INDIAN INSTITUTE OF MANAGEMENT
AHMEDABAD • INDIA

Research and Publications

Multi-processor Exact Procedures for Regular Measures of the Multi-mode RCPSP

Madhukar Dayal
Sanjay Verma

W.P. No. 2015-03-25
March 2015

INDIAN INSTITUTE OF MANAGEMENT
AHMEDABAD-380 015
INDIA

MULTI-PROCESSOR EXACT PROCEDURES FOR REGULAR MEASURES OF THE MULTI-MODE RCPSP[1]

Madhukar Dayal
Indian Institute of Management, Indore
madhukar@iimidr.ac.in

Sanjay Verma
Indian Institute of Management, Ahmedabad
sverma@iimahd.ernet.in

Abstract

The multi-mode resource-constrained project scheduling problem (MM RCPSP) is an NP-hard problem representing a generalization of the well-studied RCPSP. Depth-first tree search approach by Sprecher & Drexl (1998) is the best-known exact solution tree search procedure for this problem. In this paper we modify an existing breadth-first algorithm for multiple processors. It is a computer-cluster implementation of the breadth-first procedure which improves the solution time taken for these problem instances.

Keywords: project scheduling, regular measures, exact multi-objective solutions, breadth-first tree search, computer-cluster algorithm.

[1] This work was done as part of doctoral thesis of Prof. Madhukar Dayal at IIM Ahmedabad

1 INTRODUCTION

1.1 Overview

The project scheduling problem (PSP) has attracted the attention of a large number of researchers, especially since the World War-2. Enhancement of computational power has progressively enabled the search for solutions to large and more complex problems. However, the search for optimal solutions to projects with even a few hundred or more activities, in a reasonable time, poses a challenge even today. Various approaches such as *heuristic*, *metaheuristic*, and *exact solution approaches* – have been developed by researchers to find a solution to the PSP. Among the exact solution approaches, the branch and bound (B&B) approaches are further classified as *depth-first*, *breadth-first*, and *best-first* approaches, based on the direction chosen for search path. The appeal of the problem is enhanced by the wide range and variety of desirable objectives which can be studied – makespan, (total and minimum) lateness, earliness, tardiness, number of late jobs, flowtime, their weighted measures, and several others.

Initial scheduling approaches primarily emanated from shop floor experience and resulted in the generation of a large number of heuristics or thumb rules; a few popular ones being Earliest Due Date First (EDD), Shortest Processing Time First (SPT), First In First Out (FIFO), and Shortest Remaining Time First (SRT). These thumb rules are quite effective and yield good solutions for *simple instances* of the PSP. However, these rules are not guaranteed to yield an optimal solution.

Support for development of linear, binary and integer programming techniques, and enhancement of computational power and memory has enabled modeling and development of algorithms for problems of moderate size. However, for problems of larger sizes even the mixed integer linear programming (MILP) solvers have limitations (for example, 2^32 variables in CPLEX 12) due to the number of variables involved as it rises very rapidly for even problems of moderate size. These implementations use one approach from: the primal simplex (or dual simplex) algorithm, the network optimizer, barrier algorithm, or the sifting algorithm. MILP problem instances are solved using branch and cut procedures. Our algorithm differs in that it is a breadth-first tree generation branch and bound approach. The exploration of search for exact solutions to concurrent multiple objectives (i.e. *exact multi-objective solution*) enhances the involved challenge several folds, a feature which is missing from the best MILP solvers.

The objective of this paper is to design and develop improved exact algorithms for optimally scheduling partially ordered multi-mode activities under resource constraints, and to test whether current computational power is enough to solve real life problems. This problem is known as the Multi-Mode Resource Constrained Project Scheduling Problem (MM-RCPSP). We design and develop single processor algorithms for exact solutions to a single objective. We then extend our breadth-first tree-search approach to yield *multiple exact optimal solutions for a single objective*. Among this set of optimal single objective solutions we search for *exact multi-objective optimal solutions*. Finally, we extend the exact multi-objective breadth-first algorithm to a distributed

version (using OpenMP) for implementation over a computer-cluster. Our experiments show promising results on benchmark problem sets from PSPLIB.

1.2 Problem Statement

A *project* is a set of activities which are partially ordered by precedence relationships. An *activity* can be performed infinite number of modes, where each mode is unique and has a corresponding non-negative duration. An activity is ready to be processed only when all its predecessor activities are completed *and* the number of units of the various resource types required by it, in the mode that it is to be performed, are free and can be allocated to it. Once started, an activity is not interrupted (*non-preemptive*) and runs to its completion. The dummy (start and end) activities consume no resources and take no time. For each of its *modes*, an activity uses different types of resources, such as manpower and machinery, in different amounts, which are specified in advance. A mode specifies an activity's resource requirements for each resource type and its duration in that mode. A *resource* is an essential facilitator of an activity to be performed. It may be durable (*renewable*) or consumable (*non-renewable*). The resources are allocated exclusively to a single activity for its entire duration in the selected mode. A resource may also be *doubly constrained*, i.e. it has an overall limit of availability for the whole project, as well as, time period wise limit of consumption for each time period. The availability of each resource type is known in advance. After completion of an activity, renewable resources may be assigned to another activity, whereas, the amounts of non-renewable and doubly constrained resources decrease by the respective amounts of each of these resources consumed in completion of the activity in its assigned mode, and only the residual amounts can be used further.

Scheduling is the process of selecting the mode and committing resources to the realization of each activity, while meeting the precedence and resource restrictions, to optimize a given objective. The aim is to assign modes and start times to all activities so that the desired objective (for example, makespan, flowtime, maximum tardiness, number of tardy jobs, etc.) is optimized.

The objective to be optimized may be *regular* or *non-regular*. Regular measures are those measures for which no performance improvements will occur with delay in the start of the activities, for example, minimizing completion time or minimizing the tardiness. Non-regular measures are those measures for which the performance may improve with delay in start of the activities, for example, in objectives like minimizing the earliness-tardiness in just in time (JIT) and maximizing net present value (NPV).

In the case of non-regular measures (NPV), every mode of each activity has an associated cash flow (either inflow or outflow) at the start of the activity, for each unit time of its duration in the selected mode, and at the end of the activity. The objective in this case is to schedule all activities such that the NPV of their cash flows, at a given rate of interest is maximized.

1.3 Literature Survey

This section reviews the literature in project scheduling focusing mainly on MM-RCPSP.
While small projects and shop floor scheduling problems may use exact approaches, large

problems, being complex for the human mind or computer to comprehend and solve, are dependent on heuristics. The pursuit of one or more of several desirable objectives, simultaneously, enhances the complexity of the problem further.

Several heuristic and metaheuristic approaches have been presented in the literature to solve large scheduling problems. However, these approaches do not guarantee the yield of an optimal solution. Usually, these approaches deploy one or more checking procedures for termination of the algorithm, such as, acceptable limit on minimum percentage improvement from previously found best solution, run time bounds, and/or the number of iterations limit. In these approaches, it is possible that in multiple runs of the same algorithm using same termination criteria, and on the same problem instance and computing machine, an inferior or superior result is obtained. This clearly establishes the need for improved exact algorithms for finding the exact solutions to such problems. However, the research approaches pursuing inexact or approximate solutions is many times more than that for exact solutions. Our research attempts to cover this gap.

Exact solution approaches for the MM-RCPSP problem are few, and restricted in application to problems of small sizes only. Carefully implemented *explicit enumeration algorithms* promise the yield of an optimal solution, if one exists. These approaches are classified into three types – *depth-first*, *breadth-first*, and *best-first*. It is also possible to conceptualize hybrid approaches which merge features of more than one approach, especially in multi-thread/multiprocessor algorithms, however, this area of research is hardly explored. The resources (time and memory) consumed in these approaches make them unattractive for large problem instances. Due to this, currently, managers have to take recourse to available non-optimal approaches for solving large scale project scheduling problems.

1.3.1 Exact Approaches

Developing a mode alternative, similar to Demeulemeester and Heroelen's (1992) delay alternative, and applying a B&B procedure with search tree reduction scheme, Sprecher, Hartmann, and Drexl (1997) and Sprecher and Drexl (1998) presented algorithms for obtaining an exact solution to the MM-RCPSP. Daniels and Mazzola (1994) studied the problem in a flow shop environment where non-renewable resource allocation is of a flexible nature. They identify the properties of optimal B&B solutions and apply these to solve the problems using their iterative heuristics.

