Intelligent Scheduling of Medical Procedures

by

Yang Sui

A thesis

presented to the University of Waterloo

in fulfillment of the

thesis requirement for the degree of

Master of Applied Science

in

Mechanical Engineering

Waterloo, Ontario, Canada, 2009

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

In the Canadian universal healthcare system, public access to care is not limited by monetary or social economic factors. Rather, waiting time is the dominant factor limiting public access to healthcare. Excessive waiting lowers quality of life while waiting, and worsening of condition during the delay, which could lower the effectiveness of the planned operation. Excessive waiting has also been shown to carry economic cost.

At the core of the wait time problem is a resource scheduling and management issue. The scheduling of medical procedures is a complex and difficult task. The goal of research in this thesis is to develop the foundation models and algorithms for a resource optimization system. Such a system will help healthcare administrators intelligently schedule procedures to optimize resource utilization, identify bottlenecks and reduce patient wait times.

This thesis develops a novel framework, the MPSP model, to model medical procedures. The MPSP model is designed to be general and versatile to model a variety of different procedures. The specific procedure modeled in detail in this thesis is the haemodialysis procedure. Solving the MPSP model exactly to obtain guaranteed optimal solutions is computationally expensive and not practical for real-time scheduling. A fast, high quality evolutionary heuristic, gMASH, is developed to quickly solve large problems. The MPSP model and the gMASH heuristic form a foundation for an intelligent medical procedures scheduling and optimization system.

Acknowledgements

I would like to express gratitude to my supervisor Professor William Melek first of all for the opportunity to stand at a frontier of innovation. Secondly, I thank Professor Melek for his constant guidance, encouragement, support, and patience throughout my research.

I am grateful to Dr. Joseph Kurian and Alpha Laboratories for providing me with financial support, opportunity to participate in research conference, and access to invaluable medical knowledge and expertise. Alpha Laboratories fully supported my academic research and will continue to support me into my professional career. Much of my research continues to be strongly motivated by Dr. Kurian's inspirational vision for the Canadian healthcare system.

I thank Dr. Alexander Glazunov for freeing my mind to view the world in new, interesting and enlightened ways.

I thank my colleagues and friends for their support and occasional comic relief. Their insatiable thirst for knowledge help fuel my own passion.

Finally, I am forever grateful to my family: my mother and Ken for their unyielding support, encouragement, advice, and patience, without which, this work would not be possible.

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

1.1 Motivation

The challenge to any healthcare system is providing sufficient access to services to those that need care. In the Canadian universal healthcare system, public access to care is not limited by monetary or social economic factors. Rather, waiting time is the dominant factor limiting public access to healthcare. Waiting for care is a fact of life. No country could provide enough resources to its healthcare system to promptly meet all demand. Even countries with significant private healthcare funding experience wait time problems. [1] The wait time problem is especially common among countries with universal healthcare as discovered in a 2003 study by the Organization for Economic Cooperation and Development (OECD). In the OECD study, Canada's wait time problem was found to have worsened between 1998 and 2001. [2]

Excessive waiting has obvious adverse impact on health of the waiting patient. Adverse impact can include lower quality of life while waiting, and worsening of condition during the delay, which could lower the effectiveness of the planned operation. The impact of excessive waiting was even examined in an unprecedented case before the Supreme Court of Canada of Dr. Jacques Chaoulli and George Zeliotis v. Attorney General of Quebec and Attorney General of Canada. All justices agreed that lengthy waits for care put patients at increased risk of suffering and death and that this violated the first part of Section 7 of the Canadian Charter which grants everyone the right to life, liberty and security of the person. [3] The Supreme Court of Canada essentially granted Canadians constitutional right of timely access to health services.

Aside from the obvious adverse impact on health, excessive waiting also carries significant economic cost. In 2008 the Canadian Medical Association commissioned a study of economic impact of excessive waiting in four priority areas: joint replacement surgery, cataract surgery, CABG surgery, and MRI exam. The study estimates that in 2007, the cumulative economic cost of excessive waiting in these four areas amounted to $14.8 billion. This reduction in economic activity consequently lowered federal and

provincial government revenues by $4.4 billion. [4] The authours of this study claim that even this astonishing cost estimate was only a conservative one.

Canadian provincial governments have policies and regulations that discourage private healthcare, leaving the public system as the only choice for Canadians. [5] Systematic inefficiency, at least in the public eye, has become the status quo. Canadians can only appeal to their government for healthcare improvement. Provincial governments have initiatives in place to address the wait time problem. In Ontario for example, the Wait Time Strategy (WTS) initiative is aiming to set targets and reduce wait times in five priority areas. Since its launch in 2004, the WTS initiative has indeed lowered wait times in those five key areas. [6] Unfortunately, procedures outside of the WTS target areas continued to see increase in waiting time. [7]

Despite best of government efforts, excessive waiting time remains a daunting challenge.

At the core of the growing waiting time problem is a resource scheduling and management issue. Universal accessibility generates high demand for healthcare. The supply side of care appears to lack resources to promptly meet demand. The Wait Time Alliance report cites dire shortage in health human resources and gaps in infrastructure as challenges to improving timely access to care. [6]

There are two approaches to dealing with supply shortage: add more resources and/or make better use of existing resources. Government initiatives thus far have focused mainly on increasing funding in certain areas. There is surprisingly little academic work being done to objectively optimize utilization of existing resources. This author believes that optimizing utilization of current resources holds the greatest promise in improving the efficiency of the Canadian healthcare system. Healthcare administrators are no doubt keen to optimize their processes and will surely benefit from intelligent tools to help them achieve their goals.

The daunting waiting time challenge of the Canadian healthcare system and the gap in research into solutions to help healthcare administrators solve the problem provides motivation for this thesis.

1.2 Problem definition

The healthcare system in Canada does not appear to make optimal use of its resources. Some equipment and facilities are stressed beyond capacity while other equipment and facilities are woefully under-utilized. In the same hospital, a few floors away from crowded emergency rooms are empty operating theatres, empty surgery prep rooms, and idle imaging devices. This disparity suggests the problem could in part be caused by a poorly designed system rather than absolute scarcity of resource. The capacity may already exist to better serve patients. However, critical resources are idled by bottlenecks. An analogy can be made to the line up to get into a hockey game. The arena has the capacity to accommodate all ticket holders. Everyone will get in. The delay is simply caused by a bottleneck at the door. [8]

Scheduling of medical procedures is a difficult task. Procedures often have complex resource requirements. The high dimensionality of the problem makes manual optimization of schedules, let alone identification of bottlenecks extremely difficult. The broad and ambitious goal of this research is to design an automated, intelligent scheduling system to help healthcare administrators optimize resource utilization. The focus is on optimizing the supply side of healthcare. An intelligent system should optimize the utility of existing resources as well as identify bottlenecks so that additional resources can be effectively allocated to relieve bottlenecks. Optimization of resource utilization will reduce cost, improve patient flow, reduce patient wait time and therefore will directly improve quality of healthcare. Healthcare administrators need sophisticated tools to help them optimize the system.

The design of an intelligent optimization system is a daunting task that is far beyond the scope of a MASc thesis. However, the complex task can be broken into smaller pieces. The first step towards designing an intelligent resource optimization system is the solving of the core medical procedures scheduling problem. The next section defines the specific scheduling problem to be solved in this thesis.

1.2.1 The Medical Procedures Scheduling Problem (MPSP)

The goal of this thesis is to solve the following *Medical Procedures Scheduling Problem (MPSP)*:

> Given a hypothetical hospital department or clinic that has *m* resources. These resources can be human, equipment, supplies, room resources etc. *n* procedures, with different requirements of the *m* resources are to be scheduled at the clinic.
>
> A) If all procedures have the same priority, how many of each procedure can be scheduled into one shift (or day, or any user defined scheduling period)?
>
> B) Solve problem A) but given procedures with different priorities.
>
> C) Solve problem A) or B) but given additional constraint that some procedures must be scheduled.
>
> D) What is the schedule that maximizes the number of procedures performed?
>
> E) What is the schedule that minimizes the total patient wait time?

Procedures considered in this problem are assumed to be deterministic in nature with known and repeatable durations. That is, the MPSP considered in this thesis is a static problem. Solving of a dynamic MPSP is reserved for future work.

The goal in solving the static MPSP is to find the best schedule that satisfies certain constraints and optimizes some objectives. The user should be given the freedom to set different objectives such as maximizing number of procedures to perform in one shift and/or minimizing patient wait time. Maximizing the number of procedures to perform represents optimization of resource utilization and improvement of patient flow. The patient wait time referred to in problem E) is the patient wait time on the day that he/she is scheduled for a procedure. Minimizing patient wait time on the day of the procedure should help give patients the best experience possible.

This MPSP provides the framework for developing the core scheduling infrastructure of an intelligent scheduling system.

1.3 Literature review

1.3.1 Classical optimization

Classical methods of optimization use differential calculus to find optimum points on continuous, differentiable functions.

In a *single-variable optimization problem*, the task is to find the value of $x=x^*$ in an interval (a,b) that minimizes or maximizes a function $f(x)$. The minimum or maximum is the point at which the derivative $f'(x^*) = 0$. [9] This optimum, be it local or global, can be found analytically or iteratively using Gradient Decent or Newton's Method algorithms.

Optimum points in *Multi-variable optimization problems* are found in a similar way. Optimum points of a multi-variable function are the points where partial derivatives with respect to each variable all equal zero. [9]

Addition of constraints to *multi-variable optimization problems* greatly increases difficulty of solving such problems. For one thing, if the number of constraints exceeds the number of variables, the problem becomes overdefined and typically becomes unsolvable. Else, the problem can be solved using methods of direct substitution, constrained variation, and Lagrange multipliers. [9] Solving such problems require complex analytical solutions that become very difficult to solve as the dimension of the problem space increases.

The scheduling problem, as with many practical, real world problems, involves objective functions that may not be continuous and are often not differentiable. The scheduling problem is also limited by complex constraints. In addition, the number of variables in a practical healthcare scheduling problem is sure to be very large. Therefore, the classical optimization approach, on its own is not suitable for solving the scheduling problem.

1.3.2 Intelligent scheduling: flow-shop scheduling model

Scheduling and optimization are classical problems in the field of operations research and artificial intelligence. Researchers have tackled various scheduling

problems with great enthusiasm and fervor. The literature is very extensive. However, surprisingly little attention has been given to the specific problem of scheduling medical procedures, at least not at the depth of detail proposed in this thesis.

First introduced in 1953 by Selmer Johnson in a paper entitled: "Optimal two and three stage production schedules with setup times included," the *job-shop* or *flow-shop* problem has become by far the most popular scheduling problem for researchers. [10] In the five decades since Johnson's seminal paper, the operations research community has examined various aspects of this problem in more than 1200 published papers. The traditional flow-shop problem is defined as follows:

> *n* jobs are to be processed on *m* machines. All jobs must visit all machines in the same order. Each machine can only process one job at any given time. The processing time of job *i* on machine *j* is p_{ij} where *i=1,2…n* and *j=1,2…m*. Find the order or schedule of jobs to be processed to minimize the time (makespan) to complete all jobs. [11]

The problem and solutions can be visualized thus:

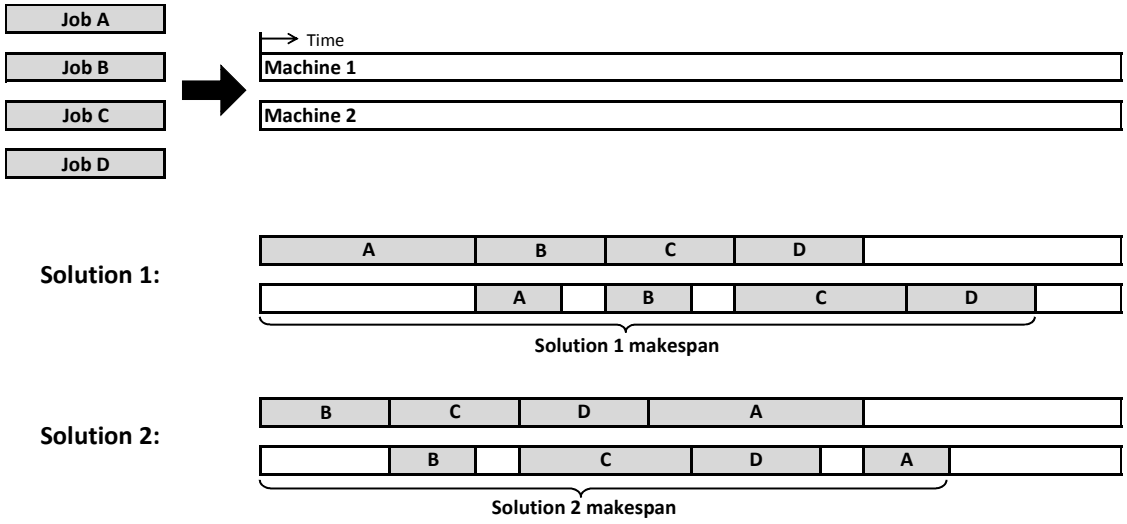


Figure 1.1: Visualization of the flow-shop problem and solution

The traditional flow-shop model is a very simplified and restrictive representation of practical situations. Grupta et al. describes 21 assumptions for the flow-shop scheduling problem. [11] [12] Those assumptions are reproduced verbatim in Table 1-1.

Table 1-1: Traditional job-shop scheduling assumptions

Assumptions concerning job	
J1	Each job is released to the shop at the beginning of the scheduling period.
J2	Each job may have its own due date which is fixed and is not subject to change.
J3	Each job is independent of each other.
J4	Each job consists of specified operations, each of which is performed by only one machine.
J5	Each job has a prescribed technological order which is the same for all jobs and is fixed.
J6	Each job (operation) requires a known and finite processing time to be processed by various machines. This processing time includes transportation and setup times, if any, and is independent of preceding and succeeding jobs.
J7	Each job is processed no more than once on any machine.
J8	Each job may have to wait between machines and thus in-process inventory is allowed.
Assumptions concerning machines	
M1	Each machine center consists of only one machine; that is, the shop has only one machine of each type.
M2	Each machine is initially idle at the beginning of the scheduling period.
M3	Each machine in the shop operates independently of other machines and thus is capable of operating at its own maximum output rate.
M4	Each machine can process at most one job at a time. This eliminates those machines that are designed to process several jobs simultaneously like multi-spindle drill.
M5	Each machine is continuously available for processing jobs throughout the scheduling period and there are no interruptions due to breakdowns, maintenance or other such causes.
Assumptions concerning operating policies	
P1	Each job is processed as early as possible. Thus, there is no intentional job waiting or machine idle time.
P2	Each job is considered an indivisible entity even though it may be composed of a number of individual units.
P3	Each job, once accepted, is processed to completion; that is, no cancellation of jobs is permitted.
P4	Each job (operation), once started on a machine, is completed to its completion before another job can start on that machine, that is, no preemptive priorities are assigned.
P5	Each job is processed on no more than one machine at a time. (This is a result of assumptions J5 and P2.)
P6	Each machine is provided with adequate waiting space for allowing jobs to wait before starting their processing.
P7	Each machine is fully allocated to the jobs under consideration for the entire scheduling period; that is, machines are not used for any other purpose throughout the scheduling period.
P8	Each machine processes jobs in the same sequence. That is, no passing or overtaking of jobs is permitted.

These assumptions dictate that only the simplest problems could be explicitly modeled as a flow-shop scheduling problem. At the very least, practical problems had to be heavily simplified to fit this model.

Over the years since the first introduction, variations on the traditional model have been created through relaxation or modification of some of the assumptions listed in Table 1-1. [13] – [19]

Solving flow-shop scheduling models is challenging due to the dimensionality of the problem. The general flow-shop model has $(n!)^m$ possible schedules. Even modest values for n and m can create an intractable problem that is too large to explicitly enumerate through.

Early research proposed using mathematical programming, specifically integer linear programming, to solve flow-shop scheduling problems. [20] – [26] Physical simulation and Monte Carlo simulation were also tried. [27] Unfortunately, the size of problems solvable at the time (early 1960s) was very small, limited by lack of computational power and lack of efficient solving algorithm. [11] Through computational complexity analysis, many well-known flow-shop problems and models were shown to belong to the NP-complete or NP-hard class of problems. [28] [29] [30]

The difficulty of the flow-shop problem and computational power limitation lead to development of heuristic techniques for finding good, near-optimal solutions for large flow-shop problems that were otherwise unsolvable at that time. Framinan et al. and Jungwattanakit et al. review some such heuristics including *constructive heuristics, fast improvement heuristics, simulated annealing, tabu search,* and *genetic algorithm heuristics.* [31] – [41] Heuristic techniques are great tools for quickly finding good solutions but can never guarantee optimality and are therefore approximation methods. There are general heuristic frameworks but effective heuristic algorithms are necessarily very problem specific. That is, flow-shop scheduling heuristics are only good for solving flow-shop problems. Despite fervent development of high quality heuristics for the flow-shop problem, the allure of guaranteed solution optimality is always strong. Recent advances in computing power and data storage capacity has reignited interest in exact, mathematical programming approaches. [11] [42] – [50]

The traditional flow-shop scheduling problem was inspired by a production line optimization problem. Therefore, one would expect that since its introduction half a century ago, the flow-shop model has been used to solve numerous practical production

problems. Unfortunately, this is not the case. Most of the research into flow-shop scheduling has been theoretical in nature. In their review of fifty years of flow-shop scheduling research, Gupta et al. found that researchers were motivated by theoretical aspects of the problem. The practical application of the flow-shop scheduling model is rare. [11] Reason for this lack of practical application is that many of the flow-shop models studied are too simple and do not accurately model real problems in industry. [11] [51] Majority of the flow-shop models research focus on the schedule makespan as the objective function. [11] The medical procedures scheduling problem requires more sophisticated objective functions to model patient wait times and resource utilization. The simple flow-shop model on its own is not enough to model the medical procedures scheduling problem. Nevertheless, the philosophy behind the flow-shop scheduling model provides a promising starting point for research in this thesis.

1.3.3 Intelligent scheduling: project planning

The planning of large, complex projects is a daunting but increasingly common challenge facing modern enterprises. Project planning typically involves scheduling a set of required jobs/tasks onto a timeline to accomplish an end goal. See Figure 1.2.

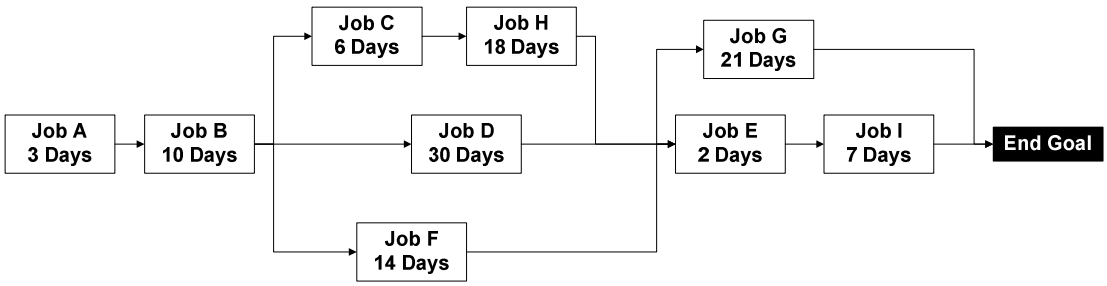


Figure 1.2: Example project timeline

Similar to flow-shop scheduling, the optimality criteria is usually project makespan. The project makespan is indirectly represented by the starting time of the very last job or task. The project planning problem, at its core is a variation on the flow-shop scheduling problem. The difference is in the setting up of constraints. While purpose of flow-shop model constraints is to enforce non-interference of jobs, project planning constraints primarily model precedence requirements of project tasks. That is,

constraints in project planning models are mainly of the nature: task x must be completed before task y can begin. Project planning problems are modeled using a binary integer formulation:

> The project completion period is divided into *w* time intervals. The decision variable $x_{i,t}$ takes on value of *1* if task *i* is scheduled to start in interval *t*. $x_{i,t}$ equals *0* otherwise. Constraints enforce precedence by ensuring that some tasks can only be scheduled in intervals after completion of other tasks.

The most basic project planning model does not consider resource requirement of tasks and schedules tasks based on their durations only. Schedules are generated by setting tasks to begin at their earliest possible start time. That is, tasks are set to begin as soon as all its precedence requirements are met. Conveniently, schedules generated in this manner can easily be modeled as finite, acyclic directed graphs. The states of the project are nodes on a graph and the tasks or activities make arcs in the same graph. Figure 1.3 shows the graph representation of the example project shown in Figure 1.2. Node 1 represents the start of the project and node 7 represents project completion. The nodes in between nodes 1 and 7 represent milestones in the project timeline.

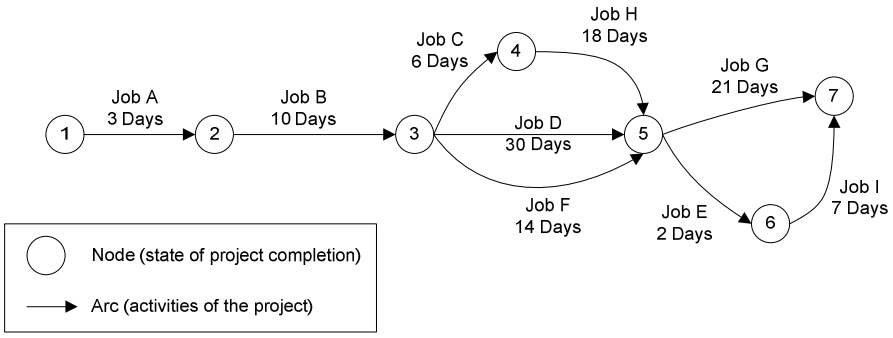


Figure 1.3: Graph representation of the example project in Figure 1.2

That graph/schedule can be analyzed using techniques within graph theory. [52] – [60] The power of graph theory can be harnessed to model uncertainty in duration of project tasks. For example, the schedule function of a project can be represented as a polyhedron. If tasks of a project have random completion times with known distributions, the probability distribution of the project makespan can be found by integrating over the contours of the polyhedron schedule function. [61]

Two other techniques: PERT and CPM also rely on graph representation of project planning models. The Program Evaluation Risk Task (PERT) technique analyzes a schedule by first estimating the possible slack between each pair of linked tasks in a project based on uncertainty of completion time of each task. The estimated slacks then contribute to the probability distribution of the overall project makespan. [62] [63] [64] Critical Path Method (CPM) is a technique that identifies bottlenecks within a given schedule. In its first, forward pass CPM schedules tasks to begin as soon as all precedence of each task is met. The project makespan can then be calculated. Then, in a second, backward pass, CPM schedules tasks at their latest possible start time without affecting the project makespan. The difference between the earliest and latest start times of each task is its float. A path from start to project completion that passes through only tasks with zero float is the *critical path*. Tasks along this critical path represent the bottlenecks of the project. CPM is a popular project planning tool in industry due to its ease of use and simplicity. [65] [56] [66]

The basic project planning model and the PERT and CPM methods to analyze that model are overly simplistic in considering only time aspect of schedules. In reality, practical problems rarely have all resources available to allow tasks to begin at their earliest possible start time. Thus a more realistic and practical problem is the *Resource-constrained Project Scheduling* problem. A review of project scheduling research reveal that resource constrained scheduling problems can be categorized into three classes: *time/cost trade-off, resource leveling,* and *resource allocation*. [67] [68] [69]

Time/cost trade-off problems consider the time aspect as the most important measure of success. The objective is to minimize the project makespan by adding resources to accelerate completion of tasks. This essentially makes completion times of tasks variable as a function of resource cost. There are potentially many different combinations of task durations that could result in the same project makespan. However, each of those schedules may result in different total project cost. The time/cost trade-off problem is then to determine the most cost effective schedule for any given project makespan. Several techniques such as linear programming and network flow algorithm exist to solve this problem. [70] The time/cost trade-off model, applied to intelligent high level management of medical resources can potentially answer questions of the following

nature: “What is the most cost effective use of resources to achieve a given patient flow?” And “How much will additional resources cost to achieve a higher level of patient flow?”

Resource leveling problems deal with situations where resource is abundantly available but one tries to maintain constant resource use or consumption. The resource leveling problem is difficult because it is dynamic in nature. That is, the problem changes during the solving process due to the tracking of resource usage level by decision variables. Dynamic scheduling problems have high dimensionality. However, techniques exist for reducing dimensionality of the problem: sub project programming; two-level sub project concept; and approximation in policy space. [71] Resource leveling is significant to medical procedures scheduling. Many resources in healthcare are human resources such as doctors and nurses. It is desirable to give human resources steady, smooth workloads to maximize fairness among the workforce and minimize unpredictability and uncertainty. In addition, many non-human resources in healthcare carry very high idle costs; Empty operating theatres and idle MRI scanners are immediate examples. It is therefore desirable to maximize operating time of such high idle cost resources.

Resource-allocation problems deal with realistic situations where resource availability is limited. The objective is to allocate the scarce resources optimally to tasks in order to minimize project makespan. The basic project planning model is extended to the resource-allocation model through the addition of resource constraints. In the integer programming formulation: each time interval in the project period is given resource availability values. Constraints are then added to ensure that demand for resources does not exceed availability of those resources in each time interval. [67] The resource-allocation problem is perhaps most applicable to healthcare scheduling. As previously discussed, a major cause for excessive wait times in the Canadian healthcare system is the scarcity of resources due to poor system design and/or actual shortage.

Similar to flow-shop scheduling problems, the resource-constrained project scheduling problem can be modeled and solved exactly using general mathematical programming. Interestingly however, research into exact solving methods resulted in development of algorithms specific to the resource-constrained project scheduling

problem. Confusingly, those algorithms are called *branch and bound (BnB)* algorithms, exactly the same name as the algorithm to solve linear programming models. The LP BnB and project scheduling BnB share the same name but are very different both in execution and application. The LP BnB algorithm for solving linear programming models will be described in detail in section 2.2.5. The project scheduling BnB algorithm is highly specific to the project scheduling problem and is basically a mathematical formalization of the manual scheduling process. Each node in the solution tree represents a partial schedule with a list of already scheduled activities and a list of candidate activities still waiting to be scheduled. Branching from a node represents different ways that partial schedule can change depending on the order of candidate activities to schedule next. The branches and the nodes of the solution tree are essentially paths that partial schedules can take to become the final schedule. The task is then to find the path or ordering of activities that results in the optimum schedule. The most popular objective value to optimize is the project makespan. [67] [68] One way to find the optimum schedule is to explicitly enumerate through all the possible orderings of activities. That option is crude and computationally costly. Research into project scheduling BnB algorithms have produced rules, branching strategies, bounding, pruning and backtracking techniques to explore the solution tree efficiently while retaining the ability to guarantee solution optimality. [68] [72] – [77]

A number of heuristic approaches were also developed to address the computational limitations of the exact, branch and bound approaches. Similar to the branch and bound method, heuristics model the problem as a network or graph with each node representing a partial schedule. Unlike branch and bound however, heuristics do not enumerate through all the possible branches from each node. Instead, at nodes, heuristics choose what activities to schedule next based on sets of priority rules. These rules include the *Minimum Job Slack (MINSLK), Resource Scheduling Method (RSM), Minimum Late Finish Time (LFT), Worst Case Slack (WCS), Greatest Resource Demand (GRD), Greatest Resource Utilization (GRU), Shortest Imminent Operation (SIO), Most Jobs Possible (MJP), Most Total Successors (MTS), Greatest Rank Positional Weight (GRPW)* and even *Select Jobs Randomly (RAN)*. [78] – [84]

More advanced, meta-search heuristics were also developed to tackle the resource constrained project scheduling problem. Meta-heuristics operate on the *activity list*: the location or ordering of activities in a schedule. Meta-heuristics employ techniques from AI such as *simulated annealing, tabu search, genetic algorithm,* and *swarm intelligence* to search through combinations of activity ordering for the best schedule. [68] [85] – [94]

Reviews of numerous heuristic procedures have found that they are highly specific to each problem. Effectiveness of each heuristic depends greatly on the setup, logic and nuances of its respective problem. [67] [68]

Research into project planning/scheduling provides a promising framework for modeling the medical procedures scheduling problem. However, even the more sophisticated resource-constrained project scheduling model is still too simple to directly model the complex workflow and resource requirements of medical procedures. Nevertheless, the philosophy underlying the project planning models will be useful in the development of a novel model for medical procedures.

1.3.4 Scheduling in healthcare: booking systems

One of very few existing practical application of AI to scheduling of medical procedures is the development of an intelligent operating room booking system by Ozkarahan. [95] Ozkarahan first recognized that scheduling of medical procedures consists of two distinct processes: *advanced scheduling*, and *allocation scheduling*. Advanced scheduling is the booking of patients or procedures to a future date. Allocation scheduling is the sequencing of procedures and activities on the day assuming all booked patients and resources are in the hospital and ready. The system proposed by Ozkarahan attempts to solve the allocation scheduling aspect of the problem:

> Given *n* procedures (surgeries), with known completion times, to be scheduled into m operating rooms (ORs), find the schedule that minimizes the total time taken to complete all procedures (makespan).

The challenge is to decide which procedures to schedule into which OR and in what order. The above problem is very similar to the flow-shop scheduling problem.

The procedures are the jobs to be scheduled onto ORs which are machines. The main difference is that jobs for the OR booking problem need only to be processed once.

Ozkarahan models the OR booking problem as an integer programming model but does not solve it exactly as an integer programming model. No exact solving methods were studied. Instead, the only solving method offered is a heuristic one. The heuristic solves the scheduling problem in two phases: loading and sequencing. The loading phase attempts to assign as many procedures, as fairly as possible to each OR. This is accomplished by first sorting the list of all procedures from longest processing time (LPT) to shortest processing time (SPT). In that order, each procedure is then assigned to an OR with the most available time. Once procedures have been assigned to ORs, the sequencing phase orders the procedures within each OR to optimize a secondary objective such as patient wait time. Ozkarahan's heuristic sequences procedures using the SPT rule. That is, procedures with the shortest processing times are processed first. This heuristic was designed based on experience of manual schedulers. Experience showed that loading longer procedures into rooms earlier usually resulted in fitting more procedures overall because shorter procedures can easily fit into capacity gaps left by longer procedures. Once procedures are loaded into rooms, experience showed that processing shorter procedures earlier minimizes overall patient wait time. Ozkarahan realizes that this heuristic cannot guarantee optimality but claims that it consistently delivers good schedules. No quantitative analysis was done to back up this claim of heuristic performance. Nevertheless, the underlying logic is reasonable.

Ozkarahan's OR booking problem thus far bears the greatest resemblance to the kind of scheduling problem that this thesis is planning to tackle. However, the OR booking problem is simplistic in considering only completion times of procedures. No consideration to resource requirements is made. This thesis research is interested in modeling resource requirements in great detail to include doctors, nurses, anesthesiologists, equipment, beds, etc. Nevertheless, this work by Ozkarahan presents valuable insights that will aid the development of a more sophisticated, novel scheduling model. Specifically, Ozkarahan recognized the high level of uncertainty in medical procedures and the importance of capturing experience of manual schedulers. The breaking up of the scheduling problem into distinct phases or processes is a good tactic

for managing problem scope. Ozkarahan also points out potential for integration with knowledge based artificial intelligence tools to improve usability of scheduling systems.

Another one of very few practical application of AI to healthcare scheduling is the development of a patient booking system by Patrick et al. [96] Patrick's medical imaging patient booking system tackles the advance scheduling aspect of the scheduling problem. The challenge for Patrick et al. was effective prioritization of significantly variable demand for CT scans. To deal with that uncertainty and variability, Patrick et al. uses a Markov Decision Process (MDP) model for their scheduling problem. This patient booking system will not be examined in much further detail because the advance scheduling aspect is outside the scope of this thesis. However, the examination of uncertainty and variability presented in this paper should aid in the development of a realistically practical scheduling system. For example, Patrick et al. references aspects of Markov Decision Process theory that can transform MDP problems with their associated uncertainty, into linear programming problems. [96] [97]

1.3.5 Scheduling in healthcare: nurse rostering

All practical healthcare scheduling problems discussed thus far have been focused on optimizing the demand side of healthcare, i.e. scheduling of patients. Indeed, most research into healthcare scheduling focus on the demand side. This section will review what little research there is into optimization of the supply side of healthcare.

To the best knowledge of this author, the nurse rostering problem is currently the only supply side optimization problem that is receiving any significant amount of attention. The nurse rostering problem is the task of allocating nurses to periods of work. Nurse rostering is a very difficult problem as it must contend with a myriad of human resource constraints. Hard constraints such as labour regulations, skill level, workload demand, etc. must be satisfied. Soft constraints such as personal preference, seniority, vacation preference, etc. while desirable, may be violated to create a feasible schedule. The Nurse rostering problem, much like flow-shop scheduling and project scheduling is first modeled using mathematical programming. The models typically start with the following binary decision variable:

$$x_{ij} = \begin{cases} 1 & if\ nurse\ i\ works\ in\ shift\ pattern\ j \\ 0 & otherwise \end{cases}$$

A great number of constraints then model the aforementioned human resource restrictions. Differences between the approaches are in the setting up of these constraints and methods to solve the problem. The models can be solved exactly using linear programming, goal programming, and constraint programming. The effectiveness of exact solvers is limited. Large problems must be greatly simplified to be solvable in reasonable amount of time. Realistic and accurate models can only solve very small sized problems. [98] – [105]

Alternatively, the nurse rostering problem can be solved approximately using heuristics. Some existing heuristics take an iterative trial and error, shuffling approach which emulates manual scheduling. Some other heuristics are rule based. More intelligent meta-heuristics using simulated annealing, tabu search, and genetic algorithms also exist. [98] [99] [106] – [114]

The core objective of the nurse rostering problem is the minimization of wasted effort while ensuring adequate coverage/service. However, typical problems are constrained so tightly by the aforementioned human resource constraints that most existing solutions are concerned first and foremost with satisfying all hard constraints. In reality therefore, optimization is lucky to even be considered as a secondary objective. Reviews of state of the art in nurse rostering research reveal that current models have difficulty modeling and satisfying all constraints and therefore do not accurately reflect real world situations. [98] [99] Research effort is still directed at clever design of constraints to better represent real-world restriction. Nevertheless, the nurse rostering problem is a resource management problem and the philosophy behind it will be useful in development of a novel formulation of the medical procedures scheduling problem.

1.3.6 Summary of literature review findings

The state of the art in scheduling healthcare currently focuses mostly on staff rostering. Some intelligent procedures scheduling models exist but are simple rule-based algorithms that capture manual scheduling logic and are very problem specific.

The flow-shop scheduling and resource constrained project scheduling problems hold the most promise for modeling the medical procedures scheduling problem. However, both are too simplistic and neither can be applied directly to the medical procedures scheduling problem. The philosophy behind them will aid development of a novel model for the medical procedures scheduling problem.

Most existing scheduling models follow similar paths of development and face similar challenges. Nearly all scheduling problems are first modeled using mathematical programming, usually as an integer problem. Therefore, integer programming is the de facto formal formulation of scheduling problems. The integer models are then solved either exactly using branch and bound or approximately using heuristics. An exact solver can guarantee a global optimum solution. However, the solving time rises rapidly with increasingly larger problem size. The size of the problem solvable depends largely upon the efficiency and cleverness of the problem formulation. Numerous heuristic techniques were developed to solve large, otherwise unsolvable problems. General heuristic frameworks exist such as simulated annealing, genetic algorithm and tabu search. However, effective heuristic algorithms are highly problem specific.

Mathematical programming is the only approach that can guarantee optimality of solutions and therefore should be deployed for problems where optimality is desired. The computational cost of mathematical programming models is high. However, recent advances in computational power and data storage capability keep mathematical programming models practical. Exact solving algorithms are also necessary for setting benchmarks for evaluating heuristic performance.

1.4 Expected thesis contributions

This thesis will develop a novel mathematical programming formulation of the static medical procedures scheduling problem (MPSP). The model will include greater level of workflow and resource requirement detail than existing flow-shop scheduling and project scheduling models. The new model will represent general medical procedures and will be customizable to represent more specific procedures. The specific procedure studied in this thesis is the haemodialysis procedure.

This thesis will then investigate solutions to the medical procedures scheduling model. Both exact and approximate solution methods will be developed and studied.

Research in this thesis will form the core models and algorithms for more sophisticated, realistic and practical scheduling systems. It is the hope of this author to help bridge the gap between theoretical research and practical application of resource scheduling techniques.

The focus of this thesis is on medical procedures. However, the contributions of this thesis will be general enough that it can be applied to any scheduling problem of similar nature.

1.5 Thesis organization

The current section outlines the motivation for work, description of problem and literature review of state of the art in scheduling research. Chapter 2 will describe the general medical procedure model as well as the more specific haemodialysis procedure model. Chapter 2 will also provide the reader with some mathematical and software background. Chapter 3 will develop, in detail the novel mathematical programming formulation of the medical procedures scheduling problem. Chapter 3 will then discuss the use of an exact solver to obtain optimum solutions to the scheduling model. Chapter 4 will develop a novel evolutionary heuristic to quickly solve the scheduling model. Chapter 5 will conclude the thesis and discuss areas for further development.

Chapter 2: Background

2.1 Medical procedures

Medical procedures are complex operations that involve interactions between patients, medical personnel, infrastructure, and resources. In order to optimize the scheduling of medical procedures, one must first design an accurate model of procedures. The ideal model should be general to represent a wide variety of different procedures yet customizable to model nuances of more specific procedures. This section 2.1 discusses the modeling of medical procedures.

2.1.1 Simple medical procedure model

As discussed earlier in section 1, research into medical procedures scheduling have largely been focused on the *advanced scheduling* aspect of the problem. That is, most existing and theoretical scheduling systems focus on booking procedures onto already scheduled resources. Each procedure has resource requirements and a scheduler looks for free time slots where those required resources are available. The availability of resources is pre-determined by shift schedules of nurses, availability of doctors, availability of equipment and rooms etc. A procedure is treated as the atomic scheduling unit. That is, the finest level of detail is at the procedure level. The only characteristics of procedures that are modeled are resource requirements and total duration.

Following the scheduling problem framework described in section 1.2.1, consider a clinic that specializes in five different procedures (P_A, P_B … P_E) that utilize different levels of the clinic's seven resources (R1, R2, … R7). The resources model both human and non-human resources such as nurses, doctors, anesthesiologists, machines, equipment and/or rooms. Required resources are assumed to be fully occupied throughout the entire procedure duration. Example procedures in the current simple model can be visualized in the following time diagrams:

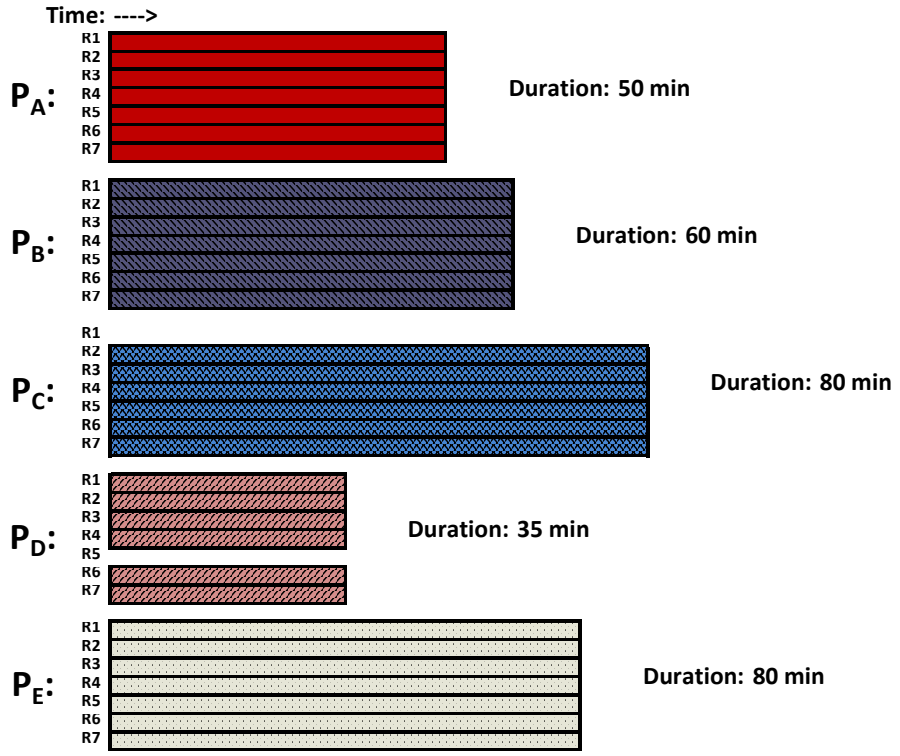


Figure 2.1: Example time diagrams of procedures under current model

In the above example, procedures P_A, P_B and P_E fully occupy all seven resources, P_C uses resources R2 through R7, and P_D uses all resources except resource R5. The weakness of this simple model is that it does not consider detailed resource utilization beyond the procedure level. As a result, procedures are treated as 'solid' resource consumption blocks. The only viable scheduling strategy is to schedule procedure end to end in a cascading manner. That is, one procedure must be completed before another procedure can begin. An example schedule under the simple model looks like the following:

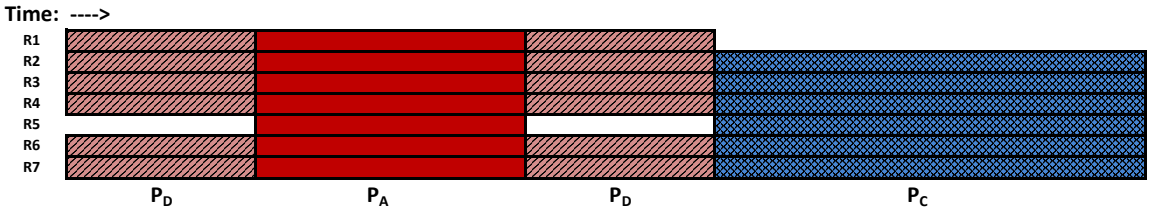


Figure 2.2: Example schedule under simple procedures model

The makespan of a cascading schedule is simply the sum of durations of procedures in the schedule. The example schedule in Figure 2.2 involves one instance of procedure P_A, one instance of P_C, and two instances of P_D. The makespan is therefore *50+80+35+35=200 min*. This simple model is similar to one used by Ozkarahan to develop a booking system for scheduling procedures into operating rooms. [95] This simple model is also the basis for the flow-shop scheduling problem.

This simple model and cascading scheduling strategy is representative of manual scheduling commonly practiced by clinic managers. A scheduling system based on this simple model does nothing more than automate the manual scheduling process. The potential for optimization is low because the simple model does not offer opportunities to implement more sophisticated scheduling strategies.

2.1.2 Improving the simple procedure model

In reality, resources are not always fully occupied throughout the entire procedure duration. For example, a typical surgery involves many resources: surgery prep nurses, prep room, anesthesiologist, surgeon, specialist, assistants, operating room etc. All those resources are not needed throughout the duration of the entire surgery. The surgeon, specialist and operating room resources are not needed until the patient is prepped. Once a patient is prepped for surgery and enters the operating room, the prep nurse and prep room resources become available. Considering prep nurse as resource R1, anesthesiologist as R2, specialist as R3, surgeon as R4, assistant nurse as R5, surgery prep room as R6, and operating room as R7, the workflow time diagrams of surgery procedures that utilize resources only when needed may look like the following:

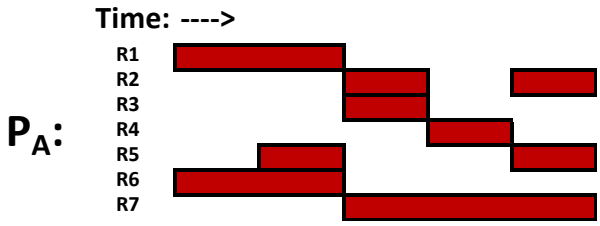


Figure 2.3: More detailed workflow time diagram of procedure P_A

The example procedure shown in Figure 2.3 reflects the complex resource requirements of real world medical procedures. Similar level of detail is added to the other four procedures of the clinic in Figure 2.4. The development of the scheduling system will be based on these more detailed representations of medical procedures.

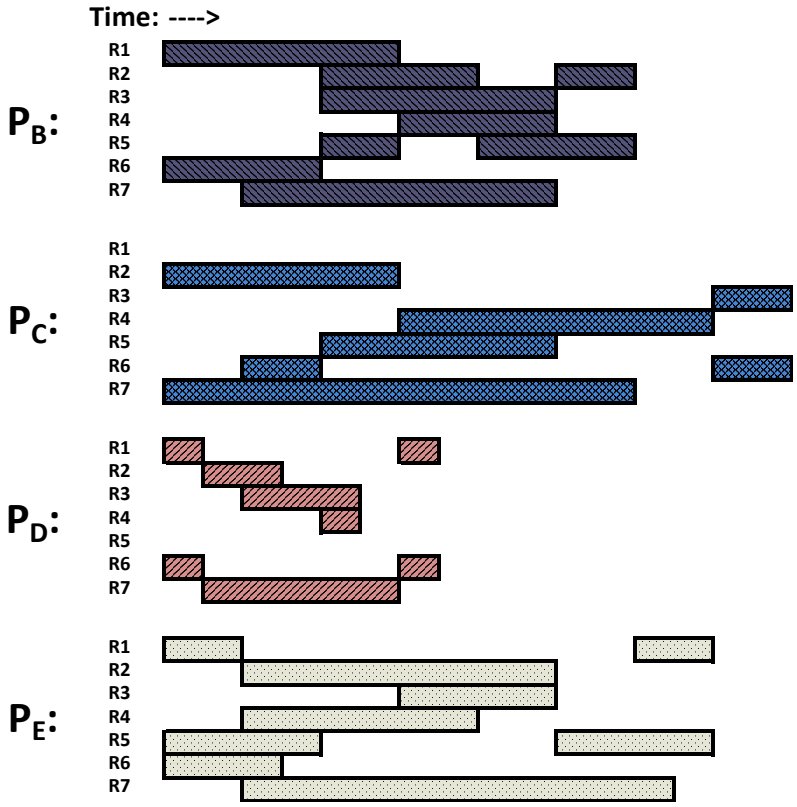


Figure 2.4: More detailed workflow time diagrams of procedures P_B through P_E

One immediately recognizes that resources that are not occupied throughout entire procedure could potentially become available to serve other procedures before the current procedure is complete. That is, there is potential for overlapping of procedures. Revisiting the simple schedule in Figure 2.2, the higher level of detail now allows the scheduler to overlap procedures to reduce makespan of the schedule:

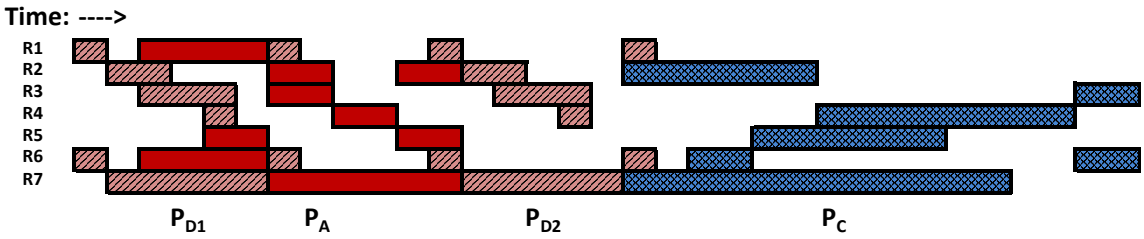


Figure 2.5: Overlap procedures to reduce schedule makespan

The order of procedures is the same for both schedules in Figure 2.2 and Figure 2.5. However, overlapping of procedures allowed by the more detailed model reduced the schedule makespan to 165 minutes from 200 minutes under the simple model. In addition to the reduction in makespan, some interesting observations can be made from the example schedule in Figure 2.5: A) the order of procedures in the schedule can have a large impact on procedure overlap and on makespan. Procedures P_{D1} and P_{D2} are two instances of the same procedure (P_D) that interact with procedure P_A differently depending on their respective position in the schedule. P_D, if scheduled ahead of P_A, overlaps significantly with P_A. If however, P_D is scheduled behind P_A, it can only overlap slightly with P_A. B) The more detailed model provides resource utilization information useful for optimization. In the above example, resource R7 is occupied 85% of the total schedule makespan and is quite obviously the limiting resource. In comparison, the next most used resource, R2 is occupied in only 42% of the total makespan. It stands to reason that one can further reduce schedule makespan by making more of resource R7 available.

