MASTER'S THESIS

A System to Test Rescheduling Algorithms

by K.M. Gerber
Advisor: J.W. Herrmann

M.S. 98-3

ISR develops, applies and teaches advanced methodologies of design and analysis to solve complex, hierarchical, heterogeneous and dynamic problems of engineering technology and systems for industry and government.

ISR is a permanent institute of the University of Maryland, within the Glenn L. Martin Institute of Technology/A. James Clark School of Engineering. It is a National Science Foundation Engineering Research Center.

Web site http://www.isr.umd.edu

ABSTRACT

Title of Thesis: A SYSTEM TO TEST RESCHEDULING ALGORITHMS

Degree candidate: Kenneth Michael Gerber

Degree and year: Master of Science, 1998

Thesis directed by: Assistant Professor Jeffrey W. Herrmann
Department of Mechanical Engineering and
Institute for Systems Research

Unpredictable events constantly force managers to reschedule projects. Ongoing research in project scheduling continues to develop new rescheduling algorithms. Until now, however, there was no standard method for testing rescheduling algorithms and analyzing their performance. This thesis explains the design and capabilities of a new system to test rescheduling algorithms. It sets a standard for analyzing rescheduling algorithms by creating a general methodology applicable to the entire testing process. The test system performs the tedious work of creating problem instances and collecting data on algorithm performance while giving the user the flexibility to use any rescheduling algorithm and the ability to perform complex analysis.

A SYSTEM TO TEST RESCHEDULING ALGORITHMS

by

Kenneth Michael Gerber

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Master of Science
1998

Advisory Committee:

Assistant Professor Jeffrey W. Herrmann, Chair
Professor Michael O. Ball
Associate Professor Michael C. Fu

DEDICATION

To my friends and family from Rockland County and Clarkstown South

ACKNOWLEDGEMENTS

I want to take this opportunity to thank Professor Herrmann, who has advised me for the past two years. During that time, I've wandered around aimlessly a lot, and he's done a good job of pointing me in the right direction. His dialogue was thought-provoking, and his comments were immeasurably helpful.

I'd also like to thank Professors Ball and Fu for serving on my committee. I was fortunate enough to also have Professor Ball as a teacher for one class. I am indebted to all my wonderful teachers, from elementary school through graduate school, as well as the staffs at all these levels.

I did most of my work in the Computer Integrated Manufacturing Lab. Without Dr. Ed Lin and the rest of the graduate students there, I probably would have given up in frustration a long time ago.

For the first year, we met frequently with members of both KPMG and the Navy down at Patuxent River. What for me was a learning experience was probably more like a waste of time for them. To Marvin, Alex, Mike, Tab, George, Lloyd, Barry, Jon, and Sam: thanks for letting the long-haired freak from Maryland pester you guys for a while.

Most importantly, I have to thank my family and friends. You supported me through this even when no one, including me, knew what was going on. And you reminded me that stuff like this really isn't important compared to other things.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

A SYSTEM TO TEST RESCHEDULING ALGORITHMS

Kenneth Michael Gerber

August 3, 1998

This comment page is not part of the dissertation.

Typeset by LaTeX using the `dissertation` class by Pablo A. Straub, University of Maryland.

Chapter 1

Introduction

A key aspect in the function of many businesses is the scheduling of projects. These projects consist of smaller tasks, each of which requires resources. These projects must be completed under certain constraints to meet some overall objective. The livelihood of many businesses strongly depends not only upon scheduling these tasks but also upon effectively maintaining high-quality schedules in a dynamic environment. For any problem of moderate scope, algorithms must be used to systematically search for a good schedule. This also applies to rescheduling, where disturbances alter the conditions under which a previous schedule was in effect.

There are a number of scheduling and rescheduling algorithms available today in the literature of operations research and production management. Testing these algorithms is an important task. It is imperative to determine which of these algorithms perform well, how well they perform, and under which circumstances certain algorithms are preferable to others. This paper will discuss exactly this point: the design of a system that tests these rescheduling algorithms.

1.1 Motivation — Pax River

The Patuxent River Naval Air Station is located at the mouth of the Patuxent River (65 miles Southeast of the Pentagon), and occupies approximately 7,950 acres on Cedar Point. The Air Station is the headquarters of the Naval Air Warfare Center Aircraft Division (NAWCAD), nearly 50 tenant activities and the U.S. Navy's only Test Pilot School. Commissioned in 1943 to centralize aircraft testing efforts, Patuxent River is now considered the U.S. Navy's premier test and evaluation, research and development center for naval aviation. The Naval Air Station has 5 runways (the longest is 11,800 feet), 50,000 square miles of air space available for operations, 5,000 square miles of controlled airspace, and 780 square miles of restricted airspace. In this area are conducted 50,500 flight hours per year and 171,546 operations per year by 12,200 personnel in 1,064 buildings worth more than $1.81 billion. [Tea96, Off97]

At the Patuxent River Naval Air Station, the Test and Evaluation (T&E) Team of the Naval Air Warfare Center Aircraft Division (NAWCAD) performs approximately 1500 tasks using 250 resources every day. For instance, one project tests an airplane's electronics under extreme heat, radiation, and moisture. Another project tests the airplane's weaponry system. Each of these complex projects is composed of simpler activities. Testing electronics must have the heat test before the radiation test, and both weapons and airplane must be checked on the ground before the weapons are tested while in flight. Renewable resources include personnel, the electromagnetic environmental effects testing facilities, and the runway. The electronics testing requires the test facility for 100 hours and 500 labor-hours of Class V engineers. Weaponry

testing requires the runway and 100 labor-hours of pilots. The manager's objectives are to finish all of the required tests on time.

Scheduling these tasks is an enormous undertaking. Furthermore, because of the nature of the business, there are frequent test cancellations, additions, and modifications, frequently with little advance warning. Also, conditions beyond the control of any of the employees, such as inclement weather, or equipment failures that force retesting, create a need for frequent rescheduling. Every day, managers must attempt to schedule the additions and also reshuffle tasks to fill the gaps that cancellations cause. Rescheduling has the potential to produce schedules that are more appropriate to the new conditions and that address the unforeseeable disturbances of the workplace. These better schedules will allocate resources to tasks in a more efficient manner. By doing this, the test facilities at Pax can better satisfy customers (the people using the test facilities) by finishing work on time, and increase their profits by accepting more jobs.

Although managers use manual techniques to reschedule tasks, they could also use computer programs to reschedule tasks. Creating or selecting a program would then lead to comparing different rescheduling algorithms.

1.2 Test System

A system for testing rescheduling algorithms would be a useful tool. The testing system works with schedules, much as they would exist at Pax in the previous example. The program has a disruptor which attempts to model some of the contingencies that may exist, and repaired schedules are analyzed based on certain criteria. A human researcher or scheduler can then take this

information and hopefully make better decisions using rescheduling algorithms in different situations.

There are many issues that the test system must address over the course of its development. The following questions address some of these issues:

- How should the system create rescheduling instances?
- What types of disturbances should occur?
- In what format should the system store the instances?
- Should the test system run the rescheduling algorithms?
- In what format should the system accept repaired schedules?
- How should the system evaluate rescheduling algorithm performance?
- In what format should the system record the results?

The potential benefits of a test system include standardizing and automating testing methodologies and a clear side-by-side comparison (in more than one dimension) of rescheduling algorithms under various scenarios. Implementing the correct algorithm will yield better schedules for any organization using rescheduling algorithms. These tangible benefits include reduced time to complete projects, reduced costs, improved customer relations, and increased long-run performance.

Ideally, a test system would become a tool that is available to researchers to test their own algorithms and to managers in industry to help them select the most appropriate rescheduling algorithms for their particular dynamic environment.

1.3 Overview of Work

This project included three major parts. First, I visited Pax River to understand the need for rescheduling, and I studied systems engineering and operations research to understand the relevant issues. I designed, created, and developed the test system. Finally, I wrote this thesis to explain the work performed.

1.3.1 Systems Engineering

Throughout the course of this project, I have used and applied the principles of systems engineering. Systems engineering has its roots during the period of World War II, and it combines elements of traditional engineering with ideas from business and management for developing large, complex systems. Systems engineering includes communication between personnel in different disciplines. Two important ideas from systems engineering are trade-off analysis to compare alternative concepts using multiple objectives and the life cycle of a system, which begins with the problem definition and ends with the system retirement.

1.3.2 Summary of System

This thesis describes a test system that evaluates the effectiveness of rescheduling algorithms. To achieve this, the system uses a data set of project definitions. An automatic schedule generator creates initial feasible schedules. A random disruptor creates unexpected disturbances and add these occurrences to the initial project definition, simulating the dynamic environment of the workplace. This infeasible schedule is the input for the

rescheduling algorithms, which create new, feasible schedules. Then, the system compares and contrasts the performance of these schedules. It measures makespan, the number of disrupted tasks, and the total task disruption time.