Hartmann and Drexl (1998) compare three B&B approaches for the MM-RCPSP and conclude that the precedence tree guided enumeration scheme performs the best. A B&B depth-first procedure for obtaining the optimal solution and its truncated version are presented by Sprecher and Drexl (1998) for obtaining exact solutions and tested on a large number of problem instances. This approach remains the best exact approach to date. They also discuss the impact of variation in several project characteristics on solution time and quality. Erenguc, Ahn and Conway (2001) presented an integer programming model and an exact solution B&B procedure adopting branching rules, minimal resource conflict sets, and node fathoming rules for improving efficiency. Heilmann (2003) has presented another exact B&B approach for small instances and a priority rule based heuristic approach (2004) for larger instances of the MM-RCPSP.

Buddhakulsomsiri and Kim (2006) apply the B&B procedure and concluded that in resource vacations and temporary resource unavailability, activity splitting can improve the optimal project makespan, and that makespan improvement is dependent on those parameters which determine resource utilization. Sabzehparvar and Seyed-Hosseini (2008) studied the problem in a mode dependent time lag environment and presented an exact algorithm. They relate the problem to a bin-packing problem and present its mixed-integer programming formulation. They also presented a geometric formulation of the problem and a B&B approach to obtain solutions to the problem instances tested.

Exact approaches are attractive because they guarantee an optimal solution. However, the adopted approaches, so far, have examined only depth-first B&B strategy enhanced with pruning and truncation rules. Breadth-first approach is extremely challenging due to the rapid expansion of the state space tree. The best-first approach too expands the search tree rather fast needing a large amount of memory. It also often explores several branches before reaching the optimal solution. The depth-first approach, once it starts backtracking, always outputs a feasible solution whenever it terminates. The other two approaches have to run to completion to produce a complete feasible solution. Best-first and depth-first approaches are able to make use of lower and upper bounds effectively in pruning branches, thus, reducing the size of the search space tree as they reach the solution. If time bound executions of the breadth-first and best-first algorithms are used, they are often likely to result in incomplete solutions and would need to be augmented by some fast heuristic schemes to generate feasible solutions. Hybrid approaches, such as a mix of metaheuristic and tree-search approaches: (a) to generate partial solutions and identify promising directions of search, and then (b) to develop the exact final solutions, may be attractive directions of research.

1.3.2 Multi-Objective Solution Approaches

Finding even a single objective optimal solution to the NP-hard MM-RCPSP problem is a computationally expensive task. However, real world situations require managers to strive to optimize multiple objectives together (for example reducing both, completion time of a project and consumption of precious resources in its completion). Not surprisingly, at times the objectives are mutually incompatible! The initial approaches adopted for Multiple Criteria Decision Making (MCDM) involved assignment of weights or priorities to various decisions. It is unrealistic to estimate the correct weights or even priorities for various criteria. Further, in decisions which are based on one or more subjective criteria, the assignment of such weights may be impractical, if not infeasible.

Analytic Hierarchy Process (AHP) is a popular approach for evaluating available options and their outcomes in MCDM. The stepwise development of a decision tree can simultaneously process objective, as well as, subjective criteria, while allowing more important criteria to be considered before others. However, it is not free from the subjectivity of estimations of mutual and relative weights of objectives considered, and from assignment of equivalents (in terms of cost or other suitable metric) to various alternatives. It also relies on the decision maker's choice or order of criteria for objective selection. The method and analysis of its results become difficult

to apply with increase in elements in the list of criteria, particularly when stochastic success rates of various options are also involved. For a detailed review of the MCDM approaches and their classification we refer the reader to Behzadian, Kazemzadeh, Albadvi, and Aghdasi (2010).

Mathematical programming approaches attempt to solve a problem separately for two (or more) single objectives, and then, within the bounds thus established by the single objective solutions, perform Pareto analysis from one solution point to the other, generating Pareto boundaries of good solutions. Typically, such analysis involves comparison of relative gain and loss, and its net effect on the objective function value, in the entire neighborhood of the hyperspace path between two known single objective optimal solutions, almost always sacrificing even the single optimality earlier achieved. For large problems, the two starting solutions themselves may not be optimal for even a single objective. The need to repeatedly solve a sub-problem many times makes these approaches computationally expensive and good only for small problems. However, for quantitative decision analysis, these approaches appear to be an interesting research direction with rapidly rising attention of researchers.

In problems which have multiple single objective optimal solutions, among which exact multi-objective optimization search is feasible, it is possible that the optimal solution points for a single objective are placed extremely far apart in the solution hyper-space. Thus, a neighborhood search, or a directed path search between two single objective optimal solutions, may actually be unable to yield an exact multi-objective optimal solution even if one exists. Further, a hyper-line connecting two exact optimal solutions for two different objectives, may actually not even touch any other single or multiple objective optimal solutions for any objective, at all.

Even though it is understood that a PSP problem may have multiple optimal solutions, the task of finding a single solution itself is so arduous that effort to find all or multiple optimal solutions has been missing in literature. MILP solvers, given a different random starting seed, may generate different solutions, though only for problems of modest sizes in a reasonable time. In project scheduling problem instances, trivial multiple optimal solutions can be rapidly developed from one optimal solution by carefully shifting the non-critical path activities within the available slack, without changing their assigned modes. However, no approach to developing multiple exact single objective solutions, and further, exact multi-objective solutions from them, for even small problems exists in literature. Our research fills this essential gap in the literature and opens new directions for further research.

1.3.3 Multi-Processor Algorithms

The advent of multi-processor architectures and availability of computer-clusters renders ever increasing computational power available for solving difficult problems. However, algorithms to exploit such architectures in solving difficult problems are lacking and research in this direction is slowly gaining momentum. To the best of our knowledge, no multiprocessor or distributed computing approach appears in the literature to obtain exact solutions for the MM-RCPSP. We develop our breadth-first algorithm for implementation over a computer-cluster and test it on a cluster of sixteen CPUs in shared memory processors (SMP) architecture. The algorithm scales very well and solves small problems extremely fast and larger problems in much less time when

compared to a single processor. To the best of our knowledge, our multi-processor exact breadth-first algorithm for the MM-RCPSP implemented over a computer-cluster is the first of its kind to be able to exploit multiple processors, generate multiple optimal solutions, and enable exact solutions to multi-objective optimization problems, with modest memory requirements in a reasonable time. At the same time, it shows promising scalability with increase in number of processors in our tests on up to sixteen processors, although not uninfluenced by Amdahl's law (Amdahl (1967) and Gustafson(1988)). However, it is a disappointment that compute-clusters being extremely expensive, this solution methodology may take some time before it becomes available for exploitation to relatively small and medium sized organizations. As per our literature review, no other exact solution tree search implementation on multiple processors is available for the MM-RCPSP.

1.3.4 Other Important Studies

A project scheduling problem may be easy if resources are abundantly available. It may also be easy if the resources are highly scarce, as in both the cases, the number of options to be explored for simultaneously performed activities is reduced, reducing the needed search in the space tree. Herroelen and De Reyck (1999) have studied the project scheduling problem's transition from easy to hard and hard to easy levels under varying levels of resource availability and network complexity. They concluded that while network complexity measures seem to reveal continuous phase transitions for project scheduling problems, the resource parameters exhibit a relatively sharp transition behavior in the problem's difficulty level. Sprecher (2000) has presented an efficient model for the problem and a solution approach which requires far less memory than other approaches.

As early studies examine only a few problem instances, there is a need to generate problems with controllable difficulty levels. Klingman, Napier and Stutz (1974) have presented a network generator, NETGEN, for generating assignment, transportation and Minimum Cost Flow networks. For project networks, Kolish, Sprecher and Drexl (1995) presented ProGen, a controlled-difficulty project network problem generator. However, in ProGen the feasibility of the problems generated is required to be tested separately. Their work is extended by Demeulemeester, Dodin, and Herroelen (1993) for generating feasible instances from the generation space. Kolisch and Sprecher (1997) proposed a continuously upgraded bank of problems including suggesting the collection of best-known solutions contributed by various researchers, the PSPLIB (http://129.187.106.231/psplib/). Drexl, Nissen, Patterson, and Salewski, (2000) have presented ProGen/JIx, which is able to generate problems for a variety of networks, including crew scheduling and timetabling.

Testing a small number of benchmark problems is arguably only a limited proof of wide applicability of any solution approach. Hence, researchers have chosen to generate a large number of test problems and apply their algorithms on these. We test our algorithms on established benchmark problem instances, as well as, on our own generated problem instances.

1.4 Motivation

Among the approaches to obtain exact solutions to the PSP, including multiple resource constrained projects, very few have considered the possibility of performing an activity in more than one mode (Multi-mode). The existing studies of the problem use heuristic and/or metaheuristic approaches to obtain good solutions. The exact solution approaches, both MILP and the depth-first B&B technique, are able to solve only problems of a limited size. Research to develop alternate techniques is, hence, continuously needed.