The additional detail in the improved model immediately provides a lot of useful information not found in the simple model. The procedures interaction information and resource utilization information form foundations from which an intelligent scheduling system can be built. Such a system will go beyond simple automation of manual scheduling and be able to intelligently schedule procedures to minimize makespan, minimize wait time, and optimize resource utilization.

Before proceeding further, an important caveat must be noted. In reality, many medical procedures such as surgeries have high degrees of uncertainty. Overlapping of surgery procedures minimizes flexibility to handle uncertainty and is therefore risky. The

realistic applicability of the deterministic model presented in this section to surgeries is questionable. However, the model provides a solid starting point for the research in this thesis. Designing a deterministic scheduling system will be much easier than designing one based on a probabilistic model. There are also plenty of medical procedures that are very predictable and behave deterministically. For example, some therapy procedures such as haemodialysis and chemotherapy have prescribed treatment lengths. Also, more routine procedures such as x-ray imaging performed by well trained and experienced personnel can have very predictable completion times. The research in this thesis can be applied directly to deterministic procedures and procedures with low level of uncertainty. Some uncertainty can be modeled indirectly by adding expected deviation to procedure durations. Modifying the deterministic model to handle procedures with more uncertainty is reserved for future work.

The improved, more detailed model of medical procedures discussed in this section will from here on in be referred to as the procedures model. The five example procedures presented in this section will continue to be used to illustrate the development of the scheduling model.

2.1.3 Specific medical procedure: Haemodialysis

The general procedures model presented in the previous section is easily customizable to fit specific procedures such as haemodialysis.

Haemodialysis is a well established and tightly constrained medical procedure. The workflow for one patient and one nurse is outlined in Figure 2.6. The corresponding activity descriptions are summarized in Table 2-1. The haemodialysis workflow data is collected and compiled by Dr. Amgad Eskander. The sources of data are workflow studies/summaries of haemodialysis units at three hospitals: Lourdes Hospital in N.Y. USA, Wollongong Hospital in Australia, and Leicester NHS Trust Hospital in the UK. Durations of activities given in Table 2-1 are mean duration values extracted by Dr. Eskander from workflow studies at the three aforementioned hospitals. [115]

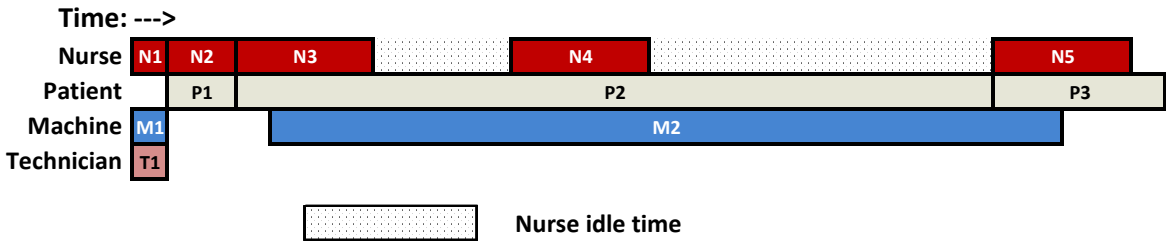


Figure 2.6: Workflow of dialysis procedure for one patient

Table 2-1: Dialysis workflow activities description

Activity	Duration (min)	Description
N1	0	Nurse is notified of patient arrival
N2	3	Nurse prepares paperwork
N3	25	Nurse prepares patient for dialysis treatment: measure temperature and bp, perform re-dialysis assessment, insert needles, secure dialysis lines, and starts dialysis machine
N4	2 – 45	Nurse administers EPO, iron IV. Nurse provides monitoring and assessment services if required
N5	30	Nurse removes needles and disconnects patient from dialysis machine. Nurse takes post-dialysis bp and weight measurements
P1	10	Patient check in with receptionist. Patient weighs him/her self
P2	200 - 300	Pre-dialysis preparation: temperature, bp measurements and re-dialysis assessment. Clean access and secure dialysis lines (25 min). Dialysis treatment period varies between different patients.
P3	35	Post-dialysis activities: disconnect dialysis lines, remove needles, wait for haemostasis, measure bp, measure weight. Patient is discharged
M1	5	Dialysis machine being primed and disinfected
M2	200 – 300	Dialysis machine is occupied by patient undergoing treatment
T1	5	Technician primes and disinfects dialysis machine

Activities must be performed in order and must not interfere with each other. For example, activity N3 must occur after N2 and may only occur if N2 is complete. Activities P2 and N3 must occur at the same time. Activities P3 and N5 must occur at the same time. Activity N4 is flexible and can be performed anytime during the patient's dialysis treatment.

A patient undergoing dialysis treatment does not require constant attention from the nurse. Once a patient starts his/her dialysis treatment, the nurse is free to serve other patients. In reality, each nurse serves multiple patients in any given shift. [115] A more realistic workflow is shown in Figure 2.7.

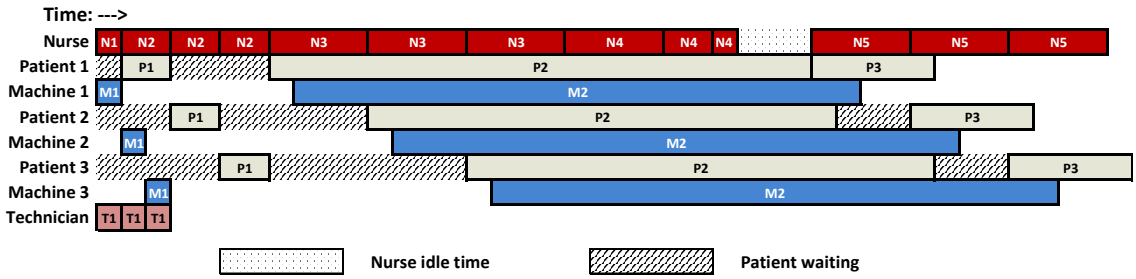


Figure 2.7: Realistic dialysis workflow of one nurse/patients grouping

Figure 2.7 shows that the general procedures model can be easily adapted to a specific procedure such as haemodialysis. The general procedures model is capable of capturing realistic details; details that would have been impossible to represent using the simple model.

Even such a simple, straightforward procedure as haemodialysis illustrates the complex nature of medical procedures and the need for a novel model to schedule them. Existing scheduling models such as flow-shop scheduling, project scheduling, and nurse rostering are not powerful enough to model medical procedures such as haemodialysis.

2.2 Mathematics background

2.2.1 Linear programming

Linear programming (LP) is a mathematical modeling tool for solving optimization problems.

The easiest way to overview linear programming is through example. Consider a car manufacturer Colonel Motors (CM). CM produces two products: small cars and SUVs. Each car cost $16,000 to build and can sell for $20,000. Each car requires 30 hours to produce parts and takes 8 hours to assemble. Each SUV cost $23,000 to build and can sell for $35,000. Each SUV require 40 hours of parts production and takes 11 hours to assemble. Each month, CM has available 30,000 parts production hours and 10,000 assembly hours. Demand for SUVs is unlimited. However, CM must produce at least 300 small cars per month in order to maintain an environmentally friendly image. CM is interested in maximizing monthly profit.

This problem can be solved using linear programming model. An LP model consists of *decision variables*, an *objective function*, and *constraints*.

Decision variables, as the name implies, describe the decisions to be made. In this case, CM must decide how many cars and SUVs to produce each month. Therefore, the decision variables for this problem are defined as follows:

$x = number\ of\ cars\ to\ produce$

$y = number\ of\ SUVs\ to\ produce$

The objective function is a linear function of the decision variables to be optimized. In this case, CM's objective value is its monthly profit. The objective function is defined as:

Maximize $4000x + 1200y$

4000 is the unit profit (selling price less production cost) of each car. 12000 is the unit profit of each SUV.

Finally, CM's production limitations are captured as constraints:

Parts production hours (C1): $30x + 40y \leq 30{,}000$

Assembly hours (C2): $8x + 11y \leq 10{,}000$

Small cars demand (C3): $x \geq 300$

Non-negativity (C4): $x, y \geq 0$

This simple LP model is two dimensional (two decision variables). Therefore, it can be represented graphically, as shown in Figure 2.8.

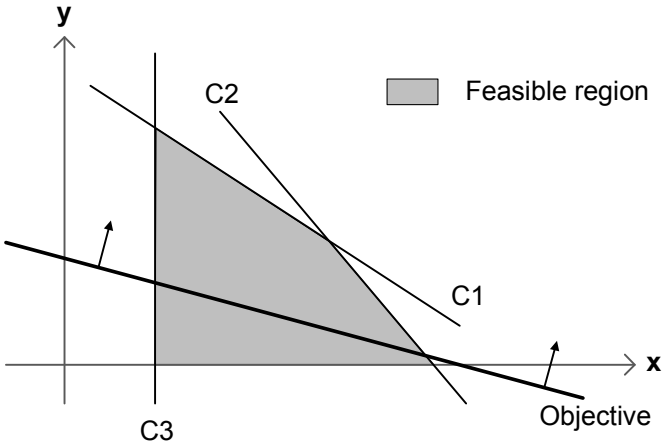


Figure 2.8: Graphical representation of LP model

The problem constraints bind the values of decision variables to a polygonal feasible region. Only in this region can a viable solution to the problem be found. The maximization nature of this problem causes the objective function to move up. The maximum objective value occurs at the corner of the feasible region shown in Figure 2.9.

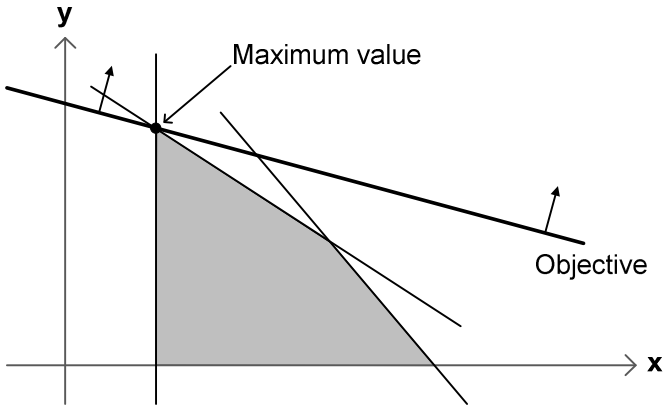


Figure 2.9: Optimal solution of LP model

At the optimal corner of the feasible region, x=300, y=562.5, and the resulting objective value is $7.95 million. This solution is easy to obtain graphically. However, problems with more than three decision variables are impossible to visualize. Fortunately, a well established algorithm exists to quickly solve LP problems: the Simplex method. The nature of linear problems guarantees that the optimal solution will

always occur at a corner of the feasible region. The Simplex method is simply a quick and intelligent way of evaluating corners.

Linear programming can similarly be used to model minimization problems. The objective function would move down in a minimization model.

2.2.2 Sensitivity analysis

Perhaps of even more interest than the optimal solution is how that optimal solution responds to changes in the problem parameters. The study of the effect of changing parameters on the optimal solution is sensitivity analysis.

Sensitivity analysis provide vital information to answer questions of managerial interests such as: "what is the bottleneck resource?" "How much more resource is required to meet a certain demand?" "How much should a company be willing to pay for additional resource?"

Consider once again the example of Colonel Motors. Changes in the profitability (coefficients in the objective function) of cars and SUVs change the slope of the objective function. At some point, the optimal solution will jump to a different corner of the feasible region. See Figure 2.10.

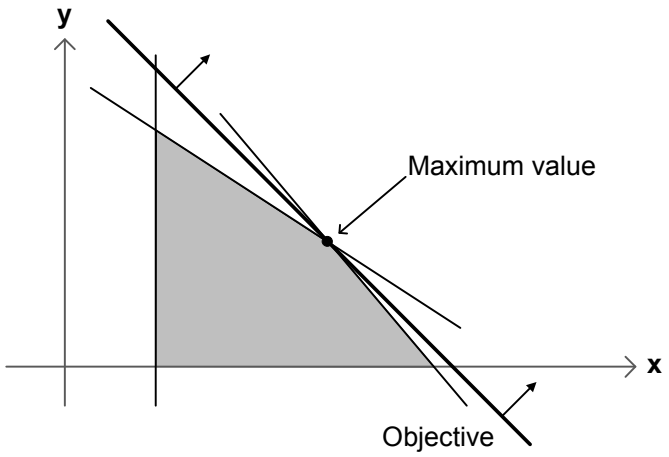


Figure 2.10: Different optimal solution due to change in objective coefficients

A sensitivity report for the CM example is show in Table 2-2. The current value of the objective coefficient on x is 4 (representing the unit profit of $4000 per car). The

'allowable increase' of that coefficient is calculated to be 5. This means that the unit profit of each car must increase by at least $5000 for the optimal solution to jump to the corner shown in Figure 2.10.

Table 2-2: Sensitivity report of optimal solution for CM

Adjustable Cells					
Name	**Final Value**	**Reduced Cost**	**Objective Coefficient**	**Allowable Increase**	**Allowable Decrease**
x	300	0	4	5	1E+30
y	525	0	12	1E+30	6.666666667
Constraints					
Name	**Final Value**	**Shadow Price**	**Constraint R.H. Side**	**Allowable Increase**	**Allowable Decrease**
Parts production	30000	0.3	30000	6636.363636	21000
Assembly	8175	0	10000	1E+30	1825
Demand	300	-5	300	700	300

This sensitivity report also reveals the parts production capacity to be the limiting resource. The shadow price on the parts production constraint is the improvement in the objective function resulting from adding one hour of parts production capacity. Therefore, that shadow price is the maximum CM should be willing to pay for an additional hour of production capacity. The shadow price on assembly hours is zero because the assembly line is not a limiting resource. In fact, the 'allowable decrease' value of the assembly constraint shows that assembly lines are sitting idle for 1825 hours each month. CM can use this information to redeploy resources.

Sensitivity analysis is an extremely useful managerial tool. Therefore, any scheduling optimization system should provide some sort of sensitivity information.

2.2.3 Integer programming

Integer programming (IP) models are LP models that require some or all of its decision variables to have integer values. For example, a product shipping problem must use an IP model because one cannot ship half a car or transport half a person. A model where all decision variables must be integers is a pure integer problem. A model where

only some decision variables need to be integer is a mixed integer problem (MIP). The difference between LP and IP problem is illustrated graphically in Figure 2.11.

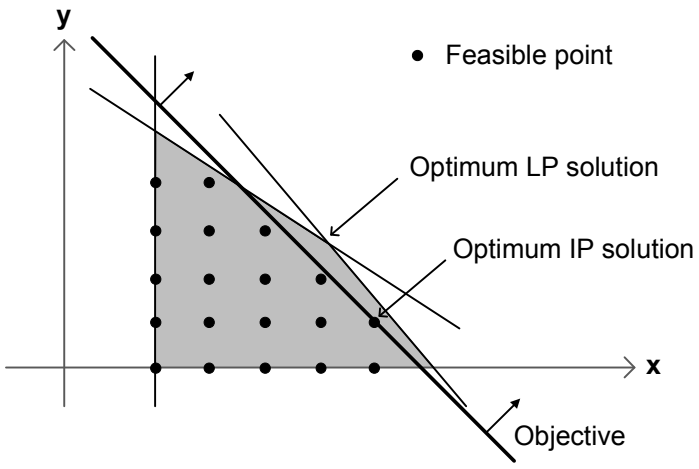


Figure 2.11: Graphical representation of IP model

Integer constraints reduce the LP feasible region to only points that have integer decision values. In all likelihood, the optimal solution will be different from the optimal of the same problem without the integer constraints.

A particularly useful type of IP is the binary 0-1 IP. In a binary 0-1 IP, some decision variables are constrained to values 0 or 1 to represent 'do' or 'don't' decisions. 0-1 decisions make possible the application of integer programming to scheduling problems.

2.2.4 Applying integer programming to scheduling problems

Consider the following simple job sequencing example: a manufacturing department uses a single machine to process three jobs. [116] Processing time and due date for each job are given in Table 2-3. Due dates are measured from the start time of the first scheduled job, which is considered day zero.

Table 2-3: Job sequencing example parameters

Job	Processing time (days)	Due date (days)	Late penalty $/day
1	5	25	19
2	20	22	12
3	15	35	34

Note that it is impossible to process all jobs before their due times. The objective of the problem is then to schedule the jobs in a sequence that minimizes the total late penalty. This is a simple example of the single processor job-shop scheduling problem.

A solution to this sequencing problem can be obtained using integer programming.

Decision variables:

$$X_j = Starting\ day\ of\ job\ j$$

$$Y_{ij} = \begin{cases} 1 & if\ job\ i\ starts\ before\ job\ j \\ 0 & otherwise \end{cases}$$

$$E_j = Number\ of\ days\ job\ j\ is\ early$$

$$L_j = Number\ of\ days\ job\ j\ is\ late$$

Data:

$$D_j = Due\ date\ of\ job\ j$$

$$P_j = Processing\ time\ of\ job\ j$$

$$C_j = Late\ penalty\ of job\ j$$

Objective Function:

$$\text{Min} \quad \sum_j C_j L_j$$

Constraints:

Non-interference

$$\text{C1} \quad X_j \geq X_j + P_i - (1 - Y_{ij})M \qquad \forall i \forall j \neq i$$

$$\text{C2} \quad X_i \geq X_j + P_j - MY_{ij} \qquad \forall i \forall j \neq i$$

Link start date with due date

C3 $X_j + P_j - D_j - L_j + E_j = 0 \qquad \forall j$

Non-negativity and binary integer

C4 $X_j, E_j, L_j \geq 0$

C5 $Y_{ij} \in (0,1)$

The binary integer variable Y_{ij}, makes this problem an integer problem. This binary variable is integral to the feasibility of the model as demonstrated by its use in the non-interference constraints.

The objective function seeks to minimize the total late penalty from all jobs.

Constraint C3 links the starting date of a job with its due date through the days early and days late variables. Constraints C4 and C5 are simply non-negativity and binary constraints standard to most binary integer problem.

More interesting are the non-interference constraints. They are the defining characteristics of a job sequencing problem. Constraints C1 and C2 use the constant *M*, which is simply a very large value. The use of *M* with a binary variable effectively gives the model the ability to turn constraints on and off. When $Y_{ij} = 1$, constraint C1 is binding because the term $(1 - Y_{ij})M$ becomes zero and C1 becomes $X_j \geq X_i + P_i$. This means if job *i* is processed before job *j*, job *j* cannot begin until job *i* is complete. Constraint C2 is rendered non-binding because the term MY_{ij} takes on a large value and C2 becomes $X_i \geq A\ negative\ value$, which will always be satisfied since X_i is also constrained to only have positive values. Similarly, if job *j* starts before job *i* (Y_{ij} = 0), constraint C2 becomes binding and constraint C1 is relaxed. In that case, job *i* will not be allowed to begin until job *j* is complete. The non-interference constraints resolve conflicts between jobs.

Typically, the constant *M* should have the smallest value required to be effective in relaxing constraints. To relax either constraints C1 or C2, M must be greater than or equal to maximum value that X_j can take. In this example, the maximum value for X_j, i.e. the latest day that job *j* can be scheduled is 35 (with jobs 2 and 3 scheduled ahead of it).

Therefore, *M* must be at minimum 35 to be effective. Ideally, *M* should be set to 35 to minimize the feasibility region of the problem, which minimizes the run time of the algorithm.

This job sequencing problem is small and simple enough to be solved using the standard Excel Solver. Solver managed to solve this problem in approximately 1 second. The optimal sequencing of jobs is summarized in Table 2-4.

Table 2-4: Optimal job sequence that minimizes late penalty

Job sequence	Starting date	Days early	Days late	Late penalty
2	0	2	0	$0
1	20	0	0	$0
3	25	0	5	$170

This demonstration of MIP performance provides motivation for the application of mixed integer programming to the scheduling and optimization of medical procedures.

2.2.5 Solving IP problems: branch and bound

Unfortunately, the introduction of integer constraints makes IPs very difficult to solve. An elegant, general solver like Simplex does not exist for IPs. There are however, several proven techniques: branch and bound, heuristics, implicit enumeration, and cutting plane. Branch and bound is a 'divide and conquer' technique that breaks the problem down to many smaller pieces that are easier to solve. Variations on the branch and bound method exist for different types of problems. Heuristics are search algorithms that are typically uniquely designed for each specific problem. Implicit enumeration is a technique that can intelligently guide the branch and bound method in certain applications. Cutting plane method 'trims' the problem by adding constraints to reduce the feasible region to a shape and size that is easier to solve. The general cutting plane method is the Gomory cut. However, one may be able to obtain higher quality, custom cuts that exploit nuances of a specific problem. Some techniques are better suited for one type of problems than another. A hybrid of cutting plane and brand and bound methods, called branch and cut, is the most common technique for solving IPs.

The branch and bound method finds the optimal solution to an IP by continuously dividing that IP into sub problems and efficiently enumerating points within feasible

regions of those sub problems. Before discussing the method further, some background and observations must be noted. First of all, a key component of branch and bound algorithm is the solving of the LP relaxation of an IP problem. An LP relaxation is identical to its respective IP except that all integer constraints on its decision variables are removed (relaxed). Secondly, feasible points of an IP are a subset of the feasible region of its respective LP relaxation. Therefore, the optimal solution or indeed any solution to the IP can only be found within the feasible region of its LP relaxation. This means that the optimal objective value of an IP cannot be better than the optimal objective value of its LP relaxation. In other words, the optimal solution to an IP is bounded by the optimal solution of its LP relaxation. Thirdly, if an LP relaxation has a solution where all decision variables have integer values, then that optimal solution to the LP relaxation is also the optimal solution to its respective IP. The relationship between an IP and its respective LP relaxation forms the foundation of the branch and bound method. [117]

To illustrate the branch and bound (BnB) solver algorithm, consider the Colonel Motors (CM) example once again. Adding integer constraints to the CM example is necessary since CM cannot sell fractions of cars or SUVs. The BnB method begins by solving the LP relaxation of the original IP.

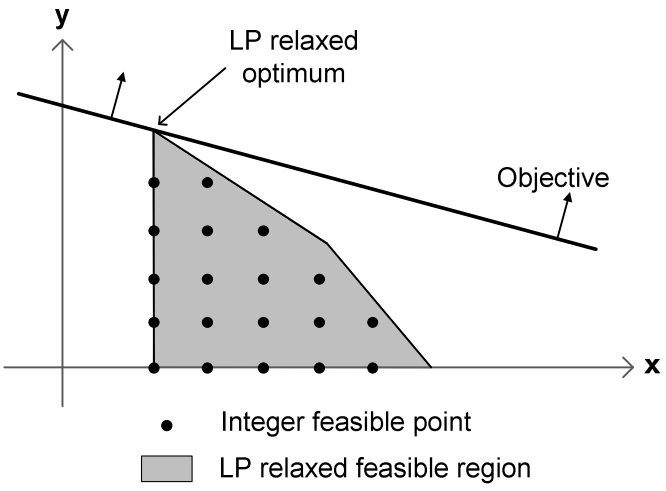


Figure 2.12: Solving the LP relaxation of the CM example

The optimal solution to the LP relaxation is *x=300, y=562.5* with an objective value *z=$7.950million*. The LP relaxed objective value of the original problem

establishes the upper bound for the optimal solution of the IP problem. That is, the optimal profit for CM is guaranteed to be less than *$7.95million*. The variable *x* has an integer value and therefore satisfies the integer constraint. The value of *y* however, violates the integer constraint. The next step in BnB method is to divide the original IP into sub-problems by "branching" on the non-integer variable. That is, BnB creates two copies of the original problem each with new constraints ("bounds") on the value of *y*. Sub-problem SP1 will restrict *y* to values greater than or equal to 563. Sub-problem SP2 will limit *y* to values less than or equal to 562. This branching and bounding eliminates the portion of the feasible region that cannot possibly contain the optimal solution. That is, due to the integer constraint, variable *y* may not take on any value between 562 and 563. The next step in BnB is to solve the LP relaxation of sub-problems SP1 and SP2.

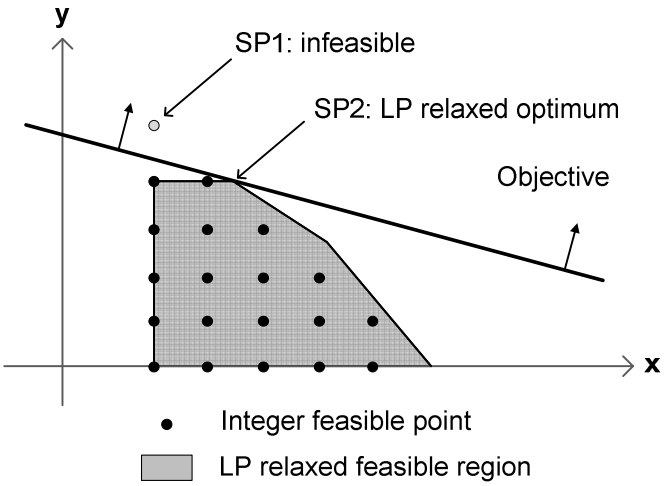


Figure 2.13: Solving LP relaxations of sub problems SP1 and SP2

Sub-problem SP1 is infeasible because its values violate constraints of the original problem. LP relaxed solution of sub-problem SP2 is *x=300.8, y=562* and *z=7.947million*. Branching and bounding of the original problem eliminated the original LP relaxed solution from the feasible region. Consequently, the upper solution bound is no longer valid and must be updated with the best LP relaxed objective value of the two sub-problems which is $7.947 million is this case.

The progress of the BnB solver is tracked using a solution *tree*. Sub-problems are nodes in the tree, the branching of variables create arcs that lead to other sub-problems (nodes). The solution tree for the CM example so far looks like the following:

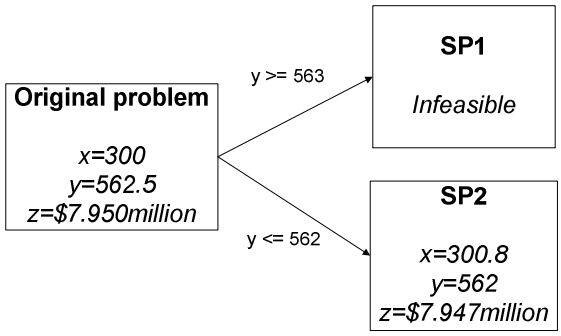


Figure 2.14: BnB solution tree of CM example so far

The branching and bounding continue until a sub-problem is found whose LP relaxed solution has all integer values. That solution is called an incumbent and represents the best feasible solution found so far. The incumbent establishes the lower bound for the original IP problem because there is no need to investigate sub-problems with worse objective values than the current incumbent. Once an incumbent is found, the BnB solver can begin eliminating (called fathoming) nodes from the solution tree. Those nodes with objective values less than the incumbent objective value are fathomed. If a better integer feasible solution is found, the incumbent gets updated which tightens the lower solution bound on the problem. The tightening of both the upper and lower solution bounds of the problem narrows the focus of the BnB solver. BnB continues until all except the incumbent node are fathomed. The remaining incumbent node is then the optimum solution to the original IP problem.

The optimal solution to the CM example is *x=300, y=562* and *z=$7.944million*. The complete solution tree in Figure 2.15 shows the progress of the BnB solver algorithm.

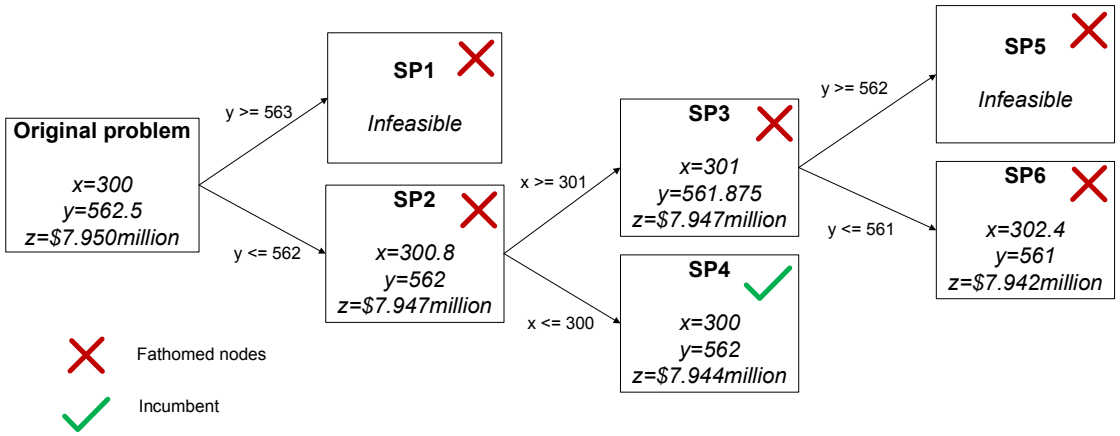


Figure 2.15: Complete solution tree for CM example

Mixed integer programming problems are solved using branch and bound exactly like pure integer problems except that the solver only branches on the variables that have integer constraints.

Branch and bound is an exact solving method. That is, by enumerating through the entire solution tree and fathoming all nodes except the best incumbent, BnB guarantees the optimality of the solution that it finds.

The CM example is very simple and can be solved easily and quickly using the BnB method. However, real life problems rarely involve only two decision variables. As the number of variables increase, the size of the BnB solution tree grows exponentially. Enumerating through exponentially growing solution trees is computationally expensive. This is the curse of dimensionality. Unfortunately, the use of BnB to solve real world problems is typically time-consuming. A user of BnB to solve an IP has the option of terminating the solving process before the optimal solution is found. By doing so, the user sacrifices the optimality guarantee and takes the best incumbent as the problem solution. Even though optimality is not guaranteed, the upper solution bound is known. Therefore, the user can calculate, with confidence, how far the incumbent solution is from the potentially optimal solution. In some applications, the reduction in solver run time may be worth sacrificing the guarantee of optimality.

The use of cutting planes is one other method of minimizing solver run time. Cutting planes add additional constraints to sub-problems to shape the feasible region to

improve solvability. For example, the common Gomory cut method adds constraints that 'cut' away fractional portions of variables to shape the feasible region such that its corners contain feasible integer solution points. Recall that solutions to LP relaxations of sub-problems are always found at the corners. Pruning the feasible region and leaving integer solutions at its corners make those integer solutions easier to find by the solver. Therefore, the cutting plane method minimizes the solution tree and theoretically improves the solver performance. [117]

A branch and bound solver that uses cutting planes to speed up its solving process is called a branch and cut solver. The branch and cut solver is currently the best, the fastest, and most efficient exact solver for IP problems and is the method of choice for solving the MIP scheduling model in this thesis.

2.2.6 Solving IP problems: heuristics

There is a class of problems called nondeterministic polynomial (NP) problems for which no known, efficient, polynomial solving algorithms exist. A subset within NP problems, called NP-hard problems are extremely difficult to solve. An example of an NP-hard problem is the classic travelling salesman problem. [117] For such NP-hard problems that are too difficult to solve using exact methods such as branch and bound, heuristic techniques can be used to quickly find good solutions. Heuristics are approximation techniques or algorithms that exploit problem structures to quickly produce high quality solutions. Heuristics sacrifice the optimality guarantee in exchange for solving speed.

Heuristics typically need to be designed to fit each specific application. However, general frameworks or strategies exist to guide heuristic design. The simplest framework uses some form of a greedy algorithm. The OR booking example discussed in section 1.3.4 uses a greedy heuristic algorithm. Recall in Ozkarahan's OR booking system, each operating room is first loaded with longest duration procedures to fit as many procedures as possible into each OR. Then the schedule in each OR is reversed so that shortest procedures are carried out first to minimize wait time of patients. A greedy heuristic can also be applied to the job-shop scheduling problem. To minimize tardiness of jobs, a greedy heuristic schedules jobs in order of their due-date. That is, jobs that are due early

are scheduled early. Greedy heuristics are extremely efficient. Their weakness however, is that for the same problem, a greedy heuristic can only produce one solution. Greedy heuristics do not explore the solution space. One way to overcome this single-solution weakness of greedy heuristics is to add some sort of exchange algorithm into the heuristic. For example, run the greedy algorithm on the job-shop scheduling problem several times, each time randomly switching the order of some jobs. This gives the heuristic ability to examine and choose from several solutions.

More sophisticated frameworks are based on intelligent search techniques developed by the artificial intelligence (AI) community. Three such frameworks are *simulated annealing, genetic search,* and *tabu search.* [117] Simulated annealing (SA) simulates the process by which atoms in a cooling system reach thermal equilibrium. SA begins at the *melting temperature* with an initial solution. The initial solution can be randomly generated or through a greedy heuristic. That initial solution is then run through a series of changes called a *cooling schedule* until a set temperature is reached. Each change in the cooling schedule can be random or specifically designed. The number of steps in the cooling schedule is typically designed to fit specific problems. A new solution at each step in the cooling schedule is evaluated and accepted or rejected based on its objective value and a probabilistic factor determined by the temperature. If a new objective value is better than the current best objective value than the new value is accepted. If a new objective value is worse than the current best objective, it still has a chance of being accepted. The reason for accepting inferior solutions is to encourage exploration of the solution space to avoid getting stuck in local optima. The probability of accepting an inferior solution is a function of temperature and decreases exponentially as the temperature falls. In other words, the heuristic encourages exploration of solution space earlier on but focuses the search towards the end of the process. The simplicity of SA and its ease of implementation make SA an attractive heuristic framework.

Genetic search is an evolutionary search technique that is inspired by the process of evolution through natural selection. Genetic algorithms (GA) begin with a population of randomly generated chromosomes. Chromosomes are encodings of potential solutions. In the case of the job-shop scheduling problem, each chromosome could represent a different ordering of jobs. A fitness function evaluates fitness of

chromosomes based on the objective value of the associated solution. In the case of the job-shop scheduling problem, a chromosome that results in a shorter makespan is fitter than those that result in longer makespans. The GA then evolves the population by selecting, recombining, evaluating, and replacing chromosomes. The typical GA generates a new population in each generation by mating chromosomes from the previous generation. The principle of 'survival of the fittest' is employed so that fitter chromosomes have higher chances of mating. Mating or reproduction is done by crossing over genetic information of parent chromosomes at some random point so that offspring chromosomes have pieces of information from each parent and therefore encode different (potentially better) solutions. See example of such a crossover in Figure 2.16. Chromosomes may also mutate by randomly changing bits of its genetic information. The evolution process is repeated until certain termination criteria are met. Typically, the evolution is stopped when the population converges on a solution or if the average fitness of the population doesn't improve for a number of generations. The general GA framework presented here is very versatile and can be fully customized to suit different problems. The user has full control over the population size, stopping criteria, crossover techniques, mutation probability, and the fitness function. A major advantage of GA is that the fitness function need not be linear, differentiable, or even continuous. This gives GA the potential to solve difficult, non-linear problems.

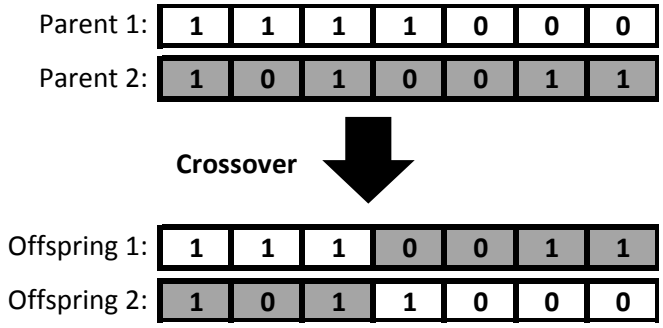


Figure 2.16: Example crossover of chromosomes

Tabu search is not based on natural or physical process. Rather, it emulates the psychology of decision making. Tabu search uses both short term and long term memory to intelligently search the solution space. Long term memory 'remembers' and focuses the search on the most promising neighborhoods in the solution space. Short term

memory ‘remembers’ where it has searched and avoids getting stuck in a local optimum. Tabu search begins with a candidate solution chosen either randomly or is an elite (good) solution taken from long term memory. A list of possible changes or moves is built for the candidate solution. In the case of the job-shop scheduling problem, a candidate solution is the ordering of jobs. Moves can be made by rearranging jobs to create a different ordering. For a set number of iterations, the algorithm evaluates the list of possible moves, makes the move that result in a better solution then evaluates the list again. The algorithm ‘remembers’ moves made during this evaluation process to avoid getting stuck cycling around a local optimum. The algorithm is repeated for any number of other candidate solutions. The best solution among the candidate solutions is chosen as the optimal solution. Tabu search has strong parallel processing potential. Candidate solutions can be evaluated by separate processors simultaneously. Though simplistic at first glance, tabu search can be effective for solving certain NP problems.

2.2.7 Linear programming software

Working with linear programming problems requires two distinct types of software: modeling and solver software. Solvers can only understand computer modeling languages such as AMPL, GAMS, and MPS among others. Modeling software help users express their LP problem in those computer modeling languages. Many commercial packages bundle the modeling and solving components together for ease of use. Commercial packages such as Premium Solver, GAMS, and LINGO are very powerful. However, they are not considered for this research due to their cost.

Free, open-source solvers are excellent alternatives to commercial packages. The drawback of open-source solvers is that they are more difficult to use than commercial solvers. Open-source solvers typically do not have easy to use GUI frontends. The user can only control solver parameters through console commands or by writing customized programs. Most open-source packages have only the solver component and leave the modeling to the user. Nevertheless, many open-source solver packages have high quality performance. The open-source solvers considered for this research are CBC, CLP, SYMPHONY, and GLPK. The Gnu Linear Programming Kit (GLPK) solver is the solver of choice for this research due to its high performance, high degree of

customizability, and relative ease of use. See Appendix B for description of the GLPK solver package.

No suitable open-source modeling software was found for this research. Therefore, a custom modeling program was written to model the medical procedures scheduling problem in the Mathematical Programming System (MPS). The custom modeling program is called *mpsWriter*. See Appendix A for a detailed description of the MPS modeling system. *mpsWriter* simply translates the MIP model logic and parameters into the MPS file format.

Integer programming scheduling models in this thesis will be presented in mathematical notation. The reader is to assume that models are converted into the MPS format by *mpsWriter* then solved using the GLPK branch and cut solver. Unless otherwise stated, the GLPK solver is run with all default options so that it behaves as a standard branch and cut solver. The numerical solutions found by GLPK will be presented in graphical format for maximum clarity.

Chapter 3: Mixed integer programming scheduling model

This chapter develops a novel mathematical formulation of the medical procedures scheduling problem (MPSP) described in section 1.2.1. A Mixed integer programming (MIP) model is first developed for very simple procedures to illustrate the core structure and functionality. The model is then enhanced to schedule more complex procedures described in section 2.1.2. The model is then enhanced further to schedule the even more complex and flexible haemodialysis procedures described in section 2.1.3. This chapter will also discuss the scalability of the MIP scheduling model.

3.1 Scheduling simple procedures – the simple MPSP model

3.1.1 Mathematical representation of simple procedures model

Before a MIP scheduling model can be developed, procedures must first be represented mathematically. Consider the following five very simple procedures (P_i):

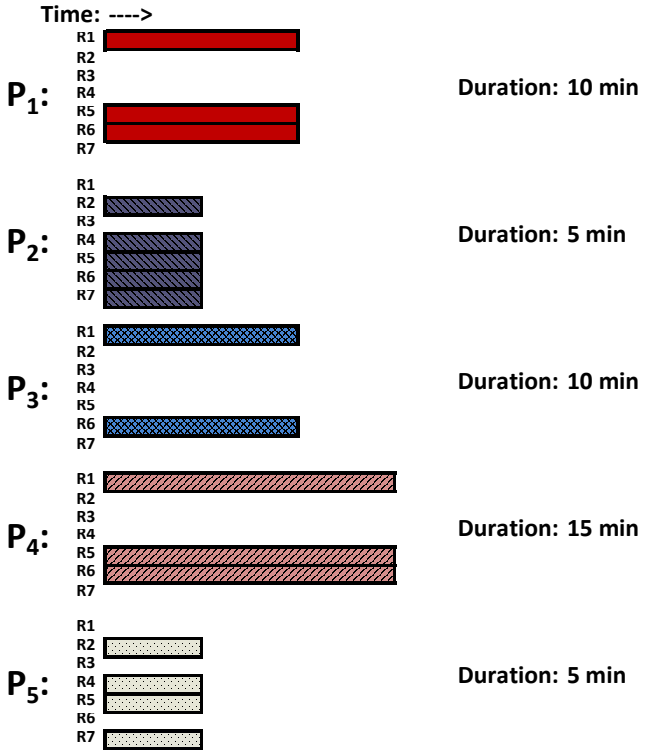


Figure 3.1: Simple example procedures

The information needed to fully represent each procedure are the procedure resource requirement and the procedure duration. Therefore, let us define a binary matrix R_{ri} to store resource requirements and define vector d_i to store procedure durations.

$$R_{ri} = \begin{cases} 1 & if\ resource\ r\ is\ required\ for\ procedure\ i \\ 0 & otherwise \end{cases}$$

$$d_i = Duration\ of\ procedure\ i$$

Durations of procedures are measured and captured in an arbitrary time unit. This time unit can represent minutes, seconds, hours or whatever denomination of time the user wishes to define. For the remainder of this thesis, the time unit will be minutes. For scheduling purposes, one needs to know the due times of procedures and to specify the time frame into which procedures are to be scheduled. The time frame or scheduling period is specified by p. Procedure due times are stored in a vector dt_i.

$$p = Scheduling\ period$$

$$dt_i = Due\ time\ of\ procedure\ i$$

The scheduling period p can be any arbitrary length of time. The user may choose to set p as the length of one shift, one day, or one week etc. The value dt_i is measured as minutes from the beginning of the scheduling period p. For example, $dt_i=0$ means procedure i is due at the beginning of the scheduling period.

Priority of each procedure is captured using two vectors: b_i and w_i.

$$b_i = Benefit\ of\ scheduling\ procedure\ i$$

$$w_i = Late\ penalty\ of\ procedure\ i$$

The value b_i captures how important it is to schedule procedure i into the current scheduling period. In cases where the scheduling period is too short to accommodate all procedures, procedures with lower b_i values should not be scheduled to make room for procedures with higher b_i values. The total benefit of the scheduled procedures is a measure of schedule quality and can be used as an objective to guide a scheduling system. The value w_i captures the priority of procedure i within the scheduling period. That is, of the procedures that are scheduled, those with higher wait time penalty w_i should be scheduled earlier than those with lower wait time penalty. The total lateness

penalty incurred by the scheduled procedures represents the impact of patient wait time and therefore also contributes to schedule quality. A scheduler should seek to minimize lateness penalty.

Thus far, the mathematical model of simple procedures shown in Figure 3.1 can be summarized in the following figure:

R_{ri}

	i=1	i=2	i=3	i=4	i=5
r=1	1	0	1	1	0
r=2	0	1	0	0	1
r=3	0	0	0	0	0
r=4	0	1	0	0	1
r=5	1	1	0	1	1
r=6	1	1	1	1	0
r=7	0	1	0	0	1

	i=1	i=2	i=3	i=4	i=5
d_i	10	5	10	15	5
dt_i	0	0	0	0	0
b_i	1	1	1	1	1
w_i	1	1	1	1	1

p	30

Figure 3.2: Parameters for mathematical model of simple example procedures

At the current stage of development, all procedures are due at the beginning of the scheduling period and all have the same priority. The scheduling period is set to 30 minutes. Note that although overlap between procedures is possible, the scheduling period is still too short to accommodate all procedures. This short scheduling period was set deliberately to test a scheduler's ability to decide which procedures to schedule and which ones not to schedule.

The next step is to develop a MIP model to schedule the simple example procedures.

3.1.2 MIP formulation of the simple MPSP model

A scheduling model must make two decisions for each procedure: 1) whether or not a procedure can fit into the scheduling period and 2) what time within the scheduling period should the procedure be scheduled at. Therefore, the first two decision variables for the MIP model are:

$$ps_i = Starting\ time\ of\ procedure\ i$$

$$x_i = \begin{cases} 1 & if\ procedure\ i \text{ can fit within the scheduling period} \\ 0 & otherwise \end{cases}$$

Additional decision variables are needed to track the lateness of procedures as well as the ordering of procedures.

$l_i = Lateness\ of\ procedure\ i$

$$y_{i,i2} = \begin{cases} 1 & if\ procedure\ i\ is\ scheduled\ before\ procedure\ i2 \\ 0 & otherwise \end{cases}$$

$$\forall \mathrm{i}, \mathrm{i2} = 1, 2, 3, \ldots \mathrm{n}$$

The MPSP problem has two competing objectives. Recall the scheduling problem defined in section 1.2.1; problems A) through D) can be summarized as an objective to maximize the total benefit of a schedule.