The system uses the tools of Microsoft Project 98 to create projects and initial schedules. It also uses Microsoft Excel for Windows 95 Version 7.0 for manipulating analysis information. The system uses Microsoft Visual Basic code for its subroutines.

The system has some limitations. The system allows only one mode per task (there is only one grouping of resources which can perform each task). The multi-mode case, which is the more general extension of the single mode case, could be an extension to the system in future work. The system does not compare the rescheduling algorithm effort (e.g. computation time). Also, there are many restrictions on how data is formatted and how the system manipulates this data.

1.3.3 Guide to Remaining Chapters

The remainder of this thesis is organized as follows: Chapter 2 discusses the background of both scheduling and rescheduling problems and presents many ideas from current literature. Chapter 3 gives a detailed mathematical formulation of the rescheduling problem. Then, Chapter 4 gives a rigorous description of the entire system, including the processes, overall functions, and individual components. Finally, Chapter 5 presents a summary of this thesis and conclusions of the research. For the benefit of the system's users, Appendix A includes a user's manual, which has step-by-step instructions.

Chapter 2

Background

This chapter reviews topics related to this project. Section 2.1 reviews the definition of systems engineering and how it relates to other engineering disciplines. Section 2.2 discusses project scheduling problems and solution techniques. Section 2.3 focuses on rescheduling and its relation to scheduling. Section 2.4 examines the comparison of different algorithms used in rescheduling. Finally, Section 2.5 summarizes the chapter.

2.1 Systems Engineering

According to Chapanis [Cha96, page 21], "systems engineering is not yet well established as an engineering discipline." Chapanis goes on to say that "engineers have not yet been able to arrive at a consensus definition of systems engineering."

Systems engineering is not a typical engineering field. Although expertise in a traditional field such as civil, mechanical, or electrical engineering is beneficial, systems engineering attempts to form methodologies that are widely applicable. Because of this, systems engineering can apply similar principles and practices to a wide range of problems, rather than

forming specific solutions to instances and having little to learn from in other instances. It often links engineering fields and frequently includes areas of economics, psychology, and business because of its stress on methodologies that are broad in scope.

Systems engineering often formulates problems as mathematical models. There are many tools, including mathematical programming, that can analyze and derive information from the mathematical model. Sensitivity analysis performed on the model analyzes its robustness, and trade-off analysis evaluates alternatives. Frequently, systems engineers must estimate the utility of a system to many different classes of people (such as user, maintainer, environmental neighbor), each of whom may have different or even competing criteria measures. Systems engineering's emphasis on financial as well as human factors and industrial psychology concerns ensures that all techniques are relevant to a wide variety of areas, be they engineering or not. A system could certainly be a computer system, an automobile, or as chemical factory, but there are other non-engineering systems as well, such as educational systems, law enforcement, and transportation systems.

This thesis uses several models. The first simplifies a real environment involving people, places, and things into a mathematical notation of tasks, resources, and allocations. Scheduling these tasks is separate from performing them. Also, this thesis assumes that time and money are directly related. The longer a project takes, the more money it costs. This thesis does not explicitly have monetary values but its capturing of temporal values is the assumed equivalent. The model's temporal component uses a discrete model that rounds events to the nearest minute.

Large systems, consisting of a collection of objects working together to perform some objective, are inherently interdisciplinary and require the collaboration of people in many diverse fields. In the construction of a new airport, for example, the runway must be structurally sound, be in close yet safe proximity to the gates, and have a control tower capable of tracking all the aircraft expected to use the airspace. Before construction begins, one must decide if a new airport is truly necessary and determine where to construct it. Such a daunting task requires a logical set of steps to run smoothly.

There are two basic models of this set of steps, the waterfall model of systems life-cycle and the spiral model of systems life-cycle [Aus97, sections 3.2–3.3]. Austin describes the difference as "[t]he waterfall model ... views the engineering process as a linear sequence of stages that includes requirements specifications, design and testing, integration, and maintenance" whereas "[t]he spiral model of systems development corresponds to a sequence of waterfall models." The waterfall model of systems life cycle development has clear and definite starts and ends to its different phases. The spiral model is more of an iterative process: a series of waterfalls.

The systems engineer must be familiar with the system life cycle of the proposed product [CBW92]:

1. Requirements development phase

   List and fully explain all the requirements of the system. These include data requirements such as input and output interaction as well as functional requirements, what the system is supposed to do. At times, it is difficult to properly ascertain the true needs of the customer. It requires information from personnel in marketing, operations, finance,

engineering, and management. Along with these go groundrules for the entire system design process and the people working on it, the scope of the project, and the proof that a solution to the problem can be found.

2. Concept development phase

   It is within this stage that different alternatives are compared. Broad classes of solutions are judged on their respective merits and a final concept, or general framework with associated attributes and technologies, is selected. Because abstract goals are difficult to tackle, the necessary functions of the proposed system are divided hierarchically into smaller and smaller sub-functions that are easier to manage than the overall objective.

3. Engineering development phase

   Here, detailed plans are made for the system. These could include blueprints, pseudocode, and process plans. There should be enough detail to ensure that actual production can begin based on these plans.

4. System development phase

   This is the real production of the system. Systems engineers typically are in charge of overseeing the different disciplines and coordinating the efforts between departments. If the system is very large, a prototype or scale model may be produced first, which would serve as a less expensive means to illustrating possible room for improvement in the final system.

5. Test and integration phase

   Together with reliability and test engineers, systems engineers verify not

only that the individual components work but also that they work well together to perform the overall objective.

6. Operations, support, and modification phase

   Any system, even those of the highest quality, will need periodic check-ups and updates. These should be expected, and they are indeed necessary for future improvements. It must be determined to what degree the system is satisfying the ever-changing needs of the customer.

7. Phase-out and replacement phase

   Eventually, the system will have run its course. At this point, depending on the physical nature of the system, it may be possible to recover some of the cost by selling for salvage or recycling individual components or larger subsystems, if not the entire system. It may be possible to keep some of the parts for the new system that will eventually replace this one.

The most important, and often most neglected aspect, is the requirements stage. It is imperative, even when engineers have a good idea of what they are doing, to detail and record what they are doing. What must this system accomplish? What would it be nice for it to accomplish? What will it not accomplish? These are important questions when determining a project's scope and size. Everything that comes after this step (the allocation of resources to the project, the analysis of costs and benefits of the system, the life cycle depiction) relies heavily on a concrete framework of the final product. This can involve a list of requirements, a work breakdown structure, or a dictionary of elements. A detailed and complete set of requirements helps the entire development process. Requirements warn of technical or financial

infeasibility. They not only indicate direction, but also suggest milestones. A schedule or time-line could provide this information.

At each stage of the system life cycle, the engineer must carefully document everything. This serves many purposes. To ensure that the system satisfies all of the requirements, the requirements must be readily available, perhaps years after they were first mentioned. Because large systems often have life spans longer than the jobs of individual workers, documentation serves to explain why things are being done, and when certain milestones need to be reached. Documentation also enables future users and maintainers of systems to properly perform their work, long after the designers and developers have completed theirs.

These steps do not focus on one instance or particular case study. Rather, they attempt to develop a methodology that is pertinent to a general class of problems. In this way, a standardized and logically organized set of ideas can be applied in completeness.

The interdisciplinary nature of systems engineering has inherent benefits. Because of its inclusion of many people from the early stages, the time and cost of the system from cradle to grave may be reduced. Changes in the wing of an aircraft for aerodynamic reasons, for example, would immediately be known to the group working on the electrical system. If this change causes a problem, the different groups may solve the problem early, rather than waiting to solve it later at a much greater expense. This early planning and revising before production, rather than creating and trying to produce and revising and trying to produce, saves the customer time and money.

2.2 Project Scheduling

Scheduling problems have been studied in great detail over the past several decades. The primary reasons are the simplicity of the problem statement, the complexity of finding the solution, and their wide applicability in many fields of business and industry. Project scheduling involves finding the best time and method to accomplish a number of tasks subject to certain constraints. Its applications range from planning the construction of a skyscraper to establishing routes and times of an airline fleet to creating schedules for space shuttle activities.

Listed below is a generic project scheduling example. It is a fictitious schedule for installing a new clock exhibit in a museum, from the example in the tutorial of Microsoft Project 4.0. There are a number of tasks, each of which requires resources. There are start and end times for each task, and precedence relations between some tasks. The two most common visual displays of a schedule are the Gantt chart and the Critical Path Method network graph. The Gantt chart (see Figure 2.1) is a bar chart focusing on the start and end times of each task. The Critical Path Method (see Figure 2.2), on the other hand, focuses on the precedence relations between tasks.