Nazareth and Bhattacharya (1993) and Nazareth, Verma, Bhattacharya and Bagchi (1999) develop and apply a breadth-first approach to solving the single mode RCPSP, which generates results comparable to the other existing algorithms. They also apply the best-first approach with comparable results. However, these approaches have not been studied or applied further to the MM-RCPSP. We study the MM-RCPSP for an exact solution for regular and non-regular measures, developing breadth-first and best-first search procedures with pruning rules that do not sacrifice the optimality of the solution. We extend our work to implement the breadth-first approach on an SMP computer-cluster using OpenMP and show that it scales extremely well.

1.5 Summary of Results

In this paper we develop exact algorithms for the multi-mode project scheduling problem with renewable and non-renewable resources. Though we do not treat doubly constrained resources separately, these can be easily modeled by considering them as a pair of joint renewable and non-renewable resources. We develop and test two different algorithmic approaches: (a) an exact breadth-first tree search approach, and (b) a monotone best-first heuristic approach, both of which yield an exact solution. We develop these algorithms to solve problems with regular performance measure, as well as, non-regular measures.

These algorithms are described with examples in Shukla and Verma (2014a and 2014b) with experimental results on problem sets from the PSPLIB. Proofs of the optimality of the algorithms are included. The methods have been compared with the depth-first method of Sprecher and Drexl (1998) using similar data structures for both approaches, and prove to be faster. Among the two, the best-first is faster on a single processor. However, the first advantage of breadth-first is in its ability to generate multiple exact single-objective solutions, from which exact multi-objective solutions can be readily obtained. Secondly, exploiting the design of our breadth-first algorithm, we develop it for implementation on an SMP architecture multi-processor compute-cluster. Extensive tests of the distributed version of breadth-first algorithm show a promising scalability.

Our non-regular measures algorithm considers the maximization of NPV. We consider the generalized case of activities with positive or negative cash flows at each time period of their being performed, as well as, at the start and end of each activity. In our analysis, we also include a component of bonus for the project to be completed earlier than its due date for up to four unit time periods. The algorithms are explained using examples. We test both these algorithms on payment schedules generated by our own problem generator using PSPLIB instances as the base problems. The results are briefly indicated in Table 1.1 below.

Table 1.1: Study of Exact Solutions to MM-RCPSP – Summary of Research Results

Multi-mode resource constrained project scheduling problem – exact solution algorithms					
Single processor results					
Problem set	*Resource type(s)*	*Measure studied*	*New algorithm(s) developed*	*Algorithm compared with*	*Brief results (fastest algorithm)*
PSPLIB set n0	Renewable	Makespan (regular measure)	Breadth-first and Best-first	Sprecher and Drexl (1998) Depth-first	Best-first, Breadth-first
PSPLIB sets j10, j12, j14, j16, j18, j20, j32	Renewable and non-renewable	Makespan (regular measure)	Breadth-first and Best-first	Sprecher and Drexl (1998) Depth-first	Best-first, Breadth-first
NPV problem sets generated using PSPLIB sets j10, j12, j14, j16, j18, j20	Renewable and non-renewable	Net present value (non-regular measure)	Breadth-first and Best-first	No exact solution algorithm to compare with	Best-first, Breadth-first
Multiprocessor results					
PSPLIB sets j10, j12, j14, j16, j18, j20, j32	Renewable and non-renewable	Makespan (regular measure)	Breadth-first	No exact solution algorithm to compare with	Faster than a single processor Breadth-first

The best-first approach outperforms the breadth-first approach on a single processor for the size of problems studied. The performance of breadth-first approach improves on multi-processor compute-cluster implementation as more processors are deployed and becomes nearly as good as the best-first approach on sixteen processors using SMP. It remains to be seen whether it outperforms the best-first monotone heuristic on even larger compute-clusters. Results obtained have been compared with CPLEX and shown. The breadth-first approach for NPV objective is also adaptable to distributed implementation on multiprocessor SMP architecture computer-cluster to solve the problems faster, though for non-regular measures multiple optimal solutions and the possibility of exact multi-objective solutions is rather rare.

2 MULTI-PROCESSOR BREADTH-FIRST ALGORITHM FOR REGULAR MEASURES

2.1 Introduction

Advances in computer architecture have enabled development of computers with multiple processors. Common desktop and laptop machines now possess multiple processor motherboards. For high performance computing requirements computer-clusters with a large number of central processing units (CPUs), each with their own arithmetic and logic unit (ALU) and substantially larger random access memory (RAM) have become available in recent years. Development of standards and programming paradigms to exploit multiple processors, such as, Message Passing Interface (MPI) and OpenMP has closely followed developments in technology. However, beyond the operating systems, relatively less development has been seen on the front of designs of algorithms and applications which exploit multiple processors. Their growth is picking up momentum and it is expected that in the coming years, as multicore systems become more affordable, matching commercial applications shall become available in the markets.

A special characteristic of our regular and non-regular measure Breadth-first algorithms is their ability to be modified for execution over multiple CPUs. In fact, the core algorithm's strategy and data structures can be gainfully deployed for solving a large number of problems of scientific and managerial interest (a few of which are suggested in the last chapter with ideas for further research). The primary interest in developing a distributed processors algorithm is many folds: (a) a problem instance can be solved faster; (b) larger problem instances can be solved in realistic time; and (c) the benefit of exact multiple objective optimization solutions can be sought, as compared to currently available practices such as MCDM or Pareto bound analysis, which sacrifice the single objective optimality. To the best of our knowledge, no other methodology exists to obtain multiple exact single objective solutions to an MM-RCPSP problem, and no other methodology exists for exact multi-objective optimization, though existence of multiple single objective solutions has been recognized. There also does not exist any methodology for exploiting multiprocessor architectures for solving the MM-RCPSP and this is the gap we address with our multiprocessor implementation.

MPI is an Application Programming Interface (API) library specification, which enables communication between processors, say connected over a LAN or Internet, thus, enabling them to pass values of variables, data structures, and/or control messages to each other. A feature that differentiates it from OpenMP, is its ability to execute code on distributed processors of even different architectures. However, a drawback is the delay due to the inherent latency of communication between processors connected over an LAN or Internet.

OpenMP enables specifying compile time directives in the code which enable parallelization of loops (typically *for* loops) when the compiled code is running on a Symmetric Multi-Processors (SMP, often also called Shared Memory Processors) system. OpenMP utilizes specification of pragmas in the code as parallelization directives, with declarations of shared and private variables. These directives are used during compilation of the code to distribute the execution of a number of threads on different processors. Both, MPI and OpenMP, are supported by large

groups of researchers, academia, vendors, implementors, and users.

2.2 Multi-Processor Breadth-first Algorithms (MPBFA)

To exploit the Breadth-first algorithm's distributable structure we develop two different versions of the Breadth-first algorithm for regular measures for tests on a computer-cluster – one using MPI and another using OpenMP. The characteristics and results of experiments with these versions are discussed below.

2.2.1 MPBFA using MPI

The MPI version requires substantial passing of data between processors. In brief, the MPI version appears handicapped by: (a) the message passing latency over the communication network, and (b) the loss in advantage of the pruning rules (especially the dominance pruning rule) due to the generation of identical states by several processors.

In our first Breadth-first MPI version (BRD-MPI-1) we solve the first level at the master node, and distribute its states for further processing to separate processors connected over a 1 GBPS LAN. After developing the child states for the incoming parent state, each processor communicates the results back to the master processor and waits for the next parent state to be assigned to it. We find that even for the small problem instances in PSPLIB, the message passing latency during communication between two processors outweighs the advantage of distributed processing by multiple processors using MPI.

In an alternate implementation (BRD-MPI-2), after developing the first level at the master processor, we consider assigning each processor a state to be processed entirely up to the final level. This version results in the generation of (thousands of the) same states by various processors at interim levels. Thus, different processors end up solving same states, separately. The collective number of states processed in such an implementation sometimes is several times that of the same problem instance when solved using a single processor! No advantage gained in wall clock time in solving the problem is noted.

We also consider an alteration wherein the processors communicate the best solution so far developed by anyone to all other processors, for use in pruning inferior partial solutions. However, by the time any processor reaches a solution state, other processors have also advanced substantially in their partitions of the search tree, and very little benefit appears to be gained. We hypothesize that an MPI version may be suitable for extremely large problem instances, where the message passing latency over a communication network is offset by the large computational time needed for solving the problem instances using a single processor. We hope the challenge of further research in this direction attracts more researchers. Our tests using OpenMP yield better results. Hybrid OpenMP and MPI versions, too, may prove beneficial for further research.