Max $\sum_i b_i x_i$ Equivalent to Min $\sum_i -b_i x_i$

The user can adjust benefit (b_i) values of individual procedures to address problems A) through D). Problem E) sets an objective to minimize the patient wait time.

Min $\sum_i w_i x_i$

These two objectives mimic the scheduling logic of a manual scheduler. One would like to schedule as many important procedures as possible without making patients wait excessively. Combining the two objectives, the objective function of the MIP scheduling model is:

Min $\sum_i (w_i l_i - b_i x_i)$

The binary decision variable x_i adds non-linear behaviour to the scheduling problem. In order to model the problem using linear programming, one must linearize the non-linear behaviour of the problem. The first two constraints of the scheduling model are such linearizations. They deal with the scheduling period limitation. The first constraint dictates that if the scheduler decides to schedule procedure *i* within scheduling period *p*, then the scheduler must ensure that procedure *i* fits completely into *p*. That is, the starting time of procedure *i* must be less than the latest possible start time for procedure *i*. The latest possible start time of procedure *i* is length of the scheduling period *p* less the duration of procedure *i*. The first constraint does not apply if the scheduler decides not to schedule procedure *i* within scheduling period *p*. The logic of the first constraint looks quite simple:

$$ps_i \leq p - d_i \qquad if \quad x_i = 1$$

However, x_i is a variable. Its value is dynamic and can change during the solving process. Unfortunately, linear programming does not allow dynamic conditional constraints. Therefore, one must use clever design techniques to enable and disable constraints when necessary. The use of *M*, first introduced in section 2.2.4, is one such clever technique. Using *M*, the first constraint is defined as:

C1 $\quad ps_i + d_i - (1 - x_i)M \leq p$

Constraint C1 is relaxed and tightened depending on the value of x_i. If $x_i=1$ then the $(1 - x_i)M$ term becomes zero and constraint C1 becomes: $ps_i + d_i \leq p$ which is the desired constraint. If $x_i=0$, then constraint C1 becomes: $ps_i + d_i - M \leq p$, which will always be true since *M* is simply a large constant. Essentially, constraint C1 is relaxed when $x_i=0$.

The second constraint in the scheduling model dictates that if the scheduler decides not to fit procedure *i* into scheduling period *p* then it must ensure the starting time of procedure *i* is outside of *p* so as not to conflict with procedures scheduled within *p*:

C2 $\quad ps_i + px_i \geq p$

Similar to constraint C1, constraint C2 is also relaxed and tightened depending o the value of x_i. If $x_i=0$, i.e. procedure *i* cannot fit into *p*, C2 becomes $ps_i \geq p$, forcing procedure *i* to be scheduled outside of the scheduling period. If however, $x_i=1$, i.e. procedure *i* is scheduled, C2 becomes $ps_i + p \geq p$, which will always be true. Constraint C2 is essentially relaxed when $x_i=1$.

Procedures that cannot fit into the scheduling period are scheduled outside of the scheduling period and treated as if they are not part of the schedule. Only procedures scheduled within the scheduling period *p* will be presented to the user as the problem solution. This technique models non-linear behaviour but maintains the linear nature of the MIP scheduling model.

The third constraint links the starting time of a procedure with its due time via its lateness variable.

C3 $ps_i - l_i = dt_i$

The next two constraints resolve resource conflicts. The logic for resolving conflicts is to ensure that between any two procedures that share the same resource, one procedure does not start until the other procedure is complete. For example, procedure *i* is to be scheduled ahead of procedure *i2*, and both share same resources. The earliest possible start time of procedure *i2* is the starting time of procedure *i* plus the duration of procedure *i*. Starting time of procedure *i2* must be later than its earliest possible start time.

C4 $ps_{i2} - ps_i + (1 - y_{i,i2})M \geq d_i \quad \forall i, i2: (R_{ri} = R_{ri2} = 1 \ \cap \ i \neq i2)$

C5 $ps_i - ps_{i2} + My_{i,i2} \geq d_{i2} \quad \forall i, i2: (R_{ri} = R_{ri2} = 1 \ \cap \ i \neq i2)$

Constraints C4 and C5 are relaxed and tightened depending on the value of $y_{i,i2}$. If procedure *i* is scheduled ahead of *i2*, i.e. $y_{i,i2}=1$, then constraint C4 is tightened and becomes $ps_{i2} - ps_i \geq d_i$ while C5 is relaxed and becomes $ps_i - ps_{i2} + M \geq d_{i2}$. If, on the other hand $y_{i,i2}=0$ then C5 is tightened and C4 is relaxed. One must be careful in setting the value of *M*. The value of *M* must be large enough to effectively relax constraints when needed but small enough to minimize the search space size. For *M* to be effective, it must have a value larger than the makespan of the worst case schedule: one that ignores overlap opportunities and cascades procedures. The makespan of the worst case cascading schedule is simply the sum of durations of all procedures, 45 minutes in this example.

The final two constraints in the scheduling model are the non-negativity and integer constraints.

C6 $ps_i, l_i, x_i, y_{i,i2} \geq 0$

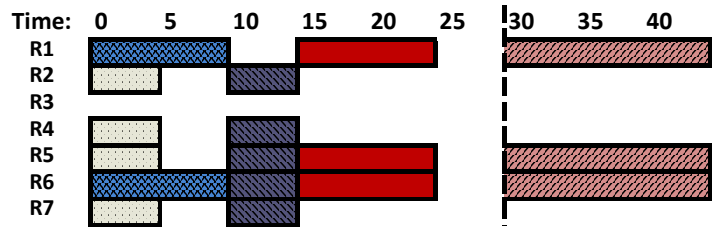
C7 $x_i, y_{i,i2} \in \{0,1\}$

The MIP model developed in this section is designated the *simple MPSP model*. Its full mathematical formulation can be found in Appendix C. The next section discusses solving of this simple MPSP model and how the optimal solution translates into a schedule.

3.1.3 Solving the simple MPSP model

The simple MPSP model is first converted into the MPS file format then solved using GLPK's branch and cut solver. See Appendix A for demonstration of converting linear programming model into the MPS file format. Specifically, see Table A-5 for the MPS file of the simple MPSP model example. See Appendix B for demonstration of the solving of linear programming models using the Gnu Linear Programming Kit (GLPK).

Solution to the simple MPSP model is the values of decision variables. Values of the starting time decision variable ps_i are all that are required to visualize the solution graphically. The optimal solution was found in less than 1 second and is shown in Figure 3.3



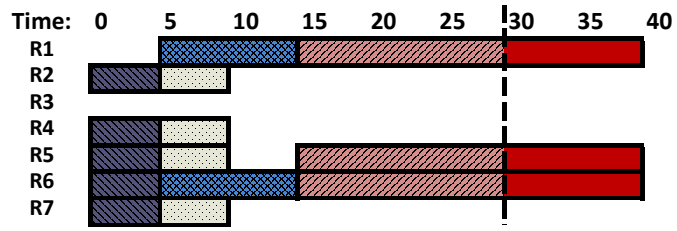
Obj val: 51

	i=1	i=2	i=3	i=4	i=5
ps_i	15	10	0	30	0
x_i	1	1	1	0	1
l_i	15	10	0	30	0
dt_i	0	0	0	0	0
b_i	1	1	1	1	1
w_i	1	1	1	1	1

Figure 3.3: Optimal solution to the simple MPSP model

Recall that the scheduling period is set at 30 minutes. The dashed line marks the end of the scheduling period. Procedure P_4 (or *i=4*) is just outside the scheduling period and is considered not scheduled. The simple MPSP model produced a conflict-free schedule that satisfies the scheduling objective function. It fit as many procedures into the scheduling period as possible and ordered the procedures to start as soon as possible to minimize total patient wait time. Procedures P_3 and P_5 were allowed to overlap because they do not share any resources.

The optimal solution can be manipulated by changing coefficients in the objective function. The effects on optimal solutions of changing objective coefficients are illustrated in Figure 3.4 and Figure 3.5.

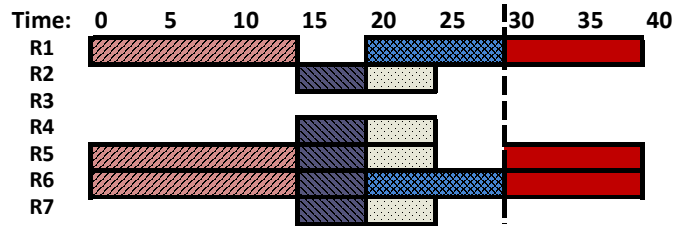


Obj val: 47

	i=1	i=2	i=3	i=4	i=5
ps_i	30	0	5	15	5
x_i	0	1	1	1	1
l_i	30	0	5	15	5
dt_i	0	0	0	0	0
b_i	1	1	1	5	1
w_i	1	1	1	1	1

Figure 3.4: Effect of increasing b_4 on the simple MPSP model's optimal solution

The example in Figure 3.4 gives procedure P_4 higher relative importance by increasing b_4 from 1 to 5. As a result, P_4 is scheduled within the scheduling period and another procedure (P_1) is bumped out. Now if one also increases the lateness penalty on procedure P4, that procedure takes on very high priority. The result is that procedure P_4 gets scheduled first, at the detriment of patient wait times in other procedures. The total wait time ($\sum l_i$) of scheduled procedures is 55 minutes in the example shown in Figure 3.5, much higher than the 25 minute total wait time in the example shown in Figure 3.4.



Obj val: 77

	i=1	i=2	i=3	i=4	i=5
ps_i	30	15	20	0	20
x_i	0	1	1	1	1
l_i	30	15	20	0	20
dt_i	0	0	0	0	0
b_i	1	1	1	5	1
w_i	1	1	1	5	1

Figure 3.5: Effect of increasing b_4 and w_4 on the simple MPSP model's optimal solution

The examples in Figure 3.4 and Figure 3.5 show that the priority of procedures can be adjusted to manipulate the optimal solution (schedule) of the model. The user has the freedom to tweak objective coefficients b_i and w_i to set relative priorities of all procedures.

The flexibility to manipulate procedure priorities to affect the optimal schedule and the ability to resolve resource conflicts are two major strengths of the simple MPSP model. Each procedure has two adjustable attributes: its relative importance (b_i) and its lateness penalty (w_i). Each attribute affects the procedure priority differently. Coefficient b_i is on the binary variable x_i and therefore has a stepwise effect on procedure priority. Coefficient w_i on the other hand, has a linear effect on priority proportional to tardiness of a procedure. Managing multiple procedures each with these two attributes is a fine balancing act. Much experience will likely be needed to fine tune the simple MPSP model for real world application.

Application of the simple MPSP model is limited to scheduling procedures that are described under the simple procedures model discussed in section 2.1.1. However, this model establishes the foundation for a more complex scheduling model. The next sections will extend the simple MPSP model into a more complex, flexible and practical scheduling model.

3.2 Scheduling complex procedures – the enhanced MPSP model

3.2.1 Mathematical representation of complex procedures

The next step in MIP model development is to add the ability to schedule complex procedures as discussed in section 2.1.2. The key to scheduling complex procedures is the breakdown of those complex procedures into simple activities. Recall example procedures presented in section 2.1.2, Figure 2.3 and Figure 2.4. They will be used here to aid the development of a more complex scheduling model.

Complex procedures can be broken down or discretized into simple activities separated at times during the procedure when resource requirements change. Such a discretization is shown in Figure 3.6.

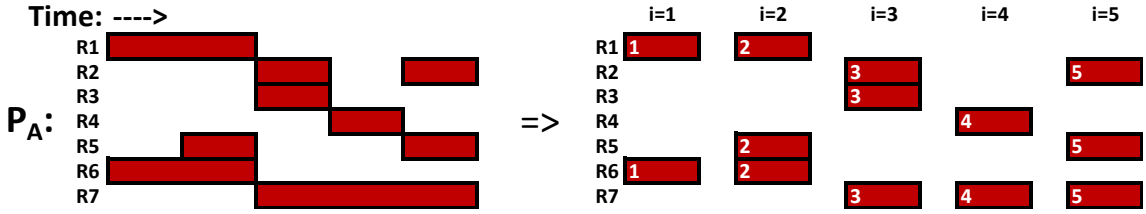


Figure 3.6: Discretizing procedure P_A into simple pieces/activities

To avoid confusion over indexing of procedures, full procedures will use alphabet index while activities within those full procedures will use a numerical index. For example, procedure P_A shown in Figure 3.6 is modeled using activities 1 through 5.

The relationships between activities are modeled by introducing a new workflow parameter: z_{ij}.

$$z_{ij} = \begin{cases} 1 & if\ activity\ i\ must\ immediately\ follow\ \text{activity}\ j \\ 2 & if\ activity\ i\ and\ j\ both\ belong\ to\ the\ same\ procedure \\ 0 & otherwise \end{cases}$$

The mathematical representation of activities of procedure P_A is shown in Figure 3.7.

R_{ri}

	i= 1	2	3	4	5
r=1	1	1	0	0	0
r=2	0	0	1	0	1
r=3	0	0	1	0	0
r=4	0	0	0	1	0
r=5	0	1	0	0	1
r=6	1	1	0	0	0
r=7	0	0	1	1	1

i=	1	2	3	4	5
d_i	10	10	10	10	10
dt_i	0	0	0	0	0
b_i	1	0	0	0	0
w_i	1	0	0	0	0

z_{ij}

	j= 1	2	3	4	5
i=1	2	2	2	2	2
i=2	1	2	2	2	2
i=3	2	1	2	2	2
i=4	2	2	1	2	2
i=5	2	2	2	1	2

Figure 3.7: Mathematical representation of activities of procedure P_A

The z_{ij} parameter ties activities together so that activities *i=1* to *i=5* are always scheduled together as a block to represent procedure P_A. The objective coefficients b_i and w_i on activities *i=2* to *i=5* are set to zero because the objective coefficients on activity *i=1* are enough to fully represent procedure P_A in the scheduling model. Any benefit or penalty on activities *i=2* to *i=5* would contribute excessively to benefit or penalty of procedure P_A.

One must be careful in setting values for the parameter z_{ij}. If for example, activity *i=2* needs to be scheduled immediately after activity *i=1*, i.e. $z_{21}=1$, then activity *i=1* cannot be scheduled after *i=2*. That is, z_{12} must equal 2. It is the responsibility of the user or modeler to ensure that this conflict does not arise. Otherwise, the resulting scheduling model will not be feasible.

The discretization of example procedures P_B through P_E is shown in Figure 3.8 through Figure 3.15.

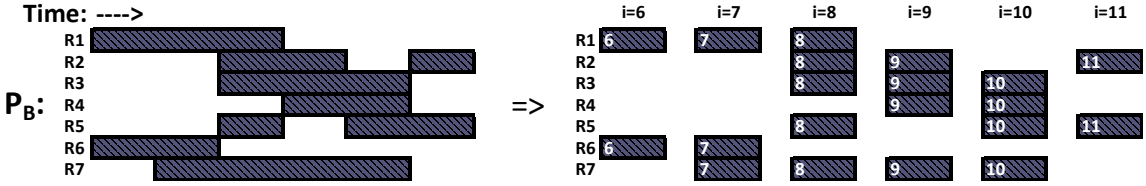


Figure 3.8: Discretization of procedure P_B

R_{ri}

	i= 6	7	8	9	10	11
r=1	1	1	1	0	0	0
r=2	0	0	1	1	0	1
r=3	0	0	1	1	1	0
r=4	0	0	0	1	1	0
r=5	0	0	1	0	1	1
r=6	1	1	0	0	0	0
r=7	0	1	1	1	1	0

	i= 6	7	8	9	10	11
d_i	10	10	10	10	10	10
dt_i	0	0	0	0	0	0
b_i	1	0	0	0	0	0
w_i	1	0	0	0	0	0

z_{ij}

	j= 6	7	8	9	10	11
i=6	2	2	2	2	2	2
i=7	1	2	2	2	2	2
i=8	2	1	2	2	2	2
i=9	2	2	1	2	2	2
i=10	2	2	2	1	2	2
i=11	2	2	2	2	1	2

Figure 3.9: Mathematical representation of activities of procedure P_B

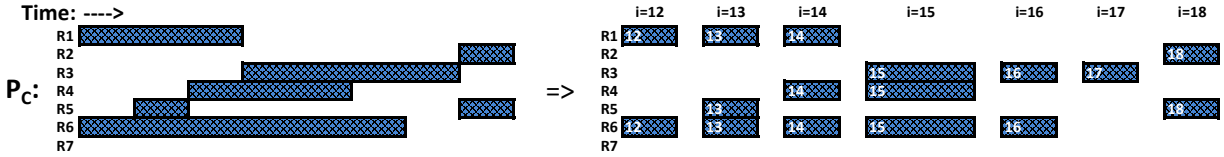


Figure 3.10: Discretization of procedure P_C

R_{ri}

	i= 12	13	14	15	16	17	18
r=1	1	1	1	0	0	0	0
r=2	0	0	0	0	0	0	1
r=3	0	0	0	1	1	1	0
r=4	0	0	1	1	0	0	0
r=5	0	1	0	0	0	0	1
r=6	1	1	1	1	1	0	0
r=7	0	0	0	0	0	0	0

	i= 12	13	14	15	16	17	18
d_i	10	10	10	20	10	10	10
dt_i	0	0	0	0	0	0	0
b_i	1	0	0	0	0	0	0
w_i	1	0	0	0	0	0	0

z_{ij}

	j= 12	13	14	15	16	17	18
i=12	2	2	2	2	2	2	2
i=13	1	2	2	2	2	2	2
i=14	2	1	2	2	2	2	2
i=15	2	2	1	2	2	2	2
i=16	2	2	2	1	2	2	2
i=17	2	2	2	2	1	2	2
i=18	2	2	2	2	2	1	2

Figure 3.11: Mathematical representation of activities of procedure P_C

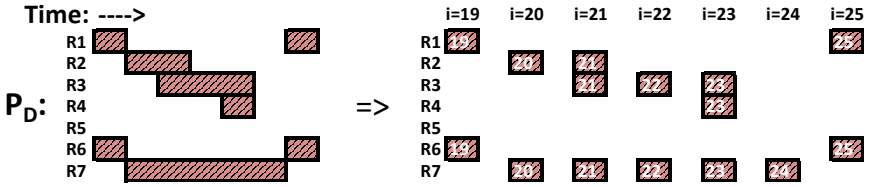


Figure 3.12: Discretization of procedure P_D

R_{ri}

	i= 19	20	21	22	23	24	25
r=1	1	0	0	0	0	0	1
r=2	0	1	1	0	0	0	0
r=3	0	0	1	1	1	0	0
r=4	0	0	0	0	1	0	0
r=5	0	0	0	0	0	0	0
r=6	1	0	0	0	0	0	1
r=7	0	1	1	1	1	1	0

i=	19	20	21	22	23	24	25
d_i	5	5	5	5	5	5	5
dt_i	0	0	0	0	0	0	0
b_i	1	0	0	0	0	0	0
w_i	1	0	0	0	0	0	0

z_{ij}

j=	19	20	21	22	23	24	25
i=19	2	2	2	2	2	2	2
i=20	1	2	2	2	2	2	2
i=21	2	1	2	2	2	2	2
i=22	2	2	1	2	2	2	2
i=23	2	2	2	1	2	2	2
i=24	2	2	2	2	1	2	2
i=25	2	2	2	2	2	1	2

Figure 3.13: Mathematical representation of activities of procedure P_D

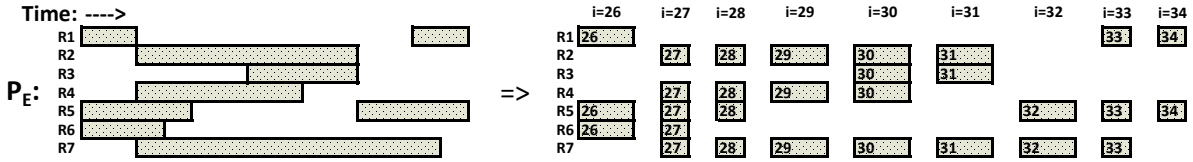


Figure 3.14: Discretization of procedure P_E

R_{ri}

	i= 26	27	28	29	30	31	32	33	34
r=1	1	0	0	0	0	0	0	1	1
r=2	0	1	1	1	1	1	0	0	0
r=3	0	0	0	0	1	1	0	0	0
r=4	0	1	1	1	1	0	0	0	0
r=5	1	1	1	0	0	0	1	1	1
r=6	1	1	0	0	0	0	0	0	0
r=7	0	1	1	1	1	1	1	1	0

i=	26	27	28	29	30	31	32	33	34
d_i	10	5	5	10	10	10	10	5	5
dt_i	0	0	0	0	0	0	0	0	0
b_i	1	0	0	0	0	0	0	0	0
w_i	1	0	0	0	0	0	0	0	0

z_{ij}

j=	26	27	28	29	30	31	32	33	34
i=26	2	2	2	2	2	2	2	2	2
i=27	1	2	2	2	2	2	2	2	2
i=28	2	1	2	2	2	2	2	2	2
i=29	2	2	1	2	2	2	2	2	2
i=30	2	2	2	1	2	2	2	2	2
i=31	2	2	2	2	1	2	2	2	2
i=32	2	2	2	2	2	1	2	2	2
i=33	2	2	2	2	2	2	1	2	2
i=34	2	2	2	2	2	2	2	1	2

Figure 3.15: Mathematical representation of activities of procedure P_E

Note that the z_{ij} matrices shown in the above figures are only those portions that relate to each procedure. The full z_{ij} matrix is shown in Figure 3.16 to emphasize that z_{ij} values between activities that do not belong to the same procedure must be 0.

z_{ij}	j=	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
	i=1	2	2	2	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	i=2	1	2	2	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	i=3	2	1	2	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	i=4	2	2	1	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	i=5	2	2	2	1	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	i=6	0	0	0	0	0	2	2	2	2	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	i=7	0	0	0	0	0	1	2	2	2	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	i=8	0	0	0	0	0	2	1	2	2	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	i=9	0	0	0	0	0	2	2	1	2	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	i=10	0	0	0	0	0	2	2	2	1	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	i=11	0	0	0	0	0	2	2	2	2	1	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	i=12	0	0	0	0	0	0	0	0	0	0	0	2	2	2	2	2	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	i=13	0	0	0	0	0	0	0	0	0	0	0	1	2	2	2	2	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	i=14	0	0	0	0	0	0	0	0	0	0	0	2	1	2	2	2	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	i=15	0	0	0	0	0	0	0	0	0	0	0	2	2	1	2	2	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	i=16	0	0	0	0	0	0	0	0	0	0	0	2	2	2	1	2	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	i=17	0	0	0	0	0	0	0	0	0	0	0	2	2	2	2	1	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	i=18	0	0	0	0	0	0	0	0	0	0	0	2	2	2	2	2	1	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	i=19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	2	2	2	2	2	2	0	0	0	0	0	0	0	0	0
	i=20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2	2	2	2	2	2	0	0	0	0	0	0	0	0	0
	i=21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	1	2	2	2	2	2	0	0	0	0	0	0	0	0	0
	i=22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	2	1	2	2	2	2	0	0	0	0	0	0	0	0	0
	i=23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	2	2	1	2	2	2	0	0	0	0	0	0	0	0	0
	i=24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	2	2	2	1	2	2	0	0	0	0	0	0	0	0	0
	i=25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	2	2	2	2	1	2	0	0	0	0	0	0	0	0	0
	i=26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	2	2	2	2	2	2	2	2
	i=27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2	2	2	2	2	2	2	2
	i=28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	1	2	2	2	2	2	2	2
	i=29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	2	1	2	2	2	2	2	2
	i=30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	2	2	1	2	2	2	2	2
	i=31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	2	2	2	1	2	2	2	2
	i=32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	2	2	2	2	1	2	2	2
	i=33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	2	2	2	2	2	1	2	2
	i=34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	2	2	2	2	2	2	1	2

Figure 3.16: Full z_{ij} matrix for activities

The next section revises the simple MPSP model to accommodate these more complex procedures.

3.2.2 MIP formulation of the enhanced MPSP model

This section begins the real contribution of this thesis. The complex procedures model presented in the previous section 3.2.1 cannot be scheduled using existing flow-shop scheduling or project scheduling models. This section builds on section 3.1.2 and continues to develop a novel mathematical formulation to model and schedule complex procedures.

The discretization of procedures into activities requires additional constraints in the scheduling model to manage the relationships between those activities. The introduction of the z_{ij} parameter makes management of those relationships possible. Two additional constraints are needed.

The first new constraint ensures that activities of the same procedure are scheduled in the correct order. That is, if activity *i* must be scheduled immediately after *j*, then the starting time of activity *i* must equal the starting time of activity *j* plus the duration of activity *j*.

C8 $\quad ps_i - ps_j = d_j \qquad \forall i,j: (z_{ij} = 1)$

The second new constraint ensures that activities of the same procedure are scheduled either all within the scheduling period or all outside of it. This prevents the partial scheduling of procedures.

C9 $\quad x_i - x_j = 0 \qquad \forall i,j: (z_{ij} = 1)$

The addition of constraint C8 makes it impossible for activities belonging to the same procedure to conflict with each other. Therefore, the resource conflict constraints for those activities are redundant. New conditions are added to the resource conflict constraints to only enforce them if $z_{ij}=0$. That is, enforce constraints C4 and C5 only for activities that do not belong to the same procedure and can potentially conflict.

C4 $\quad ps_{i2} - ps_i + (1 - y_{i,i2})M \geq d_i$

$\forall i, i2: (R_{ri} = R_{ri2} = 1 \ \cap \ z_{i2,i} = 0 \ \cap \ i \neq i2)$

C5 $\quad ps_i - ps_{i2} + My_{i,i2} \geq d_{i2}$

$\forall i, i2: (R_{ri} = R_{ri2} = 1 \ \cap \ z_{i2,i} = 0 \ \cap \ i \neq i2)$

The model developed in this section is designated the *enhanced MPSP model*. A full summary of its mathematical formulation can be found in Appendix D. The solving of the enhanced MPSP model is discussed in the next section.

3.2.3 Solving the enhanced MPSP model

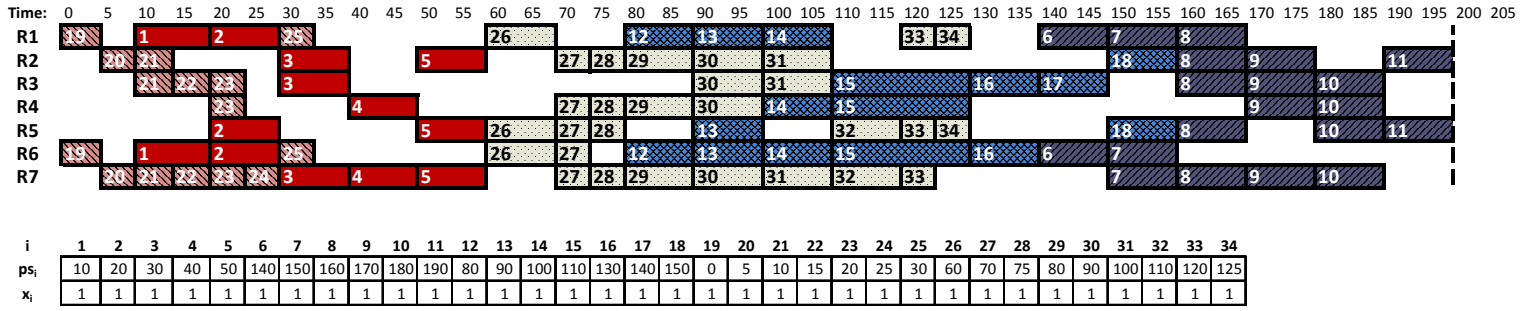
The procedures and activities defined in section 3.2.1 will be used as the examples to schedule. The sum of durations of all procedures is 295 minutes. Therefore the makespan of a cascading schedule would be 295 minutes. The scheduling period for this example is arbitrarily set at 200 minutes to test if the enhanced MPSP model can produce a better schedule with a shorter makespan than the schedule of cascading

procedures. The optimal solution to the enhanced MPSP model with scheduling period *p=200* is shown in Figure 3.17.

Surprisingly, the enhanced MPSP model was able to overlap procedures tighter than expected. As a result, all procedures were able to fit into the scheduling period. The scheduling period is then tightened to 150 minutes to force the scheduling model to decide which procedures to schedule and which to not. The optimal solution to the enhanced MPSP model with scheduling period *p=150* is shown in Figure 3.18. Procedures P_C and P_E could not fit into the short scheduling period. Consequently, P_C and P_E are placed outside of the scheduling period and not considered a part of the schedule.

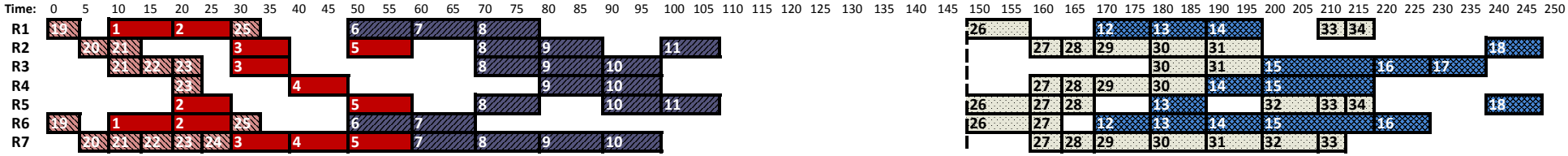
Figure 3.17 and Figure 3.18 demonstrate that the enhanced MPSP model is capable of arranging complex procedures into conflict-free schedules. The enhanced MPSP model schedules as many procedures inside the scheduling period as possible while minimizing total patient wait time. Similar to the simple MPSP model, the enhanced MPSP model allows the user to set procedure priorities to manipulate the optimal schedule. For example, the benefit coefficient b_{12} is set to 20, representing higher benefit of procedure P_C. Figure 3.19 shows the resultant schedule. Procedure P_C, previously outside of the scheduling period is now scheduled within the scheduling period. The lateness penalty can also be manipulated to change the order of procedures within the scheduling period. For example, the penalty w_6 was set to 2 to represent higher lateness penalty for procedure P_B. Figure 3.20 shows the resultant schedule. Procedure P_B is scheduled earlier in the schedule.

The next section discusses further development to the enhanced MPSP model to handle more complexity and flexibility in medical procedures.



i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
ps_i	10	20	30	40	50	140	150	160	170	180	190	80	90	100	110	130	140	150	0	5	10	15	20	25	30	60	70	75	80	90	100	110	120	125
x_i	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 3.17: Optimal solution (schedule) to the enhanced MPSP model with scheduling period p=200



i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
ps_i	10	20	30	40	50	50	60	70	80	90	100	170	180	190	200	220	230	240	0	5	10	15	20	25	30	150	160	165	170	180	190	200	210	215
x_i	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0

Figure 3.18: Optimal solution (schedule) to the enhanced MPSP model with short scheduling period p=150

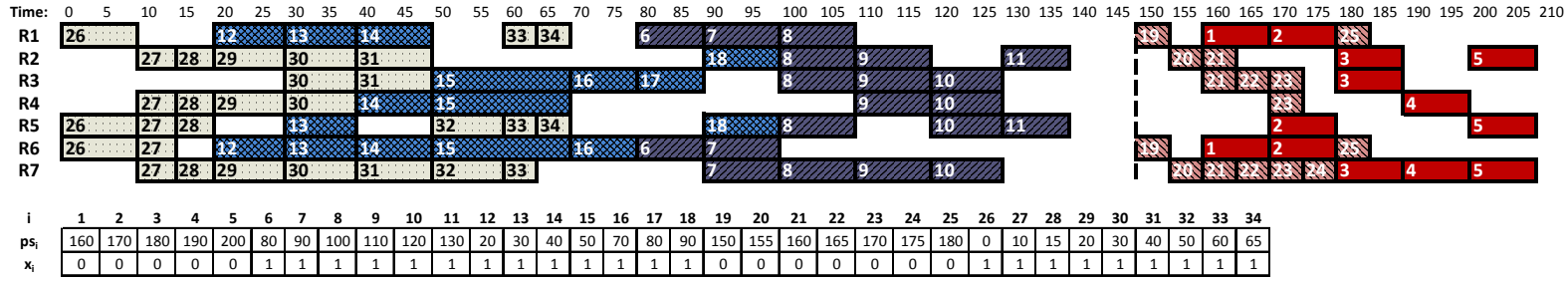


Figure 3.19: Increasing the benefit coefficient b_{12} to give priority to procedure P_C

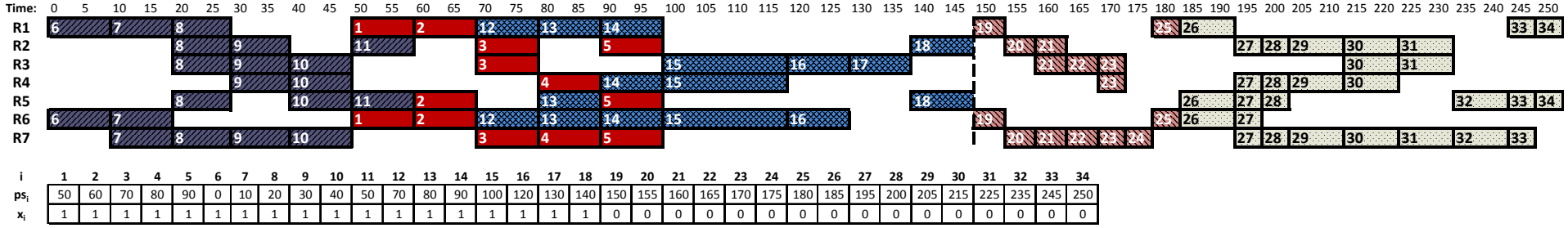


Figure 3.20: Increasing lateness penalty w_6 to give priority to procedure P_B

3.3 Scheduling flexible procedures – the final MPSP model

Both the simple and enhanced MPSP models do not allow preemption or interruption of procedures. That is, once a procedure begins, it must be completed in a fixed amount of time with no breaks in between. Some medical procedures however, are more flexible. Take haemodialysis for example. Flexibility exists between the time patient begins dialysis treatment and the time the nurse administers IV EPO. There is also flexibility between the end of the patient's treatment and the time when the nurse disconnects the patient from the machine. That is, the dialysis machine has stopped filtering at the end of the patient's prescribed treatment period but the patient must wait, still hooked up, for the nurse to disconnect him/her. An effective and practical scheduling model must be able to handle such flexibilities.

3.3.1 Modeling flexible gaps within procedures

Gaps are flexibility in the workflow where a procedure can be put on hold. Those gaps can themselves be modeled as activities. These *gap activities* can have variable durations and will typically have empty resource requirements. Gap activities can represent patient waiting and can therefore have wait time penalties associated with them.

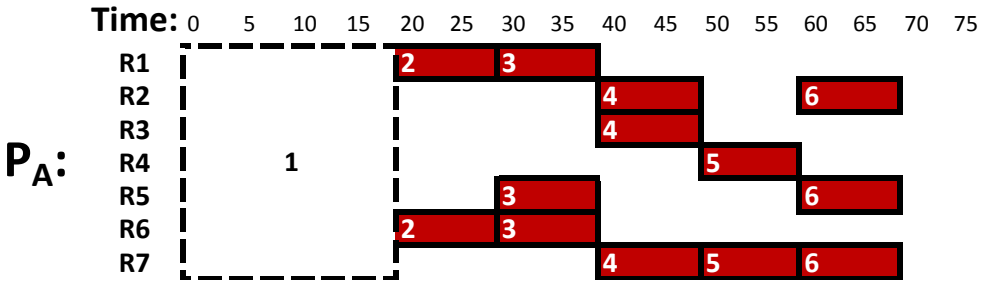


Figure 3.21: Using a gap activity to model delay between due time and start time

Figure 3.21 demonstrates the use of a gap activity to model the delay between the due time of procedure P_A (at time 0) and its scheduled time (at time 20). Gap activity 1 has no resource requirements and will therefore not conflict with any other procedures. Gap activities are subject to workflow constraints specified by the z_{ij} parameter just like any other activity. Each procedure will be modeled with a gap activity at the beginning to capture its lateness. These first gap activities or starting gap activities will be the only activities that are constrained

to begin at its due time. All other related activities will be indirectly linked to the due time via the starting gap activity and the workflow parameter z_{ij}.

Gap activities can also be used anywhere within a procedure to model situations where a procedure is put on hold and a patient waits. See Figure 3.22. For example, a patient is prepped for surgery then waits for the surgeon to be ready or for the operating room to become available.

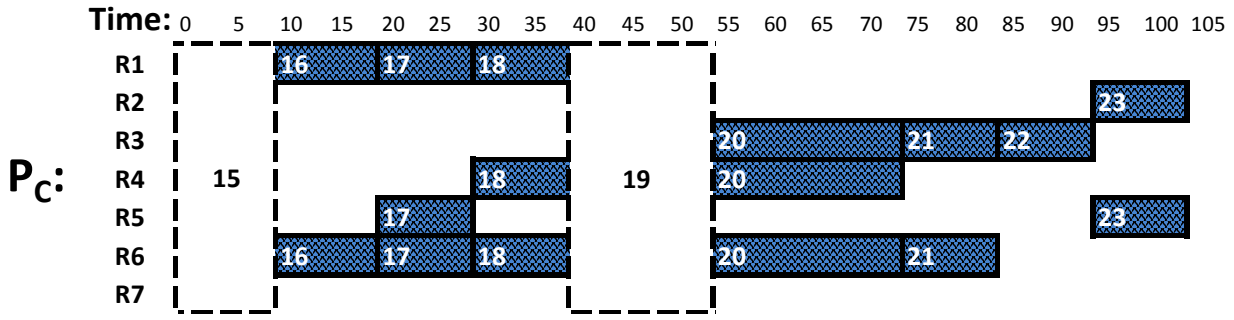


Figure 3.22: Using gap activity within a procedure

Gap activities may also have resource requirements. In the case of haemodialysis, a patient may wait to be disconnected from the dialysis machine. While waiting, the dialysis machine resource is occupied. Similar situations may arise with other resources such as rooms, beds, and equipment. Gap activities with resource requirements ensure that those resources remain occupied while a patient waits. See Figure 3.23.

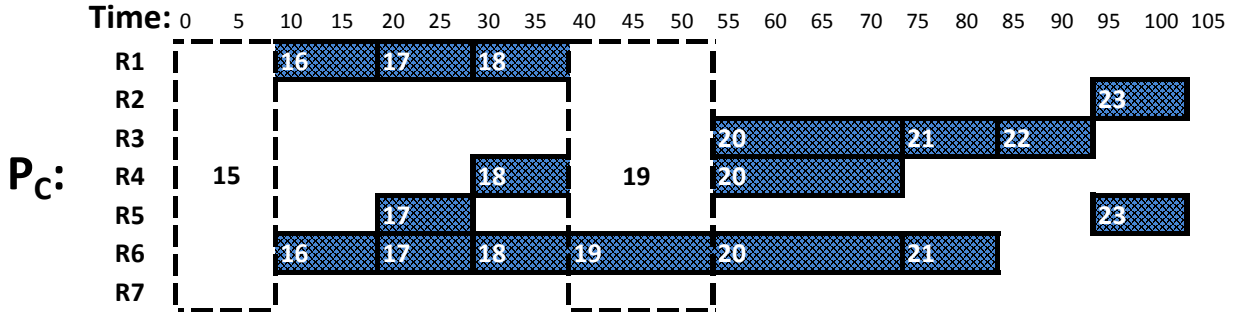
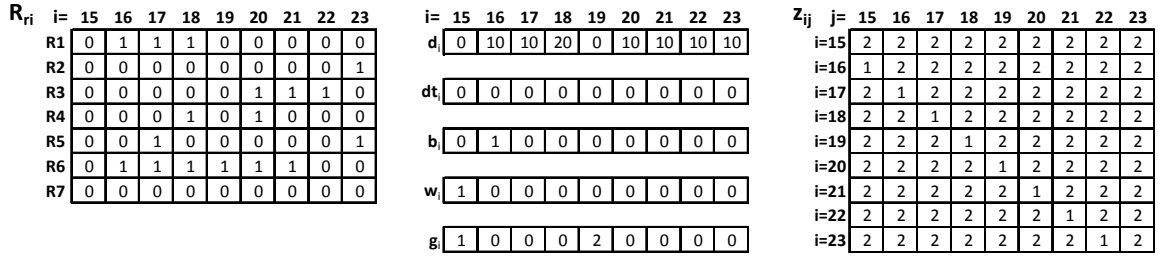


Figure 3.23: Gap activity with resource requirement

A new parameter g_i is introduced to distinguish gap activities from other activities.

$$g_i = \begin{cases} 1 & if\ activity\ i\ is\ a\ starting\ gap\ activity \\ 2 & if\ activity\ i\ is\ a\ gap\ activity\ but\ not\ starting\ gap\ activity \\ 0 & otherwise \end{cases}$$

Figure 3.24 shows the updated mathematical representation of example procedure P_C as shown in Figure 3.23.



R_{ri}	i=	15	16	17	18	19	20	21	22	23
R1		0	1	1	1	0	0	0	0	0
R2		0	0	0	0	0	0	0	0	1
R3		0	0	0	0	0	1	1	1	0
R4		0	0	0	1	0	1	0	0	0
R5		0	0	1	0	0	0	0	0	1
R6		0	1	1	1	1	1	1	0	0
R7		0	0	0	0	0	0	0	0	0

i=	15	16	17	18	19	20	21	22	23
d_i	0	10	10	20	0	10	10	10	10
dt_i	0	0	0	0	0	0	0	0	0
b_i	0	1	0	0	0	0	0	0	0
w_i	1	0	0	0	0	0	0	0	0
g_i	1	0	0	0	2	0	0	0	0

Z_{ij}	j=	15	16	17	18	19	20	21	22	23
i=15		2	2	2	2	2	2	2	2	2
i=16		1	2	2	2	2	2	2	2	2
i=17		2	1	2	2	2	2	2	2	2
i=18		2	2	1	2	2	2	2	2	2
i=19		2	2	2	1	2	2	2	2	2
i=20		2	2	2	2	1	2	2	2	2
i=21		2	2	2	2	2	1	2	2	2
i=22		2	2	2	2	2	2	1	2	2
i=23		2	2	2	2	2	2	2	1	2

Figure 3.24: Updated mathematical representation of procedure P_C

Note that the starting gap activity 15 does not have any resource requirements but gap activity 19 requires resource R6. Also note the addition of parameter g_i. Gap activities will have variable durations so the parameter d_i does not apply and is set to zero.

With added flexibility in the model, managing workflow within procedures becomes more challenging. Activities may not need to be scheduled immediately after another but may still have pre-requisite activities. For example: activity 5 must be scheduled after, but not necessarily immediately after activity 3. The workflow parameter z_{ij} is modified to capture activity pre-requisites.

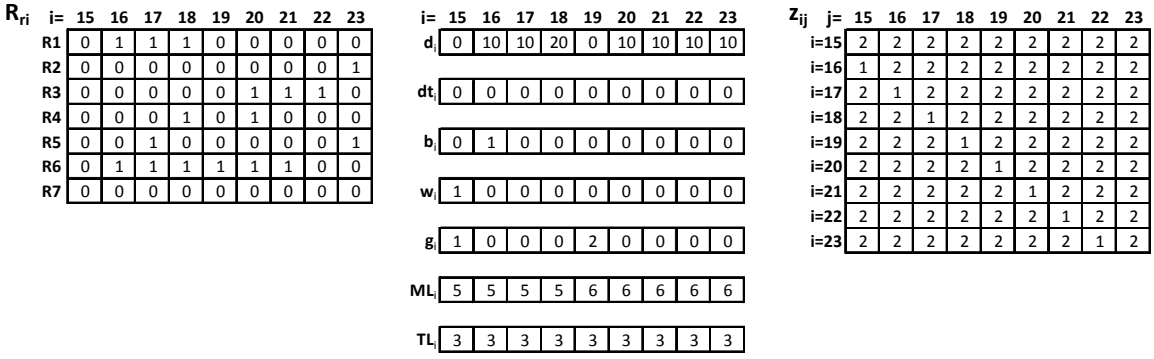
$$z_{ij} = \begin{cases} 1 & if\ activity\ i\ must\ immediately\ follow\ activity\ j \\ 2 & if\ activity\ i\ and\ j\ both\ belong\ to\ the\ same\ procedure \\ 3 & if\ activity\ j\ is\ a\ pre-requisite\ for\ activity\ i \\ 0 & otherwise \end{cases}$$

The introduction of flexibility into the model also reveals the multiple layers or levels of a procedure. The top level is the complete procedure from beginning to end. The middle level consists of portions of the procedure that cannot be interrupted and are separated by gap activities. The lower level consists of the atomic scheduling units of the procedure: the activities. New parameters ML_i and TL_i are introduced to track these levels to effectively manage the workflow of procedures.

$$ML_i = Mid\ level\ index\ of\ activity\ i$$

$$TL_i = Top\ level\ index\ of\ activity\ i$$

The final updated mathematical representation of example procedure P_C is shown in Figure 3.25.