2.2.1 Classifications

There are many different types of project scheduling problems [Boc90, Dav73, DH97, Her72, IEZ93, KD82, ÖU95, SC93]. One major division is based on whether the problem is static or dynamic. Static scheduling assumes all inputs are known a priori and remain constant during the search for a solution. In contrast, dynamic scheduling allows for changes in

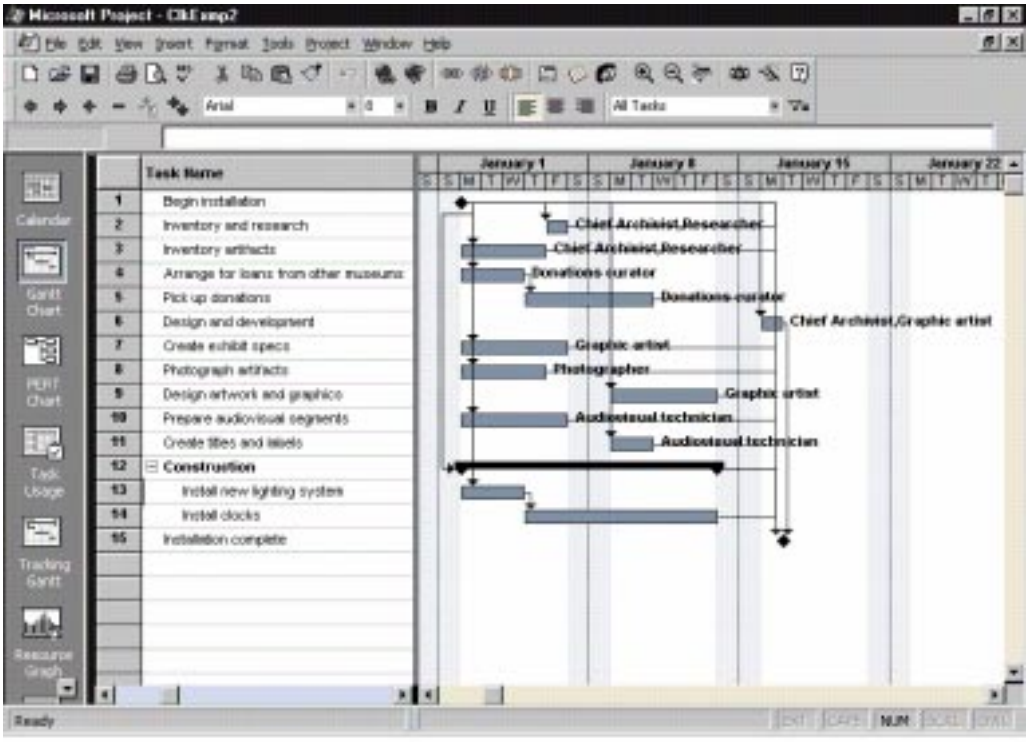

Figure 2.1: Gantt Chart Representation of a Project

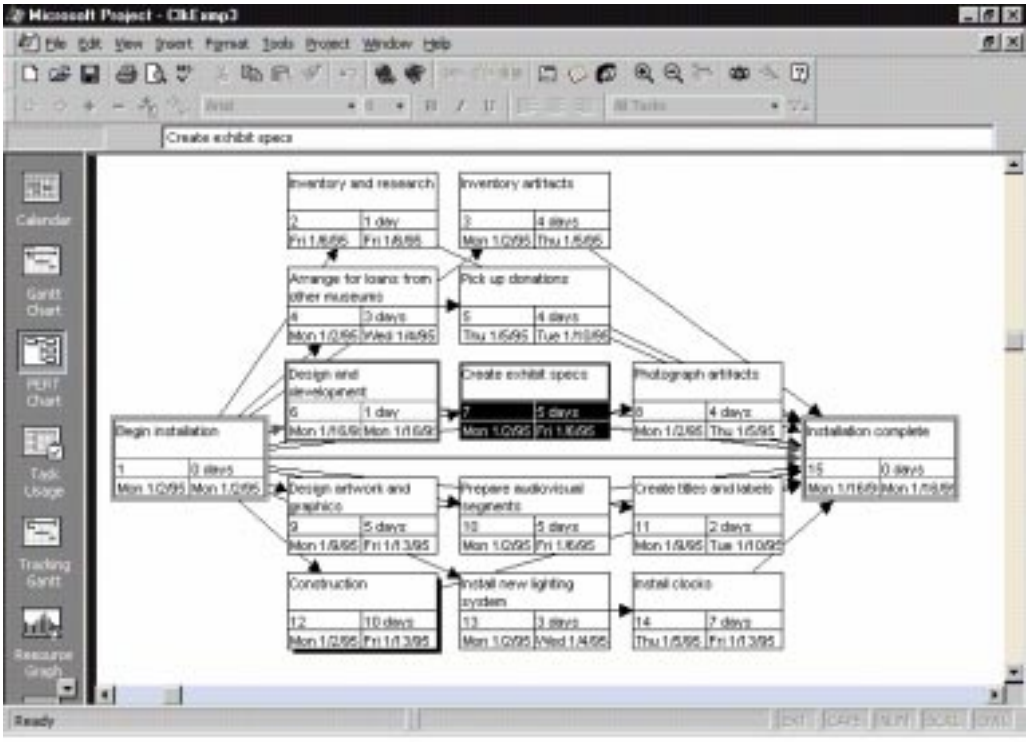

Figure 2.2: Critical Path Method Representation of a Project

the inputs (unexpected disturbances such as machines breaking down or additional customers arriving) and may be given an initial schedule, which must be modified.

The most straightforward way to include aspects of a dynamic workplace is to have the inputs be random variables, rather than known constants. This element of uncertainty frequently causes decisions to be made which may be dependent on future events and changed later on. Another way to account for uncertainty is to change the inputs (durations of tasks, starting/ending times) for a problem which already has a schedule, and then attempt to recover a feasible schedule.

All tasks involve resources. These include the people performing the tasks, as well as the equipment they use and the space within which they perform the task. Some scheduling problems assume unlimited resource availability, while more realistic problems impose limits on the resources, which is always the case in the real world. The scheduling problem could have renewable resources that have limits on the resources every time period, nonrenewable resources that have limits over the entire scheduling horizon, or doubly-constrained resources that have limits both every period and in total [SV93, ÖU95]. The limits on renewable resources may be the same for each time period, or they may vary with time [dWB92, TKE94, ÖU95]. Also, the consumption of resources may be continuous or discrete [ÖU95].

Scheduling problems may allow for preemption, the interruption of a job after it has started but before it has completed to begin another job, or state that once a job has begun, it must run to completion [SV93]. There may be multiple modes, or alternatives, of performing the same

task [Wil85, Boc90, SV93], or there may be only one way. There may be up to four types of precedence constraints, giving temporal relationships between tasks (Task A's start/finish time in relation to Task B's start/finish time), and also ready times and due dates, given real times of earliest/latest start/finish of some task [DH97]. Finally, multiple projects may be considered separately or grouped together into one larger project [ÖU95, SV93].

2.2.2 Solution Techniques

In general, the project scheduling problem is a difficult problem; most versions are NP-hard. This means that no algorithm has been developed which can solve the scheduling problem to optimality in polynomial time, and it is probable that none can be. There are therefore a number of heuristic as well as exact techniques for coming up with a solution [ÖU95]. For the static problem, one exact approach employs branch and bound using partial schedules [Pat84]. A similar enumeration technique using heuristics can be used [Her72] where the heuristics are rules which rank the activities based on some criterion and then schedule the activities based on their importance [Woo93, Her72]. It is also possible to combine heuristics [Boc90] or to try as many heuristics as possible [Wil85]. A heuristic can be applied to a schedule found by another heuristic [LW93]. These heuristics can consider such things as the duration of the tasks remaining to be scheduled, their slack time, their resource usage, or their predecessors and successors. One such scheduling program, Micro-Boss [Sad94], concentrates scheduling efforts on those resources which lead to bottlenecks. In general, heuristics can be serial-activity, where the priority is fixed at the beginning of the scheduling process, or parallel-activity,

where the priority is updated at every iteration of the scheduling process [ÖU95]. Other approaches to the scheduling problem include man-computer interaction (e.g., the leitstand [PS94]), dynamic programming [Her72], integer programming [Her72, SDK78], and hierarchical decomposition [SV93], which considers the tactical and then the operational level.

2.2.3 Objective Functions

The objective functions for project scheduling problems are typically time and money [SV93]. The most common objective function is to minimize the total project duration [DH97, Her72, Wil85, Pat84]. Other measures of time include mean lateness, mean completion time [ÖU95], and total or weighted earliness subject to maximum tardiness [TXF97]. From an economic standpoint, it is desirable to maximize net present value [SV93] or minimize total costs [Boc90]. Other managerial measures include maximizing total or average resource utilization [Wil85, JE97]. Time and money may be considered simultaneously [ÖU95], although they are usually considered separately for simplicity. In this case, it may be necessary to see how much of which additional resources may be required to meet a specific due date (the time-cost trade-off) [Wil85].