2.2.2 MPBFA using OpenMP

We next develop an implementation using OpenMP and test it over multiple processors in SMP (Shared Memory Processors) architecture. Typically, OpenMP is used for parallelizing either processing of data (a simple example being finding the average of five hundred numbers using multiple processors) or parallelization of different tasks (for example, modern operating systems).

In our distributed implementation of the Breadth-first algorithm, each processor is performing a large number of tasks in a sequence, some of which, for example, are: (a) computing the earliest finish time in a partial schedule which is provided for processing (in the real implementation this is actually done while a state is being built); (b) determining the activities which were in progress and have completed at this decision point; (c) building the new list of candidate activities; (d) building the resource satisfying sets (RSS); ... and so on. Hence, our algorithm is very different from "data parallelism" in OpenMP. Michael J. Quinn explains "data parallelism" and "task parallelism" in the book *Parallel Programming in C with OpenMP and MPI* (McGraw Hill, 2004).

Further, each of the processors in our algorithm is performing a series of tasks which are identical for all processors. Thus, it is also different from OpenMP "task parallelism", where essentially processors are performing separate or different tasks. Only on a very broad scale, features of our algorithm may be called similar to task parallelism (for example, each processor may be said to be performing the key task of: "given a partial schedule as a parent schedule, generate all its partial child schedules"). However, our implementation is substantially different from both, "data parallelism" and "task parallelism". As discussed above, our algorithm falls into a new category which characterizes "**logic parallelism**". There is a complex logically arranged series of tasks that each processor has to perform. This task involves using data from the key problem instance (global constants), as well as, results of processing from some (any) processor earlier (i.e. the partial schedules). Each processor appends its results to the search tree being generated in shared memory. These results are accessible to all other processors to use. Each processor also picks up (or is assigned) its next job from this search tree. While the iterations performed on the same problem instance could vary substantially every time, the core set of tasks and their sequence of assignment (to whichever is the next available processor) remains the same. Note that even in a run of the same problem instance on the same cluster of processors using the same algorithm, it is not necessary that the processors solve the same set of parts of the problem instance which they solved in the previous run! Also, that if a single objective optimal solution is pursued, a different optimal solution may be yielded in two different runs (provided the problem instance has multiple single objective optimal solutions)! In exact multi-objective solution's pursuit, if multiple exact multi-objective solutions exist for a problem instance, then in different runs any one of them may be yielded as the final solution!

At first we test a "master-worker" work distribution version of the Breadth-first algorithm, in which a single processor (the master processor) assigns tasks to all the remaining available worker processors. In this implementation, each processor produces and appends its results to a common tree which is being generated in the breadth-first manner, from where it is also assigned

its next task (by the master processor). Each assigned task needs substantial computational time for being processed keeping the worker processors busy. We find that in such a distribution, the "master" processor remains underutilized, as it is idle after workers have been assigned their current tasks till any-one submits its results.

Hence, we develop another scheme in which all processors work at an equivalent level (as team members). This implementation yields better results. We keep all processors in a team as equal members (working with the same rank and sharing work), picking up their "next task" from a common pool of available tasks and also appending their results to it (making sure that only one processor is appending its results at a time). This common pool is actually the state tree generated by our algorithm, and being Breadth-first in nature, simplifies the selection of subsequent tasks (from the current level being processed) and appending the results generated (which is always at least one level further). Before a new level is started, the level before it is completely processed.

Note that this leads to the intermittent processor idling (consonant with Amdahl's law, Amdahl (1967) and Gustafson (1988)) as in OpenMP's parallelization pragma the threads parallelize and close together. Thus, only when the last processor has appended its results to the main tree, all processors move to the next set of tasks. For example, when all states at a level are processed, the processors move to the next level. However, within a level, we organize the data structures in a manner that processors keep completing their task and collecting the next task, and minimal wait is encountered. An explanation of our data structure implementation and its advantages for distributed processing is now necessary to elaborate how the above drawback is partially overcome using linked lists (the parallelization of linked lists using OpenMP proves to be a little tricky!).

For the conservation of memory, our data structures organize the search tree (using linked lists) such that common information for multiple states is shared. This aids in faster implementation of our pruning rules too. A state is completely identified by a set of three data structures: two sets of the data structures are shared with several other states, and a third data structure is exclusively its own. The first shared data structure (which we call OVER) preserves the information associated with the activities completed, non-renewable resources consumed, and the candidate activities. It is appended to a linked list of another data structure (which we call PROG) containing information pertaining to its associated (feasible) sets of activities and their modes, which are in progress. These two shared data structures may belong to several states at once (in even very small problem instances of twelve or fourteen activities, they may be shared by hundreds of states). The information pertaining to each state which exclusively identifies it from other states, primarily the start times of activities scheduled, is contained in the third data structure (which we call STET). A few supporting data structures are also deployed which we avoid discussing here for brevity. As thousands of instances of each of these data structures are generated for even small problem instances, there is immense scope for parallelization provided a distributable scheme to generate the search tree can be designed. The elements of design in Breadth-first algorithm possess this characteristic.

Briefly, the way we parallelize the while loops in our algorithm is as below:

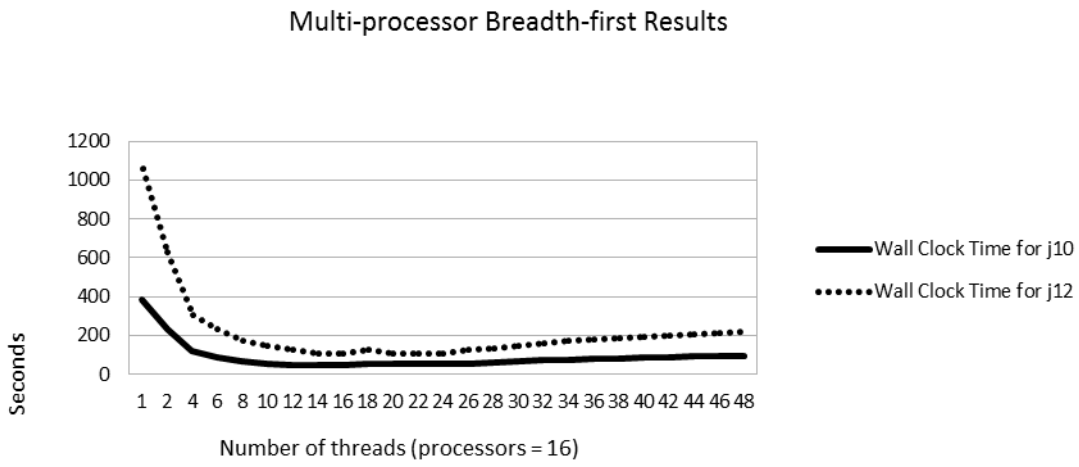

Figure 2.1: Wall Clock Time in Seconds for Entire Problem Set with Number of Threads

(a) Declare necessary thread private variables at global scope using `#pragma omp threadprivate(...)`.

(b) Set maximum number of threads using `omp_set_num_threads()` as desired for the experiment(s).

(c) Deploy parallelization directives for the while loop to be parallelized (note that our main algorithm incorporates a for loop with three levels of embedded while loops, of which we parallelize only the innermost while loop):

```
while () {...
#pragma omp parallel num_threads(NumThreads) {
#pragma omp single firstprivate(parStet) nowait {
#pragma omp task {...
}}} ...}
```

Thus, our algorithm enables the multiple processors to pick up separate nodes from the search tree, process it, and append the resultant nodes further in the search tree. By keeping a limitation on the numbers of processors and threads solving a problem in various versions, we study the ability of multiple processors to solve the MM-RCPSP problems. A critical part of our implementation involves the addition of new child states generated by a processor by comparing them to already existing child states in the search tree. At this stage, if two processors are simultaneously comparing and appending their child states to the tree, while also removing the dominated partial solution states already existing in the tree, one may cause the other to loose the trace of the last state that the processor was comparing its new child states with. Hence, in our implementation, we enable the appending of new child states for only a single processor at any given time. More research is needed to enable simultaneous appending of new child states by more than one processor at a time, for example, when the two processors are to append their new child states at different levels, which may improve the time taken to solve a problem instance. As different processors reach the stage of appending their new child states to the search tree at different times, this wait is not likely to be high, but it helps to avoid a crash of the run. We

conduct computational experiments using the PSPLIB problem sets using up to sixteen processors and up to forty-eight threads. The next section discusses the results of our OpenMP multi-processor implementation.