R_{ri}	i= 15	16	17	18	19	20	21	22	23
R1	0	1	1	1	0	0	0	0	0
R2	0	0	0	0	0	0	0	0	1
R3	0	0	0	0	0	1	1	1	0
R4	0	0	0	1	0	1	0	0	0
R5	0	0	1	0	0	0	0	0	1
R6	0	1	1	1	1	1	1	0	0
R7	0	0	0	0	0	0	0	0	0

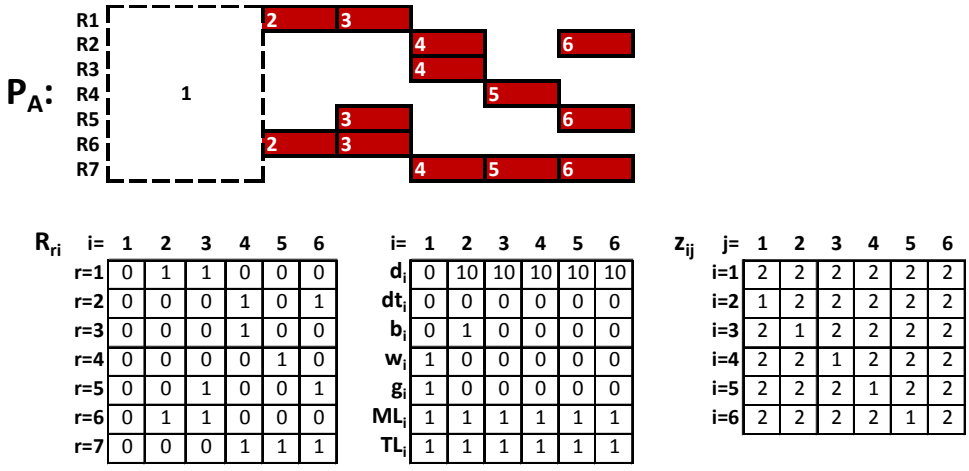
i=	15	16	17	18	19	20	21	22	23
d_i	0	10	10	20	0	10	10	10	10
dt_i	0	0	0	0	0	0	0	0	0
b_i	0	1	0	0	0	0	0	0	0
w_i	1	0	0	0	0	0	0	0	0
g_i	1	0	0	0	2	0	0	0	0
ML_i	5	5	5	5	6	6	6	6	6
TL_i	3	3	3	3	3	3	3	3	3

Z_{ij}	j= 15	16	17	18	19	20	21	22	23
i=15	2	2	2	2	2	2	2	2	2
i=16	1	2	2	2	2	2	2	2	2
i=17	2	1	2	2	2	2	2	2	2
i=18	2	2	1	2	2	2	2	2	2
i=19	2	2	2	1	2	2	2	2	2
i=20	2	2	2	2	1	2	2	2	2
i=21	2	2	2	2	2	1	2	2	2
i=22	2	2	2	2	2	2	1	2	2
i=23	2	2	2	2	2	2	2	1	2

Figure 3.25: Final updated mathematical representation of procedure P_C

Note all activities of procedure P_C belong to top level procedure 3. However, gap activity 19 separates the activities into two mid level procedures 5 and 6. This means that all activities belonging to mid level procedure 5 must be scheduled together and activities belonging to mid level procedure 6 must be scheduled together. However, there can be a gap between mid level procedures 5 and 6.

The mathematical models of example procedures P_A through P_E are shown in Figure 3.26 through Figure 3.30.

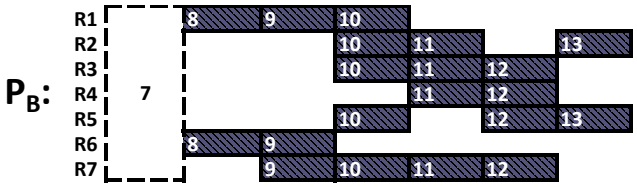


R_{ri}	i= 1	2	3	4	5	6
r=1	0	1	1	0	0	0
r=2	0	0	0	1	0	1
r=3	0	0	0	1	0	0
r=4	0	0	0	0	1	0
r=5	0	0	1	0	0	1
r=6	0	1	1	0	0	0
r=7	0	0	0	1	1	1

i=	1	2	3	4	5	6
d_i	0	10	10	10	10	10
dt_i	0	0	0	0	0	0
b_i	0	1	0	0	0	0
w_i	1	0	0	0	0	0
g_i	1	0	0	0	0	0
ML_i	1	1	1	1	1	1
TL_i	1	1	1	1	1	1

Z_{ij}	j= 1	2	3	4	5	6
i=1	2	2	2	2	2	2
i=2	1	2	2	2	2	2
i=3	2	1	2	2	2	2
i=4	2	2	1	2	2	2
i=5	2	2	2	1	2	2
i=6	2	2	2	2	1	2

Figure 3.26: Modeling procedure P_A with starting gap procedure



R_{ri}

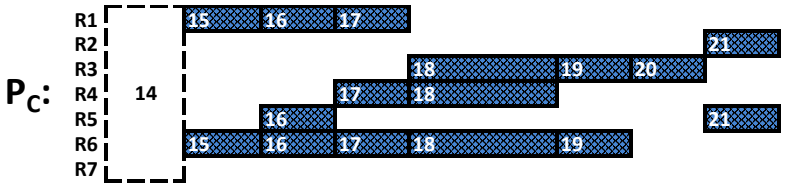
	i= 7	8	9	10	11	12	13
r=1	0	1	1	1	0	0	0
r=2	0	0	0	1	1	0	1
r=3	0	0	0	1	1	1	0
r=4	0	0	0	0	1	1	0
r=5	0	0	0	1	0	1	1
r=6	0	1	1	0	0	0	0
r=7	0	0	1	1	1	1	0

i=	7	8	9	10	11	12	13
d_i	0	10	10	10	10	10	10
dt_i	0	0	0	0	0	0	0
b_i	0	1	0	0	0	0	0
w_i	1	0	0	0	0	0	0
g_i	1	0	0	0	0	0	0
ML_i	2	2	2	2	2	2	2
TL_i	2	2	2	2	2	2	2

z_{ij}

j=	7	8	9	10	11	12	13
i=7	2	2	2	2	2	2	2
i=8	1	2	2	2	2	2	2
i=9	2	1	2	2	2	2	2
i=10	2	2	1	2	2	2	2
i=11	2	2	2	1	2	2	2
i=12	2	2	2	2	1	2	2
i=13	2	2	2	2	2	1	2

Figure 3.27: Modeling procedure P_B with starting gap procedure



R_{ri}

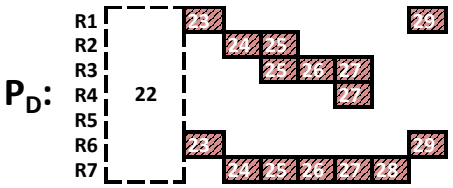
	i= 14	15	16	17	18	19	20	21
r=1	0	1	1	1	0	0	0	0
r=2	0	0	0	0	0	0	0	1
r=3	0	0	0	0	1	1	1	0
r=4	0	0	0	1	1	0	0	0
r=5	0	0	1	0	0	0	0	1
r=6	0	1	1	1	1	1	0	0
r=7	0	0	0	0	0	0	0	0

i=	14	15	16	17	18	19	20	21
d_i	0	10	10	10	20	10	10	10
dt_i	0	0	0	0	0	0	0	0
b_i	0	1	0	0	0	0	0	0
w_i	1	0	0	0	0	0	0	0
g_i	1	0	0	0	0	0	0	0
ML_i	3	3	3	3	3	3	3	3
TL_i	3	3	3	3	3	3	3	3

z_{ij}

j=	14	15	16	17	18	19	20	21
i=14	2	2	2	2	2	2	2	2
i=15	1	2	2	2	2	2	2	2
i=16	2	1	2	2	2	2	2	2
i=17	2	2	1	2	2	2	2	2
i=18	2	2	2	1	2	2	2	2
i=19	2	2	2	2	1	2	2	2
i=20	2	2	2	2	2	1	2	2
i=21	2	2	2	2	2	2	1	2

Figure 3.28: Modeling procedure P_C with starting gap procedure



R_{ri}

	i= 22	23	24	25	26	27	28	29
r=1	0	1	0	0	0	0	0	1
r=2	0	0	1	1	0	0	0	0
r=3	0	0	0	1	1	1	0	0
r=4	0	0	0	0	0	1	0	0
r=5	0	0	0	0	0	0	0	0
r=6	0	1	0	0	0	0	0	1
r=7	0	0	1	1	1	1	1	0

i=	22	23	24	25	26	27	28	29
d_i	0	5	5	5	5	5	5	5
dt_i	0	0	0	0	0	0	0	0
b_i	0	1	0	0	0	0	0	0
w_i	1	0	0	0	0	0	0	0
g_i	1	0	0	0	0	0	0	0
ML_i	4	4	4	4	4	4	4	4
TL_i	4	4	4	4	4	4	4	4

z_{ij}

j=	22	23	24	25	26	27	28	29
i=22	2	2	2	2	2	2	2	2
i=23	1	2	2	2	2	2	2	2
i=24	2	1	2	2	2	2	2	2
i=25	2	2	1	2	2	2	2	2
i=26	2	2	2	1	2	2	2	2
i=27	2	2	2	2	1	2	2	2
i=28	2	2	2	2	2	1	2	2
i=29	2	2	2	2	2	2	1	2

Figure 3.29: Modeling procedure P_D with starting gap procedure

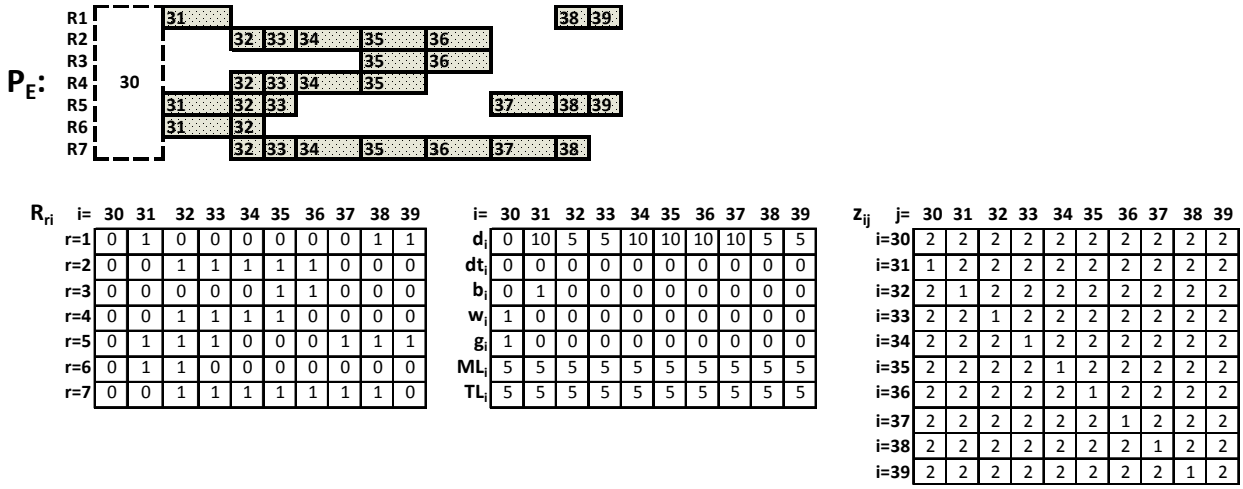


Figure 3.30: Modeling procedure P_E with starting gap procedure

The added flexibility discussed in this section allows one to now fully model the haemodialysis procedure. Consider a very small dialysis clinic with two nurses (*R1 and R2*), six dialysis machines (*R3* to *R8*), and one technician (*R9*). The entire dialysis appointment is modeled using two procedures. The technician activities constitute one procedure while the nurse activities constitute the other procedure. The status of machines is included in both procedures. Example of a dialysis appointment for one patient involving nurse *R1*, machine *R3*, and technician *R9* is illustrated in Figure 3.31. A description of each activity can be found in Table 3-1. The mathematical representation of those activities of a dialysis appointment is illustrated in Figure 3.32.

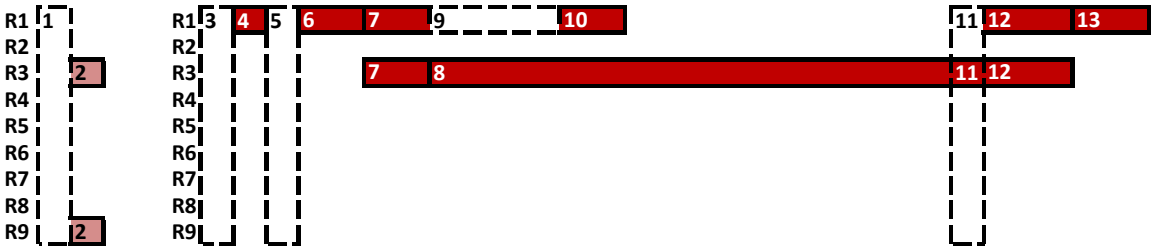


Figure 3.31: Example dialysis appointment

Table 3-1: Descriptions for activities of example dialysis appointment

Activity	Description	Duration (min)	Pre-requisite
1	Starting gap activity for technician activities	Variable	
2	Technician (*R9*) primes and disinfects the machine (*R3*). Machine is occupied	5	
3	Starting gap activity for nurse activities	Variable	
4	Nurse reviews patient paperwork	5	
5	Gap activity representing patient waiting to be called	Variable	
6	Pre-dialysis procedures: measure weight, wash access	10	
7	Pre-dialysis procedures: check temp, bp. Nurse connects patient to machine. Machine is occupied	10	Activity 2
8	Dialysis treatment. Machine is occupied, running treatment. Duration varies from patient to patient	80 – 160	
9	Gap activity to link IV service with start of treatment period	Variable	
10	Nurse provides IV and assessment services. Duration depends on which services patient needs	5 – 25	Activity 7
11	Gap activity representing patient waiting to be disconnected after treatment is complete. Machine is occupied	Variable	
12	Nurse disconnects patient from machine. Wait for haemostasis	10	Activity 10
13	Post dialysis procedure: measure bp, weight etc	10	

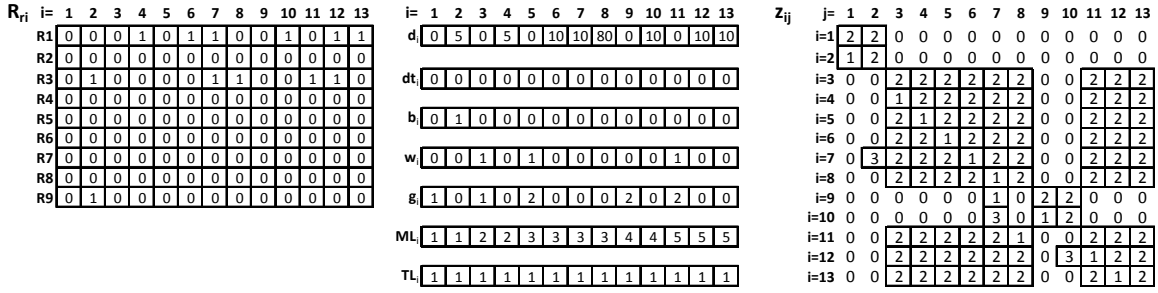


Figure 3.32: Mathematical representation of example dialysis appointment

Recall from section 2.1.3, a typical dialysis treatment lasts between 180 to 270 minutes. The dialysis treatment activity 8 in these examples has durations between 80 and 160 minutes. The model of dialysis treatment was intentionally shortened for no other reason than to fit graphically into pages of this thesis. The shortened dialysis treatment is still effective in demonstrating the scheduling model.

This example dialysis clinic has six machines; it can accommodate six patients (patients *A* through *F*) at one time. The nurse and machine assignment for patients is listed in Table 3-2. The model for appointment of patient *A* is already illustrated in Figure 3.31 and Figure 3.32. The models for appointments of patients *B* through *F* are shown in Figure 3.33 through Figure 3.37.

Table 3-2: Nurse and machine assignments for patients *A* through *F*

Patient:	A	B	C	D	E	F
Nurse:	R1	R2	R1	R2	R1	R2
Machine:	R3	R4	R5	R6	R7	R8

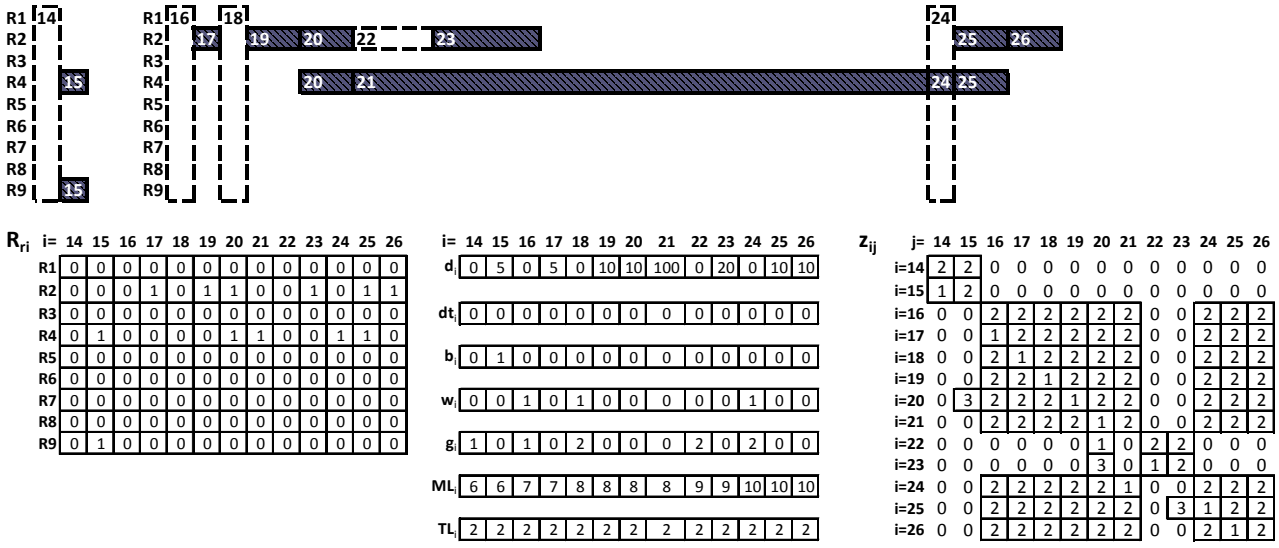


Figure 3.33: Time diagram and parameters of dialysis appointment for patient *B*

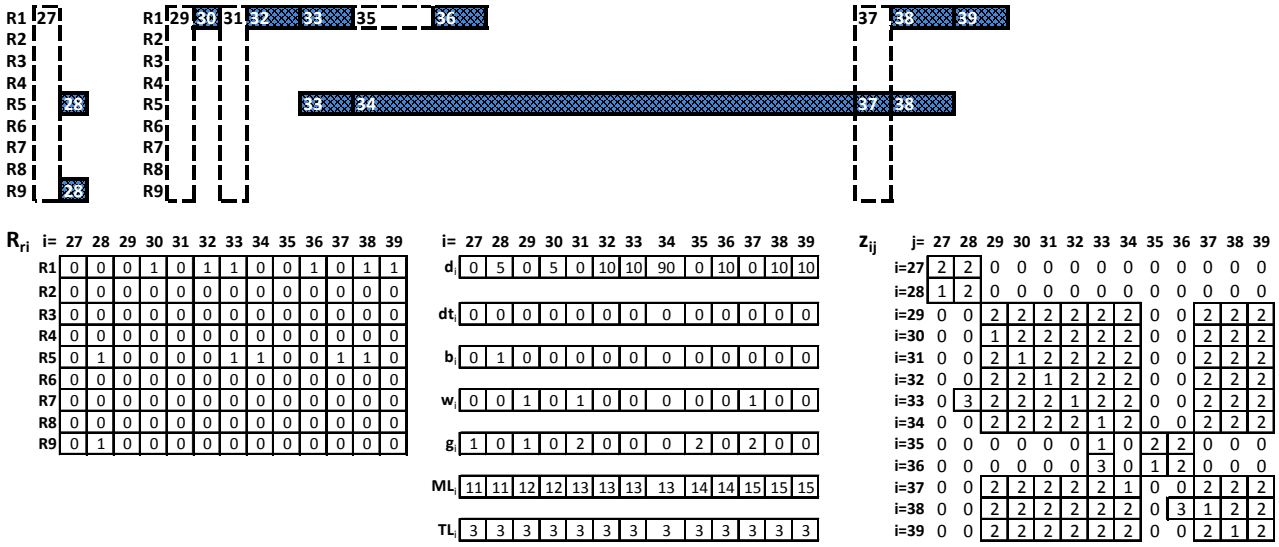


Figure 3.34: Time diagram and parameters of dialysis appointment for patient *C*

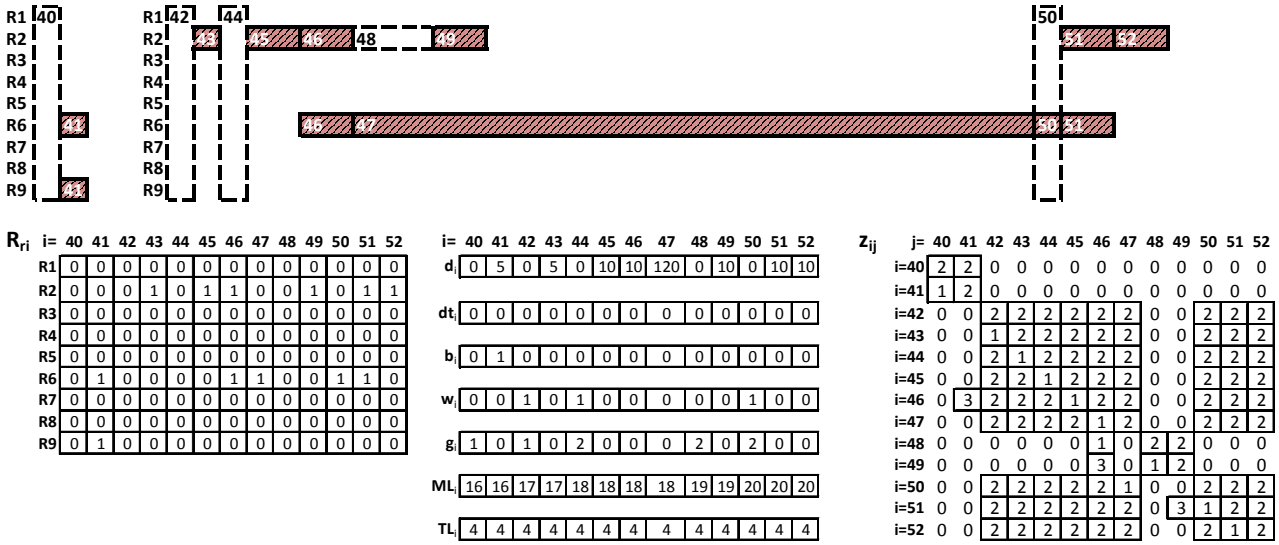


Figure 3.35: Time diagram and parameters of dialysis appointment for patient *D*

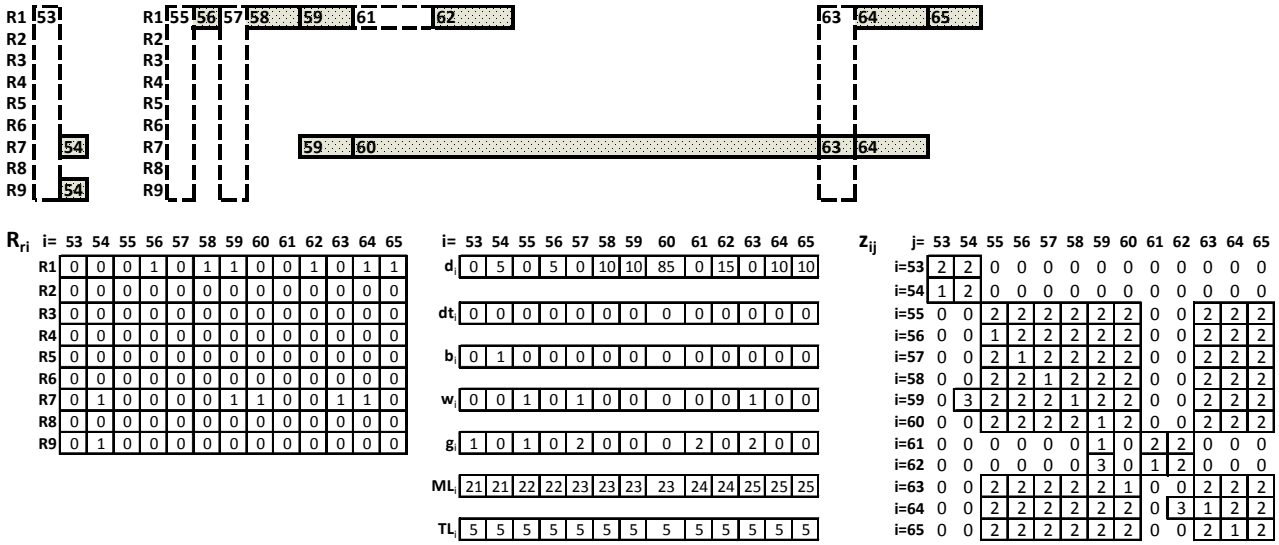


Figure 3.36: Time diagram and parameters of dialysis appointment for patient *E*

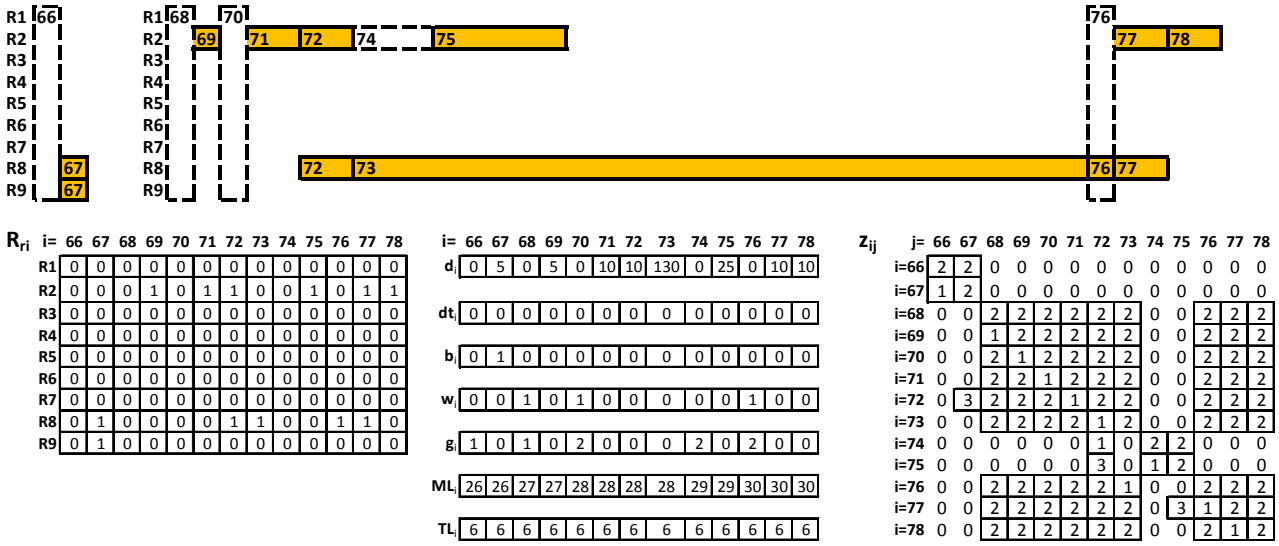


Figure 3.37: Time diagram and parameters of dialysis appointment for patient *F*

The next section discusses modification to the scheduling model to schedule flexibility introduced with the gap activities.

3.3.2 MIP formulation of the final MPSP model

The lateness variable in the enhanced MPSP model is now redundant because lateness is now represented by the duration of starting gap activities. Therefore, the first modification is to reuse variable l_i to model the variable duration of gap activities.

$$l_i = Variable\ duration\ of\ gap\ activity\ i \qquad \forall i: (g_i \neq 0)$$

Reusing variable l_i is convenient because the objective function need not be modified. However, many constraints need to be modified to accommodate the added flexibility of gap activities. Constraints C1 and C2 should only be enforced for activities that are not starting gap activities. Constraint C3 ensures that starting gap activities are always scheduled at their respective due times.

C1 $\quad ps_i + d_i - (1 - x_i)M \leq p \qquad \forall i: (g_i \neq 1)$

C2 $\quad ps_i + px_i \geq p \qquad \forall i: (g_i \neq 1)$

C3 $\quad ps_i = dt_i \qquad \forall i: (g_i = 1)$

The conflict resolution constraints C4 and C5 are each separated into two versions (*a* and *b*) to accommodate both the regular activities and the special gap activities.

C4a $ps_{i2} - ps_i + (1 - y_{i,i2})M \geq d_i$

$\forall i, i2: (R_{ri} = R_{ri2} = 1 \ \cap \ z_{i2,i} = 0 \ \cap \ g_i = 0 \ \cap \ i \neq i2)$

C4b $ps_{i2} - ps_i + (1 - y_{i,i2})M \geq l_i$

$\forall i, i2: (R_{ri} = R_{ri2} = 1 \cap z_{i2,i} = 0 \cap g_i \neq 0 \cap i \neq i2)$

C5a $ps_i - ps_{i2} + My_{i,i2} \geq d_{i2}$

$\forall i, i2: (R_{ri} = R_{ri2} = 1 \ \cap \ z_{i2,i} = 0 \ \cap \ g_{i2} = 0 \ \cap \ i \neq i2)$

C5b $ps_i - ps_{i2} + My_{i,i2} \geq l_{i2}$

$\forall i, i2: (R_{ri} = R_{ri2} = 1 \ \cap \ z_{i2,i} = 0 \ \cap \ g_{i2} \neq 0 \ \cap \ i \neq i2)$

Version *a* of both constraints C4 and C5 deal with regular activities that have fixed durations d_i, version *b* deal with gap activities with variable durations l_i.

The non-negativity (C6) and integer (C7) remain unchanged. The workflow constraint C8, similar to the conflict resolution constraints, is also separated into two versions.

C8a $ps_i - ps_j = d_j \qquad \forall i, j: (z_{ij} = 1 \ \cap \ g_j = 0)$

C8b $ps_i - ps_j = l_j \qquad \forall i, j: (z_{ij} = 1 \ \cap \ g_j \neq 0)$

Once again, version *a* deals with activities with fixed durations d_i and version *b* deals with gap activities with variable durations l_i.

The z_{ij} workflow parameter is no longer sufficient for modeling the relationship between activities for constraint C9. The enforcement criterion for constraint C9 is modified to use the TP_i parameter to group activities together.

C9 $x_i - x_j = 0 \qquad \forall i, j: (TP_i = TP_j \ \cap i \neq j)$

Finally, a new constraint is added to enforce activity pre-requisites.

C10 $y_{i,i2} = 0 \qquad \forall i, j: (z_{ij} = 3)$

Constraint C10 dictates that if activity *j* is a pre-requisite for activity *i*, then activity *j* must be scheduled before activity *i*.

This model is designated the *final MPSP model*, or *MPSP model* for short. A full summary of its mathematical formulation can be found in Appendix E. The MPSP model is this thesis' contribution of a novel scheduling model. The next section will discuss the solving of the MPSP model.

3.3.3 Solving the MPSP model

The optimal solution of the MPSP model for general procedures is shown in Figure 3.38. Despite using a different technique to model procedure flexibility, the MPSP model produces conflict-free schedules just like those produced by the enhanced MPSP model. That is, the conflict resolution capability is preserved. The strength of the MPSP model in handling complexity and flexibility is demonstrated in scheduling of dialysis procedures. Figure 3.39 shows the optimal schedule of dialysis treatment that involves 2 nurses and 6 patients. The MPSP model is able to manipulate flexibilities in the dialysis procedures to optimize the objective function.

The MPSP model retains the same objective function as the simple and enhanced MPSP models. Therefore, its behaviour can be manipulated by adjusting the benefit and lateness penalty coefficients.

The next section will discuss the scalability of the final MPSP model to schedule larger problems.

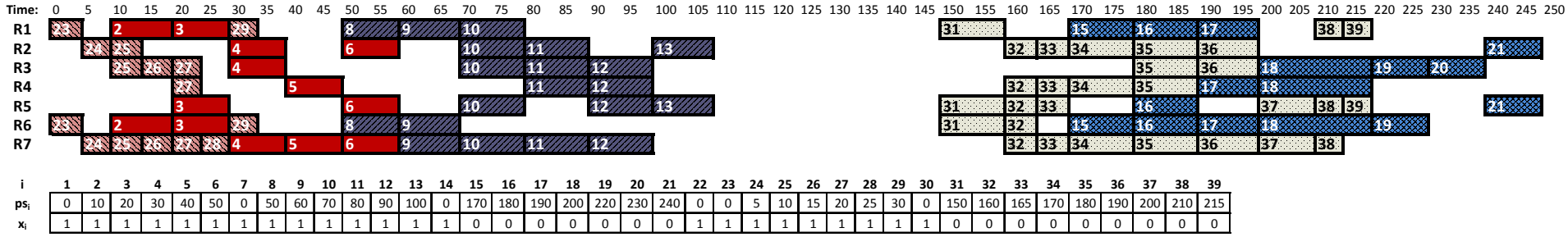


Figure 3.38: Optimal solution (schedule) to general MPSP model with p=150

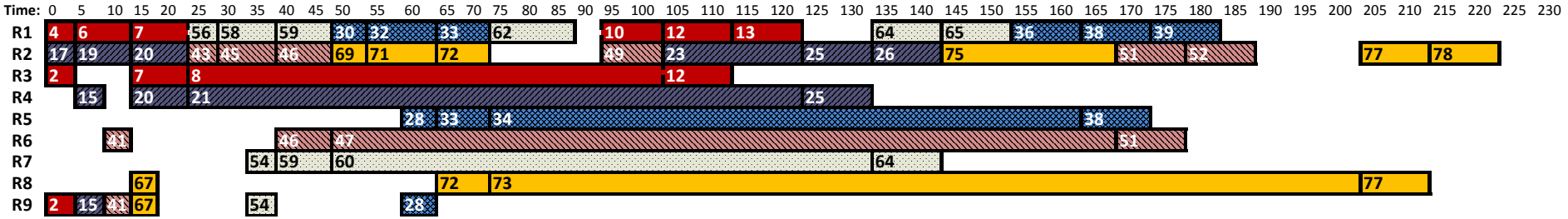


Figure 3.39: Optimal solution (schedule) of dialysis procedures with 2 nurses and 6 patients

3.4 Scalability of the MPSP model

This section will discuss the robustness of the MPSP model in handling problems of different sizes. There are two dimensions into which the MPSP model can be scaled. One can change the number of procedures to schedule and/or change the number of resources to model. The MPSP model was designed to be general and flexible to be able to model a variety of scheduling problems. The MPSP model can theoretically model any number of procedures. This gives the model freedom in scope. That is, the user can theoretically model scheduling problems of any time frame, be it one hour, one shift, one day, one week etc. The MPSP model can also model any number of resources thus allowing the user to theoretically model clinics of any size. The modeling capability of the MPSP model is infinitely scalable. Modeling additional procedures and/or resources simply requires adding more variables and constraints to the model. However, additional variables and constraints introduce more dimensions to a problem, increasing the order and size of its solution space. As a result, more effort is required to solve the problem. Therefore, though the modeling capability of the MPSP model is infinitely scalable, the solvability is most certainly not.

3.4.1 Scaling the MPSP model for general procedures

We continue with the example clinic with 7 resources. Let us see how the problem scales relative to the number of procedures to schedule. Table 3-3 together with Figure 3.40 and Figure 3.41 show the number of variables and constraints needed to model increasing number of procedures. Note that activities are the atomic scheduling units of the MPSP model. The model is scaled up by increasing the number of activities. However, the number of activities is increased by denominations that result in integer numbers of full procedures. For example, 21 activities are required to model procedures P_A, P_B, and P_C. 29 activities are required to model procedures P_A through P_D and so on. Additional instances of procedures P_A through P_E are modeled to further increase the problem scale. For example, 45 activities model 2 instances of P_A and 1 instance each of P_B through P_E; 52 activities model 2 instances each of P_A, P_B and 1 instance each of P_C through P_E and so on.

Table 3-3: Model growth as number of activities increase

Procedures	Activities	Variables	Constraints
3	21	113	247
4	29	198	428
5	39	379	814
6	45	499	1056
7	52	690	1444

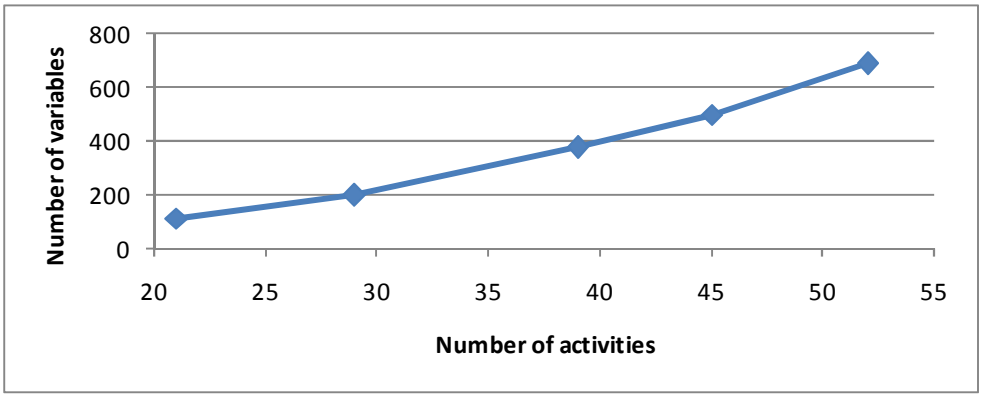


Figure 3.40: Number of variables vs. number of activities

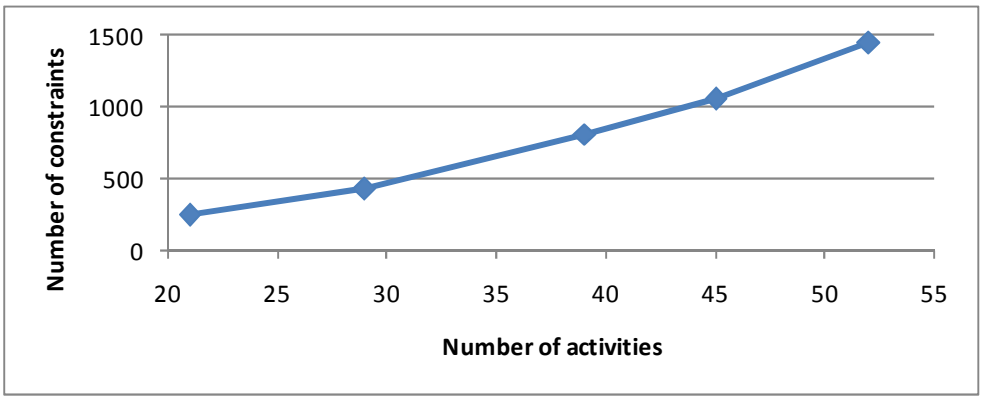


Figure 3.41: Number of constraints vs. number of activities

The data in Table 3-3 and graphs in Figure 3.40 and Figure 3.41 show that the number of variables and constraints grow almost linearly with respect to the number of activities. There is theoretically no limit to the number of procedure that is possible to model. However, the difficulty of solving increasing model size is made apparent by the data and graphs in Table 3-4, Figure 3.42 and Figure 3.43. Models are solved on a Dell Optiplex workstation with Intel Core2 Duo processor running at 2.13GHz with 2 GB memory.

Table 3-4: Effort required for solving problems of increasing size

Procedures	Activities	Run-time (s)	Memory Used (mb)
3	21	0.2	0.4
4	29	1.1	0.9
5	39	39.5	4.3
6	45	532	31.5
7	52	10936.7	198.3

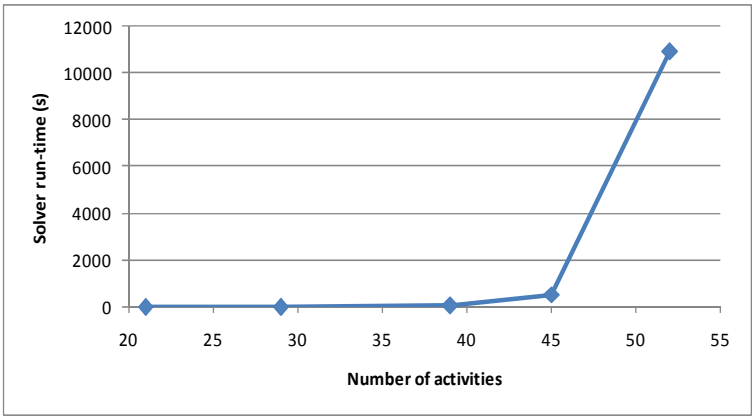


Figure 3.42: Solver run-time vs. number of activities

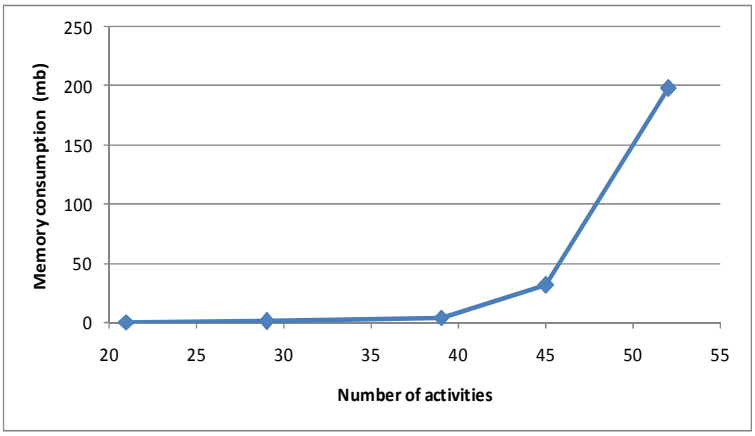


Figure 3.43: Solver memory consumption vs. number of activities

Unfortunately, the effort required to solve problems of increasing size grows exponentially. The time to solve a model with 39 activities was a reasonable 40 seconds. However, a mere 6 more activities in the model required nearly 9 minutes to solve. 7 more activities pushed the solve time to 3 hours. The memory consumption data represent the relative size of the search space of each problem. Clearly, the added dimensions of additional variables inflate the search space exponentially.

An exponential growth in solver effort was expected but not to the degree presented here. The example problems in Table 3-4 are relatively small sized problems, much smaller than real world scheduling problems. Yet solving these small problems exactly already requires impractical computational cost. Performance of the branch and cut solver must be improved for the MPSP model to be practical.

3.4.2 Scaling the MPSP model for dialysis procedures

Scaling the MPSP model to schedule dialysis procedures is an interesting exercise. A dialysis clinic is really made up of smaller groups of patients and nurses. During a typical shift, each nurse is usually assigned 3 or 4 patients under her care. Each patient is assigned his/her own machine to use throughout the treatment procedure. These patient/nurse groupings do not interfere with other groupings. Table 3-5 summarizes attempts at scaling the MPSP model for dialysis procedures. Figure 3.44 and Figure 3.45 show the growth of computational effort with growing problem size.

Table 3-5: Scaling the dialysis scheduling model

Nurses	Patients	Resources	Activities	Variables	Constraints	Run-time (s)	Memory Used (mb)
1	2	4	26	111	294	0.1	0.4
1	3	5	39	222	551	4.5	1.6
1	4	6	52	370	882	53587.5	329.8
2	4	7	48	214	538	0.6	0.9
2	6	9	78	453	1118	78386.1	373.3
3	6	10	78	345	902	119.5	12.1
3	9	14	117	675	1667	>247038.1	>1963.2

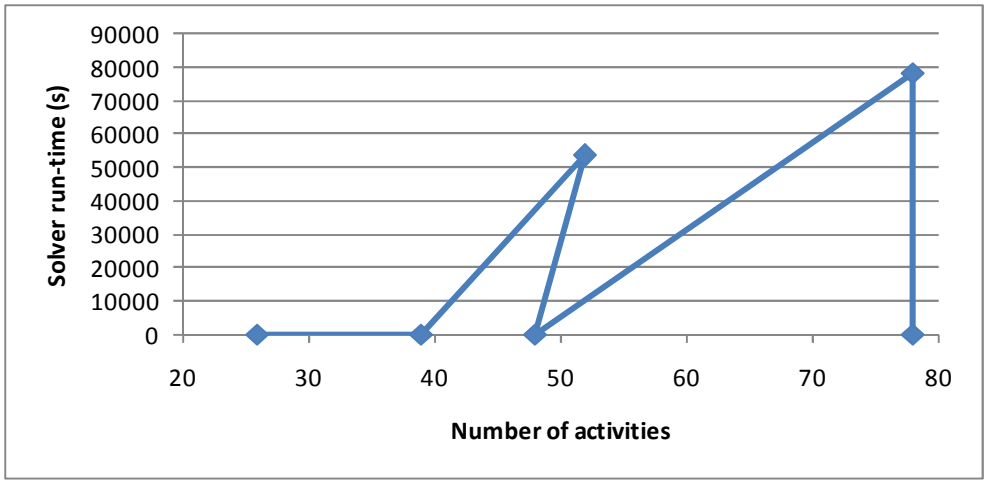


Figure 3.44: Solver run-time vs. number of activities

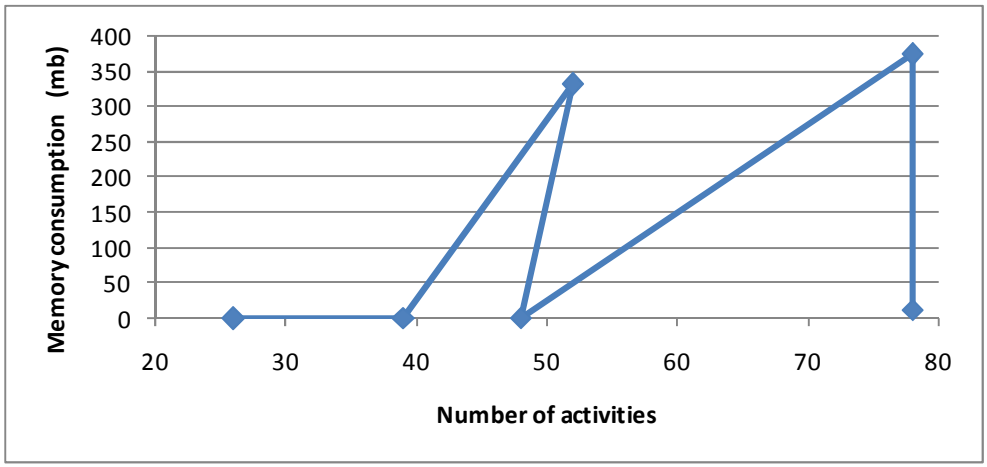


Figure 3.45: Solver memory consumption vs. number of activities

First observation is that the exponential growth in computational cost severely limits the scalability of the MPSP model for dialysis. The last example problem with 3 nurses and 9 patients was run for nearly 3 days without finding the optimal solution. In fact, the solver consumed all memory on the test computer and crashed. For all intents and purposes, the example problem with 3 nurses and 9 patients is intractable. In addition, Figure 3.44 and Figure 3.45 clearly show that the dialysis scheduling problem scales much differently than the general procedures scheduling problem. The sudden peaks and valleys in Figure 3.44 is behaviour caused by the distinct patient/nurse groupings. The characteristic with seemingly the most impact on computational effort is the number of patient per nurse. 2 patients per nurse problems

are much easier to solve than 3 patients per nurse problems. For example, 2 nurses and 6 patients case (case A) requires the same number of activities to model as the 3 nurses and 6 patients case (case B) yet case A took over 21 hours to solve compared to a brisk 2 minutes to solve case B. Case A, being a 3 patients per nurse problem is clearly much more difficult to solve than case B which is a 2 patients per nurse problem.

