMP Algorithm for the Resource Constrained Project Scheduling Problem**, C. Riley, C. Rego, H. Li. INFORMS Annual International Meeting; Austin, TX; November 7-10, 2010.

- **Antenna Design and Radiation Pattern Visualization**, A. Z. Elsherbeni, M. J. Inman, and R. C.L. Riley. The 19th Annual Review of Progress in Applied Computational Electromagnetics; Monterey, CA; March 24-28, 2003.
- **Input Interface for a Finite Difference Time Domain Electromagnetic Simulation Tool,** Chris L. Riley, Atef Z. Elsherbeni, and Charles E. Smith. Sigma Xi Student Poster Competition; University of Mississippi; 2000.
- **Input Interface for a Finite Difference Time Domain Electromagnetic Simulation Tool**, Chris L. Riley, Atef Z. Elsherbeni, Matthew J. Inman, and Charles E. Smith. Undergraduate Research Day at the Capitol Synopsis; Jackson, MS; 2000.
- **Working with Arrays of Inexpensive EIDE Disk Drives**, David Sanders, Chris Riley, Lucien Cremaldi, and Don Summers. Computing in High Energy Physics 1998 (CHEP '98); Chicago, Illinois; August 31 - September 4, 1998.