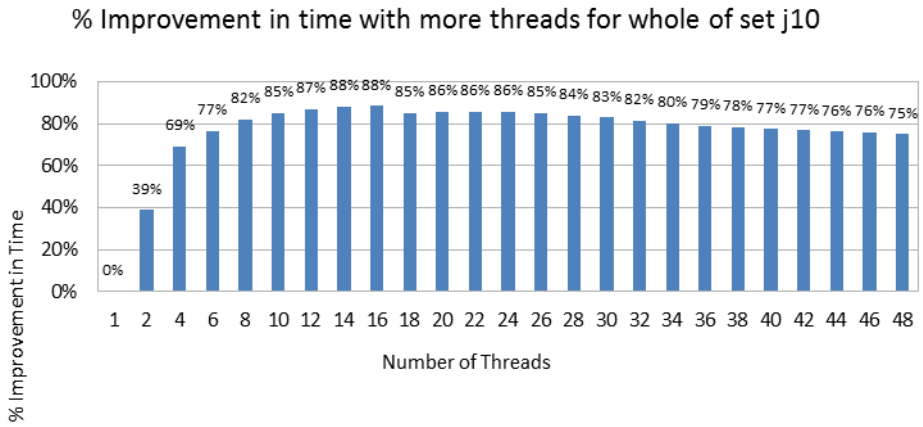

Figure 2.2: Percent Improvement in Wall Clock Time for Complete Problem Set j10 on Increasing Number of Threads

2.3 Experimental Observations

We conduct detailed tests on the j10 and j12 problem sets from the PSPLIB. These problem sets are solved using one thread, two threads, and further up to forty eight threads using all combinations at an increment of two threads. As the number of processors available on one compute-cluster we use is sixteen, the number of threads per processor is one with sixteen threads and more than one beyond it, with three threads per processor at a total of forty eight threads. The improvement in real time or wall clock time as the number of threads is increased, along with total CPU time consumed by all threads is shown in graphs below. As the number of threads per processor rises above one, the performance starts deteriorating (note that only one Arithmetic Logic Unit (ALU) is available per CPU). The problem sets j10 and j12 are solved in 298.9 and 971.42 seconds, respectively, on a single processor, while they are solved in 44.6 and 108.06 seconds using sixteen threads with sixteen CPUs, which is faster than CPLEX (on the dektop machine). The problem set j14, too, was solved by breadth using sixteen threads with sixteen CPUs in 759.6 seconds compared to 4511.8 seconds by CPLEX. Our Breadth-first approach is a tree search procedure with branch (but not bound) and pruning rules. Though CPLEX solves larger problems faster, it does not yield exact multi-objective solution like our Breadth-first algorithm.

We also test run larger PSPLIB sets j14 and j16 with one, eight and sixteen threads and the results are similar. In set j16, five problem instances are not solved by a single thread even in two

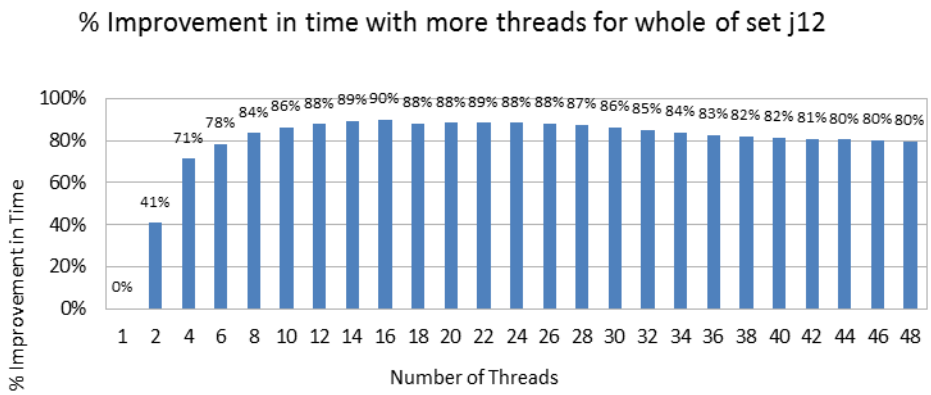

Figure 2.3: Percent Improvement in Wall Clock Time for Complete Problem Set j12 on Increasing Numbers of Threads

hours of run time each, however, the multi-processor implementation overcomes this limitation. Noticeably, the multi-processor implementation even with sixteen threads on as many processors is unable to solve the problem instances faster than our single thread Best-first algorithm! Perhaps, using even more processors this may be possible, and is fruitful due to the exact multi-objective solutions delivered by the Breadth-first algorithm. Figure 2.1 to Figure 2.3 present the results of our observations.

The improvement in real time taken to solve is observed only up to one thread per processor (note that there is only one ALU per processor), as seen in Figure 2.1 to Figure 2.3 for problem sets j10 and j12. Our experiments have been carried on a compute-cluster with sixteen

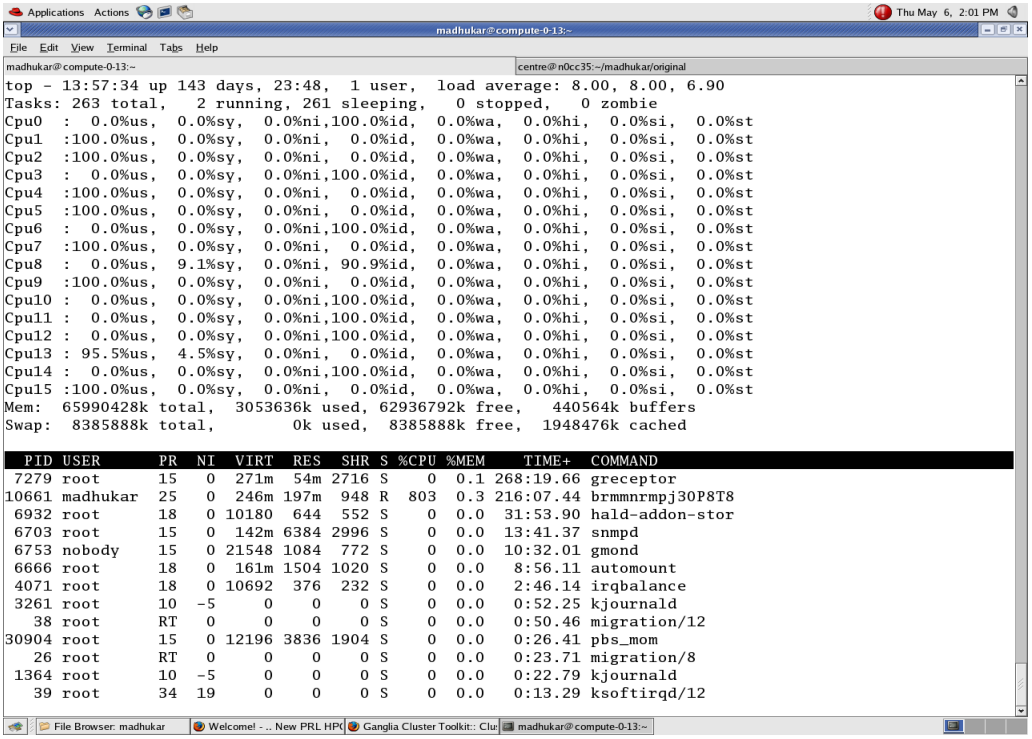

Figure 2.4: Screen Shot: Multi-processor Breadth-first Algorithm on Eight CPUs

processors. The fastest solutions in real time are obtained with one thread per processor on all sixteen processors. Time is lost due to switching of threads by processors when the number of threads rises above the number of processors. Hence there is a deterioration in the real time performance after the number of threads is increased above the number of processors (i.e. sixteen).