The dialysis scheduling problem is interesting in that the problem of scheduling an entire dialysis clinic can be decomposed into problems of scheduling multiple groupings of 1 nurse, 3 patients, and 3 machines. However, there is a link between the groupings formed by the machine technician. A clinic typically employs only 1 or 2 technicians to help operate and maintain all the machines. Technicians do not fit nicely into the neat patient/nurse groupings. If the dialysis scheduling problem is to be decomposed into multiple problems of scheduling patient/nurse groupings, the technicians must be scheduled using a different model. This thesis does not pursue any separate technician scheduling models.

This section points out that different sized dialysis scheduling problems can be solved by decomposing the problem into smaller patient/nurse groupings rather than continuously scaling up one model of the entire clinic. Unfortunately, real world dialysis scheduling problems typically involve patient/nurse groupings of 3 or 4 patients per nurse. While the 1 nurse, 3 patients problem is easily solved in 4.5 seconds, the 1 nurse, 4 patients case takes an impractically long 14.9 hours to solve. Despite the option of decomposing the dialysis scheduling problem into smaller problems, the performance of the MIP model still needs improvement.

3.5 MPSP model summary

To summarize this chapter: a novel scheduling model called the MPSP model has been developed for the medical procedures scheduling problem. The MPSP model is formulated as a linear programming problem, specifically, a mixed-integer programming problem. It is designed to be general and flexible to model a wide variety of procedures. Its flexibility has been demonstrated in modeling procedures at a haemodialysis clinic. Please see Appendix F for a brief example application of the MPSP model in scheduling a different medical procedure: PET-CT scan.

The MPSP model can be solved exactly using the branch and cut (BnC) method which guarantees solution optimality. Unfortunately, the computational cost of solving the MPSP model exactly is impractically high even for small size problems. A high quality heuristic algorithm is needed to quickly produce good solutions. A good heuristic solution can be fed into the BnC solver to reduce the solution space of the MPSP model which will lower the computational cost.

The next chapter continues the contribution of this thesis by developing a novel heuristic algorithm to quickly find high quality solutions.

Chapter 4: Heuristic scheduling algorithm

The previous section pointed out that a heuristic algorithm is needed for generating good initial solutions to feed to the branch and cut solver to improve performance and practicality of the MPSP model. This chapter develops and evaluates such a heuristic algorithm. Work in this chapter represents a major thesis contribution.

4.1 Exploiting the scheduling problem structure

Good heuristic algorithms typically exploit the structure or nuances of their respective problems. For example, a simple, effective heuristic to solve the traveling salesman problem is the nearest neighbor (NN) algorithm. NN is a greedy algorithm that simply tells the "salesman" to go to the next closest city to his/her current location.

Development of a scheduling heuristic begins with analysis of the scheduling problem structure. At the core of the medical procedures scheduling problem is really an advanced bin packing problem. Procedures are analogous to items of different shapes and sizes. The scheduling period can be considered the "bin" to pack in procedures. The deterministic procedures model presented in Figure 3.26 through Figure 3.30 in section 3.3 are items with well defined shapes and sizes. The packing of those items/procedures into the scheduling period "bin" is analogous to building a conflict-free schedule and is actually quite easy to do. The next section presents a simple algorithm to "pack the schedule bin."

4.2 Matrix shift heuristic (MASH)

Consider two bins: A and B. Bin A represents the schedule and therefore has its size defined by length of the scheduling period. Bin B has infinite size and will capture procedures that do not fit into bin A. The steps to building a conflict-free schedule are as follows:

Step 1: Begin with one procedure in the schedule either at time 0 or at its due time.

Step 2: Select another procedure from a procedures queue and attempt to place that next procedure at the beginning of the schedule or at its due time.

Step 3: Check for resource conflicts. If conflicts exist, go to Step 4. If resources are not in conflict, go to Step 5.

Step 4: Attempt to place the next procedure one time slot later. Go to Step 3.

Step 5: Slot the next procedure into the schedule. Go to Step 3.

This scheduling loop repeats until all procedures in the queue have been scheduled. The time slot can be arbitrary in length but is set at 5 minutes for examples in this thesis. The procedures queue is an arbitrary order of procedures. This algorithm schedules procedures in a serial fashion. That is, procedures are slotted into a schedule one after another. This schedule building algorithm is illustrated graphically in the following Figure 4.1 through Figure 4.4.

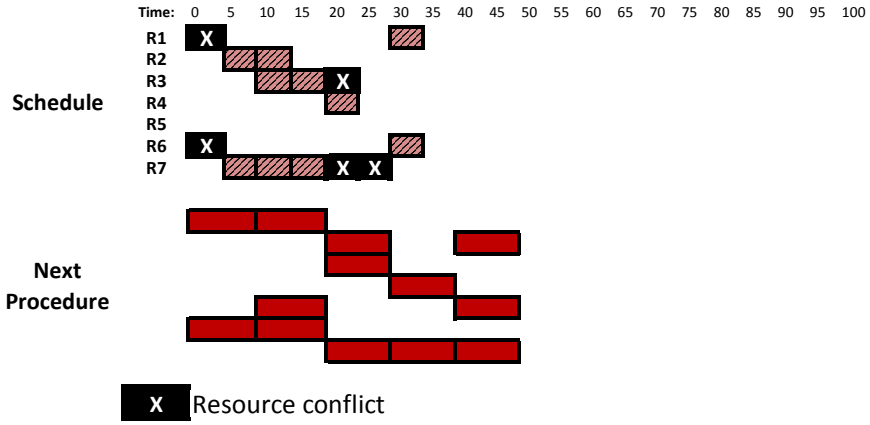


Figure 4.1: Attempt to place next procedure at the beginning of the schedule

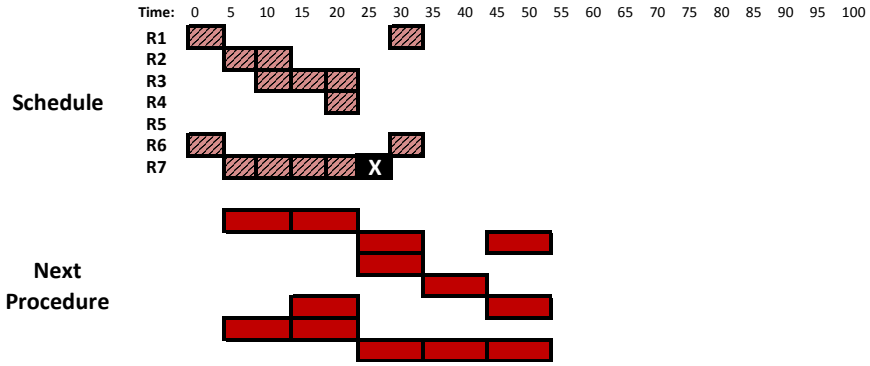


Figure 4.2: Shift next procedure one timeslot later

Figure 4.1 shows the attempt to first schedule the next procedure at time 0. There are however, resource conflicts. Shifting the next procedure by one time slot as illustrated in Figure 4.2 still does not resolve all resource conflicts.

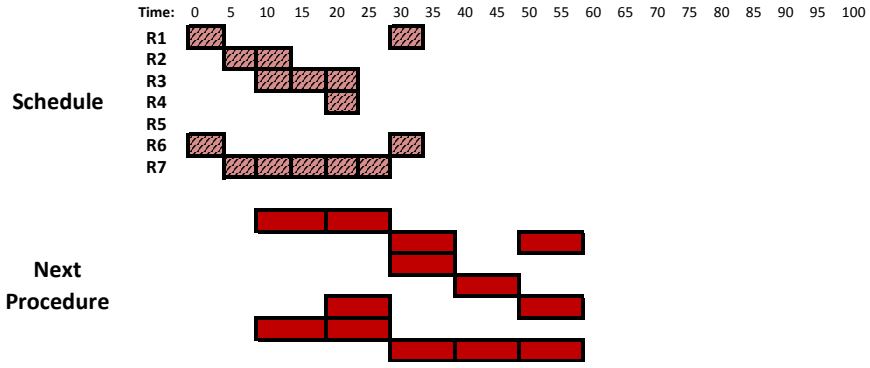


Figure 4.3: Continue shifting until next procedure does not cause resource conflicts

The next procedure is shifted later in the schedule time slot by time slot until there are no resource conflicts as shown in Figure 4.3. At which point, the next procedure is slotted into the schedule as shown in Figure 4.4. The algorithm then moves on and attempts to schedule other procedures in the queue.

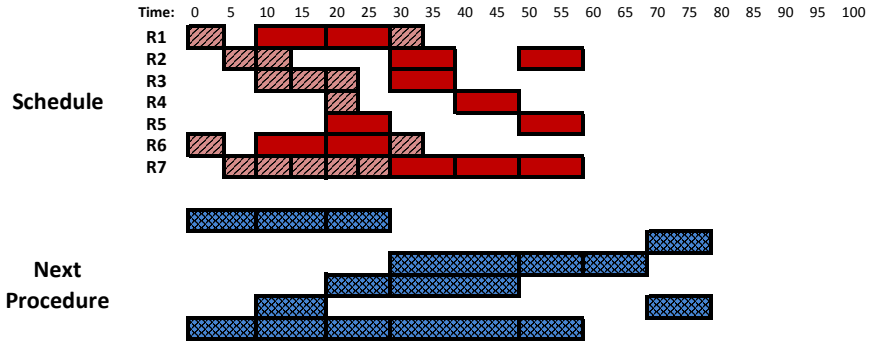


Figure 4.4: Slot next procedure into schedule and attempt to schedule other procedures

If at any time a procedure is not able to fit completely into bin A, it is immediately moved to bin B and the shifting and slotting algorithm continues in bin B. For example, the next procedure shown in Figure 4.5 has been shifted to its latest possible start time within the scheduling period but still causes resource conflicts. As a result, the algorithm immediately shifts the next procedure to the beginning of bin B as shown in Figure 4.6. Shifting a procedure into bin B signifies that it cannot fit into the scheduling period and is therefore not scheduled.

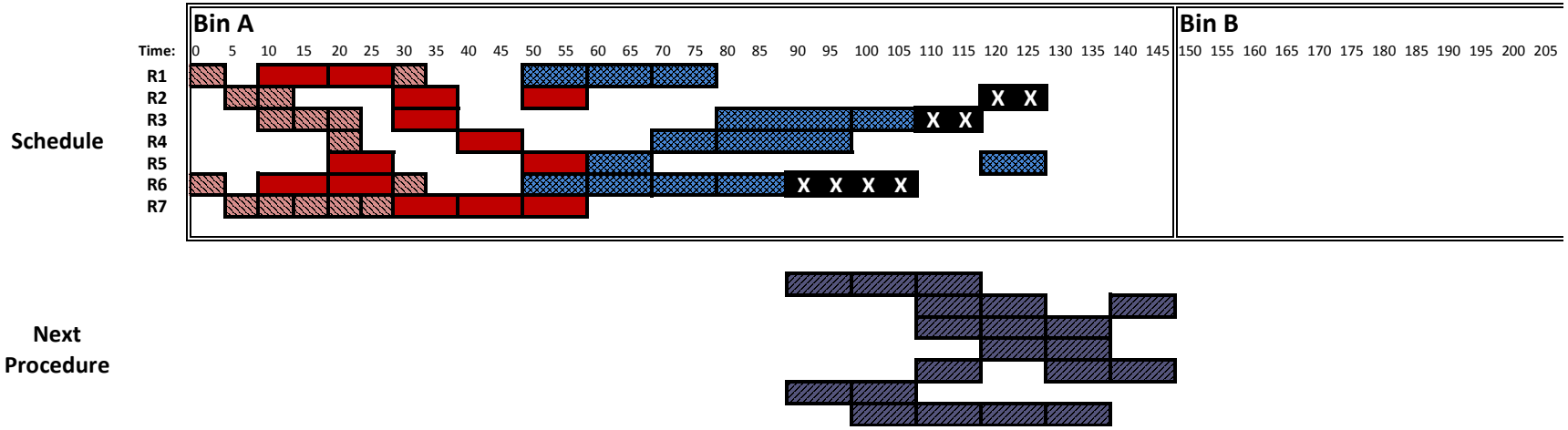


Figure 4.5: Next procedure cannot fit completely into scheduling period

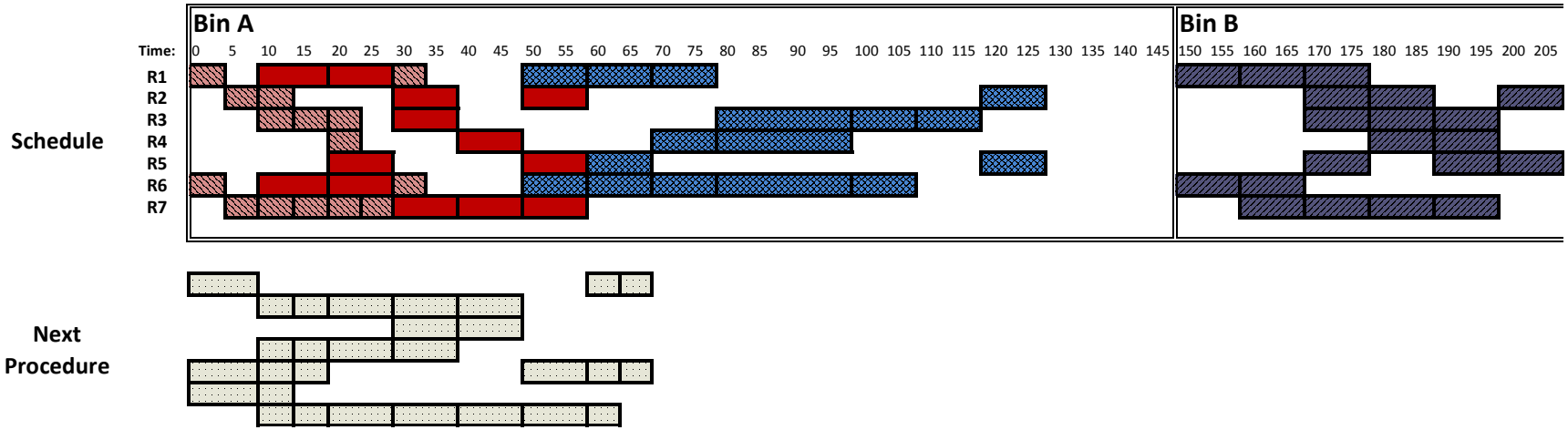
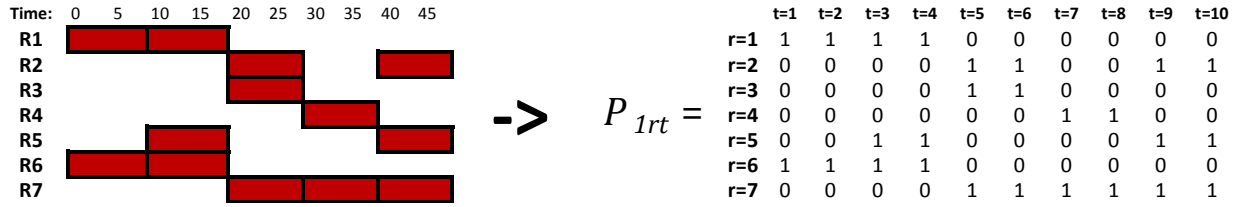


Figure 4.6: Next procedure cannot fit into bin A and is therefore scheduled in bin B

This algorithm of shifting and slotting procedures is easy to implement using binary matrices to represent the schedule and procedures. That is, procedures are modeled using a matrix:

$$P_{irt} = \begin{cases} 1 & if\ procedure\ i\ requires\ resource\ r\ at\ time\ slot\ t \\ 0 & otherwise \end{cases}$$

For example, the P_{irt} matrix for procedure P_A is shown in Figure 4.7.



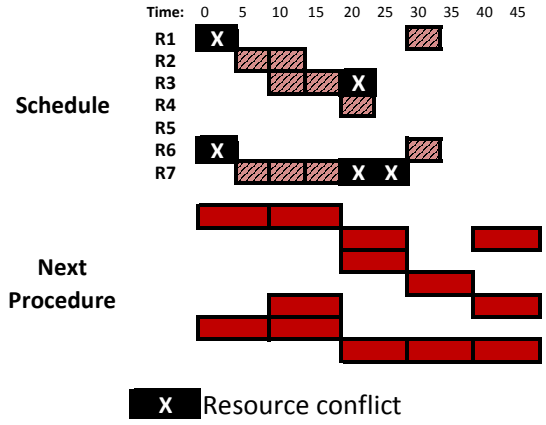
	t=1	t=2	t=3	t=4	t=5	t=6	t=7	t=8	t=9	t=10
r=1	1	1	1	1	0	0	0	0	0	0
r=2	0	0	0	0	1	1	0	0	1	1
r=3	0	0	0	0	1	1	0	0	0	0
r=4	0	0	0	0	0	0	1	1	0	0
r=5	0	0	1	1	0	0	0	0	1	1
r=6	1	1	1	1	0	0	0	0	0	0
r=7	0	0	0	0	1	1	1	1	1	1

Figure 4.7: Matrix representation of procedure P_A

The schedule is represented using a matrix:

$$S_{rt} = \begin{cases} 1 & if\ resource\ r\ is\ occupied\ in\ time\ slot\ t \\ 2 & if resource\ conflict\ exist\ for\ resource\ r\ at\ time\ slot\ t \\ 0 & if\ time\ slot\ t\ is\ free \end{cases}$$

The checking of resource conflicts is done by summing elements of the schedule matrix S_{rt} with elements of the next procedure matrix P_{irt}. Matrix representation of the situation shown in Figure 4.1 is shown in Figure 4.8.



$S_{rt} =$

	t=1	t=2	t=3	t=4	t=5	t=6	t=7	t=8	t=9	t=10
r=1	2	0	0	0	0	0	1	0	0	0
r=2	0	1	1	0	0	0	0	0	0	0
r=3	0	0	1	1	2	0	0	0	0	0
r=4	0	0	0	0	1	0	0	0	0	0
r=5	0	0	0	0	0	0	0	0	0	0
r=6	2	0	0	0	0	0	1	0	0	0
r=7	0	1	1	1	2	2	0	0	0	0

$P_{1rt} =$

r=1	1	1	1	1	0	0	0	0	0	0
r=2	0	0	0	0	1	1	0	0	1	1
r=3	0	0	0	0	1	1	0	0	0	0
r=4	0	0	0	0	0	0	1	1	0	0
r=5	0	0	1	1	0	0	0	0	1	1
r=6	1	1	1	1	0	0	0	0	0	0
r=7	0	0	0	0	1	1	1	1	1	1

Figure 4.8: Matrix representations of schedule, procedure and resource conflicts

This algorithm of shifting and slotting matrices is given the name Matrix Shift heuristic, MASH for short. MASH produces schedules that look exactly like those produced by the MPSP model. That is, solutions from MASH can easily be translated into values of decision variables for the MPSP model. However, MASH can only produce conflict free schedules without optimization considerations. The next section will discuss adding intelligence to the heuristic algorithm to search for the optimal ordering of procedures for a schedule.

4.3 Genetic matrix shift heuristic (gMASH)

The MASH heuristic provides a simple but unintelligent algorithm for building conflict free schedules. MASH is a serial scheduler. It schedules one procedure at a time. Its starting point is the procedures queue or the order of procedures to slot into a schedule. Intelligence is needed to build a good procedures queue that will lead to a good if not optimal schedule. This section uses genetic algorithm to build a good procedures queue. This combination of genetic algorithm with matrix shift heuristic is given the name gMASH, short for genetic matrix shift.

Genetic algorithm is a flexible and powerful search tool. Its fitness or objective function need not be linear or differentiable. It is especially suited for the inherently non-linear and non-differentiable problem of scheduling. Figure 4.9 outlines the basic structure of the genetic heuristic algorithm gMASH.

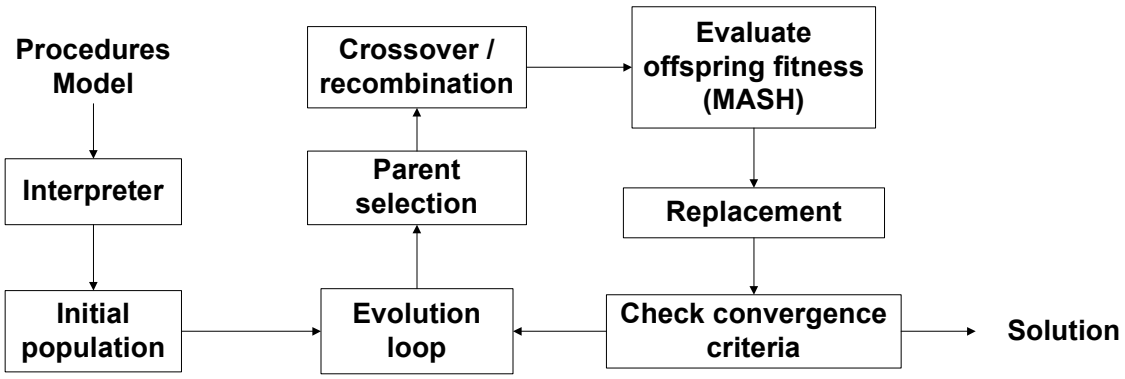


Figure 4.9: Basic structure of the gMASH heuristic

Actually, Figure 4.9 shows a general genetic algorithm structure. The only thing that makes gMASH unique from the general GA structure is the use of MASH as its fitness function. gMASH first interprets the procedures model, captures the number of procedures, their benefit

coefficients and lateness penalties, and builds the matrix representations of procedures. gMASH then initializes a random population of chromosomes or solutions. That population is then evolved through crossover, recombination and replacement. The strongest chromosome or the best solution at the end of the evolutionary loop is the heuristic solution.

4.3.1 Encoding solutions into chromosomes

Solutions to gMASH are encoded in chromosomes. An example of a chromosome for a 7 procedures problem is shown in Figure 4.10. The chromosome has two parts. Part A encodes the ordering of procedures using priority values. Allele 1 of part A contains the priority value of procedure P_A, allele 2 contains priority of procedure P_B and so on for each procedure. Procedures with higher priority values are placed earlier in the procedures queue and will therefore be scheduled earlier. The scheduling order in the Figure 4.10 example is: P_F, P_B, P_D, P_C, P_A, P_E, P_G. Alleles in part B encode the x_i values of procedures. In the Figure 4.10 example, procedures P_A, P_B, P_C, and P_D will be scheduled within the scheduling period while procedures P_E, P_F and P_G will be placed outside of the scheduling period. The length or the number of alleles in each chromosome is twice the number of procedures.

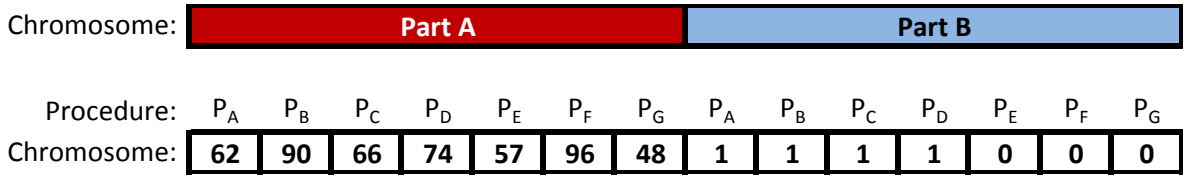


Figure 4.10: Example chromosome for a 7 procedures problem

The initial population is simply *N* randomly generated chromosomes. *N* is the population size and remains constant throughout the course of evolution. The population size is a customizable parameter that can have tremendous impact on the performance of a genetic algorithm. Its effect will be studied in a later section. Initially, gMASH will use a population multiplier of 10 as suggested by experts in the genetic algorithms field. [118] That is, the population will be set at 10 times the dimensionality of the problem. Dimensionality of the MPSP model is the number of procedures to schedule. So for example, a problem of scheduling 7 procedures will use a population size of 70 chromosomes.

4.3.2 Fitness function

The fitness of a chromosome is the quality of the schedule resulting from the procedures queue encoded in that chromosome. The fitness function is simply the matrix shift heuristic MASH. The fitness value is the schedule quality calculated by MASH. The quality of a schedule is judged using the same objective function as the MPSP model. A fit chromosome is one which results in a schedule that minimizes lateness penalty and maximizes benefit.

4.3.3 Chromosome repair function

Recall that the flexible procedures model presented in section 3.3.1 includes precedence relationships between procedures. gMASH maintains precedence relations with a chromosome repair function. A repair function was designed specifically for gMASH. The repair function checks the workflow matrix z_{ij} for precedence relations and swaps priority values between procedures to enforce precedence. The repair function also synchronizes parts A and B of each chromosome. That is, if the fitness function finds that the procedures queue encoded in part A does not agree with x_i values encoded in part B, part B will be repaired accordingly. The chromosome repair function ensures that the population contains only feasible solutions.

4.3.4 Recombination / replacement

In each iteration or generation of the evolutionary loop, two parent chromosomes are recombined to produce offspring chromosomes. One parent is the strongest chromosome of the population; the other parent is a randomly chosen chromosome. The recombination is done through a simple random crossover. That is, the two parent chromosomes exchange genetic information between two randomly chosen crossover points. Figure 4.11 shows an example recombination of two parent chromosomes.

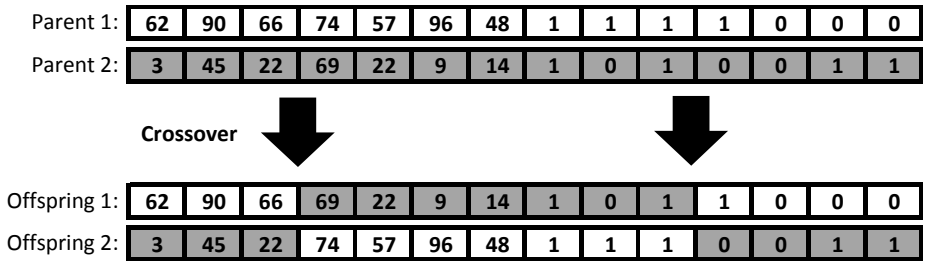


Figure 4.11: Crossover / recombination of parent chromosomes

gMASH uses a survival of the fittest replacement strategy. That is, the fitness values of the resulting offspring chromosomes are compared to those of the weakest chromosomes. If an offspring chromosome is stronger than the weakest chromosome, it replaces that weakest chromosome in the population.

4.3.5 Convergence

gMASH tracks the average fitness of the population through the evolutionary loop. If the average population fitness stays constant for 10 generations, it is assumed that the population has converged on a solution and the evolutionary loop is stopped. A converged population is typically homogenous where every member in the population encodes the same solution.

4.4 Performance of gMASH

4.4.1 Heuristic solution vs. exact solution for general procedures

The performance of gMASH is compared to performance of the BnC method in solving the same MPSP model problems presented in section 3.4.1. Table 4-1 summarizes the results of that comparison. Figure 4.12 compares speeds of gMASH and the exact BnC method.

Table 4-1: gMASH performance vs. BnC method in scheduling general procedures

		BnC method		gMASH		
Problem	Activities	Exact solution	Solver Run-time (s)	Population Size	Heuristic solution	Run-time (s)
A	21	97	0.2	30	97	0.000
B	29	156	1.1	40	156	0.031
C	39	285	39.5	50	285	0.110
D	45	444	532.0	60	444	0.203
E	52	653	10936.7	70	673	0.469

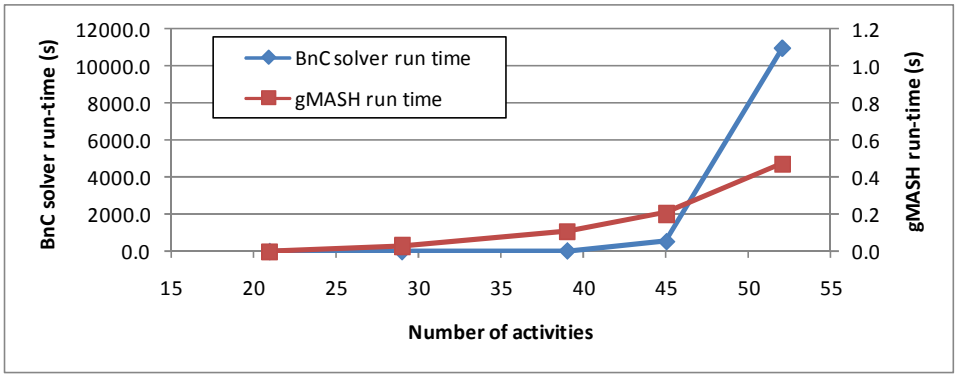


Figure 4.12: gMASH run-time compared to BnC solver run-time

One immediately notices that the gMASH heuristic is significantly faster than the BnC solver in solving the same MPSP models. Problem E that took the BnC solver 3 hours to solve was solved by gMASH in half a second. The growth of gMASH run-time with increasing problem size is also much more manageable than the BnC solver's n^{th} degree exponential growth. The second observation is that the gMASH heuristic solutions are very good. In fact, for problems A through D, gMASH actually produced the optimal solution. gMASH failed to produce the optimal solution to problem E. Instead, the gMASH solution has a 3% error from optimal. Repeat iterations of the gMASH heuristic for problem E resulted in different solutions, all iterations produced solutions within 4.5% error of the exact, optimal solution and some iterations did arrive at the optimal solution. This variability in solution reveals a weakness of genetic algorithms in that they can sometimes converge prematurely on local optima. As a result, the optimality of gMASH solutions cannot be guaranteed.

4.4.2 Heuristic solution vs. exact solution for dialysis procedures

Performance of gMASH is compared to the BnC solver for scheduling the same dialysis procedures described in section 3.4.2.

Table 4-2: gMASH performance vs. BnC method in scheduling dialysis procedures

				BnC solver		gMASH		
Problem	Nurses	Patients	Activities	Exact solution	Run-time (s)	Population Size	Heuristic solution	Run-time (s)
F	1	2	26	21	0.1	100	21	0.015
G	1	3	39	69	4.5	150	69	0.109
H	1	4	52	162	53587.5	200	197	0.547
I	2	4	48	42	0.6	200	42	0.125
J	2	6	78	138	78386.1	300	153	0.844
K	3	6	78	63	119.5	300	68	0.735
L	3	9	117	212 (best incumbent)	>247038.1	450	232	3.969

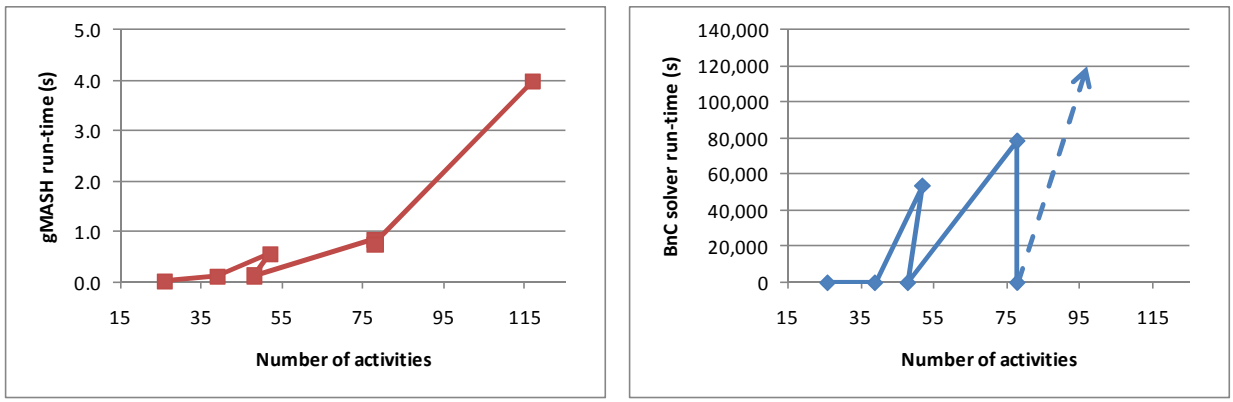


Figure 4.13: gMASH run time compared with BnC solver run time

Once again, gMASH is orders of magnitude faster than the exact, BnC solver. The MPSP model for dialysis quickly becomes impractical to solve exactly. Problem L with 3 nurses and 9 patients became intractable for the BnC solver and crashed the test computer. gMASH, on the other hand solved that same problem L within 4 seconds. Similar to general procedures, gMASH cannot guarantee optimality but its solutions can be very good. For the easy problems F, G, and I, gMASH converged on the optimum solution. The error in gMASH solution was 21.6% for problem H, 10.9% for problem J, and 7.9% for problem K. The exact solution to problem L was never found. The best incumbent (potential solution) found by the BnC solver at the time of crash has an objective value of 212. gMASH converged to a solution that is 9.4% from that best incumbent. Dialysis procedures are noticeably more difficult to schedule than general procedures. That difficulty highlights gMASH's weakness in not being able to guarantee optimality. However, the accuracy of gMASH can be improved by manipulating its population size. Increasing the population size is akin to casting a finer net over the solution space. A more thorough search increases the probability of gMASH converging on the optimal solution. Manipulation of population size to improve gMASH performance will be studied in a later section. But before that, let us first compare the quality of gMASH solutions to manually generated schedules.

4.4.3 gMASH heuristic solutions vs. manual scheduling of dialysis procedures

The ideal baseline measure of scheduler performance for this thesis is the manually generated schedule. The question is: can an intelligent scheduler produce a better schedule than

a human scheduler? It was not necessary to compare the performance of the BnC solver with a human scheduler because the MPSP model when solved exactly guarantees the optimality of its solutions. However, the evolutionary heuristic gMASH presented in this chapter cannot guarantee the optimality of its solutions. Therefore, the worthiness of gMASH should ideally be judged based on its performance compared to a human scheduler.

The general MPSP model in this thesis was designed to be very flexible in order to model a wide variety of procedures. In the real world, different procedures are scheduled using different strategies. For example, surgical procedures that share resources are minimally overlapped to accommodate the high degree of uncertainty in those procedures. In contrast, more deterministic procedures such as some medical imaging procedures can benefit from as much overlap as possible to maximize patient flow. The scheduling model and gMASH can be adapted to mimic different scheduling strategies. Therefore, it is difficult to make general conclusions about the performance of gMASH. Its performance is different in each unique application.

The specific application studied in this thesis is the scheduling of haemodialysis procedures. As discussed in section 3.4, the dialysis scheduling problem can really be broken down into smaller nurse/patient groupings. Realistically, the maximum number of patients assigned to 1 nurse does not exceed 4. [115] Therefore, the 4 patients per nurse grouping represents a very difficult problem to solve and will serve as the testing ground of gMASH's performance. The problem is defined in the following Figure 4.14 through Figure 4.17 and Table 4-3.

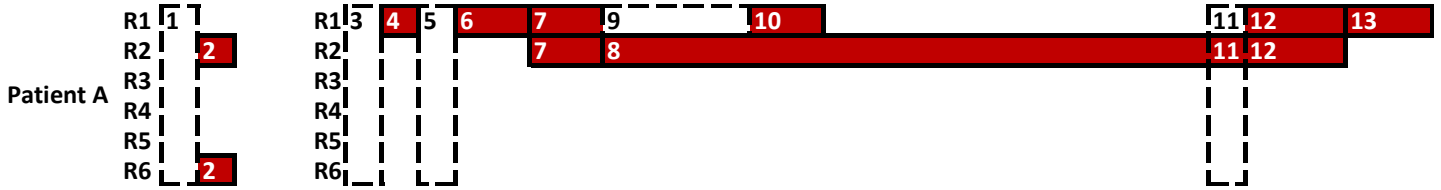


Figure 4.14: Dialysis procedure workflow of patient A

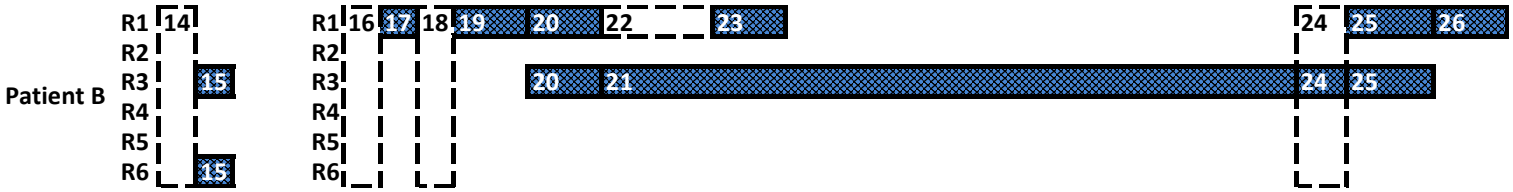


Figure 4.15: Dialysis procedure workflow of patient B

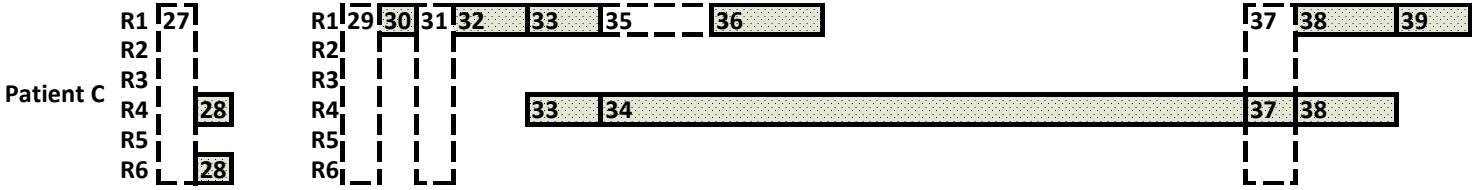


Figure 4.16: Dialysis procedure workflow of patient C

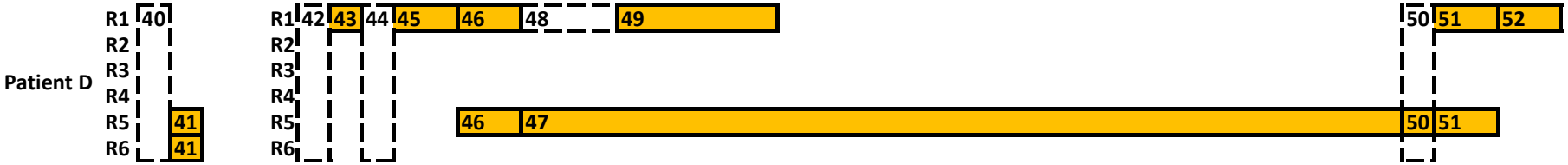


Figure 4.17: Dialysis procedure workflow of patient D

Table 4-3: Descriptions of dialysis procedure workflow

Patient	Activity	Description
A	2	Technician prepares dialysis machine R2
	4	Nurse prepares paperwork
	6	Pre-dialysis activities
	7	Patient A is connected to machine R2
	8	Machine R2 administers dialysis treatment
	10	Nurse administers IV EPO and assessment services
	12	Patient A is disconnected from machine R2
	13	Post-dialysis activities
B	15	Technician prepares dialysis machine R3
	17	Nurse prepares paperwork
	19	Pre-dialysis activities
	20	Patient B is connected to machine R3
	21	Machine R3 administers dialysis treatment
	23	Nurse administers IV EPO and assessment services
	25	Patient B is disconnected from machine R3
	26	Post-dialysis activities
C	28	Technician prepares dialysis machine R4
	30	Nurse prepares paperwork
	32	Pre-dialysis activities
	33	Patient C is connected to machine R4
	34	Machine R4 administers dialysis treatment
	36	Nurse administers IV EPO and assessment services
	38	Patient C is disconnected from machine R4
	39	Post-dialysis activities
D	41	Technician prepares dialysis machine R5
	43	Nurse prepares paperwork
	45	Pre-dialysis activities
	46	Patient D is connected to machine R5
	47	Machine R5 administers dialysis treatment
	49	Nurse administers IV EPO and assessment services
	51	Patient D is disconnected from machine R5
	52	Post-dialysis activities

Procedures with dashed outlines are gap activities and represent flexibilities in the dialysis workflow. Procedures between gap activities must be scheduled together as a block.

The first baseline schedule (Baseline 1) is generated based on an unintelligent, cascading strategy. The nurse will serve each patient sequentially, performing one block of activities for one patient then moving on to perform one activities block for another patient. For example, the nurse prepares paperwork for patient A then for patient B, patient C, then patient D. At which time, the nurse returns to patient A and perform the next block of activities: the pre-dialysis activities and so on for all patients. Baseline 1 is illustrated in Figure 4.18.

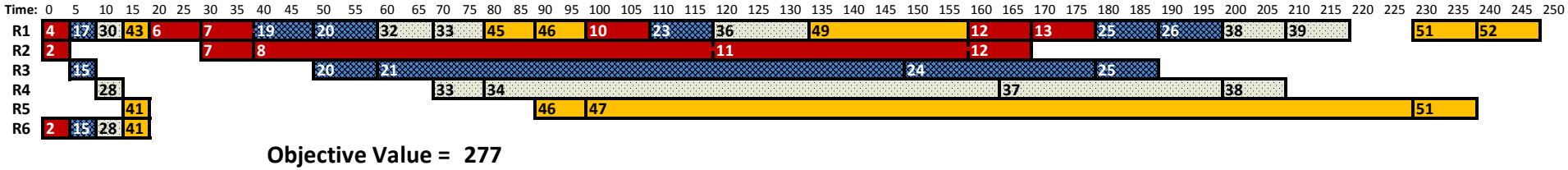


Figure 4.18: Manually generated schedule Baseline 1

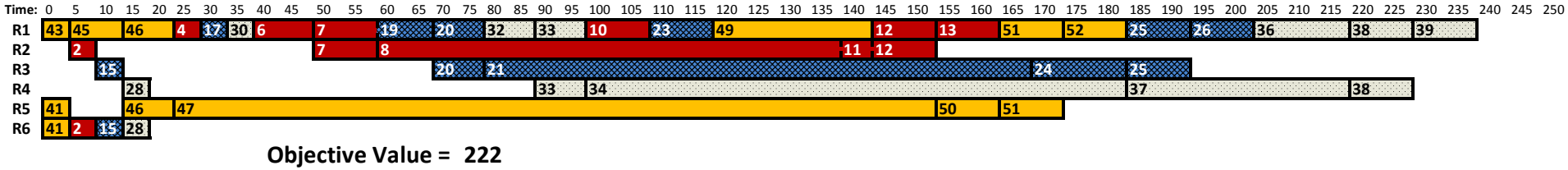


Figure 4.19: Manually generated schedule Baseline 2

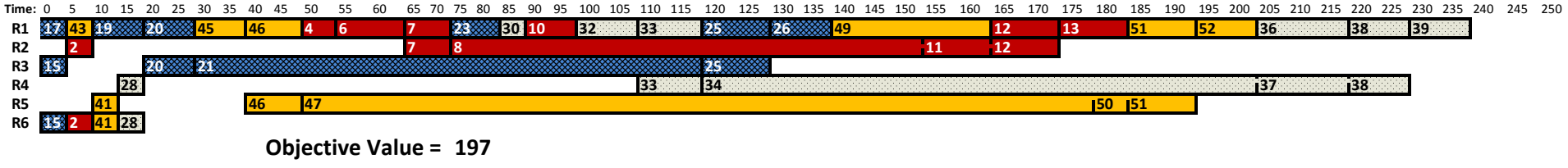


Figure 4.20: Heuristic gMASH solution with small population (multiplier = 10)

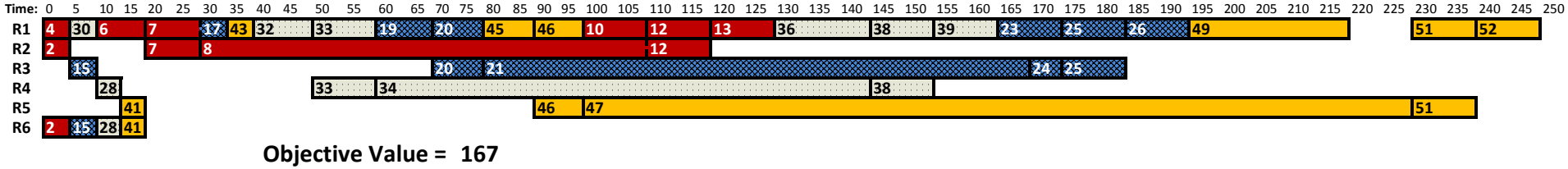


Figure 4.21: Heuristic gMASH solution with larger population (multiplier = 50)

Baseline 1 is a typical starting point for the manual scheduling process. A manual scheduler can then rearrange activities within that baseline schedule to try to improve it either through trial and error or through experience. For example, the second baseline schedule (Baseline 2) improves upon Baseline 1 by adding intelligence and logic into the schedule. One notices that patient D has a relatively long dialysis treatment period (activity 47). Baseline 2 schedules patient D to begin treatment first. The reason is that the shorter treatment periods of other patients are likely able to fit into the long treatment period of patient D and the total wait time of all patients will likely be less. Baseline 2 is illustrated in Figure 4.19. The simple injection of logic into Baseline 2 improved upon Baseline 1's objective value by 55. In this case, that improvement in objective value represents a 55-minute reduction in total patient wait time.

Scheduling patients with longer treatment periods earlier is not guaranteed to improve the schedule. One was simply lucky that this particular intuition worked to one's advantage. The hidden rules for improving schedules are no doubt very complex and problem specific. A manual scheduler or clinic manager may be able to infer some of those rules through experience. Unfortunately, that experience is very difficult to capture. An experienced scheduler of dialysis procedures was not available for this thesis. As a result, Baseline 2 is the best attempt at a manual schedule.

With baseline manual schedules established, the performance of gMASH can be evaluated. Figure 4.20 presents a schedule produced by gMASH. gMASH improved upon Baseline 2 by reducing total patient wait time by 15 minutes. Not a spectacular improvement but note that this solution was obtained using a relatively small population (population multiplier of 10). A wonderful feature of gMASH and indeed all genetic algorithms is that the quality of its solutions is related to its population size. Increasing the size of population increases the probability of converging on the optimal solution. Figure 4.21 presents a schedule produced by gMASH using a larger population (population multiplier of 50). The total patient wait time is reduced a further 40 minutes resulting in an objective value of 167. That is a 15% improvement over gMASH with a small population and a 25% improvement over the manual, Baseline 2 schedule. Also note that this solution is within approximately 3% of the exact solution which has an objective value of 162. In this case, that 3% error from the exact, optimal solution translates to mere 5 more minutes of total patient wait time.

In this example, gMASH absolutely outperforms manual scheduling. However, manual scheduling is a fine art and is highly subjective. An experienced scheduler can likely produce a schedule that is better than Baseline 2. He/she may, through experience or trial and error, even produce the optimal schedule. Manual scheduling is ideally the benchmark for testing the gMASH heuristic. Unfortunately, it is nearly impossible to objectively test gMASH against manual scheduling due to the subjective nature of manual scheduling. The only objective test of gMASH performance is a study of its repeatability and solution quality.