2.2.4 Size of Problem

The problem size that scheduling algorithms can handle varies greatly. For difficult exact approaches such as integer programming, Herroelen had less than 20 tasks [Her72]. For rescheduling, Zweben compared approaches on

instances with 20, 50, 100, and 500 tasks and solved a largest problem with 1453 tasks and 176 temporal constraints [ZDDD93]. For simulation, Belz modeled a factory with 700 parts and less than 100 work centers [BM96]. Most researchers have studied problems with at most 100 activities. However, project scheduling in industry can easily involve thousands of tasks and tens of thousands of constraints.

2.3 Rescheduling

Project and production scheduling are very important parts of the more general planning process. In some ways, scheduling can be seen as an academic exercise. Project scheduling involves finding the best time and method to accomplish a number of tasks subject to certain constraints. Although it certainly is not easy, the scheduler typically knows the inputs and how to evaluate different schedules based on their merits. As is often the case with textbook problems, however, there are complications in the real world which often add many layers of difficulty to the problem.

Even if a schedule has been formulated, it is rarely the case that the initial schedule will be the one followed throughout the life of the project. Unexpected disturbances create a need for a different schedule. A machine may break down, an item's due date may change, or actual progress may slip behind (or move ahead of) the planned schedule. One method to deal with these problems is rescheduling. A new schedule can address the problems that the old schedule was unable to foresee. In addition, the new schedule can take advantage of some of the opportunities made available in the workplace. This essentially means that rescheduling can more accurately keep track of changes

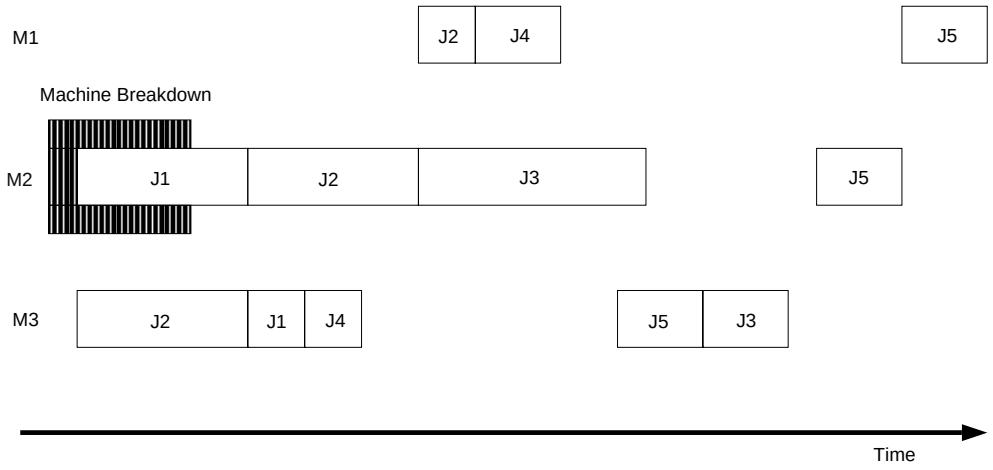

Figure 2.3: An Example of an Infeasible Schedule

in the inputs to the model. This allows for greater control of the project.

When a disturbance causes a schedule to become infeasible, a project manager needs to reschedule the unfinished tasks. There are two fundamental approaches to rescheduling. Opportunistic scheduling creates another schedule by solving the new problem with the new data. Conventional optimization methods and heuristics for project scheduling can be used. However, this ignores the information in the initial schedule. Turnpike rescheduling, on the other hand, alters the existing schedule. Because high-quality feasible schedules can be produced quickly with turnpike scheduling, it is the more common approach. Turnpike scheduling can use simple heuristics, perform complicated searches, or take hierarchical viewpoints.

Many researchers have developed rescheduling algorithms and heuristics [SOS95]. Typically these take an infeasible schedule (usually produced by resource overallocation, as in Figure 2.3) and create a new feasible schedule. Some of the more common heuristics are listed below.

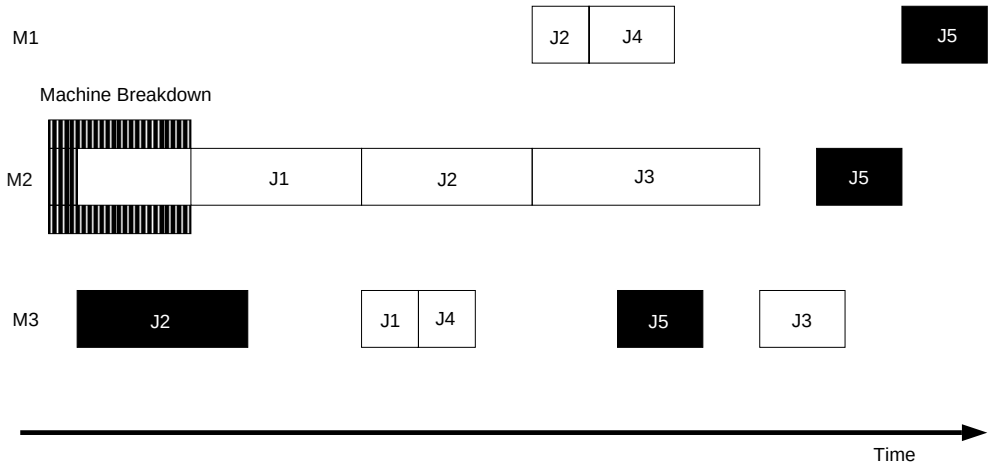

Figure 2.4: Simple "Right Shifting" Feasible Schedule

- Right Shifting — This heuristic delays the start time of those activities that contribute to resource overallocation and then schedules all unfinished tasks as soon as possible. The later tasks can either keep their respective order, or they can be unscheduled to create a new static scheduling problem, with the unscheduled jobs to be scheduled to begin some time after the first overallocation is resolved. Figure 2.4 gives an example.

- Right Shifting and Jumping — This heuristic is similar to Right Shifting. Right Shifting can have a domino effect that delays every job after the initial resource overallocation. Right Shifting and Jumping is more of a potential leapfrog, where some but not necessarily all of the subsequent jobs will be affected. Here again, there is the choice between forward bumping or the creation of a new scheduling problem. Figure 2.5 gives an example.

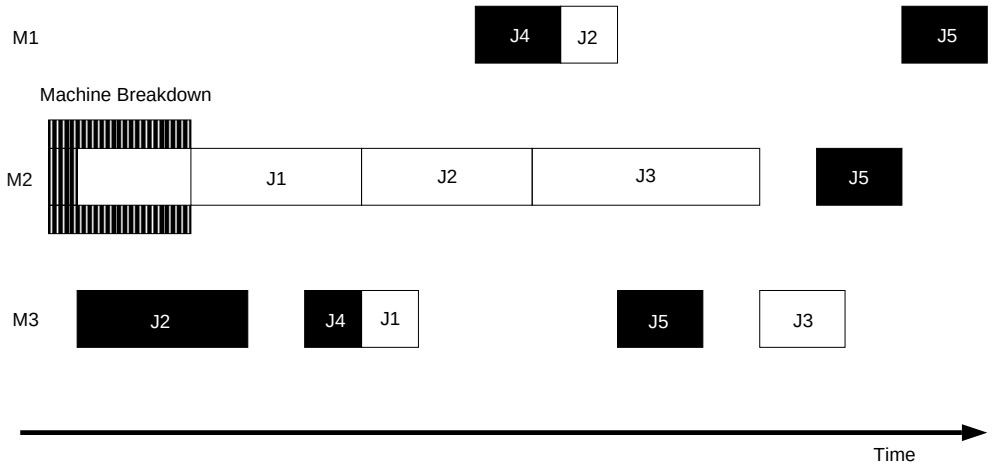

Figure 2.5: "Right Shifting and Jumping" Feasible Schedule

- More complicated rescheduling algorithms can look at unscheduling additional jobs to the ones listed above. The idea is that the improvement in the new schedule will offset the work done. Based on heuristics from Sadeh, job $J2$ on machine $M3$ could be a candidate to unschedule and then add to the set of jobs to reschedule. Taken to the extreme, all of the jobs which have not finished could be unscheduled, and then rescheduled. This entails the largest amount of rescheduling work, but produces the potentially best schedules. This is opportunistic scheduling, where a complete new project scheduling is created.