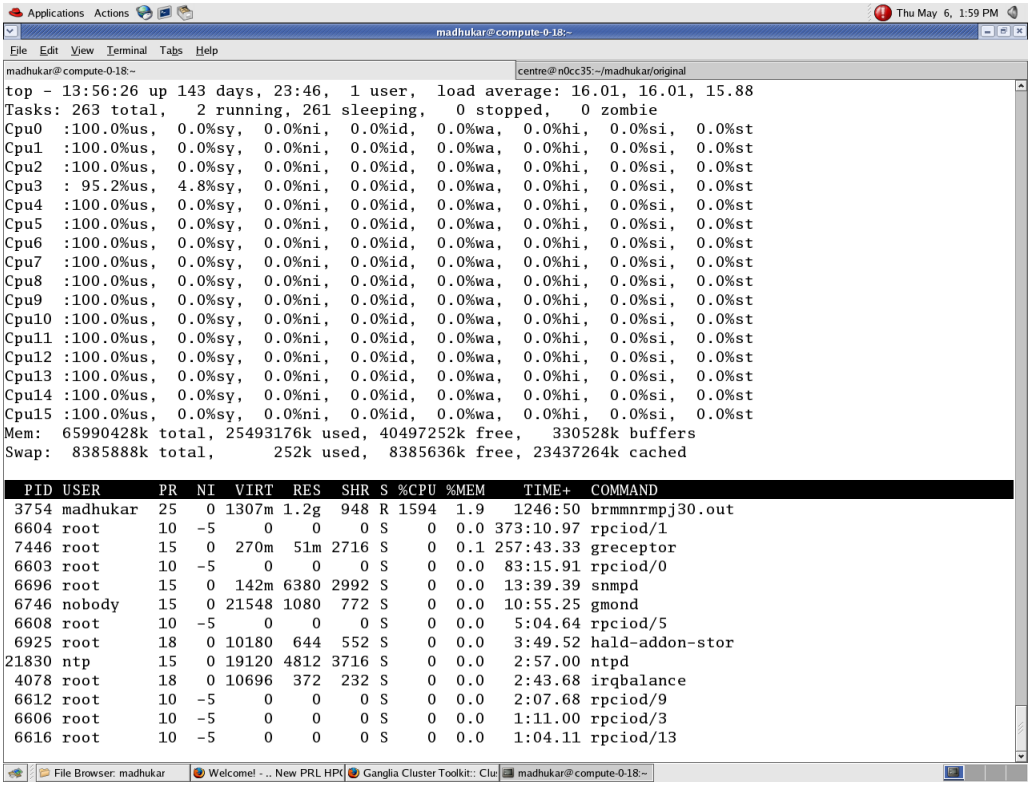

Figure 2.5: Screen Shot: Multi-processor Breadth-first Algorithm on Sixteen CPUs

During the runs we observe some more characteristics of multicore operating systems. A screen shot of the processors' usage (shown below) demonstrates some behavior of the processors. A small element of processing power is consumed by the (non-GUI thin version of the) operating system processes too. At the times when a processor is assigned no task by the algorithm (and is waiting for another processor(s) to complete), it is engaged in the default (user or system) idle process. This is visible in the screen shots shown in Figure 2.4 and Figure 2.5.

In Figure 2.4, the task with process ID 10661 (COMMAND column showing 'brmmnrmpj30P8T8') running on eight processors is our algorithm solving a problem instance from the PSPLIB j30 set using eight threads. In Figure 2.5 another j30 set problem instance is under process using sixteen processors and sixteen threads. The %CPU column in system statistics columns shows the CPU utilization over past few second(s) (in this case one second) and %MEM shows the percentage of system RAM used by the process. The TIME+ column shows the

cumulative time consumed by all processors (all threads) together and letter R in column S shows the sleeping or running status of the job. Note that when all sixteen processors are engaged, for its own tasks the operating system may wait till a processor is idle or interrupt a processor depending on the system tasks' priority. We execute our experiment runs at normal user priority in Red Hat Linux Enterprise 5 environment (which is twenty five). A particular, though small, percentage of CPU time is regularly needed for the operating system tasks. This is much higher for operating systems with GUI, though compute-clusters are mounted with relatively thinner versions of the operating system being primarily designed for computational purposes. A screen shot of all the sixteen processors engaged by the algorithm is shown in Figure 2.5 below with the task number 3754 of our interest.
Some more interesting observations and analysis from the parallel processors runs are as follows:

(a) As the number of processors is increased, the problem instances are solved faster. However, the incremental benefit of adding a processor gradually diminishes (diminishing return on investment, as indicated by Amdahl's law). This is due to one or more of several reasons: (i) Different (parent) states need different processing times. (ii) The parent states assigned to threads are in the order they were earlier appended in the tree. Any one of them could be longest for processing. (iii) The threads in a parallel OpenMP construct are required to culminate and terminate together, else some of them idle till the last one completes its assigned task. Thus, the single latest finishing thread determines when the next set of tasks can be picked up by all threads, and to that time other threads need to be idle.

(b) The order of generation and addition of states to the search tree may become different in different runs on the same problem instance. As different partial solutions, i.e. parent states, are processed simultaneously by different processors, the order in which they append their results to the search tree is different from that in a single processor case. Hence, the states which dominate other states in a single thread implementation, as they were generated first, may sometimes themselves be dominated in a multi-processors implementation by other states which came to be generated first!

(c) The above phenomenon leads to another interesting observation. The final solution yielded for a problem instance may be different in different runs of the same problem instance! If single objective optimization is pursued, even for the same problem instance in different runs, different optimal solutions may be yielded, depending up on which processor appended its results to the tree first, which later dominates some other partial schedules! In exact multi-objective solutions this situation arises if multiple exact multi-objective solutions for the problem instance exist and during different runs, different partial solutions were appended to the search tree first.

(d) Increasing the number of threads beyond one per processor appears to be of no advantage as the algorithm requires intensive use of the ALU (for mathematical and logical operations), of which only one per processor is available. As soon as a processor is assigned two or more tasks, the switching of processing from one thread to another wastes much more time than is gained by having added one more thread. The situation is worse for three threads assigned to each available processor. This appears to be the reason for the rise in the time taken to solve the problem instances by increasing the number of threads to more than the number of processors.

The attraction of optimal solutions to real life size problems has supported research on project scheduling (and many other families of similar NP-hard problems) since several decades. Exact solutions have, however, remained elusive due to several bottlenecks, such as, speed of processors and the size of memory. Advances in past few years have raised both of these limits, though some constraints are now being faced, such as in increasing the CPU clock speed, and it may be some time before these are overcome. The available memory address space in a computing system with a 32 bit architecture was barely 4 GB (i.e. 2^32), which for 64 bit architecture has risen to 4 G times 4 GB. Multi-processor mother-boards and operating systems have driven out erstwhile single processor computers from the market almost entirely. Hence, the platform is now ripe for development of more and more distributable algorithms. Our implementation of the Breadth-first algorithm over multiple processors is a small step in this direction.

2.4 Summary

The Breadth-first regular measures algorithm is readily extendable to implementation over multiple processors. It is then able to solve problem instances faster or solve larger problems in a reasonable clock time. Many different design schemes for such implementations are possible. In this chapter we have described our OpenMP multiple processor algorithm including its comparison with MPI versions and presented the results of our experiments. While the multiple processor Breadth-first implementation is unable to solve the problem instances faster than the single processor Best-first algorithm, even when using up to sixteen processors, it is attractive as it yields exact multi-objective solutions commensurate with the needs of managers. A few alterations to the data structure designs and implementation on larger number of clusters may be able to yield solutions to real life problems of up to a few hundred activities, and is worthy of further research (we elaborate this in the last chapter).

The development on the front of multi-processor algorithms has been somewhat slower, and is on road to gain momentum. Distributable algorithms and applications which exploit the availability of multiple processors and large amounts of memory, are now expected to attract more attention in the research domain. It is anticipated that solution methodologies to solve larger problems would become available through these directions of research in many more problem domains. We hope the results of our research will prove to be an additional step in this direction and stimulate more attention to distributed computing algorithms in the future.

3 CONCLUSION

3.1 Summary of Results

In this research we have explored the breadth-first and best-first tree-search algorithms in solving the multi-mode resource constrained project scheduling problem (MM-RCPSP). Further, we have developed the breadth-first algorithm for implementation on a compute-cluster using OpenMP and deploying logic parallelism as compared to normal data or task parallelism. The algorithms are designed and tested for regular and non-regular performance measures and described in Chapters 2 and 3 respectively. Experimental comparison with the best-known tree-

search algorithm, the depth-first algorithm of Sprecher and Drexl (1998) is presented. Both the algorithms solve the PSPLIB problems much faster when tested on a single processor in a cluster with Quad-Core AMD® Opteron® Processor 8360 SE, 2511.578 MHz running a thin (non-GUI) version of Red Hat Enterprise Linux 5.

The breadth-first algorithm reveals *multiple optimal solutions* for regular measures, and by re-traversing the last level of the solution space tree, an *exact multi-objective optimal solution* can be found to minimize the non-renewable resources consumed for several possible multi-objective goals. Simple modifications can easily enable search for an exact multi-objective optimal solution which optimizes consumption of non-renewable resources (a) by their given priority, (b) by their given unit costs, or (c) by their relative weights or costs. Supporting examples have been used to describe the algorithms in detail and proof of optimality of the algorithm provided where necessary.