The next section will discuss the effect of population size on gMASH performance and quality of its solutions.

4.4.4 gMASH performance and solution quality

The stochastic search element inherent in genetic algorithms means gMASH can potentially produce a different solution every time it runs. Despite being a directed search algorithm, an element of randomness will always be present. Since gMASH cannot guarantee solution optimality, the user must be convinced of gMASH's reliability in other ways. One way is to compare gMASH solutions to known optimal solutions. We have already seen this comparison in sections 4.4.1 and 4.4.2. Unfortunately, problems to which exact optimal solutions are known are relatively simple and not large enough for practical application to medical procedures scheduling. An efficient exact solver for larger MPSP models does not exist. Therefore, exact optimal solutions for larger MPSP models are not available for comparison with heuristic solutions.

Another way to evaluate the reliability of gMASH is to study its repeatability and the distribution and quality of its solutions. The gMASH parameter with the greatest effect on solution quality is the population size. This section studies that effect. The study will be carried out on general problems A through E presented in Table 4-1 of section 4.4.1 and dialysis problems F through L presented in Table 4-2 of section 4.4.2. In addition, larger general problems involving 10 and 20 procedures will be studied. A larger dialysis problem involving 5 nurses, 20 patients, and 2 technicians will also be studied. The parameters of the population size study are outlined in the following Table 4-4 and Table 4-5.

Table 4-4: Population size study parameters for general procedure problems

Problem	Procedures	Multiplier values tested	Iterations
A	3	10, 30, 50	1000
B	4	10, 30, 50	1000
C	5	10, 30, 50	1000
D	6	10, 30, 50	1000
E	7	10, 30, 50	1000
E2	10	10, 30, 50, 100, 200	1000
E3	20	10, 30, 50, 100, 200	1000

Table 4-5: Population size study parameters for dialysis procedure problems

Problem	Nurses	Patients	Procedures	Multiplier values tested	Iterations
F	1	2	10	10, 30, 50, 100, 200	1000
G	1	3	15	10, 30, 50, 100, 200	1000
H	1	4	20	10, 30, 50, 100, 200	1000
I	2	4	20	10, 30, 50, 100, 200	1000
J	2	6	30	10, 30, 50, 100, 200	1000
K	3	6	30	10, 30, 50, 100, 200	1000
L	3	9	45	10, 30, 50, 100, 200	1000
L2	5	20	100	30, 50, 100, 200	100

Recall that the dimensionality of a problem is the number of procedures to schedule. The population size is proportional to the problem dimensionality via the population size multiplier. Population sizes studied range from the small (multiplier value of 10) to the very large (multiplier value of 200).

gMASH is run through 1,000 iterations for each problem at each multiplier value except problem L2. Problem L2 is the largest and most difficult problem to solve in this thesis. Although gMASH is relatively fast, solving problem L2 1,000 times at each population multiplier value for a total of 4,000 iterations is still impractical. Therefore, problem L2 is solved only 100 times at each population multiplier value.

The immediate effect of increasing the population size is higher computational cost. Figure 4.22 shows that for the general procedures problem, the computational cost increases linearly with increasing population size. The slope of that linear relationship also increases as the number of procedures to schedule increase, indicating that the problem is becoming more difficult to solve. Figure 4.23 and Figure 4.24 show the growth of computational cost for solving larger problems E2 and E3 with very large population multipliers up to 200. Again, a linear

relationship is observed. This means that the computational cost of using gMASH to schedule general procedures is predictable and does not grow exponentially.

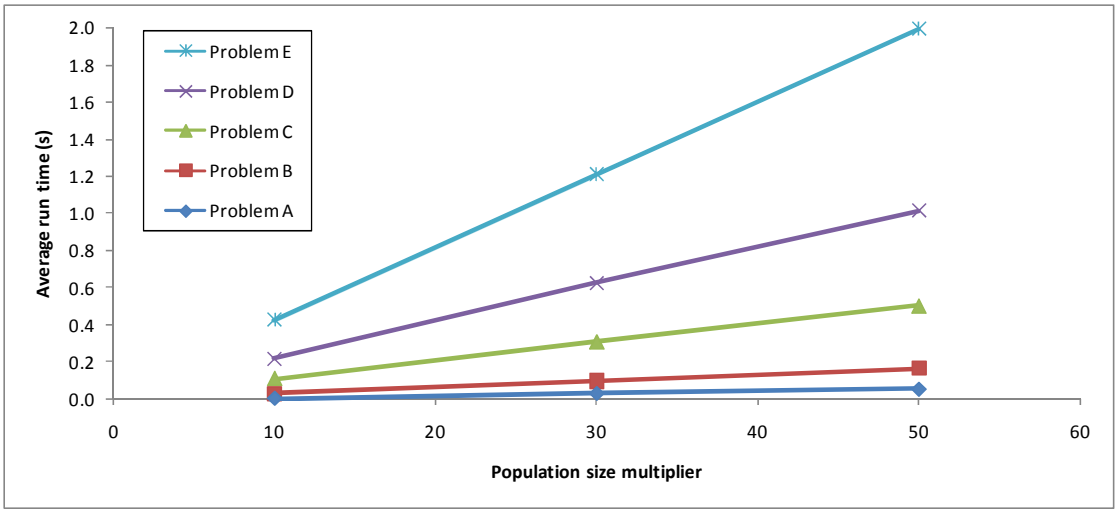


Figure 4.22: Increased computational cost due to larger population size for problems A through E

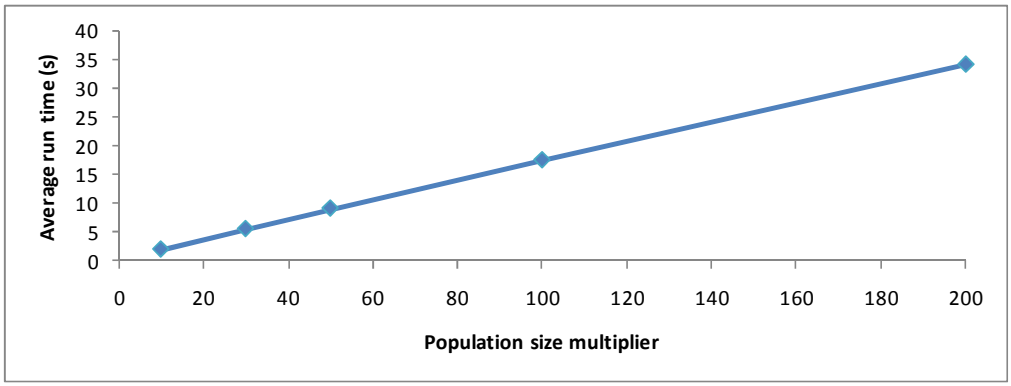


Figure 4.23: Increased computational cost due to larger population size for problem E2

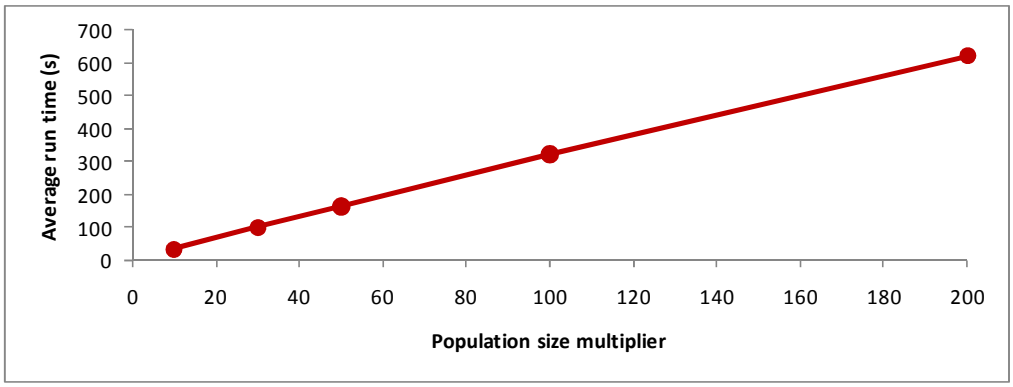


Figure 4.24: Increased computational cost due to larger population size for problem E3

A similarly linear growth in computational cost as population increase is experienced by gMASH in solving dialysis problems. Figure 4.25 shows computational effort growth for solving problems F through K. Figure 4.26 and Figure 4.27 show computational effort growth for solving problems L and L2 respectively.

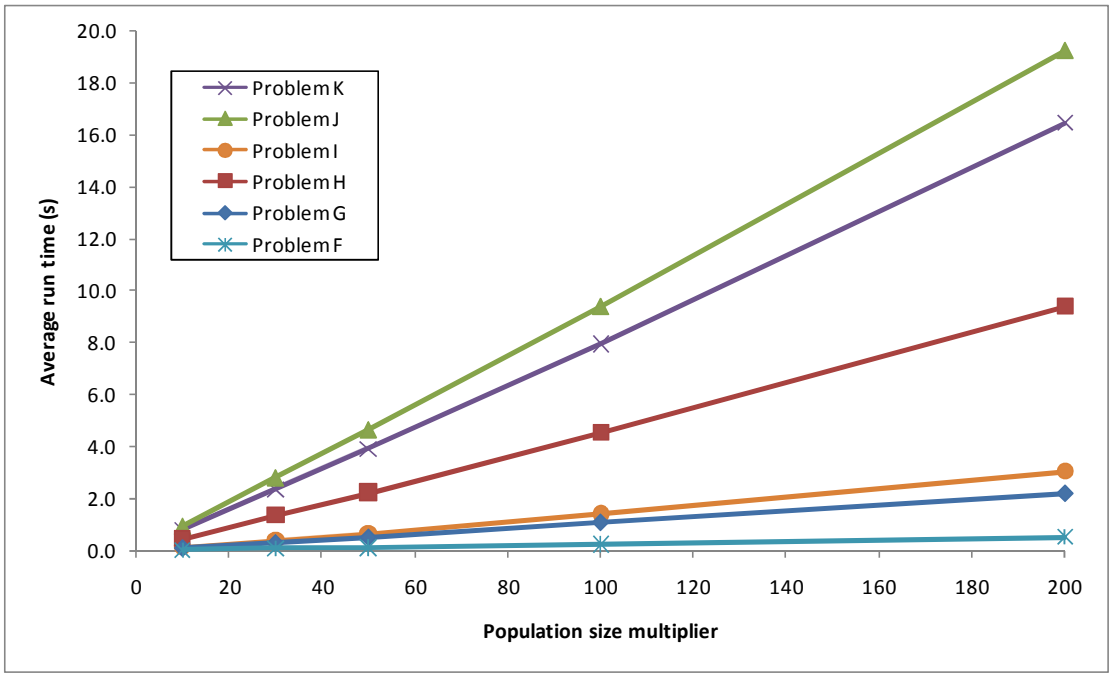


Figure 4.25: Increased computational cost due to larger population size for problems F through K

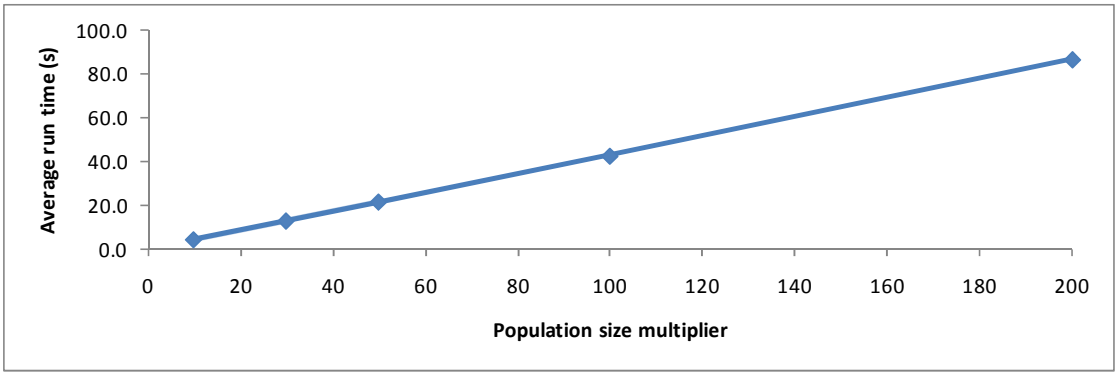


Figure 4.26: Increased computational cost due to larger population size for problems L

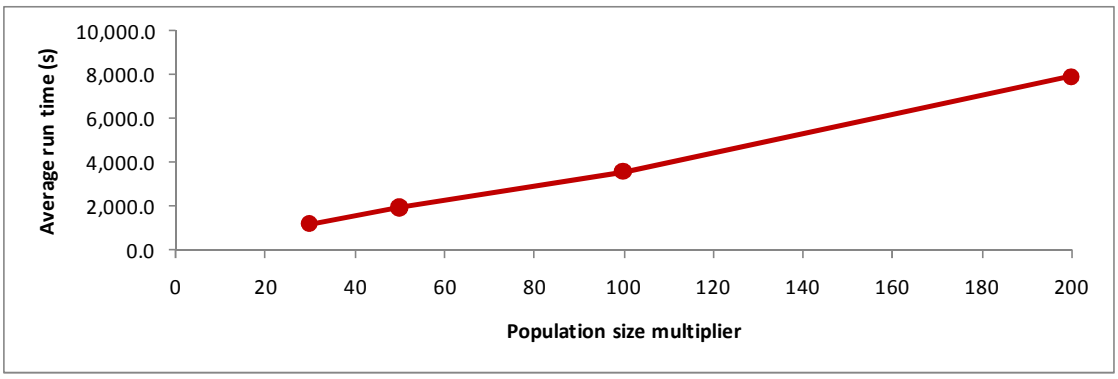


Figure 4.27: Increased computational cost due to larger population size for problems L2

With increased computational cost of using larger populations, one would expect the solution quality to improve. The solution quality is a difficult metric to define. For the small, easy problems A through K where the exact optimal solution is known, quality of the heuristic solution can be defined as expected error from optimal solution. The expected error from optimal is the average heuristic error from optimal solution over 1,000 iterations. Table 4-6 and Figure 4.28 show the average error from optimal of problems A through E. gMASH produces very high quality solutions for these small, easy problems A through E. Even with a small population multiplier of 10, gMASH produced solutions that are expected to be very close to optimal. gMASH is very fast and produces high quality solutions and is therefore far superior to the exact BnC method for solving the easy problems A through E.

Table 4-6: Average % error of problems A through E with different population sizes

	Procedures	Average % error at multiplier value:		
		10	30	50
Problem A	3	0.06%	0.00%	0.00%
Problem B	4	0.71%	0.01%	0.00%
Problem C	5	4.79%	0.29%	0.03%
Problem D	6	2.60%	0.45%	0.14%
Problem E	7	3.50%	1.38%	0.86%

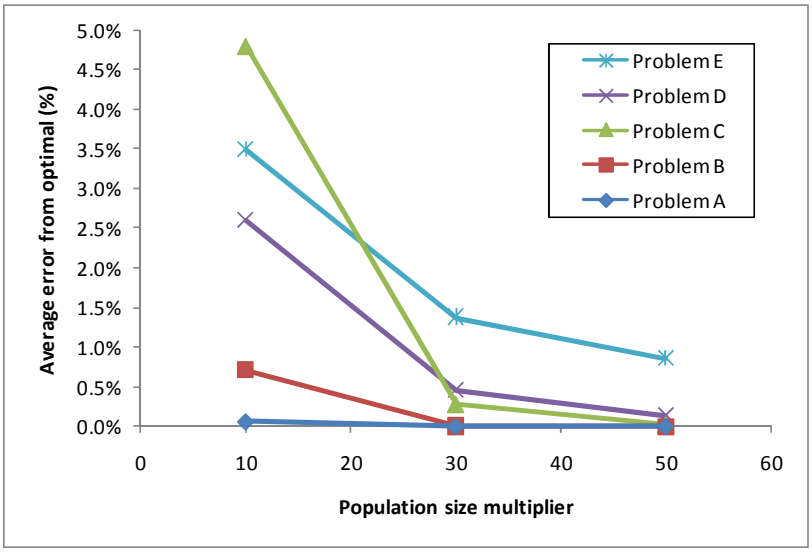


Figure 4.28: Reduction of average % error of problems A through E as population size increase

Similar behaviour is observed for dialysis scheduling problems. The expected error or average % error from optimal in solving problems F through K decreases with larger population sizes. See Table 4-7 and Figure 4.29.

Table 4-7: Average % error of problems F through K with different population sizes

	Nurses	Patients	Procedures	Average % error at multiplier value:				
				10	30	50	100	200
Problem F	1	2	10	0.10%	0.00%	0.00%	0.00%	0.00%
Problem G	1	3	15	3.80%	1.46%	0.61%	0.17%	0.03%
Problem H	1	4	20	21.78%	17.13%	15.28%	12.96%	10.89%
Problem I	2	4	20	0.05%	0.00%	0.00%	0.00%	0.00%
Problem J	2	6	30	9.21%	5.83%	4.71%	3.45%	2.33%
Problem K	3	6	30	6.43%	3.40%	2.45%	1.43%	0.94%

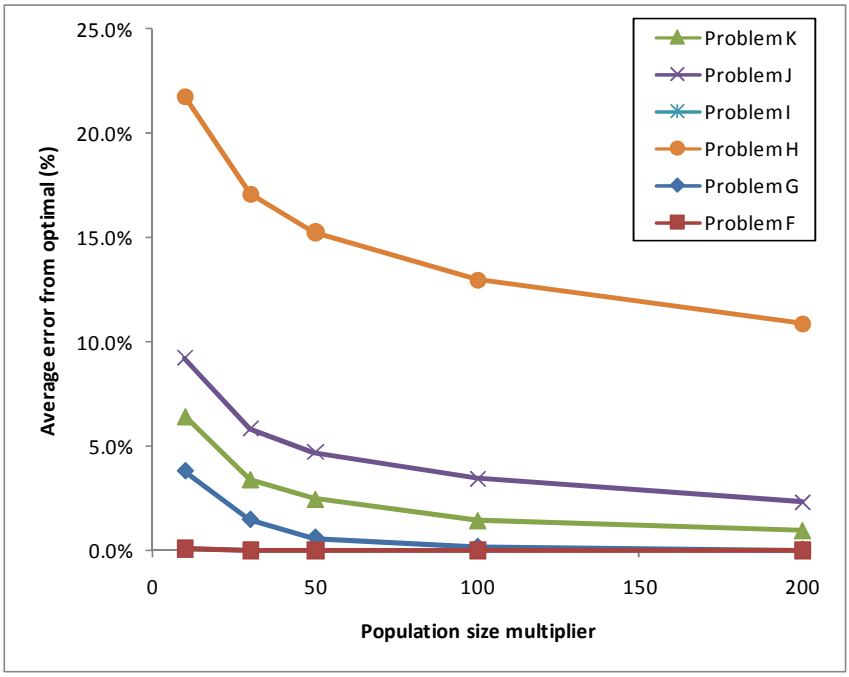


Figure 4.29: Reduction of average % error of problems F through K as population size increase

For larger problems E2, E3 L, and L2, the exact optimal solutions are not available. The solution quality in those cases may be defined as % error from best solution found. The assumption is that the best solution found in thousands of iterations is the optimal solution. Although it must be emphasized that optimality cannot be guaranteed. Table 4-8 and Figure 4.30 show the decrease of expected error in solving problems E2 and E3 using different population sizes.

Table 4-8: Average % error of problems E2 and E3 with different population sizes

	Procedures	Average % error at multiplier value:				
		10	30	50	100	200
Problem E2	10	8.06%	3.65%	2.29%	1.31%	0.59%
Problem E3	20	12.83%	8.49%	6.71%	5.03%	3.81%

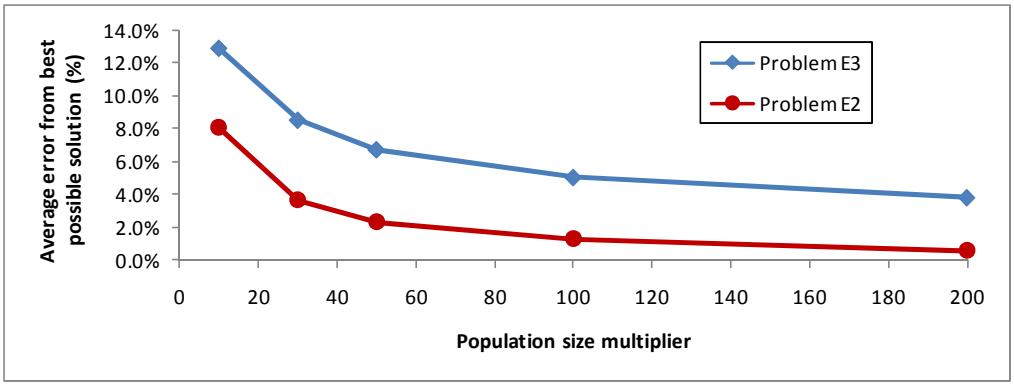


Figure 4.30: Reduction of average % error of problems E2 and E3 as population size increase

Table 4-9 and Figure 4.31 show a similar decrease in expected error in solving problems L and L2.

Table 4-9: Average % error of problems L and L2 with different population sizes

	Nurses	Patients	Procedures	Average % error at multiplier value:				
				10	30	50	100	200
Problem L	3	9	45	14.91%	10.71%	8.75%	7.18%	5.25%
Problem L2	5	20	100		5.48%	4.95%	4.23%	3.55%

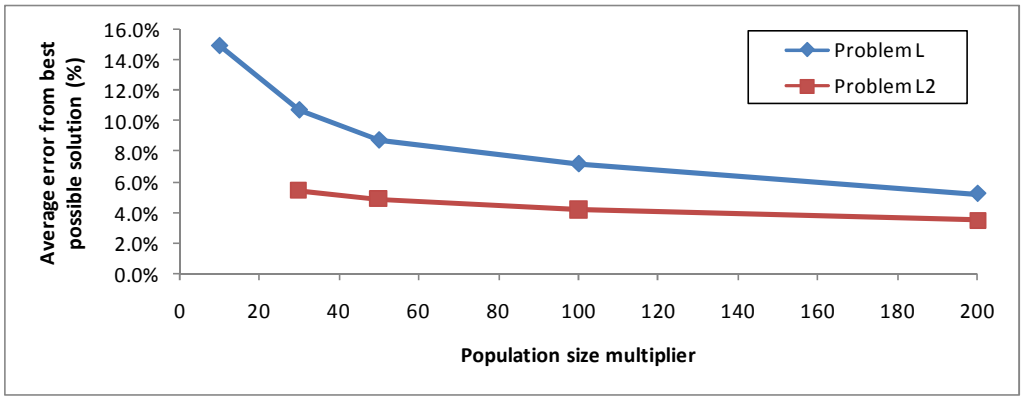


Figure 4.31: Reduction of average % error of problems L and L2 as population size increase

It has been shown that increasing the population size of gMASH decreases the expected error of its solutions but at increased computational cost. The user has freedom to decide the tradeoff between accuracy and speed.

This study of gMASH solution quality thus far also shows once again that the dialysis procedures scheduling problem can be more difficult than the general procedures scheduling problem. Each dialysis appointment for one patient requires 5 procedures to model. 30 procedures are needed to model one shift of a clinic serving 6 patients simultaneously. Therefore the dimensionality of the dialysis scheduling problem can be high. In addition, the dialysis workflow contains many precedence requirements. The chromosome repair function of gMASH ensures that precedence is enforced. However, some diversity in the population is lost as many originally different chromosomes are necessarily repaired to encode the same solution. As a result, a large population is needed to produce high quality solutions. The added computational cost associated with large populations contributes to the difficulty of scheduling dialysis procedures.

gMASH reliably produces high quality solutions for easy problems. Average gMASH solution error to problems A through E are close to zero even using a modest population size multiplier of 30. In other words, gMASH has a very high probability of producing the optimal solution in those easy problems. In larger problems E2, E3, L, and L2 however, the expected error of gMASH solutions is higher. It is worth investigating the repeatability of gMASH and distribution of its solutions for those problems.

We begin with problem E2 of scheduling 10 procedures. Table 4-10 shows the frequency of gMASH solutions out of 1,000 iterations at each population size multiplier value that fell within certain % error ranges. Recall that exact optimal solution is not available for problem E2. The error referred to in Table 4-10 is the error from best solution found. Although optimality cannot be guaranteed, one can be confident that the best solution found in 5,000 iterations of gMASH is very likely optimal. Figure 4.32 through Figure 4.34 are visualizations of the gMASH solution distribution shown in Table 4-10.

Table 4-10: Distribution of gMASH solutions for problem E2 at different multiplier values

% error	Frequency at multiplier value:				
	10	30	50	100	200
0 - 5%	350	715	861	936	987
5- 10%	365	225	119	54	13
10 - 15%	146	51	19	9	0
15 - 20%	103	8	1	1	0
20 - 25%	34	1	0	0	0
25 - 30%	2	0	0	0	0
30 - 35%	0	0	0	0	0
35 - 40%	0	0	0	0	0
>40%	0	0	0	0	0

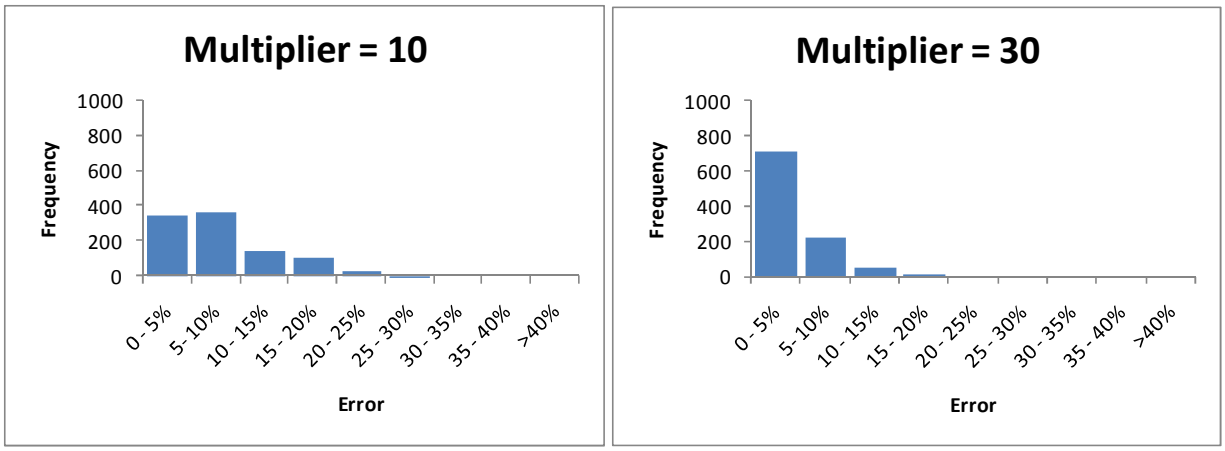


Figure 4.32: gMASH solution distribution for problem E2 at multiplier values of 10 and 30

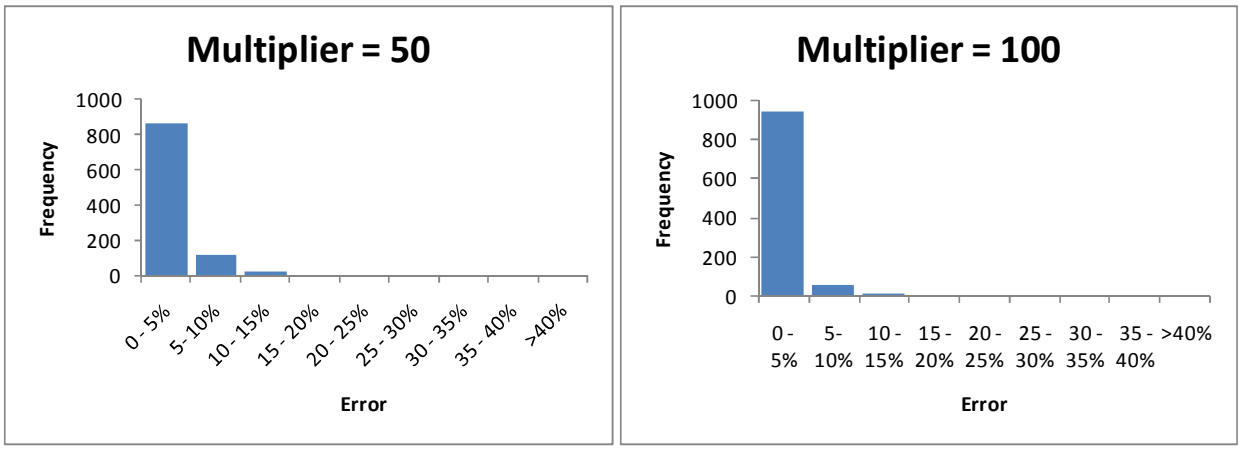


Figure 4.33: gMASH solution distribution for problem E2 at multiplier values of 50 and 100

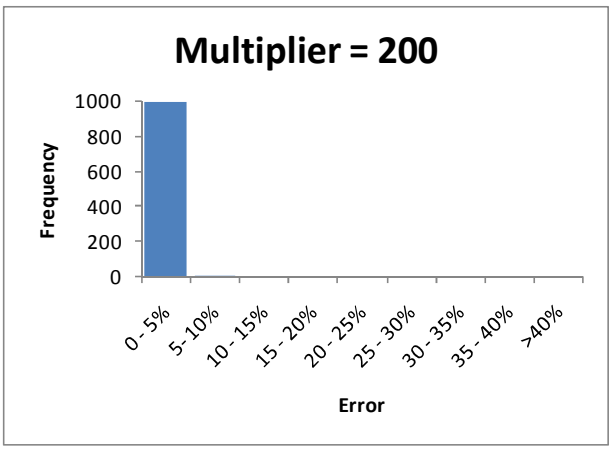


Figure 4.34: gMASH solution distribution for problem E2 at multiplier value of 200

The gMASH solution distributions in Figure 4.32 through Figure 4.34 show that as population size increase, the probability of obtaining a high quality solution increases. With a small population (multiplier of 10) 350 out of 1,000 iterations produced solutions with less than 5% error. In other words: a small population has 35.0% chance of producing a very high quality solution with less than 5% error. That probability of obtaining very high quality solutions increased to 71.5% with a population size multiplier of 30, 86.1% with multiplier of 50, and 93.6% with multiplier of 100. Finally, gMASH with a very large population size (multiplier of 200) has 98.7% chance of producing very high quality solutions. This improvement in solution quality is illustrated in Figure 4.35 along with average gMASH run-time. The reader is reminded of the rising computational cost associated with higher solution quality.

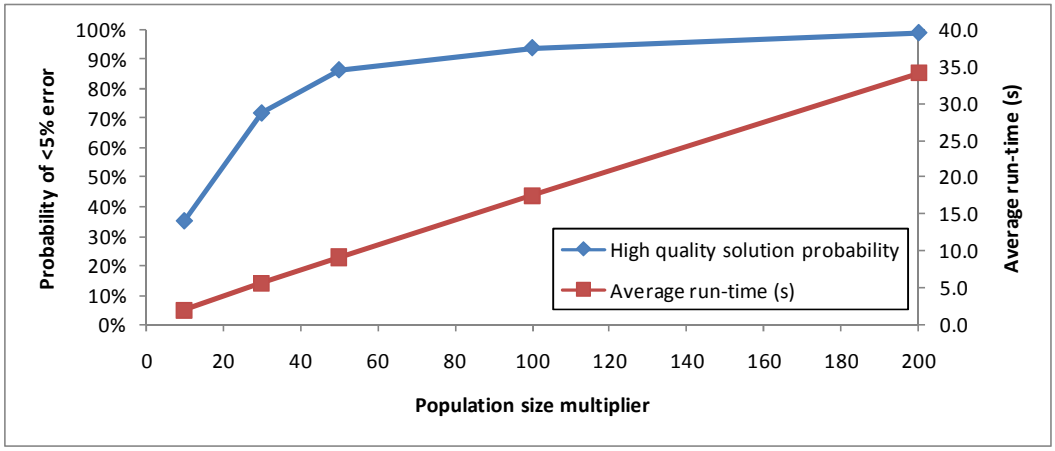


Figure 4.35: Rising problem E2 solution quality and computational cost with larger population

Similar behaviour is observed when solving problem E3 with different population sizes. Table 4-11 and Figure 4.36 through Figure 4.38 summarize and visualize the distribution of gMASH solutions for problem E3.

Table 4-11: Distribution of gMASH solutions for problem E3 at different multiplier values

% error	Frequency at multiplier value:				
	10	30	50	100	200
0 - 5%	29	196	336	560	740
5- 10%	256	467	503	379	238
10 - 15%	405	283	144	59	22
15 - 20%	243	53	17	2	0
20 - 25%	62	1	0	0	0
25 - 30%	5	0	0	0	0
30 - 35%	0	0	0	0	0
35 - 40%	0	0	0	0	0
>40%	0	0	0	0	0

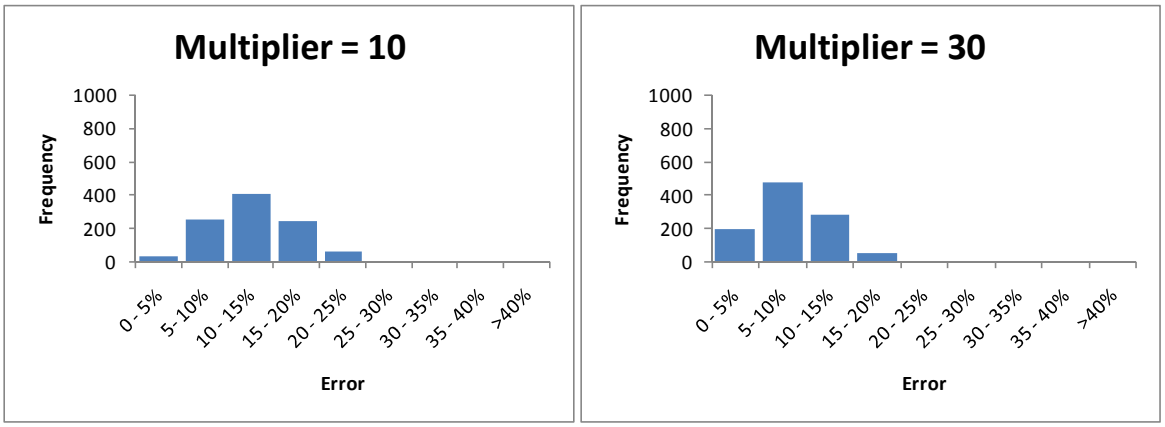


Figure 4.36: gMASH solution distribution for problem E3 at multiplier values of 10 and 30

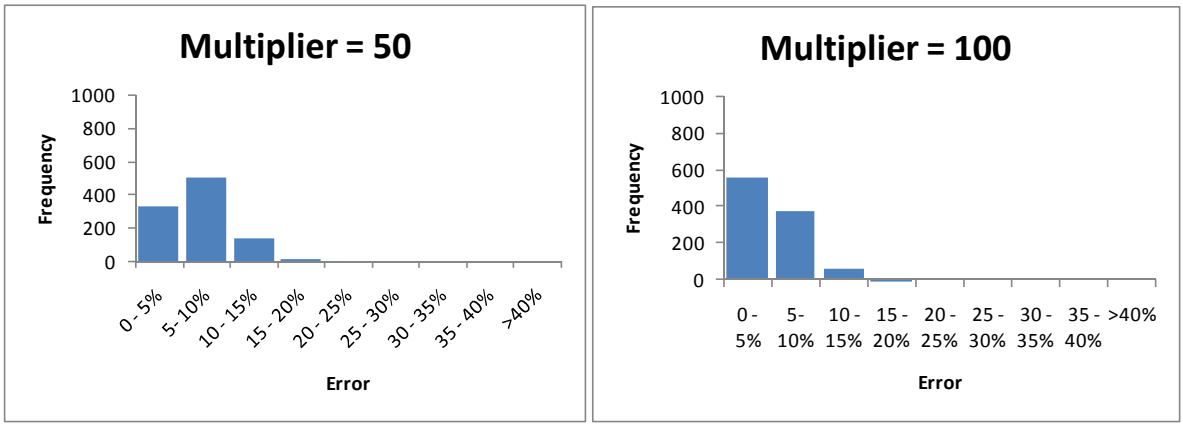


Figure 4.37: gMASH solution distribution for problem E3 at multiplier values of 50 and 100

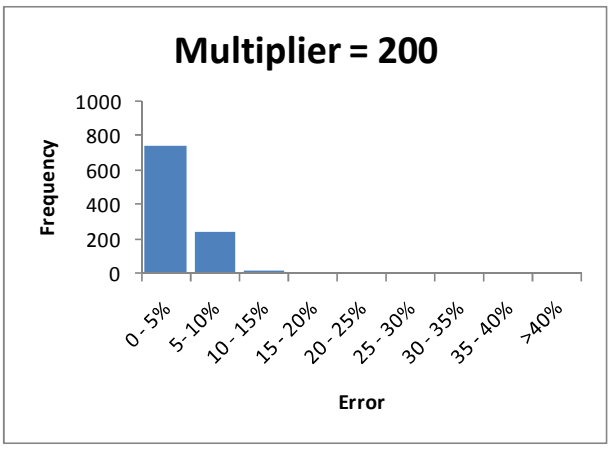


Figure 4.38: gMASH solution distribution for problem E3 at multiplier value of 200

Figure 4.36 shows that when solving the difficult problem E3, a small population size produces poor quality solutions. At a multiplier value of 10, gMASH has a 40.5% chance of producing a solution with between 10 and 15% error, but only 2.9% chance of producing high quality solutions with less than 5% error. With increasing population size however, the solution quality gradually improves to the point a very large population with multiplier value of 200 has 74.0% chance of producing a very high quality solution. Figure 4.36 through Figure 4.38 show the shifting of gMASH solution quality towards lower error. Figure 4.39 shows the increase in solution quality with larger population size and once again reminds the reader of the associated rise in computational cost.

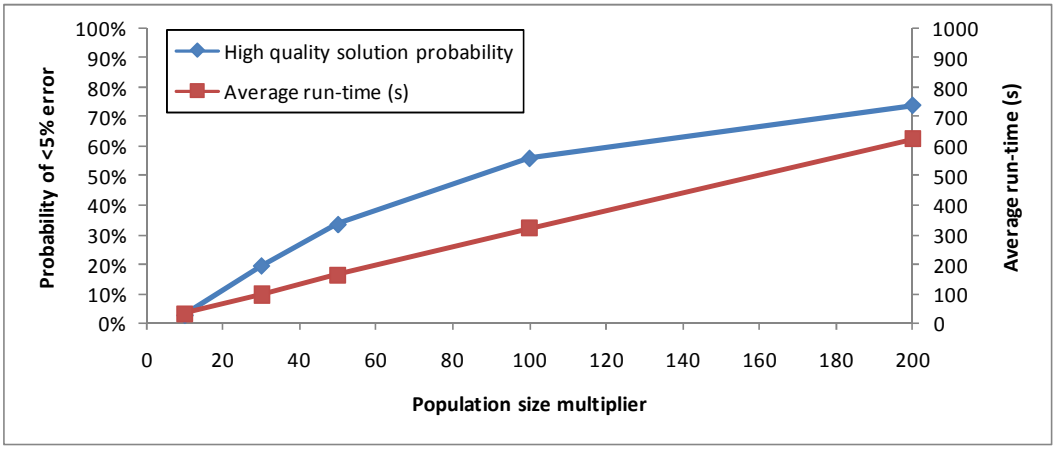


Figure 4.39: Rising problem E3 solution quality and computational cost with larger population

The same relationship between population size and solution quality extends into the larger and more difficult to solve dialysis problems L and L2. Although as discussed earlier, gMASH population for dialysis problems are not as diverse as in the general procedures problems; as a result, the improvement in solution quality with increasing population size will not be as impressive as with general procedures problems.

Table 4-12 and Figure 4.40 through Figure 4.42 summarize and illustrate the distribution of gMASH solutions for problem L at different population size multiplier values.

Table 4-12: Distribution of gMASH solutions for problem L at different multiplier values

% error	Frequency at multiplier value:				
	10	30	50	100	200
0 - 5%	13	125	263	418	694
5- 10%	180	409	454	421	248
10 - 15%	369	338	232	144	52
15 - 20%	326	118	49	16	6
20 - 25%	105	10	2	1	0
25 - 30%	7	0	0	0	0
30 - 35%	0	0	0	0	0
35 - 40%	0	0	0	0	0
>40%	0	0	0	0	0

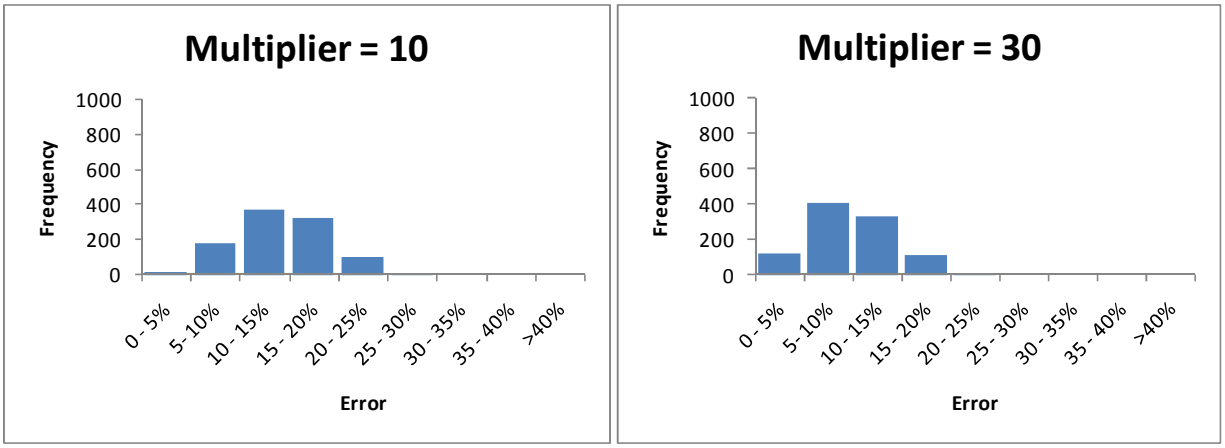


Figure 4.40: gMASH solution distribution for problem L at multiplier values of 10 and 30

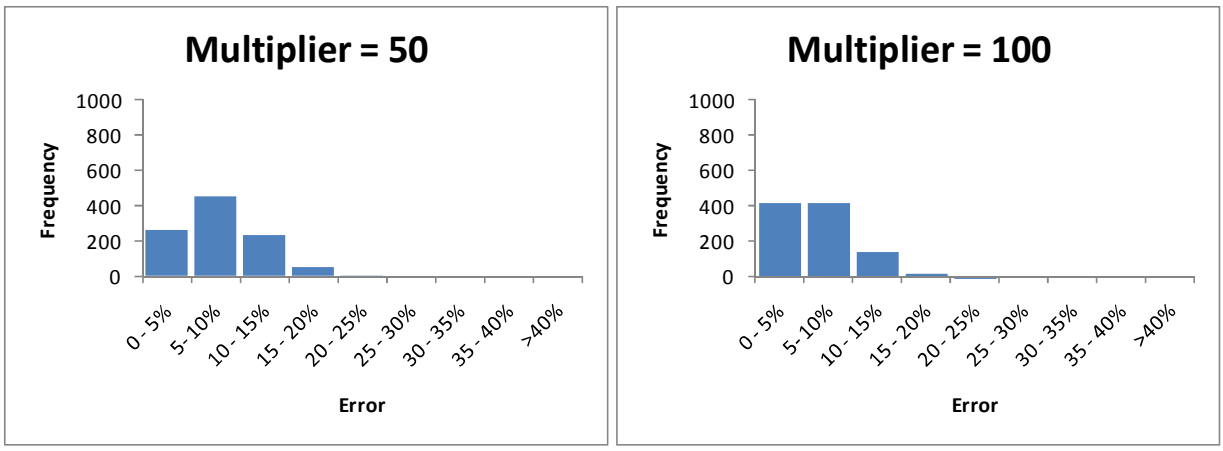


Figure 4.41: gMASH solution distribution for problem L at multiplier values of 50 and 100

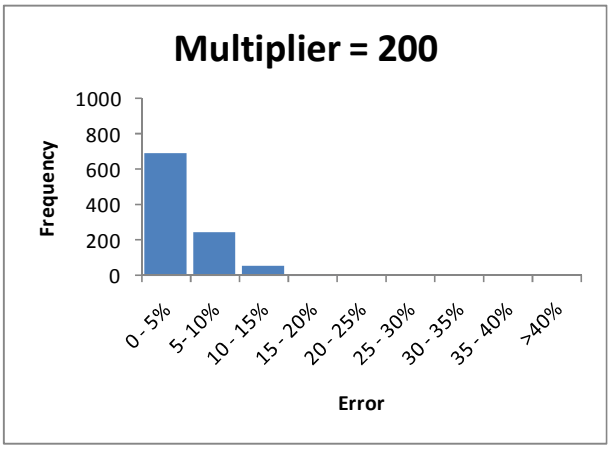


Figure 4.42: gMASH solution distribution for problem L at multiplier value of 200

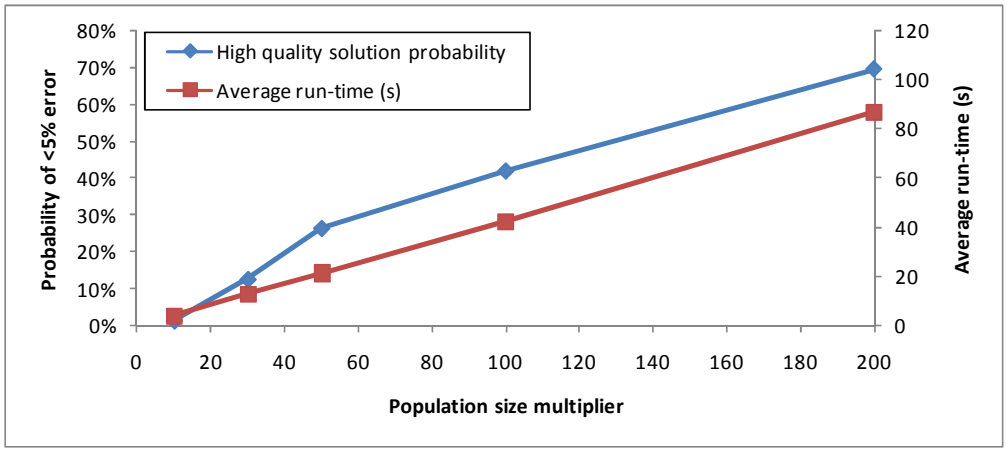


Figure 4.43: Rising problem L solution quality and computational cost with larger population

Figure 4.43 shows the improvement in solution quality and increase in computational cost with increasing population size. The difficulty of solving problem L is obvious as a very large population is required to achieve 69.4% chance of obtaining a very high quality solution with less than 5% error. Note however, that gMASH still far exceeds the performance of the exact BnC solver. Recall that the BnC solver ran for nearly three days, consumed all memory on the test computer, and crashed without finding the optimal solution. The best solution BnC found at the time of crash had an objective value of 212. The best solution found by gMASH in this study had an objective value of 207, which is 2.4% better than the BnC solution. Even using a very large population size, gMASH's average run-time is 86.6 seconds, orders of magnitude lower than the exact BnC solver. A little bit of uncertainty in solution quality is a small sacrifice for that low computational cost. The reader is reminded that freedom to decide the tradeoff between solution quality and computational cost rests with the user. The uncertainty in solution quality can also easily be covered by iterating gMASH multiple times for the same problem to increase the probability of obtaining a high quality solution. The low computational cost makes repeat iterations of gMASH practical.