Rather than using simple heuristics, many researchers perform searches to create new schedules. Here, advances in genetic algorithms [JE97] and simulation [BM96] are helping the scheduling process. Zweben [ZDDD93] improves a complete but possibly flawed schedule by employing constraint-based searches to find infeasibilities and repair each infeasibility by attempting to move tasks. Although this iterative repair is intuitively

appealing, repairing one part of the schedule by moving tasks may create new infeasibilities somewhere else. Some of Zweben's search strategies are:

> "Random Repair: The system randomly selects a task to reassign and then selects a random assignment for that task between its earliest and latest start times.
>
> "Random Constraint Repair: The system behaves identically to the random repair method except that it only repairs tasks with violated constraints. This repair exploits the blame assignment quality of constraint representations because it focuses the repairs on those tasks involved in constraint violations.
>
> "Heuristic Repair: The system repairs ten random constraints per iteration using the heuristic constraint knowledge discussed earlier to generate and select candidate repairs.
>
> "Lookahead Repair: The system uses the same constraint knowledge as the heuristic repair method to generate repairs, but then instead of scoring them, it performs lookahead. It tries each generated repair and selects the one resulting in the lowest cost. This method is a form of the MIN-CONFLICTS heuristic that exploits constraint knowledge to restrict the candidates for lookahead."
>
> [ZDDD93, page 1591]

Another criteria to differentiate between rescheduling approaches is monolithic versus hierarchical. In the monolithic approach, all of the tasks and resources are considered at the same time to find start and end times for each task with appropriate allocations of resources. On the other hand, hierarchical approaches separate the levels of abstraction to control and scheduling, where the control level is responsible for the allocation of resources, and once this is determined, the scheduling level finds start and end times for the tasks. Similar to Zweben's constraint-directed approach is another knowledge-based approach which uses rules in an expert system. This hierarchical approach separates scheduling activities from allocating resources [YS93]. This dual-level management has both resource rules and activity scheduling rules. A similar approach has been developed in the Micro-Boss scheduling program [Sad94], which recognizes disruptions and then proceeds to classify them as simple problems that the scheduling level can handle quickly or as more serious disturbances that require the control level to repair the infeasibility.

2.4 Algorithm Comparison

After developing rescheduling algorithms, a researcher needs to judge their performance based on some criteria. Often the researcher uses the rescheduling algorithms to solve a set of benchmark problems and measures the schedules for each algorithm and each instance. Then the researcher uses statistical measures to draw conclusions. For example, if two rescheduling heuristics are used on each of ten scheduling instances, then there will be twenty makespans. The first set of ten makespans must be compared to the second set of ten makespans to see if there is a difference in performance between the two

heuristics. Many of these theoretical tests, such as t-tests and Wilcoxon signed rank tests, are given in [GS85] and [LK91]. Also, the researcher can analyze the CPU time required to find a solution.

Sadeh [Sad94] compares his algorithm to 39 combinations of dispatching rules. He evaluates the tardiness and inventory costs of the resulting schedules. He randomly creates ten instances for each of eight problem classes, for a total of 80 scheduling problems. Each problem class has either a high or a low value for each of three scheduling parameters. Each problem has 20 jobs and 5 resources for a total of 100 operations. His schedules are 20% less expensive than the ones created by the heuristics. He also compares his algorithm to the actual decisions made by a shop-floor. The hypothetical results of his algorithm performed better than the already-made decisions of the shop-floor. These two examples of analysis illustrate the major ways of analyzing algorithms. Either instances must be created to facilitate a side-by-side comparison between algorithms, or the algorithm's results are compared to what an actual system already did. Both of these comparison are common [ZDDD93, CK88, JE97]. However, many papers in the literature only present qualitative comparisons, such as how easy is it for a human scheduler to use an interactive scheduling system [BM96, PHY$^+$92].

Analyzing a rescheduling algorithm is similar to analyzing any other algorithm. Once a model is made, it must be tested according to some criterion determined by the researcher. These results are published and must be verifiable by others in the same practice. Further standards for reporting computational results can be found in [CDM79].

Measuring the performance of a rescheduling algorithm is not

straightforward. On one hand, the repaired schedule that it produces can be evaluated just as any other schedule. Makespan should be small, net present value should be high, and so forth. However, it is often beneficial if the repaired schedule is similar to the initial schedule. This minimizes changes to the operation of the project and leads to smoother transitions and better results. Finally, rescheduling algorithms should execute quickly, be transportable, and be reliable, like all algorithms.

2.5 Summary

This chapter reviewed literature relevant to the system. It discussed principles of systems engineering and the system development life cycle. The most important step is the requirements development stage. This chapter introduced the project scheduling problem and the project rescheduling problem. Many rescheduling approaches and heuristics have been proposed. The two basic approaches are opportunistic scheduling and turnpike scheduling. Finally, this chapter reviewed methods for comparing algorithms.

Chapter 3

Problem Description

This chapter gives the precise mathematical model of the rescheduling problem. The formulation describes single-mode project scheduling with preemption. Section 3.1 explains the notation and the variables. Section 3.2 then describes the relations between the variables, including the constraints and the objective. Section 3.3 defines disturbances, and Section 3.4 uses this to more fully develop the rescheduling problem. Finally, Section 3.5 summarizes the chapter.

3.1 Notation

The project scheduling problem has a set of jobs (or tasks) that require a set of resources.

Given

- J : the set of jobs
- $d_j \geq 0$: the duration of job j
- R : the set of renewable resources
- $K_r \geq 0$: the number of units of renewable resource r available in each period

- $k_{jr} \geq 0$: the number of units of renewable resource r used by job j each period the job is in process
- P_j : the set of immediate predecessors of job j
- g_j : the ready time of job j. Job j must begin no earlier than g_j.
- h_j : the due date of job j. Job j must finish no later than h_j.

Decision variables (for all $j \in J$, $t = 0, 1, 2, 3, \ldots$):

$$x_{jt} = \begin{cases} 1 & \text{if job } j \text{ is performed during period } t \\ 0 & \text{otherwise} \end{cases}$$

Define : $s(j)$ = the start time of job j

$f(j)$ = the finish time of job j

$\overline{T}$ = the makespan of the project

3.2 Formulation

The following constraints describe the project scheduling problem P:

$$\sum_{j=1}^{J} k_{jr} x_{jt} \leq K_r \qquad \forall r \in R, \forall t \tag{3.1}$$

$$\sum_{t=1}^{\overline{T}} x_{jt} = d_j, \qquad \forall j \in J \tag{3.2}$$

$$f(h) + 1 \leq s(j), \qquad \forall h \in P_j, \forall j \in J \tag{3.3}$$

$$s(j) \geq g_j, \qquad \forall j \in J \tag{3.4}$$

$$f(j) + 1 \leq h_j, \qquad \forall j \in J \tag{3.5}$$

$$s(j) = \min\{t : x_{jt} = 1\} \tag{3.6}$$

$$f(j) = \max\{t : x_{jt} = 1\} \tag{3.7}$$

$$\overline{T} = \max\{f(j)\} \tag{3.8}$$

Equation 3.1 guarantees that no resource is overutilized in any period. Equation 3.2 guarantees that each job is completed. This formulation allows preemption. Equation 3.3 guarantees that each job begins after all of its predecessors complete. Equation 3.4 guarantees that no job begins before its ready time. Equation 3.5 guarantees that each job completes before its due date. Equations 3.6 and 3.7 formally define the start and finish time of a job. Equation 3.8 formally defines the project makespan.

A feasible schedule S is a set of values for all x_{jt} such that Equations 3.1– 3.8 are satisfied.

3.3 Disturbances

Given a feasible schedule S for problem P, a disturbance disrupts the schedule by changing the constraints and adding variables. This creates a new problem

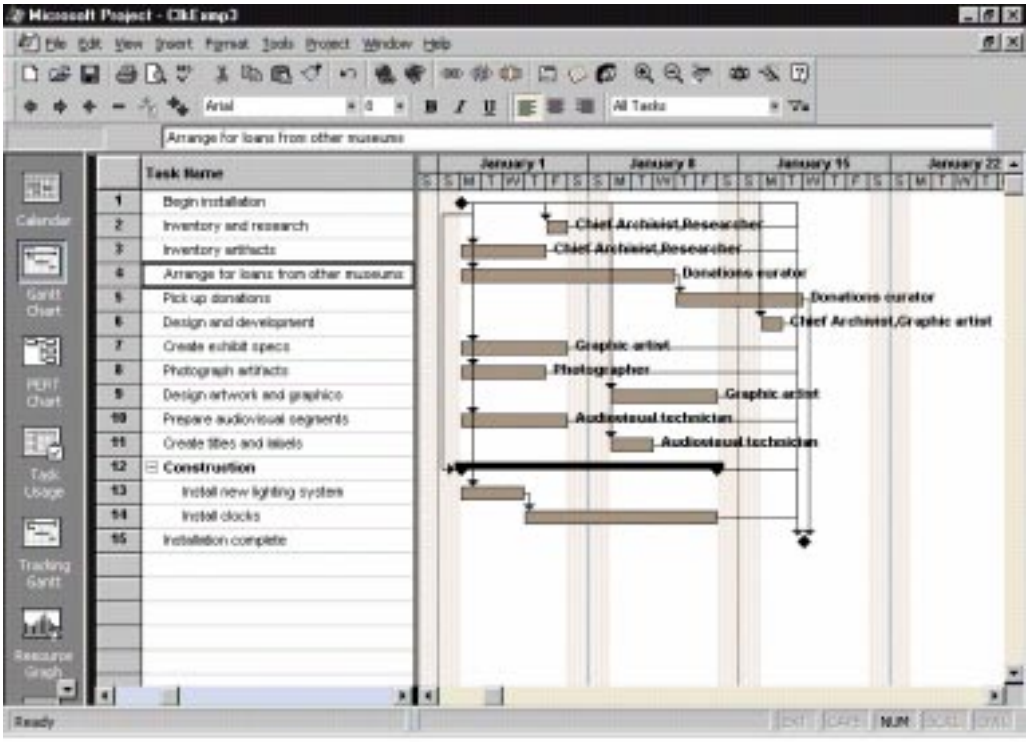

Figure 3.1: Delay Disturbance

P^*. The existing solution may be infeasible. Although there are many possible disturbances, we will consider these three types:

- Delay — A job takes longer than originally expected. This increases that job's duration from d_j to d_j^*. Let $t^* = f(j)$.