Being extremely difficult problem sets to solve, no exact solution approaches for non-regular measures for MM-RCPSP exist in literature. Standard problem sets for non-regular measures are also missing from libraries of such problems. We test the algorithms for non-regular performance measures on payment schedules generated using our own generator, where the PSPLIB problem instances are used as the base problems. The breadth-first algorithm for non-regular measures is also extensible to multi-processor SMP implementation. As the difference in the performance of best-first for non-regular measures vis-a-vis breadth-first is not as large as in regular measures, the multi-processor implementation of breadth-first for non-regular measures may yield superior performance to best-first with only a few processors.

Chapter 4 discusses the extension of the breadth-first algorithm for implementation on a distributed computing environment (using a computing-cluster) with OpenMP and results of its extensive tests with up to sixteen processors in SMP (shared memory processors) architecture. The new implementation demonstrates promising down scaling in real time taken to solve the problem instances enabling larger problem instances to be solved.

3.2 Extension to other problems

The breadth-first and best-first, as well as, the distributed breadth-first approach can be extended to several other problems of interest, such as, design and configuration problems (for example, sheet cutting, vessel loading, etc.), bin packing and partitioning problems, combinatorial mathematics problems (such as, perfect square placement and number partitioning), Bioinformatics (such as, DNA word design), etc. These are all known to be very tough families of problems to solve, and lack applications of exact tree-search algorithms. We refer the interested reader to The TPTP (Thousands of Problems for Theorem Provers) website (http://www.cs.miami.edu/~tptp/) which also provides links to many other problem libraries including the PSPLIB and ORLib.

3.3 Scope for Further Research

The work in this paper, especially the compute-cluster implementations, promise a mechanism for solving larger problem instances, both, faster and yielding multiple optimal solutions, among

which exact multi-objective solutions can be found. The computational requirements for exact solutions to even small problem instances are extremely large. We envisage further research for development in the following directions as fruitful:

(a) **Shared renewable resources:** An interesting model of the problem to study would be the consideration of renewable resources which are shared among activities, as is typically the case in real life problems. Consider for example, resources such as a common pool vehicle or a photocopying machine in an office. The estimation of relative utilization of a resource by different activities (in its various modes) and a wait time if the resource is engaged may pose some difficulty.

(b) **Dynamic and stochastic problem models:** Extension of these algorithms is also possible to cases in which activities are dynamically arriving (dynamic project scheduling) where job duration becomes known only when it becomes available for processing, and to stochastic project scheduling models.

(c) **Improved pruning rules:** Identification of more powerful pruning rules to reduce the size of the search tree may enhance the overall performance of both algorithms. For example, computation of minimum requirements of non-renewable resources needed for completion of remaining activities and its comparison with the actual residuals of these non-renewable resources, may be able to reduce the size of the search tree by pruning some of its branches.

(d) **Hybrid algorithms:** Use of a good available upper bound, computed through other heuristic algorithms, could help in cutting large portions of the tree as searched by the breadth-first approach. Such an integration could yield faster algorithms and is worth exploration.

(e) **Inclusion of resource vacations/withdrawals:** An extension towards consideration of periods of resource vacations (for example a machine falling under repair for some time) or renewable resource withdrawals (for example, diversion of a renewable resource towards another project in a firm processing multiple projects) would further the algorithm towards solving problems closer to real life situations. Similarly, interim addition of renewable resources may also be considered for bringing the problem closer to real problems.

(f) **Improved data structures:** Although the memory requirements of these algorithms are modest for the standard test problem sets, these would be much larger for bigger problems, especially for the best-first algorithm. Improved data structure implementations could aid in reducing this requirement as well as in reducing computational effort involved in pruning rules, thus enabling larger problems to be solved with limited memory usage.

(g) **Hybrid OpenMP and MPI implementations:** It is noticed that MPI implementations of the algorithm over distributed processors are handicapped by the message passing latency between two processors over a network, which may be as high as 3 to 5 milliseconds in 100/1000 MBPS Ethernet networks of today. Hybrid (MPI and OpenMP) implementations of the algorithms may be able to solve larger problems, as the message passing latency would be countered by the gain in processing speed and expected resulting reduction in overall solution time. For approaching

solutions to large, real life, problems, this appears to be a promising field of research. Depth-first approach using OpenMP and/or MPI may also yield promising results.

4 REFERENCES

Alcaraz, J., C. Maroto, and R. Ruiz, 2002. A New Genetic Algorithm for the Multi-Mode Resource-Constrained Project Scheduling Problem. Dpto de Estadistica e Investigacion Operativa, Universidad Politecnica de Valencia, Spain.

Alcaraz, J., C. Maroto, and R. Ruiz, 2003a. Multi-Mode Resource-Constrained Project Scheduling Problem: An Advanced Genetic Algorithms. The Fifth Metaheuristics International Conference, Kyoto, Japan.

Alcaraz, J., C. Maroto, and R. Ruiz, 2003b. Solving the Multi-Mode Resource- Constrained Project Scheduling Problem with Genetic Algorithms. The Journal of the Operational Research Society, 54(6): 614-626.

Amdahl, G. M., 1967. Validity of the single-processor approach to achieving large scale computing capabilities. In AFIPS Conference Proceedings (Atlantic City, N.J., Apr. 18–20), 30: 483–485.

Ballestin, F., and Blanco, R., 2011. Theoretical and practical fundamentals for multi- objective optimisation in resource-constrained project scheduling problems. Computers & Operations Research, 38(1): 51-62.

Behzadian, M., Kazemzadeh, R. B., Albadvi, A., and Aghdasi, M., 2010. PROMETHEE: A comprehensive literature review on methodologies and applications. European Journal of Operational Research, 200(1): 198-215.

Berman, E. B., 1964. Resource Allocation in a Pert Network under Continuous Activity Time-Cost Functions. Management Science, 10(4): 734-745.

Bouleimen, K. and H. Lecocq, 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.

Buddhakulsomsiri, J. and D. Kim (2006). Properties of multi-mode resource-constrained project scheduling problems with resource vacations and activity splitting. European Journal of Operational Research, 175(1): 279-295.

Buddhakulsomsiri, J. and D. Kim, 2007. Priority rule-based heuristic for multi-mode resource-constrained project scheduling problems with resource vacations and activity splitting. European Journal of Operational Research, 178(2): 374-390.

Chyu, C.C., Chen, A.H.L., and Lin, X.H., 2005. A Hybrid Ant Colony Approach to Multi- mode

Resource-Constrained Project Scheduling Problems with non-renewable types. *1st* International Conference on Operations and Supply Chain Management.

Daniels, R. L. and J. B. Mazzola, 1994. Flow Shop Scheduling with Resource Flexibility. Operations Research, 42(3): 504-522.

Dayanand, N., and Padman, R., 1997. On modelling payments in projects. Journal of the Operational Research Society, 48(9): 906-918.

De Reyck, B., E. Demeulemeester, and W. Herroelen, 1998. Local search methods for the discrete time/resource trade-off problem in project networks. Naval Research Logistics, 45(6): 553-578.

De Reyck, B. and W. Herroelen, 1999. The multi-mode resource-constrained project scheduling problem with generalized precedence relations. European Journal of Operational Research, 119(2): 538-556.

Demeulemeester, E., B. Dodin, and W. Herroelen, 1993. A random activity network generator. Operations Research, 41(5): 972-980.

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

Drexl, A. and J. Gruenewald, 1993. Non-Preemptive Multi-Mode Resource Constrained Project Scheduling. IIE Transactions, 25(5): 74-81.

Drexl, A., R. Nissen, J.F. Patterson, and F. Salewski, 2000. ProGen/JIx–An instance generator for resource-constrained project scheduling problems with partially renewable resources and further extensions. European Journal of Operational Research, 125(1): 59-72.

Erenguc, S. S., T. Ahn, and D.G. Conway, 2001. The resource constrained project scheduling problem with multiple crashable modes: An exact solution method. Naval Research Logistics, 48(2): 107-127.

Gustafson, J. L., 1988. Reevaluating Amdahl's law. Communications of the ACM, 31(5): 532-533.

Hartmann, S., 2001. Project Scheduling with Multiple Modes: A Genetic Algorithm. Annals of Operations Research, 102(1): 111-135.

Hartmann, S. and A. Drexl, 1998. Project Scheduling with Multiple Modes: A Comparison of Exact Algorithms. Networks, 32: 283-297.

He, Z. and Y. Xu, 2008. Multi-mode project payment scheduling problems with bonus– penalty

structure. European Journal of Operational Research, 189(3): 1191-1207.

Heilmann, R., 2003. A Branch-and-bound Procedure for the Multi-mode Resource- constrained Project Scheduling Problem with Minimum and Maximum Time Lags. European Journal of Operational Research, 144(2): 348-365.