Finally, the largest, most difficult dialysis scheduling problem L2 involves 5 nurses, 20 patients, 2 technicians and requires 100 procedures to model. Table 4-13, Figure 4.44 and Figure 4.45 summarize and illustrate the distribution of gMASH solutions for problem L2 at different population size multiplier values.

Table 4-13: Distribution of gMASH solutions for problem L2 at different multiplier values

% error	Frequency at multiplier value:			
	30	50	100	200
0 - 5%	29	54	78	83
5- 10%	69	44	22	17
10 - 15%	2	2	0	0
15 - 20%	0	0	0	0
20 - 25%	0	0	0	0
25 - 30%	0	0	0	0
30 - 35%	0	0	0	0
35 - 40%	0	0	0	0
>40%	0	0	0	0

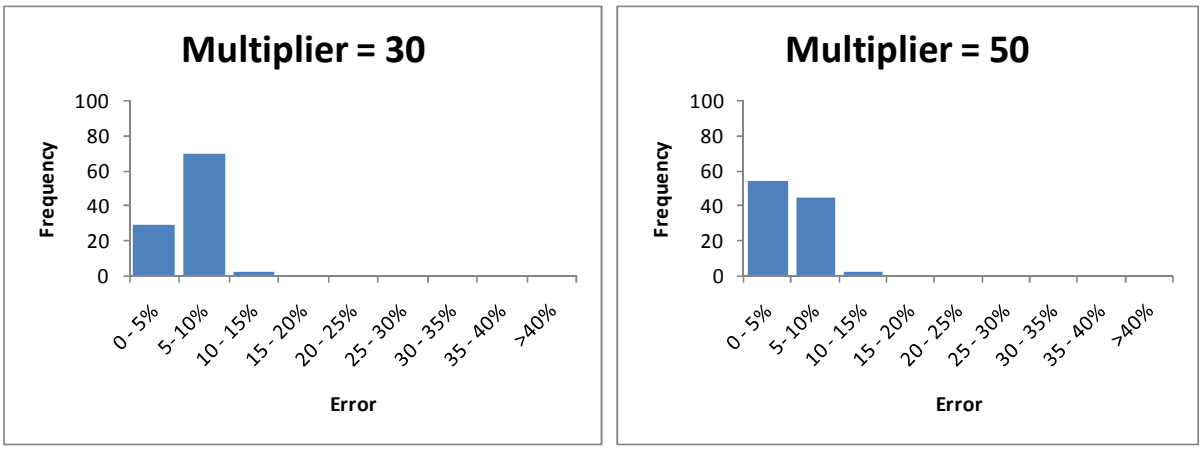


Figure 4.44: gMASH solution distribution for problem L2 at multiplier values of 30 and 50

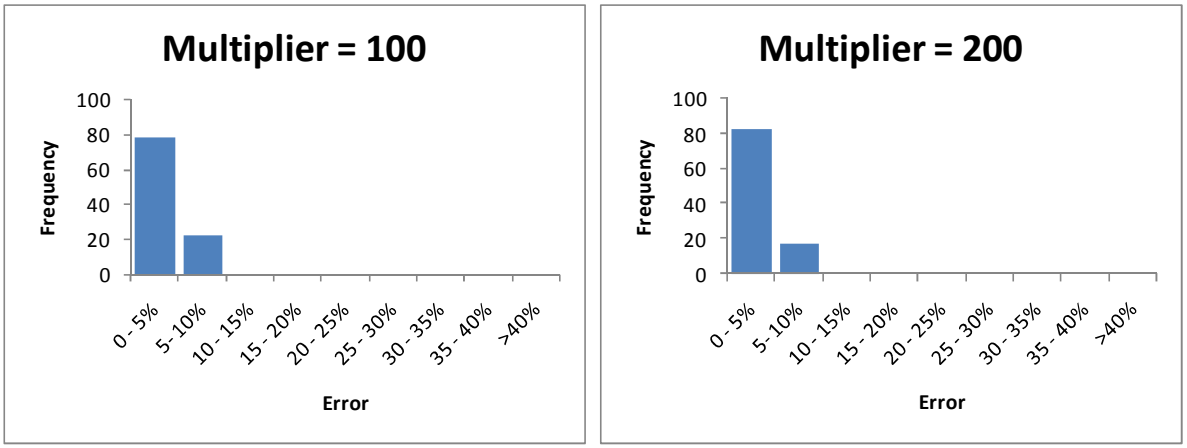


Figure 4.45: gMASH solution distribution for problem L2 at multiplier values of 100 and 200

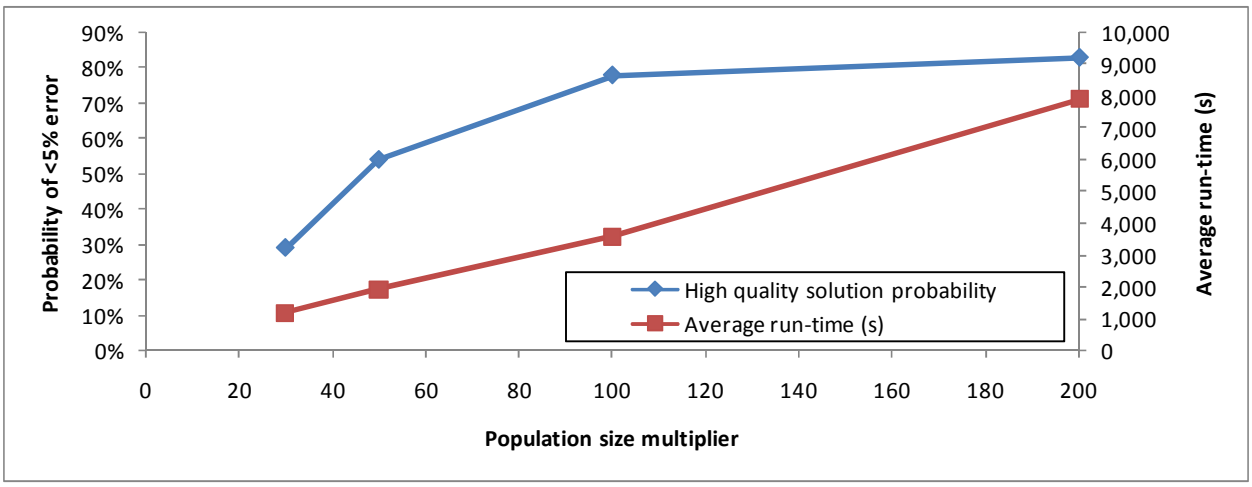


Figure 4.46: Rising problem L2 solution quality and computational cost with larger population

Figure 4.46 shows the improvement in solution quality and increase in computational cost with increasing population size. Note that the computational cost is starting to become significant. Solving problem L2 with a very large population (multiplier value of 200) takes on average 8,000 seconds which is approximately three and quarter hours. The user's decision on trade-off between solution quality and computational cost becomes more difficult to make in larger problems.

In summary, although gMASH cannot guarantee solution optimality, its solution quality can be reliable and predictable. gMASH is capable of producing very high quality solutions if large population size is used. The computational cost of gMASH is orders of magnitude lower than the exact BnC solver and therefore much more practical. The user has freedom to decide the trade-off between solution quality and computational cost. gMASH is a competent algorithm for solving the medical procedures scheduling problem.

4.5 Why is gMASH fast?

The fast performance of gMASH is attributed to its exploitation of the problem structure. The true decision variables in a scheduling problem are the starting times of individual procedures. However, procedure starting times are subject to resource conflict constraints. As a result, the continuous solution space of starting times of procedures includes mostly infeasible solutions. gMASH decomposes the scheduling problem variables into simply the ordering of procedures. In doing so, gMASH created a solution space that not only is smaller but is

populated entirely by feasible solutions. That is, any ordering of procedures results in a feasible solution. All gMASH has to do is then to find the optimal ordering of procedures.

Genetic algorithms maintain multiple strings or solutions in its population. That is, genetic algorithms can evaluate multiple solutions simultaneously. gMASH takes advantage of GA's powerful parallel processing feature to efficiently search through the solution space. gMASH is also a directed search algorithm that guides the search toward optimal solutions without having to iterate through the entire solution space.

The exploitation of problem structure combined with genetic algorithms' powerful parallel processing feature are responsible for gMASH's high performance.

4.6 Scalability of gMASH

Similar to the MPSP model, gMASH has infinite modeling capabilities. That is, procedures are modeled as simple 0-1 matrices and therefore gMASH can be used to model any number of procedures and clinics of any size. The usefulness of gMASH, like the MPSP model is limited by the solvability of larger sized models. The speed of gMASH is linearly proportional to the size of the population. Larger problems require larger population size to solve. Unfortunately, the population size required to achieve a certain level of solution quality grows exponentially with increasing problem size. As a result, the computational effort required to obtain high quality solutions grows exponentially with increasing problem size. The relationship between computational effort and problem size depends on the difficulty level of the particular problem. However, the growth is much more manageable than the n^{th} degree exponential growth in computational cost of solving the MPSP model exactly. In addition, the computational cost of gMASH is orders of magnitude less than the computational cost of the exact BnC solver in solving the same problem. Therefore, gMASH can be scaled to solve much larger problems than can be solved using BnC. For example, using population multiplier of 50, gMASH was able to schedule 50 general procedures in approximately 34 minutes. Recall that the MPSP model became intractable for problems with more than 7 procedures. Again using population multiplier of 50, gMASH scheduled 5 nurses, 20 patients, and 2 technicians also in approximately 34 minutes. gMASH is a much more practical algorithm for solving the MPSP model than the exact BnC method.

4.7 gMASH improves branch and cut solver performance

The original purpose of the gMASH heuristic was to generate a good solution to feed into the BnC solver to improve scalability of the exact MPSP model. gMASH has since emerged as a competent scheduler on its own but nevertheless, let us see if it can help improve the BnC solver performance.

In addition to finding a good initial solution, gMASH can also estimate a tight initial solution bound. This is done by iterating gMASH for the same problem multiple times but ignoring conflicts in a different resource at each iteration. For the example clinic with 7 resources, gMASH is iterated 8 times (iterations 0 to 7). The first iteration, iteration 0 is a regular run of gMASH to obtain the good initial solution. The next iteration, iteration 1 runs gMASH again but ignores conflicts in resource R1. This simulates unlimited availability of resource R1. The next iteration, iteration 2 ignores resource R2 and so on through to iteration 7 which ignores resource R7. These iterations reveal the bottleneck resource of the problem. For example, let's say in ignoring resource R4, iteration 4 produced a schedule that is much better than any other iteration; we then know that resource R4 is the limiting resource. The objective value of that best schedule represents a boundary for the objective value of the current problem. That is, the current problem cannot possibly have a better solution than the same problem with its limiting resource relaxed. In providing both a good initial solution and a tight solution bound, gMASH focuses the branch and cut solver onto a narrow band in the MPSP model solution space. This is best illustrated with an example. Consider problem E in Table 4-1. The exact solution of 653 took the BnC solver 3 hours to find. For the same problem E, gMASH found a sub-optimal but good initial solution with objective value of 673 and a solution bound of 593. Feeding that initial solution and solution bound to the BnC solver, the run-time was reduced to 44 minutes. The 76% reduction in computational cost is significant but not as dramatic as hoped.

Even in the best case scenario where the initial solution is the optimal solution with objective value of 653 and a very tight solution bound of 603 is enforced, the BnC solver still ran for 36 minutes to confirm optimality of that initial solution.

Previously intractable problems can now be solved. The BnC solver worked on the problem of scheduling 8 procedures with 60 activities for over 24 hours without finding the optimal solution. With help of initial solution and bound from gMASH, that problem was solved

in 6 hours. This means that gMASH is able to help the BnC solver scale up to solve slightly larger MPSP models. However, 6 hours is still an unacceptable solve-time for the simple problem of scheduling 8 procedures. Also, the problem of scheduling 9 procedures, even with help from gMASH, ran for over 24 hours without finding the optimal solution. Solving the 9 procedures scheduling problem exactly, for all intents and purposes, is considered intractable.

Feeding the BnC solver with a good initial solution and tight solution bound from gMASH does indeed improve its performance. However, that improvement does not break the n^{th} degree exponential growth of the computational cost for solving the MPSP model exactly. The MPSP model still becomes intractable too quickly. Unfortunately, even with help from gMASH, solving the MPSP model exactly is not practical for real-time scheduling. The focus of future work should be on improving performance of the gMASH heuristic as a standalone scheduler in its own right.

However, there is merit in pursuing exact, guaranteed optimal solutions. The general procedures model was designed to be very flexible to model a variety of different procedures. Different procedures require different scheduling strategies. Different scheduling strategies change the nature of the scheduling problem which affects the performance of gMASH. For each different scheduling problem, the BnC solver should be used to find exact, optimal solution for small problems. Those solutions become reliable benchmarks to help customize genetic algorithm parameters to optimize gMASH performance for larger problems.

Chapter 5: Conclusion

5.1 Exact or good enough?

The MPSP model developed in this thesis is flexible enough to model a variety of problems. It can be solved exactly using branch and cut (BnC) method to guarantee optimality of its solutions. The healthcare industry is relatively resistant to computerization of some of its processes not out of old fashion but out of liability concern. Therefore, if software is to succeed in the healthcare industry, it must be proven superior to established techniques. The promise of exact, optimal solutions is very enticing. Healthcare administrators will have little reservation in accepting help from software tools that guarantees to improve their processes. Unfortunately, that guarantee carries enormous computational cost. Each new variable in the MPSP model adds a new dimension to the problem. Additional dimensions inflate the solution space exponentially which makes the optimal solution much more difficult to find. Scheduling is a deceptively complex problem. Each additional procedure to schedule requires many variables to model. As a result, scaling the MPSP model up to schedule more procedures rapidly inflates the dimensionality of the problem. Solving the MPSP model exactly takes impractically long time even for small scheduling problems involving only 5 or 6 procedures. The MPSP model becomes intractable for problems involving 7 or 8 procedures. With help of good initial solution and tight solution bound from gMASH, the BnC solver can solve slightly larger MPSP models; unfortunately, not enough to solve practical sized real world problems. The guarantee of optimality is not worth the impractical computational cost. Therefore, solving the MPSP model exactly using BnC method is not practical for real time scheduling. However, small MPSP models should be solved exactly using BnC method to set performance benchmarks that help optimize parameters of the evolutionary heuristic gMASH.

The gMASH evolutionary heuristic was originally intended as an assistant tool to find good initial solutions to improve performance of the BnC solver in solving MPSP model; which it does, quite well. However, the performance of gMASH turned out to be so good and customizable that it emerged as a strong, competent scheduler on its own. gMASH, due to its genetic algorithm nature cannot guarantee optimality but its solutions can be very good. The size of the population can be adjusted to increase probability of finding the exact optimal solution and

to decrease the expected error of the heuristic solution. gMASH's ability to produce near optimal solutions give users confidence that its solutions are better than manually generated schedules. Most importantly, gMASH produces high quality solutions very quickly with much less effort than manual scheduling. gMASH does not experience n^{th} degree exponential growth in computational cost required to solve the MPSP model exactly. The run time of gMASH is orders of magnitude lower than that of the BnC solver. Therefore, gMASH can solve much larger and more practical MPSP models than the BnC solver.

gMASH produces very good solutions very quickly. Its performance is predictable and customizable. Its computational cost growth is manageable. It can be scaled up to solve practically sized problems. The user is given freedom to decide trade-off between solution quality and computational cost. In conclusion: the evolutionary heuristic gMASH developed in this thesis is a practical and highly competent algorithm for solving the MPSP model.

5.2 Future work

The contribution of this thesis should be expanded further in three areas: core algorithm, modeling scope, and business intelligence.

5.2.1 Improve core algorithm

gMASH, at its core, is a basic genetic algorithm. Its effectiveness is proven but there should be room for improvement. Future work should investigate different GA parameters such as initial population generation strategy, chromosome selection strategy, mutation rate, crossover strategy, replacement strategy, and convergence criteria. For example, the initial population can be seeded with good solutions found using simple priority rule based heuristics like in project scheduling problems. That should result in an initial population of very good solutions. The GA will start off closer to the optimal solution and therefore should converge very quickly. More advanced evolutionary techniques such as adaptive niching [119] should also be investigated.

gMASH can be further enhanced with other search techniques. For example, the genetic algorithm search aspect of gMASH quickly explores as much of the solution space as possible to identify neighborhoods that the optimal solution potentially resides in. A local search such as tabu search or simulated annealing can then be used to thoroughly explore those neighborhoods to pin-point the optimal solution.

5.2.2 Expand modeling scope

The modeling capability of the MPSP model has been demonstrated for general procedures and one specific procedure: haemodialysis. Future work should study workflows of other medical procedures for the MPSP model to model. For example: medical imaging, surgery, neonatal care etc. The true flexibility of the MPSP model capability should be tested on a wide variety of different procedures. The modeling capability and versatility of the MPSP model should be continuously improved.

Research effort should be directed at studying uncertainty in scheduling of medical procedures. The MPSP model should be improved to handle uncertainty to become a truly realistic and effective medical procedures model. For example, dynamic scheduling capability should be implemented to handle unexpected delays, emergency procedures, or unexpected loss of resources. gMASH should then be enhanced to solve the more realistic MPSP model.

5.2.3 Develop business intelligence

The MPSP model, in its current form, is nothing more than an automated scheduler. More work is required to develop a resource management tool to intelligently use the MPSP model and gMASH. Such a tool should be able to identify bottleneck resources and inform the user how much more of that resource is required to relieve the bottleneck. Conversely, the tool should also identify overabundant resources so that an administrator can redistribute and make better use of those resources. The tool should be able to simulate scenarios to help administrators plan for unexpected change. The tool should have database memory and take feedback so that learning functions can be implemented. Adding business intelligence to the MPSP model and gMASH is of less academic interest but will result in a valuable management tool that can help healthcare administrators optimize resource utilization and reduce patient wait times in the system.

Appendix A. Mathematical Programming System (MPS)

The MPS format is the de facto file format for presenting linear programming and mixed integer programming models. It is column oriented so systems of equations typical of linear programming models must be translated into the following format:

		COLUMNS (Variables)						RHS
		X_1	X_2	X_3	X_4	...	X_n	
ROWS (Equations)	$Equation_1$	a_{11}	a_{12}	a_{13}	a_{14}		a_{1n}	S_1
	$Equation_2$	a_{21}	a_{22}	a_{23}	a_{24}		a_{2n}	S_2
	$Equation_3$	a_{31}	a_{32}	a_{33}	a_{34}		a_{3n}	S_3
	...							...
	$Equation_m$	a_{m1}	a_{m2}	a_{m3}	a_{m4}		a_{mn}	S_m

Figure A.1: System of equations in column oriented format

ROWS capture the objective function and constraints of an LP model. COLUMNS capture the decisions variables. a_{11} to a_{mn} are the coefficients of variables in each equation. RHS capture the right hand side values of the equations.

Translation of a linear programming (LP) model into the mps format is best illustrated through example. Recall the Colonel Motors (CM) production planning problem presented in section 2.2.1:

Maximize	$4000x + 1200y$
Parts production hours (C1):	$30x + 40y \leq 30{,}000$
Assembly hours (C2):	$8x + 11y \leq 10{,}000$
Small cars demand (C3):	$x \geq 300$
Non-negativity (C4):	$x, y \geq 0$

Translated into column oriented format, the CM problem looks like the following:

		COLUMNS		RHS
		X	Y	
ROWS	Objective	4000	1200	
	C1	30	40	30000
	C2	8	11	10000
	C3	1	0	300

Figure A.2: The CM production planning model in column oriented format

The MPS file format has 5 main sections: NAME, ROWS, COLUMNS, RHS, and BOUNDS. The NAME section is used to specify a name for the model or problem; "PRODUCTION" for example.

Table A-1: NAME section of MPS file format

```
NAME        PRODUCTION
```

The ROWS section defines names for the rows or equations of the model. The letter preceding the name is a flag that defines the nature of the equation. N indicates that the row is the objective function to be maximized or minimized. L indicates that the row is a less-than-or-equal-to relation. G specifies greater-than-or-equal-to. E specifies equality.

Table A-2: ROWS section of MPS file format

```
ROWS
 N  OBJECTIVE
 L  C1
 L  C2
 G  C3
```

The COLUMNS section defines names for columns and the values of their coefficients in applicable rows. The following example means the variable X appears in OBJECTIVE with a coefficient of 4,000, in C1 with coefficient of 30, in C2 with coefficient of 8 and in C3 with coefficient of 1. The variable Y appears in OBJECTIVE with coefficient of 1,200, in C1 with coefficient of 40 and in C2 with coefficient of 11.

Table A-3: COLUMNS section of MPS file format

```
COLUMNS
    X      OBJECTIVE    4000
    X      C1           30
    X      C2           8
    X      C3           1
    Y      OBJECTIVE    1200
    Y      C1           40
    Y      C2           11
```

The RHS section defines the right hand side values of the rows or equations.

Table A-4: RHS section of MPS file format

```
RHS
    RHS1   C1    30000
    RHS1   C2    10000
    RHS1   C3    300
```

The BOUNDS section can be used to define range of allowable values for columns or variables. This section is usually used to specify integer or binary constraints on variables.

The following is an example MPS file for the simple procedures presented in section 3.1.1 and simple MPSP model presented in section 3.1.2.

Table A-5: MPS file for the example simple MPSP model problem

```
NAME           SIMPLE_MPSP
ROWS
 N  OBJECTIVE
 L  C1(1)
 L  C1(2)
 L  C1(3)
 L  C1(4)
 L  C1(5)
 G  C2(1)
 G  C2(2)
 G  C2(3)
 G  C2(4)
 G  C2(5)
 E  C3(1)
 E  C3(2)
 E  C3(3)
 E  C3(4)
```

```
 E  C3(5)
 G  C4(1,1,3)
 G  C4(1,1,4)
 G  C4(1,3,4)
 G  C4(2,2,5)
 G  C4(4,2,5)
 G  C4(5,1,2)
 G  C4(5,1,4)
 G  C4(5,1,5)
 G  C4(5,2,4)
 G  C4(5,2,5)
 G  C4(5,4,5)
 G  C4(6,1,2)
 G  C4(6,1,3)
 G  C4(6,1,4)
 G  C4(6,2,3)
 G  C4(6,2,4)
 G  C4(6,3,4)
 G  C4(7,2,5)
 G  C5(1,1,3)
 G  C5(1,1,4)
 G  C5(1,3,4)
 G  C5(2,2,5)
 G  C5(4,2,5)
 G  C5(5,1,2)
 G  C5(5,1,4)
 G  C5(5,1,5)
 G  C5(5,2,4)
 G  C5(5,2,5)
 G  C5(5,4,5)
 G  C5(6,1,2)
 G  C5(6,1,3)
 G  C5(6,1,4)
 G  C5(6,2,3)
 G  C5(6,2,4)
 G  C5(6,3,4)
 G  C5(7,2,5)
COLUMNS
    PS(1)     C1(1)     1
    PS(1)     C2(1)     1
    PS(1)     C3(1)     1
    PS(1)     C4(1,1,3) -1
    PS(1)     C5(1,1,3) 1
    PS(1)     C4(6,1,2) -1
    PS(1)     C5(6,1,2) 1
    PS(1)     C4(6,1,3) -1
    PS(1)     C5(6,1,3) 1
    PS(1)     C4(6,1,4) -1
    PS(1)     C5(6,1,4) 1
    PS(2)     C1(2)     1
    PS(2)     C2(2)     1
```

```
    PS(2)      C3(3)      1
    PS(2)      C4(2,2,5) -1
    PS(2)      C5(2,2,5) 1
    PS(2)      C4(4,2,5) -1
    PS(2)      C5(4,2,5) 1
    PS(2)      C4(5,2,4) -1
    PS(2)      C5(5,2,4) 1
    PS(2)      C4(5,2,5) -1
    PS(2)      C5(5,2,5) 1
    PS(2)      C4(6,2,3) -1
    PS(2)      C5(6,2,3) 1
    PS(2)      C4(6,2,4) -1
    PS(2)      C5(6,2,4) 1
    PS(2)      C4(7,2,5) -1
    PS(2)      C5(7,2,5) 1
    PS(2)      C4(5,1,2) 1
    PS(2)      C5(5,1,2) -1
    PS(2)      C4(6,1,2) 1
    PS(2)      C5(6,1,2) -1
    PS(3)      C1(3)      1
    PS(3)      C2(3)      1
    PS(3)      C3(3)      1
    PS(3)      C4(1,3,4) -1
    PS(3)      C5(1,3,4) 1
    PS(3)      C4(6,3,4) -1
    PS(3)      C5(6,3,4) 1
    PS(3)      C4(1,1,3) 1
    PS(3)      C5(1,1,3) -1
    PS(3)      C4(6,1,3) 1
    PS(3)      C5(6,1,3) -1
    PS(3)      C4(6,2,3) 1
    PS(3)      C5(6,2,3) -1
    PS(4)      C1(4)      1
    PS(4)      C2(4)      1
    PS(4)      C3(4)      1
    PS(4)      C4(5,4,5) -1
    PS(4)      C5(5,4,5) 1
    PS(4)      C4(1,1,4) 1
    PS(4)      C5(1,1,4) -1
    PS(4)      C4(1,3,4) 1
    PS(4)      C5(1,3,4) -1
    PS(4)      C4(5,1,4) 1
    PS(4)      C5(5,1,4) -1
    PS(4)      C4(5,2,4) 1
    PS(4)      C5(5,2,4) -1
    PS(4)      C4(6,1,4) 1
    PS(4)      C5(6,1,4) -1
    PS(4)      C4(6,2,4) 1
    PS(4)      C5(6,2,4) -1
    PS(4)      C4(6,3,4) 1
    PS(4)      C5(6,3,4) -1
```

```
    PS(5)      C1(5)      1
    PS(5)      C2(5)      1
    PS(5)      C3(5)      1
    PS(5)      C4(2,2,5)  1
    PS(5)      C5(2,2,5)  -1
    PS(5)      C4(4,2,5)  1
    PS(5)      C5(4,2,5)  -1
    PS(5)      C4(5,1,5)  1
    PS(5)      C5(5,1,5)  -1
    PS(5)      C4(5,2,5)  1
    PS(5)      C5(5,2,5)  -1
    PS(5)      C4(5,4,5)  1
    PS(5)      C5(5,4,5)  -1
    PS(5)      C4(7,2,5)  1
    PS(5)      C5(7,2,5)  -1
    L(1)       C3(1)      -1
    L(2)       C3(2)      -1
    L(3)       C3(3)      -1
    L(4)       C3(4)      -1
    L(5)       C3(5)      -1
    X(1)       C1(1)      45
    X(1)       C2(1)      30
    X(2)       C1(2)      45
    X(2)       C2(2)      30
    X(3)       C1(3)      45
    X(3)       C2(3)      30
    X(4)       C1(4)      45
    X(4)       C2(4)      30
    X(5)       C1(5)      45
    X(5)       C2(5)      30
    Y(1,3)     C4(1,1,3)  -45
    Y(1,3)     C5(1,1,3)  45
    Y(1,3)     C4(6,1,3)  -45
    Y(1,3)     C5(6,1,3)  45
    Y(1,4)     C4(1,1,4)  -45
    Y(1,4)     C5(1,1,4)  45
    Y(1,4)     C4(5,1,4)  -45
    Y(1,4)     C5(5,1,4)  45
    Y(1,4)     C4(6,1,4)  -45
    Y(1,4)     C5(6,1,4)  45
    Y(1,5)     C4(5,1,5)  -45
    Y(1,5)     C5(5,1,5)  45
    Y(2,3)     C4(6,2,3)  -45
    Y(2,3)     C5(6,2,3)  45
    Y(2,4)     C4(6,2,4)  -45
    Y(2,4)     C5(6,2,4)  45
    Y(2,5)     C4(2,2,5)  -45
    Y(2,5)     C5(2,2,5)  45
    Y(2,5)     C4(4,2,5)  -45
    Y(2,5)     C5(4,2,5)  45
    Y(2,5)     C4(5,2,5)  -45
```

```
    Y(2,5)    C5(5,2,5) 45
    Y(2,5)    C4(7,2,5) -45
    Y(2,5)    C5(7,2,5) 45
    Y(3,4)    C4(1,3,4) -45
    Y(3,4)    C5(1,3,4) 45
    Y(3,4)    C4(6,3,4) -45
    Y(3,4)    C5(6,3,4) 45
    Y(4,5)    C4(5,4,5) -45
    Y(4,5)    C5(5,4,5) 45
RHS
    RHS1      C1(1)     65
    RHS1      C1(2)     70
    RHS1      C1(3)     65
    RHS1      C1(4)     60
    RHS1      C1(5)     70
    RHS1      C2(1)     30
    RHS1      C2(2)     30
    RHS1      C2(3)     30
    RHS1      C2(4)     30
    RHS1      C2(5)     30
    RHS1      C3(1)     0
    RHS1      C3(2)     0
    RHS1      C3(3)     0
    RHS1      C3(4)     0
    RHS1      C3(5)     0
    RHS1      C4(1,1,3) -35
    RHS1      C4(1,1,4) -35
    RHS1      C4(1,3,4) -35
    RHS1      C4(2,2,5) -40
    RHS1      C4(4,2,5) -40
    RHS1      C4(5,1,2) -35
    RHS1      C4(5,1,4) -35
    RHS1      C4(5,1,5) -35
    RHS1      C4(5,2,4) -40
    RHS1      C4(5,2,5) -40
    RHS1      C4(5,4,5) -30
    RHS1      C4(6,1,2) -35
    RHS1      C4(6,1,3) -35
    RHS1      C4(6,1,4) -35
    RHS1      C4(6,2,3) -40
    RHS1      C4(6,2,4) -40
    RHS1      C4(6,3,4) -35
    RHS1      C4(7,2,5) -40
    RHS1      C5(1,1,3) 10
    RHS1      C5(1,1,4) 15
    RHS1      C5(1,3,4) 15
    RHS1      C5(2,2,5) 5
    RHS1      C5(4,2,5) 5
    RHS1      C5(5,1,2) 5
    RHS1      C5(5,1,4) 15
    RHS1      C5(5,1,5) 5
```

```
    RHS1      C5(5,2,4) 15
    RHS1      C5(5,2,5) 5
    RHS1      C5(5,4,5) 5
    RHS1      C5(6,1,2) 5
    RHS1      C5(6,1,3) 10
    RHS1      C5(6,1,4) 15
    RHS1      C5(6,2,3) 10
    RHS1      C5(6,2,4) 15
    RHS1      C5(6,3,4) 15
    RHS1      C5(7,2,5) 5
BOUNDS
 BV BND1      X(1)
 BV BND1      X(2)
 BV BND1      X(3)
 BV BND1      X(4)
 BV BND1      X(5)
 BV BND1      Y(1,3)
 BV BND1      Y(1,4)
 BV BND1      Y(1,5)
 BV BND1      Y(2,3)
 BV BND1      Y(2,4)
 BV BND1      Y(2,5)
 BV BND1      Y(3,4)
 BV BND1      Y(4,5)
ENDATA
```

Appendix B. Gnu Linear Programming Kit (GLPK)

GLPK is a powerful, highly customizable, open-source large scale linear programming solver package. It is a callable library of routines written in the ANSI C programming language. [120] GLPK can solve linear programming problems in a variety of mathematical programming languages including the MPS file format. GLPK comes packaged with a stand-alone executable *glpsol.exe* that combines the routines together into a standard, easy to use solver.

For example, the simple MPSP model example (*simple_MPSP.mps*) presented in Appendix A is solved using the following console command.

Table B-1: Calling glpsol.exe to solve model defined in simple_MPSP.mps

```
glpsol --freemps simple_MPSP.mps -o solution.txt
```

The above commands calls the *glpsol.exe* executable to read and solve the model defined in the file *simple_MPSP.mps* and output the solution into the file *solution.txt*. The solver progress is output onto the screen. The user also has the option of logging the progress in a text file by adding the command "--log *progress.txt*" to the *glpsol.exe* call. The progress of solving the example problem defined in *simple_MPSP.mps* is shown in Table B-2. The solution to *simple_MPSP.mps* is presented in Table B-3.

Table B-2: Solver progress for the problem defined in simple_MPSP.mps

```
C:\temp\GLPK>glpsol --freemps simple_MPSP.mps -o solution.txt
glp_read_mps: reading problem data from `alpha.mps'...
glp_read_mps: problem GENERAL
glp_read_mps: 106 rows, 50 columns, 300 non-zeros
glp_read_mps: 23 integer columns, all of which are binary
glp_read_mps: 540 records were read
ipp_basic_tech:  1 row(s) and 0 column(s) removed
ipp_reduce_bnds: 2 pass(es) made, 27 bound(s) reduced
ipp_basic_tech:  0 row(s) and 0 column(s) removed
ipp_reduce_coef: 1 pass(es) made, 0 coefficient(s) reduced
glp_intopt: presolved MIP has 105 rows, 50 columns, 290 non-zeros
glp_intopt: 23 integer columns, all of which are binary
Scaling...
 A: min|aij| = 1.000e+000  max|aij| = 6.000e+001  ratio = 6.000e+001
GM: min|aij| = 5.027e-001  max|aij| = 1.989e+000  ratio = 3.957e+000
```

```
EQ: min|aij| = 2.527e-001  max|aij| = 1.000e+000  ratio = 3.957e+000
2N: min|aij| = 2.500e-001  max|aij| = 1.000e+000  ratio = 4.000e+000
Crashing...
Size of triangular part = 105
Solving LP relaxation...
      0: obj =  0.000000000e+000  infeas = 1.000e+002 (0)
*    40: obj =  1.666666667e-001  infeas = 0.000e+000 (0)
*    41: obj =  0.000000000e+000  infeas = 0.000e+000 (0)
OPTIMAL SOLUTION FOUND
Integer optimization begins...
+    41: mip =     not found yet >=              -inf        (1; 0)
+    81: >>>>>  5.100000000e+001 >=  0.000000000e+000 100.0% (10; 0)
+   186: mip =  5.100000000e+001 >=     tree is empty   0.0% (0; 57)
INTEGER OPTIMAL SOLUTION FOUND
Time used:   0.1 secs
Memory used: 0.2 Mb (225598 bytes)
lpx_print_mip: writing MIP problem solution to `solution.txt'...

C:\temp\GLPK>
```

Table B-3: Solution to simple_MPSP.mps

```
Problem:    GENERAL
Rows:       106
Columns:    50 (23 integer, 23 binary)
Non-zeros:  300
Status:     INTEGER OPTIMAL
Objective:  OBJECTIVE = 51 (MINimum)

   No.   Row name        Activity     Lower bound   Upper bound
------ ------------    ------------- ------------- -------------
     1 OBJECTIVE                  51
     2 C1(1)                      75                          80
     3 C1(2)                      70                          85
     4 C1(3)                      60                          80
     5 C1(4)                      30                          75
     6 C1(5)                      60                          85
     7 C2(1)                      45            30
     8 C2(2)                      40            30
     9 C2(3)                      30            30
    10 C2(4)                      30            30
    11 C2(5)                      30            30
    12 C3(1,1)                     0             0             =
    13 C3(1,3)                     0             0             =
    14 C3(1,4)                     0             0             =
    15 C3(2,2)                     0             0             =
    16 C3(2,5)                     0             0             =
    17 C3(4,2)                     0             0             =
    18 C3(4,5)                     0             0             =
```

19	C3(5,1)	0	0	=
20	C3(5,2)	0	0	=
21	C3(5,4)	0	0	=
22	C3(5,5)	0	0	=
23	C3(6,1)	0	0	=
24	C3(6,2)	0	0	=
25	C3(6,3)	0	0	=
26	C3(6,4)	0	0	=
27	C3(7,2)	0	0	=
28	C3(7,5)	0	0	=
29	C4(1)	0	0	=
30	C4(2)	0	0	=
31	C4(3)	0	0	=
32	C4(4)	0	0	=
33	C4(5)	0	0	=
34	C5(1,1,3)	-15	-50	
35	C5(1,1,4)	-45	-50	
36	C5(1,3,1)	-45	-50	
37	C5(1,3,4)	-30	-50	
38	C5(1,4,1)	-15	-45	
39	C5(1,4,3)	-30	-45	
40	C5(2,2,5)	-10	-55	
41	C5(2,5,2)	-50	-55	
42	C5(4,2,5)	-10	-55	
43	C5(4,5,2)	-50	-55	
44	C5(5,1,2)	-5	-50	
45	C5(5,1,4)	-45	-50	
46	C5(5,1,5)	-15	-50	
47	C5(5,2,1)	-55	-55	
48	C5(5,2,4)	-40	-55	
49	C5(5,2,5)	-10	-55	
50	C5(5,4,1)	-15	-45	
51	C5(5,4,2)	-20	-45	
52	C5(5,4,5)	-30	-45	
53	C5(5,5,1)	-45	-55	
54	C5(5,5,2)	-50	-55	
55	C5(5,5,4)	-30	-55	
56	C5(6,1,2)	-5	-50	
57	C5(6,1,3)	-15	-50	
58	C5(6,1,4)	-45	-50	
59	C5(6,2,1)	-55	-55	
60	C5(6,2,3)	-10	-55	
61	C5(6,2,4)	-40	-55	
62	C5(6,3,1)	-45	-50	
63	C5(6,3,2)	-50	-50	
64	C5(6,3,4)	-30	-50	
65	C5(6,4,1)	-15	-45	
66	C5(6,4,2)	-20	-45	
67	C5(6,4,3)	-30	-45	
68	C5(7,2,5)	-10	-55	
69	C5(7,5,2)	-50	-55	

70	C6(1,1,3)	15	10	
71	C6(1,1,4)	45	15	
72	C6(1,3,1)	45	10	
73	C6(1,3,4)	30	15	
74	C6(1,4,1)	15	10	
75	C6(1,4,3)	30	10	
76	C6(2,2,5)	10	5	
77	C6(2,5,2)	50	5	
78	C6(4,2,5)	10	5	
79	C6(4,5,2)	50	5	
80	C6(5,1,2)	5	5	
81	C6(5,1,4)	45	15	
82	C6(5,1,5)	15	5	
83	C6(5,2,1)	55	10	
84	C6(5,2,4)	40	15	
85	C6(5,2,5)	10	5	
86	C6(5,4,1)	15	10	
87	C6(5,4,2)	20	5	
88	C6(5,4,5)	30	5	
89	C6(5,5,1)	45	10	
90	C6(5,5,2)	50	5	
91	C6(5,5,4)	30	15	
92	C6(6,1,2)	5	5	
93	C6(6,1,3)	15	10	
94	C6(6,1,4)	45	15	
95	C6(6,2,1)	55	10	
96	C6(6,2,3)	10	10	
97	C6(6,2,4)	40	15	
98	C6(6,3,1)	45	10	
99	C6(6,3,2)	50	5	
100	C6(6,3,4)	30	15	
101	C6(6,4,1)	15	10	
102	C6(6,4,2)	20	5	
103	C6(6,4,3)	30	10	
104	C6(7,2,5)	10	5	
105	C6(7,5,2)	50	5	
106	C9	51	0	

No.	Column name	Activity	Lower bound	Upper bound
1	PS(1)	15	0	
2	PS(2)	10	0	
3	PS(3)	0	0	
4	PS(4)	30	0	
5	PS(5)	0	0	
6	RS(1,1)	15	0	
7	RS(1,3)	0	0	
8	RS(1,4)	30	0	
9	RS(2,2)	10	0	
10	RS(2,5)	0	0	
11	RS(4,2)	10	0	

```
    12 RS(4,5)                    0              0
    13 RS(5,1)                   15              0
    14 RS(5,2)                   10              0
    15 RS(5,4)                   30              0
    16 RS(5,5)                    0              0
    17 RS(6,1)                   15              0
    18 RS(6,2)                   10              0
    19 RS(6,3)                    0              0
    20 RS(6,4)                   30              0
    21 RS(7,2)                   10              0
    22 RS(7,5)                    0              0
    23 L(1)                      15              0
    24 L(2)                      10              0
    25 L(3)                       0              0
    26 L(4)                      30              0
    27 L(5)                       0              0
    28 X(1)          *            1              0              1
    29 X(2)          *            1              0              1
    30 X(3)          *            1              0              1
    31 X(4)          *            0              0              1
    32 X(5)          *            1              0              1
    33 Y(1,2)        *            0              0              1
    34 Y(1,3)        *            0              0              1
    35 Y(1,4)        *            1              0              1
    36 Y(1,5)        *            0              0              1
    37 Y(2,1)        *            1              0              1
    38 Y(2,3)        *            0              0              1
    39 Y(2,4)        *            1              0              1
    40 Y(2,5)        *            0              0              1
    41 Y(3,1)        *            1              0              1
    42 Y(3,2)        *            1              0              1
    43 Y(3,4)        *            1              0              1
    44 Y(4,1)        *            0              0              1
    45 Y(4,2)        *            0              0              1
    46 Y(4,3)        *            0              0              1
    47 Y(4,5)        *            0              0              1
    48 Y(5,1)        *            1              0              1
    49 Y(5,2)        *            1              0              1
    50 Y(5,4)        *            1              0              1

Integer feasibility conditions:

INT.PE: max.abs.err. = 0.00e+000 on row 0
        max.rel.err. = 0.00e+000 on row 0
        High quality

INT.PB: max.abs.err. = 0.00e+000 on row 0
        max.rel.err. = 0.00e+000 on row 0
        High quality

End of output
```

The solution file contains a lot of information about the problem solution. However, we only need values of the variable (or column) ps_i to visualize the solution. The values of ps_i and the resulting schedule are shown in Figure B.1.

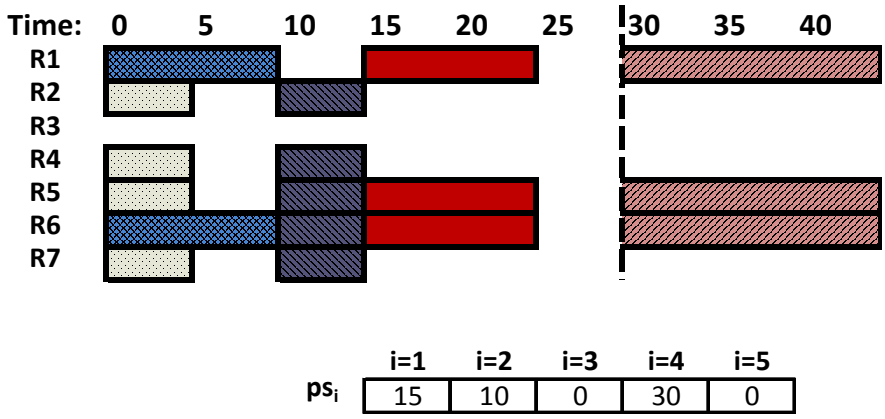


Figure B.1: Visualizing the solution to alpha.mps

All MIP models discussed in this thesis are solved using GLPK's *glpsol.exe* stand-alone solver except those models that are helped along with good initial solutions and tight solution bounds. The stand-alone *glpsol.exe* cannot be warm started with an initial solution. However, the underlying solver routine can accommodate initial solutions. A custom program was developed to read in initial solutions and to pass them into the GLPK solver routine. Aside from warm starting with an initial solution, the custom program behaves exactly like *glpsol.exe*.