  Figure 3.1 gives an example of a delay: the fourth task now requires eight instead of three days (as it did in Figure 2.1).

- New Task — A new task is added to the project at time t^*. The new job set is J^*. The new job j has ready time $g_j = t^*, P_j = \{\}$, resource requirements $k_{jr} \forall r \in R$. The new job may be a predecessor for some jobs n $(s(n) > g_j)$; if so, $P_n^* = P_n \cup \{j\}$.

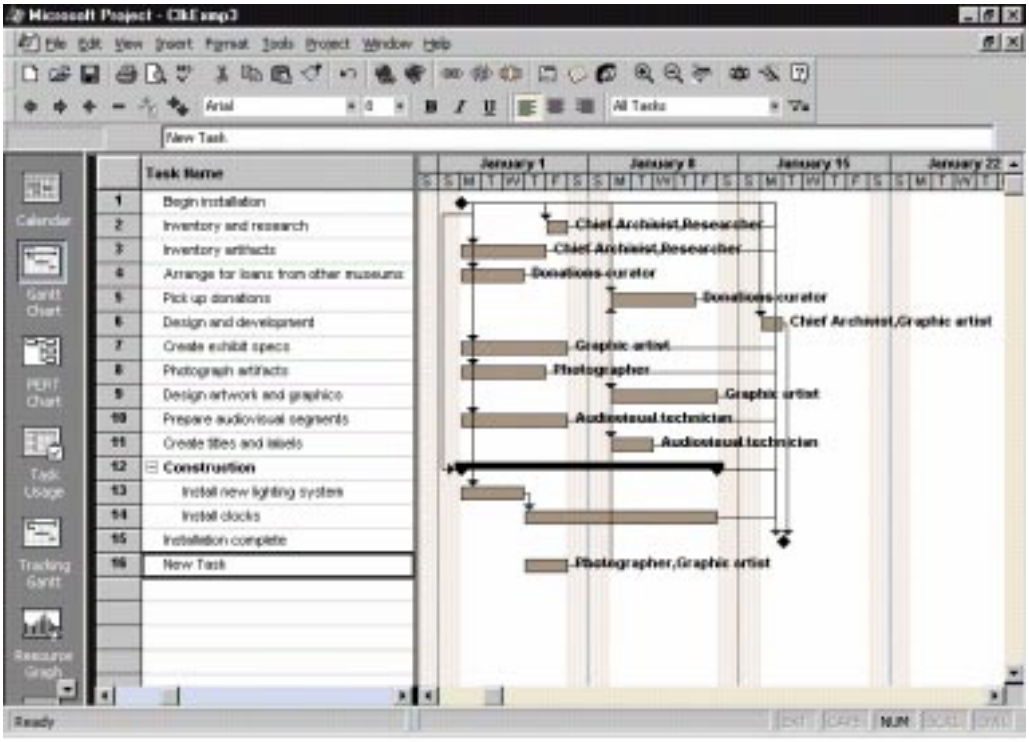

Figure 3.2: New Task Disturbance

Figure 3.2 gives an example of a new task, which is a predecessor for the fifth task.

- Failure — A single unit of resource r^* breaks down at time t^* (the beginning of period t^*) and the repair operation requires b periods. In effect, this changes the job set to J^*. The new job j has ready time $g_j = t^*$, due date $h_j = t^* + b$, duration $d_j = b$. $P_j = \{\}$. It is not a predecessor for any other job.

$$k_{jr} = \begin{cases} 1 & \text{if } r = r^* \\ 0 & \text{otherwise} \end{cases}$$

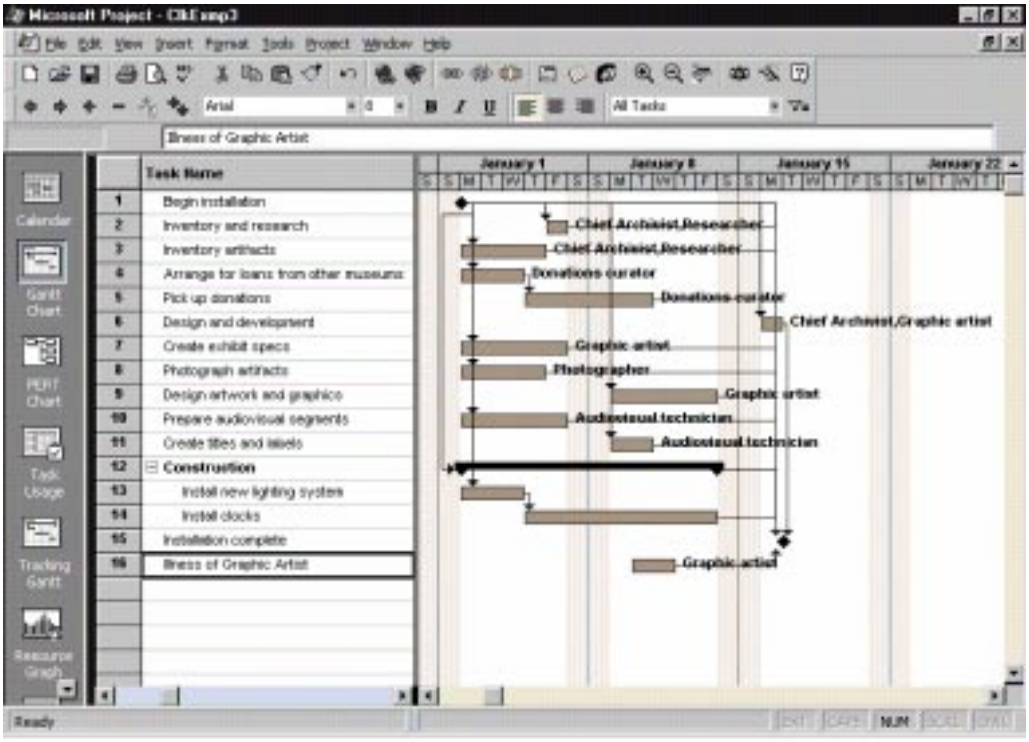

Figure 3.3: Failure Disturbance

Figure 3.3 gives an example of a failure: the sixteenth task represents the artist's illness.

3.4 Rescheduling Problem

The rescheduling problem is to find a repaired schedule S^* for the new problem P^*. We wish to satisfy all constraints of the new problem. This new problem uses the data from the original problem P and the disturbance, which changes the problem data. The new schedule S^* will be the same as the original schedule S for all time $t < t^*$, since the disturbance occurs at t^*. That is,

$$x_{jt} = x^*_{jt} \qquad \forall j \in J, t < t^*$$

There are many objectives for the repaired schedule. The most important objective is feasibility; the new schedule S^* must satisfy all of the constraints. We will measure three other objectives: the schedule's makespan $\overline{T}^*$, the number of jobs disrupted $\sum_{j \in J^*} U^*(j)$, and the total disruption time $\sum_{j \in J^*} D^*(j)$.

$$U^*(j) = \begin{cases} 1 & \text{if } D^*(j) > 0 \\ 0 & \text{if } D^*(j) = 0 \end{cases}$$

$$D^*(j) = \begin{cases} \left|(f^*(j) - d_j^*) - (f(j) - d_j)\right| & \text{if } j \in J \\ \left|(f^*(j) - d_j^*) - t^*\right| & \text{if } j \notin J \end{cases}$$

These last two stability measures evaluate the similarity between the original schedule and the repaired schedule. $D^*(j)$ measures the disruption due to postponing the job or preempting the job.

3.5 Summary

This chapter presented the rescheduling problem. It formulated the project scheduling problem, described three specific disturbances, and discussed the objectives of the rescheduling problem. The test system will use these objectives to compare rescheduling algorithms.

Chapter 4

System Description

This chapter describes the system that we designed and implemented. Section 4.1 discusses the requirements development. Section 4.2 explores different concepts and gives some of the tradeoffs during this phase. Section 4.3 provides details on the components performing the system's first major function, instance creation. Section 4.4 does the same for the system's second major function, schedule analysis. Section 4.5 then describes the integration of individual components. Section 4.6 discusses the system testing. Section 4.7 outlines the major sources of documentation. Finally, Section 4.8 summarizes the chapter.