Heilmann, R., 2004. A Priority Rule Method for the Multi-Mode Project Scheduling Problem MRCPSP/max. Retrieved May 23, 2008 from: http://www.mathematik.uni-osnabrueck.de/research/OR/pms2000/abstract/heilmann127.de.ps

Herroelen, W. and B. D. Reyck, 1999. Phase Transitions in Project Scheduling. The Journal of the Operational Research Society, 50(2): 148-156.

Icmeli, O., and Erenguc, S. S., 1996. A branch and bound procedure for the resourceconstrained project scheduling problem with discounted cash flows. Management Science, 42(10): 1395-1408.

Józefowska, J., M. Mika, R. Rózycki, G. Waligóra, and J. Weglarz, 1999. Solving the Multi-Mode Resource-Constrained Project Scheduling Problem by Simulated Annealing. Seventh International Workshop on Project Management and Scheduling.

Józefowska, J., M. Mika, R. Rózycki, G. Waligóra, and J. Weglarz, 2001. Simulated Annealing for Multi-Mode Resource-Constrained Project Scheduling. Annals of Operations Research, 102(1): 137-155.

Kavalak, N., 2005. Client-Contractor Bargaining Problem in the Context of Multi-Mode Project Scheduling with Limited Resources. Masters Thesis, Sabanci University.

Klingman, D., A. Napier, J. Stutz, 1974. NETGEN: A Program for Generating Large Scale Capacitated Assignment, Transportation, and Minimum Cost Flow Network Problems. Management Science, 20(5): 814-821.

Kolisch, R. and A. Drexl, 1997. Local search for nonpreemptive multi-mode resource-constrained project scheduling. IIE Transactions, 29(11): 987-999.

Kolisch, R. and A. Sprecher, 1996. PSPLIB - A project scheduling problem library : OR Software - ORSEP Operations Research Software Exchange Program. European Journal of Operational Research, 96(1): 205-216.

Kolisch, R., A. Sprecher, A. Drexl, 1995. Characterization and Generation of a General Class of Resource-Constrained Project Scheduling Problems. Management Science, 41(10): 1693-1703.

Leachman, R., 1980. Multiple Resource Leveling in Construction Systems through Variation of

Activity Intensities. Retrieved May 23, 2008 from: http://stinet.dtic.mil/oai/oaiverb=getRecord&metadataPrefix=html&identifier=ADA089988

Leachman, R., A. Dincerler, and S. Kim, 1990. Resource-Constrained Scheduling of Projects with Variable-Intensity Activities. IIE Transactions, 22(1): 31-40.

Lova, A., Tormos, P., and F. Barber, 2006. Multi-Mode Resource Constrained Project Scheduling-Scheduling, Priority rules and mode selection rules. Artificial Intelligence, 10(30): 69-86.

Mika, M., Waligóra, G., and J. Weglarz, 2008. Tabu search for multi-mode resource- constrained project scheduling with schedule-dependent setup times. European Journal of Operational Research, 187(3): 1238-1250.

Mori, M. and C. Tseng, 1997. A genetic algorithm for multi-mode resource constrained project scheduling problem. European Journal of Operational Research, 100(1): 134-141.

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

Nazareth T., and Bhattacharya S., 1993. A breadth-first search scheme to solve the resource constrained project scheduling problem, Proc IFIP 93 on Systems Modelling and Optimizaiton, Compeigne, France, July 5-9, 641-644.

Özdamar, L. and H. Dündar, 1997. A flexible heuristic for a multi-mode capital constrained project scheduling problem with probabilistic cash inflows. Computers and Operations Research, 24(12): 1187-1200.

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

Peteghem, V. V. and M. Vanhoucke, 2008. A Genetic Algorithm for the Multi-Mode Resource Constrained Project Scheduling Problem. Working Papers of Faculty of Economics and Business Administration, Ghent University Ghent University, Belgium. Retrieved May 23, 2008 from: http://www.FEB.UGent.be/fac/research/WP/Papers/wp_08_494.pdf

Prashant Reddy, J., S. Kumanan, O.V. Krishnaiah Chetty, 2001. Application of Petri Nets and a Genetic Algorithm to Multi-Mode Multi-Resource Constrained Project Scheduling. The International Journal of Advanced Manufacturing Technology, 17(4): 305-314.

Pulat, P. S. and S. J. Horn, 1996. Time-resource trade off problem [project scheduling].Engineering Management, IEEE Transactions on, 43(4): 411-417.

Quinn, Michael J. Parallel Programming in C with OpenMP and MPI. McGraw Hill, 2004.

Sabzehparvar, M. and S. Seyed-Hosseini, 2008. A mathematical model for the multi- mode resource-constrained project scheduling problem with mode dependent time lags. The Journal of Supercomputing, 44(3): 257-273.

Schirmer, A., 1996. New Insights on the Complexity of Resource-Constrained Project Scheduling-Two Cases of Multi-Mode Scheduling. Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, 391. Retrieved May 23, 2008 from: http://citeseer.ist.psu.edu/46793.html

Shan, M., Q. Hong, and W. Juan, 2007. Multi-Mode Multi-Project Scheduling Problem for Mould Production in MC Enterprise. Wireless Communications, Networking and Mobile Computing, 2007 (WiCom 2007), International Conference on: 5311- 5315.

Shan, M., J. Wu, and D. Peng, 2007. Particle Swarm and Ant Colony Algorithms Hybridized for Multi-Mode Resource-constrained Project Scheduling Problem with Minimum Time Lag. Wireless Communications, Networking and Mobile Computing, 2007 (WiCom 2007), International Conference on: 5893-5897.

Shukla, M. and Verma, S, 2014, Breadth-first and Best-first Exact Procedures for Regular Measures of the Multi-mode RCPSP. Working Paper, IIM Ahmedabad, W.P. No. 2014-10-4.

Shukla, M. and Verma, S, 2015, Breadth-first and Best-first Exact Procedures for Regular Measures of the Multi-mode RCPSP. Working Paper, IIM Ahmedabad, W.P. No. 2015-03-06.

Sprecher, A., 2000. Scheduling Resource-Constrained Projects Competitively at Modest Memory Requirements. Management Science, 46(5): 710-723.

Sprecher, A. and A. Drexl, 1998. Solving Multi-mode resource-constrained project scheduling by a simple, general and powerful sequencing algorithm. European Journal of Operational Research, 107(2): 431-450.

Sprecher, A., Kolisch, R., and A. Drexl, 1995. Semi-active, active, and non-delayschedules for the resource-constrained project scheduling problem. European Journal of Operational Research, 80(1): 94-102.

Sprecher, A., Hartmann, S., and A. Drexl, 1997. An exact algorithm for project scheduling with multiple modes. OR Spectrum, 19(3): 195-203.

Sunde, L. and S. Lichtenberg, 1995. Net-present-value cost/time tradeoff. International Journal of Project Management, 13(1): 45-49.

Talbot, F. B., 1982. Resource-Constrained Project Scheduling with Time-Resource Trade offs: The Non-preemptive Case. Management Science, 28(10): 1197-1210.

Ulusoy, G., S. S. Funda, and S. Sahin, 2001. Four Payment Models for the Multi-Mode Resource Constrained Project Scheduling Problem with Discounted Cash Flows. Annals of Operations Research, 102(1-4).

Ulusoy, G. and S. Sahin, 1998. Three Different Payment Programs for the Multi-Mode Resource Constrained Project Scheduling Problem with Discounted Cash Flows: A Genetic Algorithm Approach. Retrieved May 23, 2008 from: http://www.mathematik.uni-osnabrueck.de/research/OR/pms2000/abstract/ulusoy90.tr.ps

Vanhoucke, M., Demeulemeester, E., and W. Herroelen, 2001. On maximizing the net present value of a project under renewable resource constraints. Management Science, 47(8): 1113-1121.

Dhavale, N. P., Verma, S., and A. Bagchi, 2003. Scheduling Partially Ordered Jobs Under Resource Constraints To Optimize Non-Regular Performance Measures. Working Paper No. 2003-07-03, IIM Ahmedabad.

Voss, S. and A. Witt, 2007. Hybrid flow shop scheduling as a multi-mode multi-project scheduling problem with batching requirements: A real-world application. International Journal of Production Economics, 105(2): 445-458.

Waligóra, G., 2008. Discrete-continuous project scheduling with discounted cash flows-A tabu search approach. Computers & Operations Research, 35(7): 2141-2153.

Weglarz, J., Józefowska, J., Mika, M., and Waligóra, G., 2010. Project scheduling with finite or infinite number of activity processing modes-a survey. European Journal of Operational Research (In press).