Appendix C. The simple MPSP model

Decision Variables

$ps_i = Starting\ time\ of\ procedure\ i$

$x_i = \begin{cases} 1 & if\ procedure\ i \text{ can fit within the scheduling period} \\ 0 & otherwise \end{cases}$

$l_i = Lateness\ of\ procedure\ i$

$y_{i,i2} = \begin{cases} 1 & if\ procedure\ i\ is\ scheduled\ before\ procedure\ i2 \\ 0 & otherwise \end{cases}$

$$\forall i, i2 = 1, 2, 3, \dots n$$

Data

$R_{ri} = \begin{cases} 1 & if\ resource\ r\ is\ required\ for\ procedure\ i \\ 0 & otherwise \end{cases}$

$d_i = Duration\ of\ procedure\ i$

$p = Scheduling\ period$

$dt_i = Due\ time\ of\ procedure\ i$

$b_i = Benefit\ of\ scheduling\ procedure\ i$

$w_i = Late\ penalty\ of\ procedure\ i$

Objective Function

Min $\sum_i (w_i l_i - b_i x_i)$

Constraints

C1 $ps_i + d_i - (1 - x_i)M \leq p$

C2 $ps_i + px_i \geq p$

C3 $ps_i - l_i = dt_i$

C4 $ps_{i2} - ps_i + (1 - y_{i,i2})M \geq d_i \quad \forall i, i2: (R_{ri} = R_{ri2} = 1 \cap i \neq i2)$

C5 $ps_i - ps_{i2} + My_{i,i2} \geq d_{i2} \quad \forall i, i2: (R_{ri} = R_{ri2} = 1 \cap i \neq i2)$

C6 $ps_i, l_i, x_i, y_{i,i2} \geq 0$

C7 $x_i, y_{i,i2} \in \{0,1\}$

Appendix D. The enhanced MPSP model

Decision Variables

$ps_i = Starting\ time\ of\ activity\ i$

$x_i = \begin{cases} 1 & if\ activity\ i \text{ can fit within the scheduling period} \\ 0 & otherwise \end{cases}$

$l_i = Lateness\ of\ activity\ i$

$y_{i,i2} = \begin{cases} 1 & if\ activity\ i\ is\ scheduled\ before\ activity\ i2 \\ 0 & otherwise \end{cases}$

$$\forall i, i2 = 1, 2, 3, \dots n$$

Data

$R_{ri} = \begin{cases} 1 & if\ resource\ r\ is\ required\ for\ activity\ i \\ 0 & otherwise \end{cases}$

$d_i = Duration\ of\ activity\ i$

$p = Scheduling\ period$

$dt_i = Due\ time\ of\ activity\ i$

$b_i = Benefit\ of\ scheduling\ activity\ i$

$w_i = Late\ penalty\ of\ activity\ i$

$z_{ij} = \begin{cases} 1 & if\ activity\ i\ must\ immediately\ follow \text{ activity } j \\ 2 & if\ activity\ i\ and\ j\ both\ belong\ to\ the\ same\ procedure \\ 0 & otherwise \end{cases}$

Objective Function

Min $\sum_i (w_i l_i - b_i x_i)$

Constraints

C1 $ps_i + d_i - (1 - x_i)M \leq p$

C2 $ps_i + px_i \geq p$

C3 $ps_i - l_i = dt_i$

C4 $ps_{i2} - ps_i + (1 - y_{i,i2})M \geq d_i$
$\forall i, i2: (R_{ri} = R_{ri2} = 1 \ \cap \ z_{i2,i} = 0 \ \cap \ i \neq i2)$

C5 $ps_i - ps_{i2} + My_{i,i2} \geq d_{i2}$
$\forall i, i2: (R_{ri} = R_{ri2} = 1 \ \cap \ z_{i2,i} = 0 \ \cap \ i \neq i2)$

C6 $ps_i, l_i, x_i, y_{i,i2} \geq 0$

C7 $x_i, y_{i,i2} \in \{0,1\}$

C8 $ps_i - ps_j = d_j \qquad \forall i,j: (z_{ij} = 1)$

C9 $x_i - x_j = 0 \qquad \forall i,j: (z_{ij} = 1)$

Appendix E. The final MPSP model

Decision Variables

$ps_i = Starting\ time\ of\ activity\ i$

$x_i = \begin{cases} 1 & if\ activity\ i \text{ can fit within the scheduling period} \\ 0 & otherwise \end{cases}$

$l_i = Variable\ duration\ of\ gap\ activity\ i \qquad \forall i: (g_i \neq 0)$

$y_{i,i2} = \begin{cases} 1 & if\ activity\ i\ is\ scheduled\ before\ activity\ i2 \\ 0 & otherwise \end{cases}$

$$\forall i, i2 = 1, 2, 3, \dots n$$

Data

$R_{ri} = \begin{cases} 1 & if\ resource\ r\ is\ required\ for\ activity\ i \\ 0 & otherwise \end{cases}$

$d_i = Duration\ of\ activity\ i$

$p = Scheduling\ period$

$dt_i = Due\ time\ of\ activity\ i$

$b_i = Benefit\ of\ scheduling\ activity\ i$

$w_i = Late\ penalty\ of\ activity\ i$

$z_{ij} = \begin{cases} 1 & if\ activity\ i\ must\ immediately\ follow\ activity\ j \\ 2 & if\ activity\ i\ and\ j\ both\ belong\ to\ the\ same\ procedure \\ 3 & if\ activity\ j\ is\ a\ pre-requisite\ for\ activity\ i \\ 0 & otherwise \end{cases}$

$g_i = \begin{cases} 1 & if\ activity\ i\ is\ a\ starting\ gap\ activity \\ 2 & if\ activity\ i\ is\ a\ gap\ activity\ but\ not\ starting\ gap\ activity \\ 0 & otherwise \end{cases}$

$ML_i = Mid\ level\ index\ of\ activity\ i$

$TL_i = Top\ level\ index\ of\ activity\ i$

Objective Function

Min $\quad \sum_i (w_i l_i - b_i x_i)$

Constraints

C1 $ps_i + d_i - (1 - x_i)M \leq p \qquad \forall i: (g_i \neq 1)$

C2 $ps_i + px_i \geq p \qquad \forall i: (g_i \neq 1)$

C3 $ps_i = dt_i \qquad \forall i: (g_i = 1)$

C4a $ps_{i2} - ps_i + (1 - y_{i,i2})M \geq d_i$
$\forall i, i2: (R_{ri} = R_{ri2} = 1 \ \cap \ z_{i2,i} = 0 \ \cap \ g_i = 0 \ \cap \ i \neq i2)$

C4b $ps_{i2} - ps_i + (1 - y_{i,i2})M \geq l_i$
$\forall i, i2: (R_{ri} = R_{ri2} = 1 \cap z_{i2,i} = 0 \cap g_i \neq 0 \cap i \neq i2)$

C5a $ps_i - ps_{i2} + My_{i,i2} \geq d_{i2}$
$\forall i, i2: (R_{ri} = R_{ri2} = 1 \ \cap \ z_{i2,i} = 0 \ \cap \ g_{i2} = 0 \ \cap \ i \neq i2)$

C5b $ps_i - ps_{i2} + My_{i,i2} \geq l_{i2}$
$\forall i, i2: (R_{ri} = R_{ri2} = 1 \ \cap \ z_{i2,i} = 0 \ \cap \ g_{i2} \neq 0 \ \cap \ i \neq i2)$

C6 $ps_i, l_i, x_i, y_{i,i2} \geq 0$

C7 $x_i, y_{i,i2} \in \{0,1\}$

C8a $ps_i - ps_j = d_j \qquad \forall i, j: (z_{ij} = 1 \ \cap \ g_j = 0)$

C8b $ps_i - ps_j = l_j \qquad \forall i, j: (z_{ij} = 1 \ \cap \ g_j \neq 0)$

C9 $x_i - x_j = 0 \qquad \forall i, j: (TP_i = TP_j \ \cap i \neq j)$

C10 $y_{i,i2} = 0 \qquad \forall i, j: (z_{ij} = 3)$

Appendix F. Applying the MPSP model to PET-CT procedure

The positron emission tomography – computed tomographic (PET-CT) scan is a complex medical imaging procedure. This section will demonstrate the versatility of the MPSP model by modeling one shift at a small PET-CT clinic.

This example clinic has at its disposal 1 scanner, 2 nurses/technicians and 4 private uptake rooms. A patient's typical appointment workflow is outlined in Table F-1. [121]

Table F-1: PET-CT appointment description

Activity	Description	Duration (min)
A	Extensive interview with nurse/technician. Review medical history, learn more about the scanning procedure, resolve any misunderstandings about the procedure, and review any other important information. Place catheter for serum glucose assay and begin FDG infusion.	40
B	FDG uptake phase and oral contrast ingestion. Very little to no interaction with staff.	60 - 120
C	Escorted to scan room and positioned.	15
D	PET-CT scan.	35
E	Return to uptake room or waiting area (not necessarily the same room in the uptake phase) for post scan assessment, interview and catheter removal. Patient departs.	25

The duration values given in Table F-1 are average expected values but in reality they are variable depending on individual patient needs. The MPSP model is currently a deterministic model and can only model known activity durations so the average expected durations will be used. The variability in the FDG uptake phase (activity B) however, is significant and must be modeled. FDG, short for fluorodeoxyglucose, is a radioactive tracer isotope that is injected into the patient and absorbed by tissue. That absorption or uptake takes time. The minimum uptake time is 60 minutes; however, longer uptake time up to 120 minutes can improve picture contrast. Some clinics prefer longer uptake phase. [121] This example will model the uptake phase as an activity with flexible duration so that the uptake phase may be extended as desired. Flexibility in activities should also ease scheduling of shared resources.

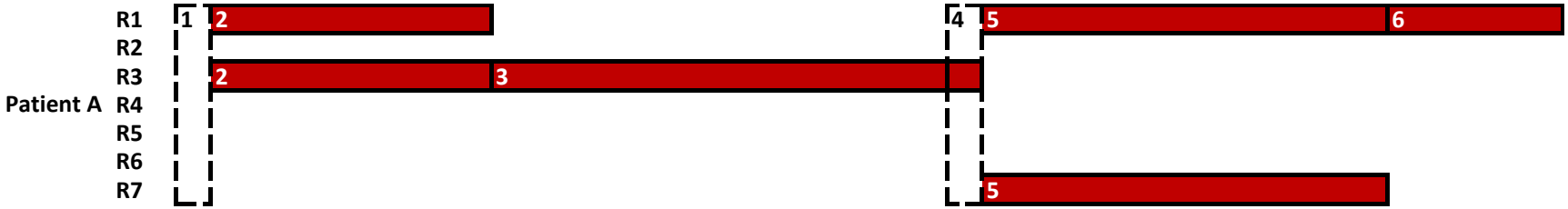
The resources of the clinic are modeled as shown in Table F-2. Figure F.1 through Figure F.4 show the modeling of PET-CT appointment workflow for patients A through D. Descriptions of the modeled activities are found in Table F-3.

Table F-2: Modeling clinic resources

	Resource
R1	Nurse/technician 1
R2	Nurse/technician 2
R3	Uptake room 1
R4	Uptake room 2
R5	Uptake room 3
R6	Uptake room 4
R7	CT scanner

Table F-3: MPSP model of PET-CT workflow description

Patient	Activity	Description	Duration (min)
A	1	Starting gap activity representing the patient waiting to be called.	Variable
	2	Pre-scan interview with nurse/technician 1. Occupies uptake room 1.	40
	3	Uptake phase. Minimum uptake time requirement. Occupies uptake room 1.	60
	4	Gap activity. Uptake phase continued. Additional uptake time as desired. Occupies uptake room 1.	Variable
	5	Patient escorted to scan room and positioned on the scanner. Scan process. Occupies CT scanner.	50
	6	Post scan assessment, interview, and catheter removal. Patient departs.	25
B	7	Starting gap activity representing the patient waiting to be called.	Variable
	8	Pre-scan interview with nurse/technician 2. Occupies uptake room 2.	40
	9	Uptake phase. Minimum uptake time requirement. Occupies uptake room 2.	60
	10	Gap activity. Uptake phase continued. Additional uptake time as desired. Occupies uptake room 2.	Variable
	11	Patient escorted to scan room and positioned on the scanner. Scan process. Occupies CT scanner.	50
	12	Post scan assessment, interview, and catheter removal. Patient departs.	25
C	13	Starting gap activity representing the patient waiting to be called.	Variable
	14	Pre-scan interview with nurse/technician 1. Occupies uptake room 3.	40
	15	Uptake phase. Minimum uptake time requirement. Occupies uptake room 3.	60
	16	Gap activity. Uptake phase continued. Additional uptake time as desired. Occupies uptake room 3.	Variable
	17	Patient escorted to scan room and positioned on the scanner. Scan process. Occupies CT scanner.	50
	18	Post scan assessment, interview, and catheter removal. Patient departs.	25
D	19	Starting gap activity representing the patient waiting to be called.	Variable
	20	Pre-scan interview with nurse/technician 2. Occupies uptake room 4.	40
	21	Uptake phase. Minimum uptake time requirement. Occupies uptake room 4.	60
	22	Gap activity. Uptake phase continued. Additional uptake time as desired. Occupies uptake room 4.	Variable
	23	Patient escorted to scan room and positioned on the scanner. Scan process. Occupies CT scanner.	50
	24	Post scan assessment, interview, and catheter removal. Patient departs.	25



R_{ri}

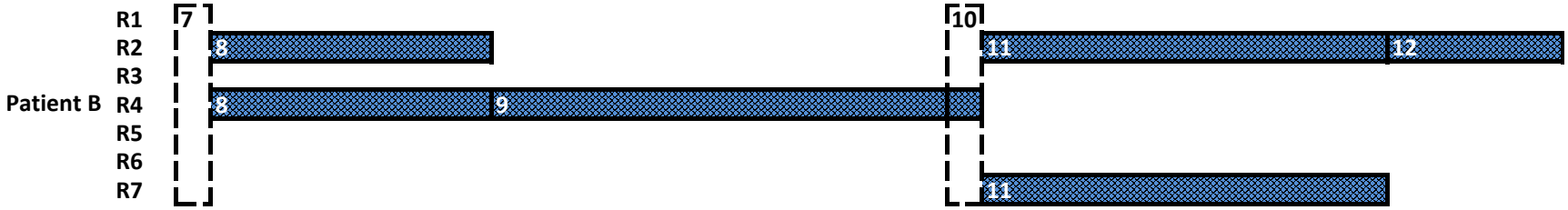
	i= 1	2	3	4	5	6
R1	0	1	0	0	1	1
R2	0	0	0	0	0	0
R3	0	1	1	1	0	0
R4	0	0	0	0	0	0
R5	0	0	0	0	0	0
R6	0	0	0	0	0	0
R7	0	0	0	0	1	0

i=	1	2	3	4	5	6
d_i	0	40	60	0	50	25
dt_i	0	0	0	0	0	0
b_i	0	1	0	0	0	0
w_i	1	0	0	0	0	0
g_i	1	0	0	2	0	0
ML_i	1	1	1	2	2	2
TL_i	1	1	1	1	1	1

Z_{ij}

j=	1	2	3	4	5	6
i=1	2	2	2	0	0	0
i=2	1	2	2	0	0	0
i=3	2	1	2	0	0	0
i=4	0	0	3	2	2	2
i=5	0	0	0	1	2	2
i=6	0	0	0	2	1	2

Figure F.1: PET-CT Workflow for patient A



R_{ri}

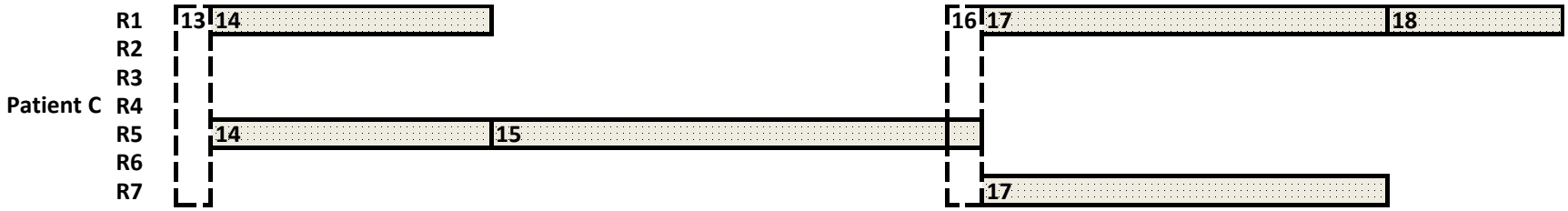
	i= 7	8	9	10	11	12
R1	0	0	0	0	0	0
R2	0	1	0	0	1	1
R3	0	0	0	0	0	0
R4	0	1	1	1	0	0
R5	0	0	0	0	0	0
R6	0	0	0	0	0	0
R7	0	0	0	0	1	0

i=	7	8	9	10	11	12
d_i	0	40	60	0	50	25
dt_i	0	0	0	0	0	0
b_i	0	1	0	0	0	0
w_i	1	0	0	0	0	0
g_i	1	0	0	2	0	0
ML_i	3	3	3	4	4	4
TL_i	2	2	2	2	2	2

Z_{ij}

j=	7	8	9	10	11	12
i=7	2	2	2	0	0	0
i=8	1	2	2	0	0	0
i=9	2	1	2	0	0	0
i=10	0	0	3	2	2	2
i=11	0	0	0	1	2	2
i=12	0	0	0	2	1	2

Figure F.2: PET-CT Workflow for patient B



R_{ri}

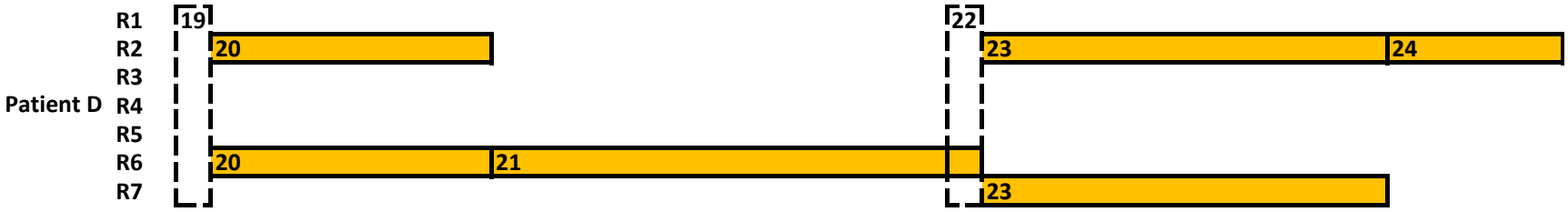
	i= 13	14	15	16	17	18
R1	0	1	0	0	1	1
R2	0	0	0	0	0	0
R3	0	0	0	0	0	0
R4	0	0	0	0	0	0
R5	0	1	1	1	0	0
R6	0	0	0	0	0	0
R7	0	0	0	0	1	0

i=	13	14	15	16	17	18
d_i	0	40	60	0	50	25
dt_i	0	0	0	0	0	0
b_i	0	1	0	0	0	0
w_i	1	0	0	0	0	0
g_i	1	0	0	2	0	0
ML_i	5	5	5	6	6	6
TL_i	3	3	3	3	3	3

Z_{ij}

	j= 13	14	15	16	17	18
i=13	2	2	2	0	0	0
i=14	1	2	2	0	0	0
i=15	2	1	2	0	0	0
i=16	0	0	3	2	2	2
i=17	0	0	0	1	2	2
i=18	0	0	0	2	1	2

Figure F.3: PET-CT Workflow for patient C



R_{ri}

	i= 19	20	21	22	23	24
R1	0	0	0	0	0	0
R2	0	1	0	0	1	1
R3	0	0	0	0	0	0
R4	0	0	0	0	0	0
R5	0	0	0	0	0	0
R6	0	1	1	1	0	0
R7	0	0	0	0	1	0

i=	19	20	21	22	23	24
d_i	0	40	60	0	50	25
dt_i	0	0	0	0	0	0
b_i	0	1	0	0	0	0
w_i	1	0	0	0	0	0
g_i	1	0	0	2	0	0
ML_i	7	7	7	8	8	8
TL_i	4	4	4	4	4	4

Z_{ij}

	j= 19	20	21	22	23	24
i=19	2	2	2	0	0	0
i=20	1	2	2	0	0	0
i=21	2	1	2	0	0	0
i=22	0	0	3	2	2	2
i=23	0	0	0	1	2	2
i=24	0	0	0	2	1	2

Figure F.4: PET-CT Workflow for patient D

Additional patients can be modeled by repeating the models for patients A through D.

The model was set up to guarantee satisfaction of the minimum uptake time of 60 minutes but allows flexible uptake time. That is, for patient A, activity 3 has a fixed 60-minute duration. The next activity, activity 4 has variable duration to allow a prolonged uptake phase. At this point, the user has several options to handle that flexible uptake phase. One may add hard constraints to the MIP formulation to enforce that the additional uptake time (i.e. duration of activity 4) does not exceed 60 minutes. Like so:

$$l_4 \leq 60$$

If the MIP formulation is modified, gMASH's fitness function and chromosome repair function would need to be modified as well to enforce the additional constraints.

Another option is to manipulate the flexible uptake time with coefficients in the objective function. Allowing longer uptake phase improves scan quality at the cost of patient throughput. In some circumstances, a clinic administrator may favour patient throughput over higher picture quality. In that case, the user may attach a wait time penalty to activity 4. The model will then try to minimize the duration of activity 4 like any other gap activity. This approach does not require modification to the MIP formulation or to the gMASH heuristic.

The MPSP model for the PET-CT procedure presented so far can easily be converted into the MIP model presented in Appendix E and solved exactly using GLPK's branch and cut solver or solved approximately using gMASH. The following figures show attempts of scheduling 6 PET-CT appointments into an 8-hour shift. Figure F.5 shows the exact solution obtained by the BnC solver. Figure F.6 shows gMASH's heuristic solution. Both solutions are conflict-free schedules that minimize patient wait time thus demonstrating the MPSP model's versatility in modeling medical procedures.

An important caveat must now be noted here. Medical procedures are complex, each have properties and nuances that are unique to that procedure. The MPSP model was designed to be general to model a variety of procedure but it should be used only as a starting point for building more sophisticated, realistic and practical systems. Extensive research and study is needed to understand the unique properties of each medical procedure. Only then can the MPSP model be adapted to accurately model and schedule those procedures.

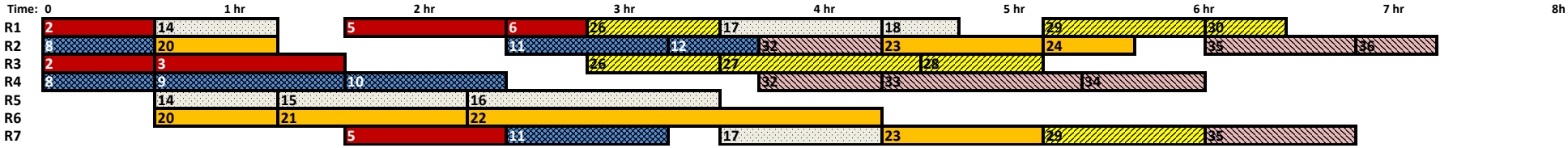


Figure F.5: Scheduling 6 PET-CT appointments, solving exactly using BnC

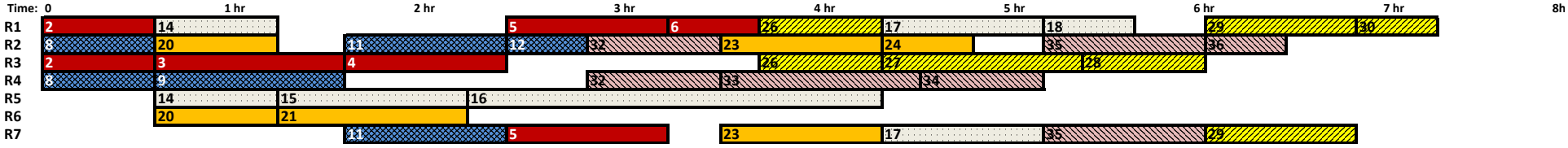


Figure F.6: Scheduling 6 PET-CT appointments, solving approximately using gMASH

References

[1] Marcel Saulinier, Sam Shortt, and Emily Gruenwoldt, "The Taming of the Queue: Toward a Cure for Health Care Wait Times," Canadian Medical Association, Canadian Nurse Association, 2004.

[2] J Hurst and L Siciliani, "Tackling Excessive Waiting Times for Elective Surgery: A Comparison of Policies in Twelve OECD Countries," *OECD Health Working Papers*, vol. 6, 2003.

[3] Chaoulli v. Quebec (Attorney General), *2005 SCC 35 [2005] 1 S.C.R. 791*.

[4] Ernie Stokes and Robin Romerville, "The Economic Cost of Wait Times in Canada," The Centre for Spatial Economics, Canadian Medical Association, 2008.

[5] Colleen M. Flood and Tom Archibald, "The illegality of Private Health Care in Canada," *Canadian Medical Association Journal*, vol. 164, no. 6, pp. 825-830, 2001.

[6] Wait Time Alliance Report 2007, "Time for Progress: New benchmarks for achieving meaningful reductions in wait times," Canadian Medical Association, 2007.

[7] Ontario Medical Association, "The Wait Time Strategy Review of Activities: Update #9 - October 30, 2007," Ontario Medical Association, 2007.

[8] Michael Rachlis, "Public Solutions to Health Care Wait Lists," Canadian Centre for Policy Alternatives, 2005.

[9] Singiresu S. Rao, *Engineering Optimization. Theory and Practice*, 3rd ed.: Wiley Eastern Limited, New Age International Publishers Ltd, 1996.

[10] S. M. Johnson, "Optimal Two and Three Stage Production Schedules with Setup Times Included," *The Rand Corporation*, 1953.

[11] Jatinder N.D. Gupta and Edward F. Stafford Jr, "Flowshop scheduling research after five decades," *European Journal of Operational Research*, vol. 169, no. 3, pp. 699-711, 2006.

[12] Jatinder N. D. Gupta, "A Review of Flowshop Scheduling Research," in *Disaggregation Problems in Manufacturing and Service Organizations*. The Hague: Martinus Nijhoff, 1979, pp. 363-388.

[13] Richard D. Smith and Richard A. Dudek, "A General Algorithm for Solution of the n-Job, M-Machine Sequencing Problem of the Flow Shop," *Operations Research*, vol. 15, no. 1, pp. 71-82, January 1967.

[14] Jatinder N. D. Gupta, "An Improved Combinatorial Algorithm for the Flowshop-Scheduling Problem," *Operations Research*, vol. 19, no. 7, pp. 1753-1758, November 1971.

[15] Wlodzimierz Szwarc, "Elimination Methods in the m× n Flow-Shop Scheduling Problem,"

Operations Research, vol. 21, no. 6, pp. 1250-1259, November 1973.

[16] Czeslaw Smutnicki, "Some results of the worst-case analysis for flow shop scheduling," *European Journal of Operational Research*, vol. 109, no. 1, pp. 66-87, August 1998.

[17] T. C. Edwin Cheng, Jatinder N. D. Gupta, and Guoqing Wang, "A review of flowshop scheduling research with setup times," *Production and Operations Management*, vol. 9, no. 3, pp. 262-283, September 2000.

[18] C. N. Potts and L. N. Van Wassenhove, "Integrating Scheduling with Batching and Lot-Sizing: A Review of Algorithms and Complexity," *The Journal of the Operational Research Society*, vol. 43, no. 5, pp. 395-406, May 1992.

[19] Chris N. Potts and Mikhail Y. Kovalyov, "Scheduling with batching: A review," *European Journal of Operational Research*, vol. 120, no. 2, pp. 228-249, January 2000.

[20] Alan S. Manne, "On the Job-Shop Shceduling Problem," *Operations Research*, vol. 8, no. 2, pp. 219-223, 1960.

[21] Z. A. Lomnicki, "A Branch-and-Bound Algorithm for the Exact Solution of the Three-Machine Scheduling Problem," *OR*, vol. 16, no. 1, pp. 89-100, March 1965.

[22] Edward Ignall and Linus Schrage, "Application of the Branch and Bound Technique to Some Flow-Shop Scheduling Problems," *Operations Research*, vol. 13, no. 3, pp. 400-412, May 1965.

[23] A. P. G. Brown and Z. A. Lomnicki, "Some Applications of the "Branch-and-Bound" Algorithm to the Machine Scheduling Problem," *OR*, vol. 17, no. 2, pp. 173-186, June 1966.

[24] G. B. McMahon and P. G. Burton, "Flow-Shop Scheduling with the Branch-And-Bound Method," *Operations Research*, vol. 15, no. 3, pp. 473-481, May 1966.

[25] Kenneth R. Baker, "A Comparative Study of Flow-Shop Algorithms," *Operations Research*, vol. 23, no. 1, pp. 62-73, January 1975.

[26] B. J. Lageweg, J. K. Lenstra, and A. H. G. Rinnooy Kan, "Bounding Scheme for the Permutation Flow-Shop Problem," *Operations Research*, vol. 26, no. 1, pp. 52-67, January 1978.

[27] Roger L. Sisson, "Methods of Sequencing in Job Shops - A Review," *Operations Research*, vol. 7, no. 1, pp. 10-29, 1959.

[28] M. R. Garey, D. S. Johnson, and Ravi Sethi, "The Complexity of Flowshop and Jobshop Scheduling," *Mathematics of Operations Research*, vol. 1, no. 2, pp. 117-129, May 1976.

[29] Sartaj Sahni and Yookun Cho, "Complexity of Scheduling Shops with No Wait in Process," *Mathematics of Operations Research*, vol. 4, no. 4, pp. 448-457, November 1979.

[30] Teofilo Gonzalez and Sartaj Sahni, "Flowshop and Jobshop Schedules: Complexity and

Approximation," *Operations Research*, vol. 26, no. 1, pp. 36-52, January 1978.

[31] J. M. Framinan, J. N. D. Gupta, and R. Leisten, "A Review and Classification of Heuristics for Permutation Flow-Shop Scheduling with Makespan Objective," *The Journal of the Operational Research Society*, vol. 55, no. 12, pp. 1243-1255, December 2004.

[32] Jitti Jungwattanakit, Manop Reodecha, Paveena Chaovalitwongse, and Frank Werner, "A comparison of scheduling algorithms for flexible flow shop problems with unrelated parallel machines, setup times and dual criteria," *Computers & Operations Research*, vol. 36, pp. 358-378, 2009.

[33] D. S. Palmer, "Sequencing Jobs Through a Multi-Stage Process in the Minimum Total Time -- A Quick Method of Obtaining a Near Optimum," *OR*, vol. 16, no. 1, pp. 101-107, March 1965.

[34] Muhammad Nawaz, E. Emory Enscore Jr., and Inyong Ham, "A Heuristic Algorithm for the m-Machine, n-Job Flow-shop Sequencing Problem," *OMEGA: The International Journal of Management Science*, vol. 11, no. 1, pp. 91-95, 1983.

[35] Marino Widmer and Alain Hertz, "A new heuristic method for the flow shop sequencing problem," *European Journal of Operational Research*, vol. 41, no. 2, pp. 186-193, August 1989.

[36] S. Sarin and M. Lefoka, "Scheduling Heuristic for the n-Job m-Machine Flow Shop," *OMEGA: International Journal of Management Science*, vol. 21, no. 2, pp. 229-234, March 1993.

[37] Joao Vitor Moccellin, "A New Heuristic Method for the Permutation Flow Shop Scheduling Problem," *The Journal of The Operational Research Society*, vol. 46, no. 7, pp. 883-886, July 1995.

[38] Helena Ramalhinho Lourenco, "Sevast'yanov's algorithm for the flow-shop scheduling problem," *European Journal of Operational Research*, vol. 91, no. 1, pp. 176-189, May 1996.

[39] Eugeniusz Nowicki and Czeslaw Smutnicki, "The flow shop with parallel machines: A tabu search approach," *European Journal of Operational Research*, vol. 106, no. 2, pp. 226-253, April 1998.

[40] Christos Koulamas, "A new constructive heuristic for the flowshop scheduling problem," *European Journal of Operational Research*, vol. 105, no. 1, pp. 66-71, February 1998.

[41] Ruben Ruiz, Concepcion Maroto, and Javier Alcaraz, "Solving the flowshop scheduling problem with sequence dependent setup times using advanced metaheuristics," *European Journal of Operational Research*, vol. 165, no. 1, pp. 34-54, March 2005.

[42] Willem J. Selen and David D. Hott, "A Mixed-Integer Goal-Programming Formulation of the Standard Flow-Shop Scheduling Problem," *The Journal of the Operational Research Society*, vol. 37, no. 12, pp. 1121-1128, December 1986.

[43] Edward F. Stafford, "On the Development of a Mixed-Integer Linear Programming Model for the Flowshop Sequencing Problem," *The Journal of the Operational Research Society*, vol. 39, no. 12, pp. 1163-1174, December 1988.

[44] Shaukat A. Brah and John L. Hunsucker, "Branch and bound algorithm for the flow shop with multiple processors," *European Journal of Operational Research*, vol. 51, no. 1, pp. 88-99, March 1991.

[45] O. Moursli and Y. Pochet, "A branch-and-bound algorithm for the hybrid flowshop," *International Journal of Production Economics*, vol. 64, no. 1, pp. 113-125, March 2000.

[46] F. Della Croce, M. Ghirardi, and R. Tadei, "An improved branch-and-bound algorithm for the two machine total completion time flow shop problem," *European Journal of Operational Research*, vol. 139, no. 2, pp. 293-301, June 2002.

[47] Edward F. Stafford Jr. and Fan T. Tseng, "Two models for a family of flowshop sequencing problems," *European Journal of Operational Research*, vol. 142, no. 2, pp. 282-293, October 2002.

[48] Vincent T'kindt, Jatinder N. D. Gupta, and Jean-Charles Billaut, "Two-machine "owshop scheduling with a secondary criterion," *Computers & Operations Research*, vol. 30, no. 4, pp. 505-526, April 2003.

[49] Sang M. Lee and Arben A. Asllani, "Job scheduling with dual criteria and sequence-dependent setups: mathematical versus genetic programming," *OMEGA: The International Journal of Management Science*, vol. 32, no. 2, pp. 145-153, April 2004.

[50] Fan T. Tseng, Edward F. Stafford Jr., and Jatinder N. D. Gupta, "An empirical analysis of integer programming formulations for the permutation flowshop," *OMEGA: The International Journal of Management Science*, vol. 32, no. 4, pp. 285-293, August 2004.

[51] R. A. Dudek, S. S. Panwalkar, and M. L. Smith, "The Lessons of Flowshop Scheduling Research," *Operations Research*, vol. 40, no. 1, pp. 7-13, January 1992.

[52] Gary Chartrand, *Introductory Graph Theory*. Mineola, NY, USA: Dover Publications Inc., 1985.

[53] Jonathan L. Gross and Jay Yellen, *Graph Theory and its Applications*, 2nd ed. Boca Raton, FL, USA: Chapman & Hall/CRC, 2006.

[54] Richard W. Conway, William L. Maxwell, and Louis W. Miller, *Theory of Scheduling*`. Mineola, NY, USA: Dover Publications Inc., 2003.

[55] Jouko Seppanen and James M. Moore, "Facilities Planning with Graph Theory," *Management Science*, vol. 17, no. 4, pp. 242-253, December 1970.

[56] Walter H. Kohler, "A Preliminary Evaluation of the Critical Path Method for Scheduling Tasks on Multiprocessor Systems," *IEEE Transactions on Computers*, vol. 24, no. 12, pp. 1235-1238, December 1975.

[57] Ravi Sethi, "Scheduling Graphs on Two Processors," *SIAM Journal on Computing*, vol. 5,

no. 1, pp. 73-82, March 1976.

[58] Yu-Kwong Kwok and Ishfaq Ahmad, "Dynamic Critical-Path Scheduling: An Effective Technique for Allocating Task Graphs to Multiprocessors," *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, no. 5, pp. 506-521, May 1996.

[59] Avrim L. Blum and Merrick L. Furst, "ast planning through planning graph analysis," *Artificial Intelligence*, vol. 90, no. 1, pp. 281-300, February 1997.

[60] Hong-Chao Zhang and Enhao Lin, "A hybrid-graph approach for automated setup planning in CAPP," *Robotics and Computer-Integrated Manufacturing*, vol. 15, no. 1, pp. 89-100, February 1999.

[61] J. O. Mayhugh, "On the mathematical theory of schedules," *Management Science*, vol. 11, no. 2, pp. 289-307, 1964.

[62] Special Projects Office, Bureau of Naval Weapons, "PERT: Summary Report, Phase I," Department of the Navy, Washington D.C., 1958.

[63] J. W. Pocock, "PERT as an Analytical Aid for Program Planning-Its Payoff and Problems," *Operations Research*, vol. 10, no. 6, pp. 893-903, November 1962.

[64] Richard M. van Slyke, "Monte Carlo Methods and the PERT Problem," *Operations Research*, vol. 11, no. 5, pp. 839-860, September 1963.

[65] James M. Antill and Ronald W. Woodhead, *Critical Path Methods in Constructive Practice*, 4th ed.: Wiley - IEEE, 1990.

[66] Ali Jaafari, "Criticism of CPM for Project Planning Analysis," *Journal of Construction Engineering and Management*, vol. 110, no. 2, pp. 222-233, June 1984.

[67] Willy S. Herroelen, "Resource-constrained Project Scheduling - the State of the Art," *Operations Research Quarterly*, vol. 23, no. 3, pp. 261-275, 1972.

[68] Peter Brucker, Andreas Drexl, Rolf Mohring, Klaus Neumann, and Erwin Pesch, "Resource-constrained project scheduling: Notation, classification, models, and methods," *European Journal of Operational Research*, vol. 112, no. 1, pp. 3-41, January 1999.

[69] Erik L. Demeulemeester and Willy S. Herroelen, "New Benchmark Results for the Resource-Constrained Project Scheduling Problem," *Management Science*, vol. 43, no. 11, pp. 1485-1492, November 1997.

[70] H. J. Laue, "Efficient methods for the allocation of resources in project networks," *Unternehmensforshung*, vol. 12, p. 133, 1968.

[71] R. Petrovic, "Optimization of resource allocation in project planning," *Operations Research*, vol. 16, no. 3, pp. 559-568, 1968.

[72] Nicos Christofides, R. Alvarez-Valdes, and J. M. Tamarit, "Project scheduling with resource constraints: a branch and bound approach," *European Journal of Operational Research*, vol. 29, no. 3, pp. 262-273, June 1987.

[73] Erik Demeulemeester and Willy Herroelen, "A Branch-and-Bound Procedure for the Multiple Resource-Constrained Project Scheduling Problem," *Management Science*, vol. 38, no. 12, pp. 1803-1818, December 1992.

[74] Aristide Mingozzi, Vittorio Maniezzo, Salvatore Ricciardelli, and Lucio Bianco, "An Exact Algorithm for the Resource-Constrained Project Scheduling Problem Based on a New mathematical Formulation," *Management Science*, vol. 44, no. 5, pp. 714-729, May 1998.

[75] Peter Brucker, Sigrid Knust, Arno Schoo, and Olaf Thiele, "A branch and bound algorithm for the resource-constrained project scheduling problem," *European Journal of Operational Research*, vol. 107, no. 2, pp. 272-288, June 1998.

[76] Arno Sprecher, "Scheduling Resource-Constrained Projects Competitively at Modest Memory Requirements," *Management Science*, vol. 46, no. 5, pp. 710-723, May 2000.

[77] Genmou Jiang and Jonathan Shi, "Exact Algorithm for Solving Project Scheduling Problems under Multiple Resource Constraints," *Journal of Construction and Engineering Management*, vol. 131, no. 9, pp. 986-992, September 2005.

[78] Edward W. Davis and James H. Patterson, "A Comparison of Heuristic and Optimum Solutions in Resource-Constrained Project Scheduling," *Management Science*, vol. 21, no. 8, pp. 944-955, April 1975.

[79] I. Kurtulus and E. W. Davis, "Multi-Project Scheduling: Categorization of Heuristic Rules Performance," *Management Science*, vol. 28, no. 2, pp. 161-172, February 1982.

[80] Rainer Kolisch, "Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation," *European Journal of Operational Research*, vol. 90, no. 2, pp. 320-333, April 1996.

[81] Rainer Kolisch, "Efficient priority rules for the resource-constrained project scheduling problem," *Journal of Operations Management*, vol. 14, no. 3, pp. 179-192, September 1996.

[82] Jan Weglarz, *Project scheduling: recent models, algorithms, and applications*. Norwell, Massachusetts, USA: Kluwer Academic Publishers, 1999.

[83] Fayez F. Boctor, "Some efficient multi-heuristic procedures for resource-constrained project scheduling," *European Journal of Operational Research*, vol. 49, no. 1, pp. 3-13, November 1990.

[84] Rainer Kolisch and Sonke Hartmann, "Experimental investigation of heuristics for resource-constrained project scheduling: An update," *European Journal of Operational Research*, vol. 174, no. 1, pp. 23-37, October 2006.

[85] K. Bouleimen and H. Lecocq, "A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version," *European Journal of Operational Research*, vol. 149, no. 2, pp. 268-281, September 2003.

[86] Vicente Valls, Francisco Ballestin, and Sacramento Quintanilla, "Justification and RCPSP: A technique that pays," *European Journal of Operational Research*, vol. 165, no. 2, pp. 375-386, September 2005.

[87] Paul R. Thomas and Said Salhi, "A Tabu Search Approach for the Resource Constrained Project Scheduling Problem," *Journal of Heuristics*, vol. 4, no. 2, pp. 123-139, July 1998.

[88] Robert Klein, "Project scheduling with time-varying resource constraints," *International Journal of Production Research*, vol. 38, no. 16, pp. 3937-3952, 2000.

[89] Sonke Hartmann, "A Competitive Genetic Algorithm for Resource-Constrained Project Scheduling," *Naval Research Logistics*, vol. 45, no. 7, pp. 733-750, October 1998.

[90] J. Alcaraz and C. Maroto, "A robust genetic algorithm for resource allocation in project scheduling," *Annals of Operations Research*, vol. 102, no. 1, pp. 83-109, February 2001.

[91] Y. Cengiz Toklu, "Application of genetic algorithms to construction scheduling with or without resource constraints," *Canadian Journal of Civil Engineering*, vol. 29, no. 3, pp. 421-429, June 2002.

[92] Sonke Hartmann, "A Self-Adapting Genetic Algorithm for Project Scheduling under Resource Constraints," *Naval Research Logistics*, vol. 49, no. 5, pp. 433-448, August 2002.

[93] Khalil S. Hindi, Hongbo Yang, and Krzysztof Fleszar, "An Evolutionary Algorithm for Resource-Constrained Project Scheduling," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 5, pp. 512-518, October 2002.

[94] Daniel Merkle, Martin Middendorf, and Hartmut Schmeck, "Ant Colony Optimization for Resource-Constrained Project Scheduling," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 4, pp. 333-346, August 2002.

[95] Irem Ozkarahan, "Allocation of Surgical Procedures to Operating Rooms," *Journal of Medical Systems*, vol. 19, no. 4, pp. 333-352, 1995.

[96] Jonathan Patrick, Martin L. Puterman, and Maurice Queyranne, "Dynamic Multipriority Patient Scheduling for a Diagnostic Resource," *Operations Research*, vol. 56, no. 6, pp. 1507-1525, 2008.

[97] M. Puterman, *Markov Decision Processes*. New York: John Wiley and Sons, 1994.

[98] B. Cheang, H. Li, A. Lim, and B. Rodrigues, "Nurse rostering problems – a bibliographic survey," *European Journal of Operational Research*, vol. 151, no. 3, pp. 447-460, December 2003.

[99] Edmund K. Burke, Patrick De Causmaecker, Greet Vanden Berghe, and Hendrik Van Landeghem, "The State of the Art of Nurse Rostering," *Journal of Scheduling*, vol. 7, no. 6, pp. 441-499, 2004.

[100] A. T. Ernst, H. Jiang, M. Krishnamoorthy, and D. Sier, "Staff scheduling and rostering: A review of applications, methods and models," *European Journal of Operational Research*,

vol. 153, no. 1, pp. 3-27, February 2004.

[101] Lori S. Franz, Hope M. Baker, G. Keong Leong, and Terry R. Rakes, "A mathematical model for scheduling and staffing multiclinic health regions," *European Journal of Operational Research*, vol. 41, no. 3, pp. 277-289, August 1989.

[102] Irem Ozkarahan, "A Disaggregation Model of a Flexible Nurse Scheduling Support System," *Socio-Economic Planning Sciences*, vol. 25, no. 1, pp. 9-26, January 1991.

[103] Ilham Berrada, Jacques A. Ferland, and Philippe Michelon, "A Multi-objective Approach to Nurse Scheduling with both Hard and Soft Constraints," *Socio-Economic Planning Sciences*, vol. 30, no. 3, pp. 183-193, September 1996.

[104] Brigitte Jaumard, Frederic Semet, and Tsevi Vovor, "A generalized linear programming model for nurse scheduling," *European Journal of Operational Research*, vol. 107, no. 1, pp. 1-18, May 1998.

[105] Harvey H. Millar and Mona Kiragu, "Cyclic and non-cyclic scheduling of 12 h shift nurses by network programming," *European Journal of Operational Research*, vol. 104, no. 3, pp. 582-592, February 1998.

[106] Mihoko Okada and Masahiko Okada, "Prolog-Based System for Nursing Staff Scheduling Implemented on a Personal Computer," *Computers and Biomedical Research*, vol. 21, no. 1, pp. 53-63, February 1988.

[107] Michael M. Kostreva and Karen S. B. Jennings, "Nurse scheduling on a microcomputer," *Computers & Operations Research*, vol. 18, no. 8, pp. 731-739, January 1991.

[108] Sabah U. Randhawa and Darwin Sitompul, "A heuristic-based computerized nurse scheduling system," *Computers & Operations Research*, vol. 20, no. 8, pp. 837-844, October 1993.

[109] Michael J. Brusco and Larry W. Jacobs, "Cost analysis of alternative formulations for personnel scheduling in continuously operating organizations," *European Journal of Operational Research*, vol. 86, no. 2, pp. 249-261, October 1995.

[110] Gary M. Thompson, "A simulated-annealing heuristic for shift scheduling using non-continuously available employees," *Computers & Operations Research*, vol. 23, no. 3, pp. 275-288, March 1996.

[111] Kathryn A. Dowsland, "Nurse scheduling with tabu search and strategic oscillation," *European Journal of Operational Research*, vol. 106, no. 2, pp. 393-407, March 1997.

[112] Koji Nonobe and Toshihide Ibaraki, "A tabu search approach to the constraint satisfaction problem as a general problem solver," *European Journal of Operational Research*, vol. 106, no. 2, pp. 599-623, March 1997.

[113] Uwe Aickelin and Kathryn A. Dowsland, "Exploiting problem structure in a genetic algorithm approach to a nurse rostering problem," *Journal of Scheduling*, vol. 3, no. 3, pp. 139-153, May 2000.

[114] Edmund Burke, Peter Cowling, Patrick De Causmaecker, and Greet Vanden Berghe, "A Memetic Approach to the Nurse Rostering Problem," *Applied Intelligence*, vol. 15, no. 3, pp. 199-214, November 2001.

[115] Dr. Amgad Eskander, *Workflow of dialysis unit*., 2008.

[116] Juan Vera, *MSCI 603 - Principles of Operations Research course notes*.: University of Waterloo, 2008.

[117] Wayne L. Winston and Munirpallam Venkataramanan, *Introduction to Methematical Programming*, 4th ed.: Brooks/Cole, Thomson Learning, Inc, 2003.

[118] Kenneth Price and Rainer Storn, "Differential Evolution - A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces," *Journal of Global Optimization*, vol. 11, pp. 341 - 359, 1997.

[119] Saleh Tabandeh, William W. Melek, and Christopher M. Clark, "An adaptive niching genetic algorithm approach for generating multiple solutions of serial manipulator inverse kinematics with applications to modular robots," *Robotica*, vol. 00, pp. 1-15, May 2009.

[120] Andrew Makhorin. (2008, October) GLPK (GNU Linear Programming Kit). [Online]. http://www.gnu.org/software/glpk/

[121] Todd Faasse and Paul. Shreve, "Positron Emission Tomography–Computed Tomography Patient Management and Workflow," *Seminars in Ultrasound, CT, and MRI*, vol. 29, no. 4, pp. 277-282.