4.1 Requirements Development

As mentioned already in Section 2.1, it is imperative to discern the system requirements from the user's point of view. What do the buyer, user, and maintainer of the system want it to do? For this project, what would a research scientist want the testing system to do? All of these requirements include primary requirements (what the user explicitly says during brainstorming sessions and initial interviews) and secondary requirements

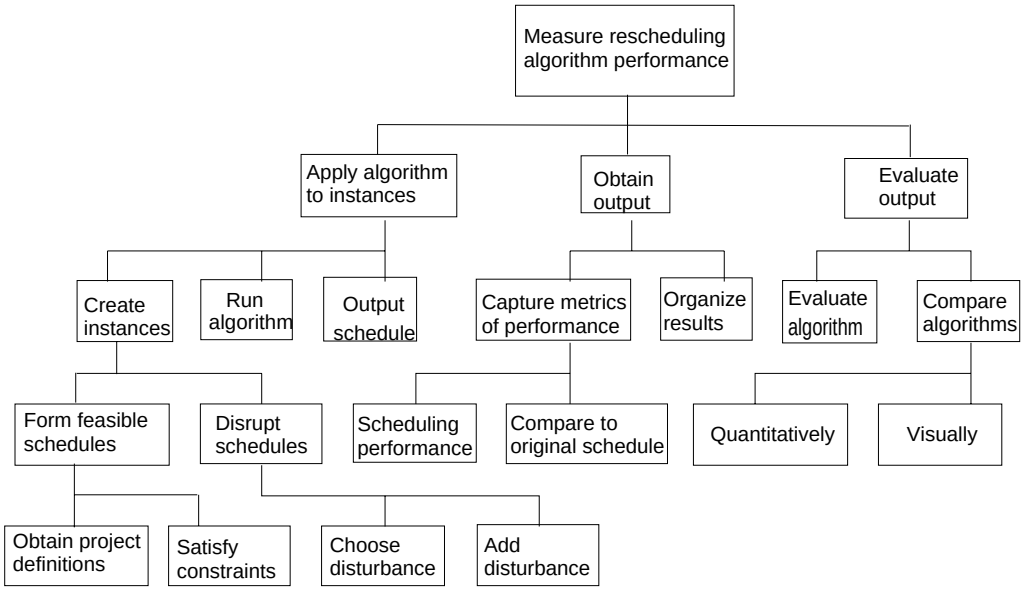

Figure 4.1: Functional Decomposition of User Functions

(what the user implicitly means, but does not explicitly say). Another group of requirements emerges after a system concept has been proposed. These deal with the requirements that the system has to satisfy in order to meet the user requirements.

From brainstorming sessions and researching relevant literature, I assembled the major user requirements for this system. They can be organized as in Figure 4.1, which outlines the major user requirements in a hierarchical fashion.

At the highest level, the system should compare rescheduling algorithms. This is the purpose of the system. We can divide this function into three actions: using the rescheduling algorithm to solve some rescheduling problems, measuring the schedule performance, and evaluating the performance. Each of these actions consists of various tasks, as Figure 4.1 shows.

To accomplish the overall objective of evaluating rescheduling algorithms, the system needs to be able to do the following:

- Recognize an initial project. The system needs to accept inputs of tasks, resources, and their interrelations and then create an initial feasible schedule for this project. This entails individual checks for resource allocation in each period, and the satisfaction of temporal constraints, ready times, and due dates.

- Create a random disturbance and add it to the initial schedule. These disturbances [Kra85] can be categorized as delay, new task, and failure, as in Section 3.3.

- Perform replications. Because researchers cannot draw valid conclusions unless there is a sizable data set, it is important for the system to automatically generate its own problems.

- Monitor and record the performance of each heuristic for each rescheduling instance. Measures of performance include feasibility of the repaired schedule, makespan, number of jobs disrupted, and total disruption time.

The performance of the testing system may be measured by how varied the input can be, how detailed the analysis of the output is (here, the output is the performance of the algorithms. So, how easily can the algorithms be compared?), how stable the system is, and most importantly, how much does it save (time and money) to future researchers attempting to create and test their own algorithms.

For a company to develop a product, it must be worthwhile. Although this statement is almost trivial, this is often difficult to measure. Non-monetary considerations and externalities often don't compute easily in an accounting framework. Furthermore, there are a number of ways to calculate financial situations (including net present value, cost-benefit ratio, and payback period) that give contradictory outcomes on whether to proceed with projects, and which projects are the most attractive. For this project, the expenditures are rather small, while the potential benefits (with an assumed moderate probability of success) outweigh them. The main expenditures are the labor hours of computer programming and a licensing fee for the newest version of Microsoft Project. The potential benefits are increased productivity of all companies using rescheduling.

4.2 Concept Development

The next step in the system development process is the concept development stage. We have the user requirements, and we brainstormed about the different technologies and applications we could use to satisfy these requirements. The process began at an abstract level, and as more and more decisions and trade-offs were made, the plan became more and more detailed. Because the system was novel, we followed a spiral, rather than a waterfall, process. We made incremental changes and continuously modified prototypes, rather than knowing ahead of time the basic structure of the design process.

4.2.1 System Architecture

An important initial step in designing the testing procedure is deciding the basic architecture. I knew most of the requirements of the system. Now I had to concentrate on what the system would be, and how it would satisfy those requirements. I had two major decisions to make. One was which software to use, and the other was where to have the rescheduling algorithm.

The program should be useful to as many researchers as possible. For this reason, I chose to use the Windows operating system, because it is the most common operating system in the world. Unfortunately, no clear-cut similarity existed for a particular software program. Program management software is not as prevalent as word processing or spreadsheet software, nor is there a most popular software program. I wanted the program to not only interact with some project management tool but also some program to store and organize results. I had the option of creating my own project management software, or using one of several existing software packages, such as FastTrack Schedule, Harvard Project Manager, VisiSchedule, or Project Scheduler. Instead, I chose Microsoft Project, which has several advantages. It has built-in capabilities to manage projects and has robust online help. Microsoft's Visual Basic for Applications has an inherent knowledge of its data structures, and Project resembles other programs in the popular Microsoft suite.

Choosing an appropriate concept and associated technology created another large set of requirements. One was that the system should interact with the software applications Microsoft Project 98 for Windows 95 as the front-end for the user and Microsoft Excel for the storage and manipulation of analysis results. The main data exchanges within the system are between these

programs and plain text. The system attempts to minimize the amount of data that the user must enter and interpret so that the user can focus on the results of the testing.

Where will the rescheduling algorithm be? This question deals with the scope of the system. Is the rescheduling algorithm within, or external to, the system? One choice is to include the rescheduling algorithm in the system, while the other choice is to exclude it. There are consequences to each of these alternatives. The first alternative would suggest a Microsoft Visual Basic Application program that interacts solely with Microsoft Project and performs all the functions, including the rescheduling (see Figure 4.2). The second alternative suggests using Microsoft Visual Basic to translate between Microsoft Project and some specified input and output formats that rescheduling algorithms can understand (see Figure 4.3).

The primary advantage of the first approach is that it includes all elements in one integrated program. Each function could be a different subroutine: one for extracting information, another for creating disturbances, and so on. There would be no wasted effort translating data or switching between applications. In contrast, the second approach would have intermediate file formats. Running the rescheduling algorithm would require the user to switch to another application. Advantage: Approach #1.

Approach #1 requires that the rescheduling algorithm be a Visual Basic program. In the second approach, the researcher is free to use any modern computer programming language to code the rescheduling algorithm. Because of this flexibility, the second approach may be more practical for a wider variety of researchers. However, this added flexibility comes at a cost of increasing the

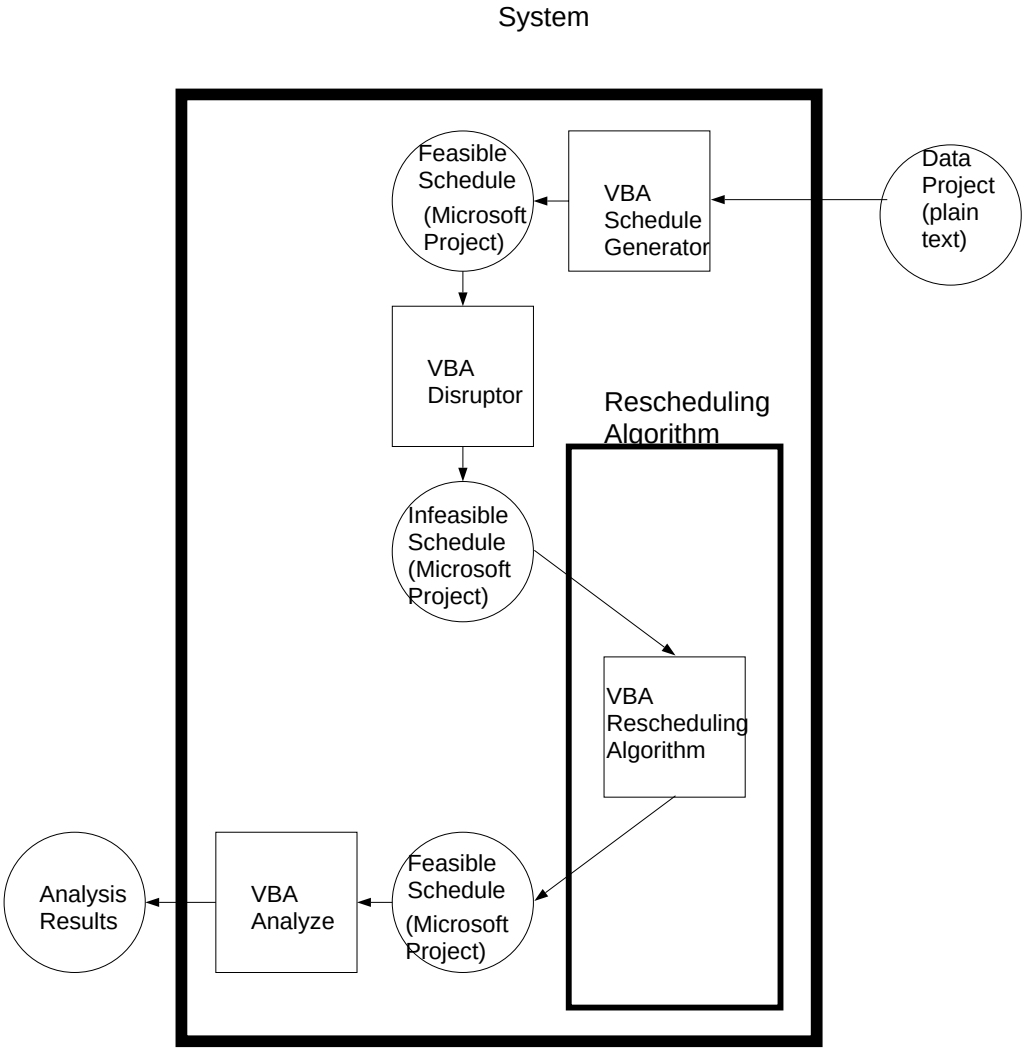

Figure 4.2: One Concept for the Architecture (VBA = Visual Basic for Applications)

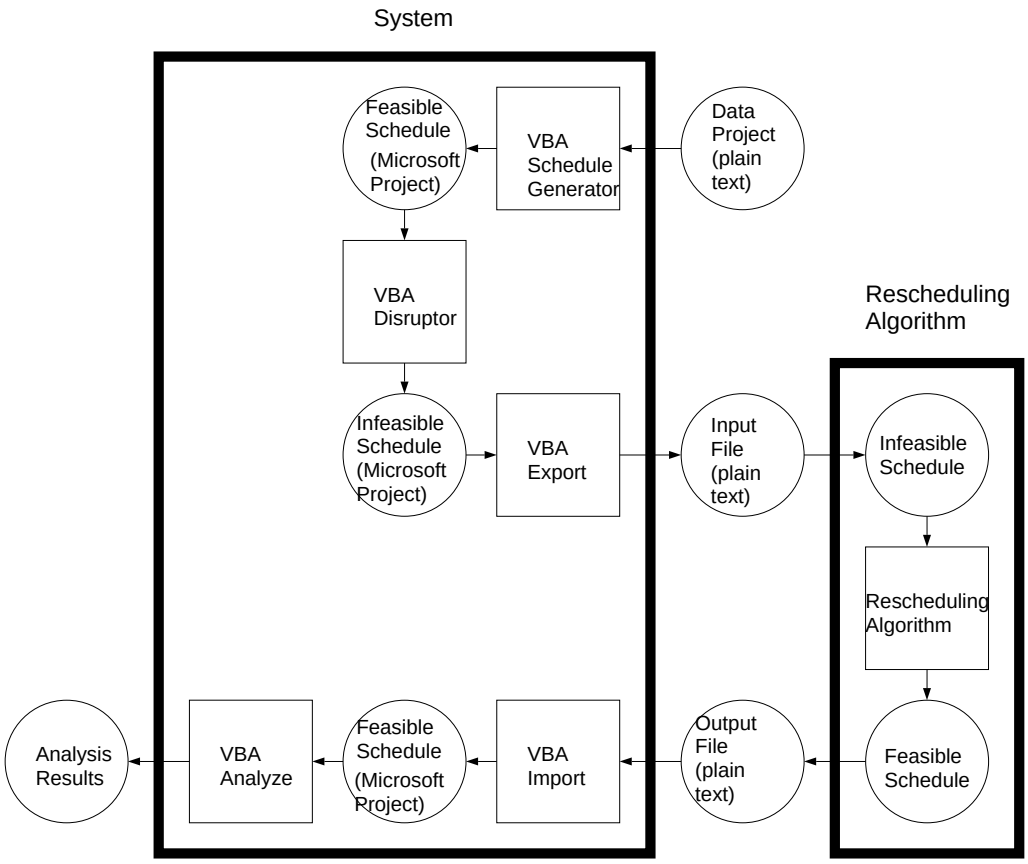

Figure 4.3: Another Concept for the Architecture

complexity of the system: having translators. Advantage: Approach #2.

What other advantages and disadvantages do these two approaches have?

The testing system will lead, we hope, to a standard for testing rescheduling algorithms. Increased use of standards promotes efficiency and facilitates collaboration between different teams and projects. The second approach supports this goal because it requires standards for the input and output file formats. In looking for sample problems, instances, and schedules, I came across text files that are formatted to easily describe the activities and resources. This is certainly a convenient and useful way to store and transmit information. Standardizing file formats facilitates communication between different groups and teams, an important systems engineering objective.

The choice of architecture will greatly influence the rescheduling algorithms that can be tested. In the first approach, the system can test only those algorithms that can be coded in Visual Basic. Coding algorithms in Visual Basic may be a difficult task. The second approach is more flexible, since it can compare any rescheduling algorithm, regardless of its complexity or its computer programming language. Researchers can choose whatever language is easiest for them.

The architecture influences the system's ability to compare computational effort. In the first approach, the system can run the rescheduling algorithm and measure its computation time directly. However, in the second approach, the system does not run the rescheduling algorithm, and it cannot measure the effort expended by the algorithm.

These results are summarized in Table 4.1.

Criteria	Approach 1	Approach 2
Integration	+	-
Flexibility	-	+
Ease of programming	-	+
Standardization	-	+
Facilitation of analysis	○	○

Table 4.1: Comparison of Approaches Regarding Rescheduling Algorithm

After careful consideration, I selected the second approach. The researcher can use any computer programming language he wants, and the system exchanges information using standard file formats to facilitate this.

The entire testing system is a macro written in Microsoft Visual Basic for Applications associated with a project in Microsoft Project for Windows. The macro calls individual subroutines that perform definite, specific functions and that are easily translatable into requirements of either the primary overall user requirements, secondary requirements that are derived from the user's implicit meaning or the particular choice of concept, and helper subroutines that perform smaller functions needed by the other subroutines. The user of the system sees only a simple interface when the system is running, but the details of any of the routines can be viewed in full by using the VBA editor. In this way, the user has the capability to not only better understand the system, but add and modify it at his discretion.

The subroutines work together to perform as many of the functions efficiently and as automatically as possible, to allow for the researcher to

concentrate on the rescheduling algorithms and their associated results, rather than on the testing procedures and protocols. Most of the data management and computations are labor-intensive and repetitive, and are therefore better suited for computers. Again, there is the flexibility to run individual subroutines. Each critical subroutine corresponds to a box in Figure 4.3. Each box can be thought of as a task in the progression through different states in the testing process. These tasks can be labeled as Reading In, Creating, Disrupting, Writing Out, and Analyzing.

4.2.2 Maintainability

I addressed issues of maintainability during the development of the computer program. It took several months to code the program, and it was important to keep track of progress and document the work I had done. When researchers begin to work with the program, I want them to understand how the program works, and modify it as they see fit. For these reasons, there were several conventions that I adhered to.

The system has documented code. Each subroutine is labeled and a brief summary describes its function and origin. Also, during some of the longer portions of code, individual lines or groups of lines are documented.

Rather than having variables of x and y floating around, the system adheres to a naming convention for all variables and routines. In this way, the user can easily see the scope and type of each variable, in addition to what sort of information it contains. This makes it almost possible to read the lines of code and know instinctively what is being done. Many of the suggestion for this came from programming style books, such as [War95, Cor96, PVTM96, Cor95].

There are several points during the execution of the program that a subroutines calls a random number generator. Sometimes this is needed for integers, other times for real numbers, and still other times for dates. The concept is so similar, however, that the system combines all of them into one large global routine. In this way, if the user needs to change something, he can change it once, and have its effect felt everywhere. This is similar to hardcoding as few directory and file names as possible.

4.2.3 Baselines and Interim Plans

For future analysis, it would be best to save as many snapshots of the schedules during the testing process as possible. For this, the system uses the features of Microsoft Project dealing with baselines and interim schedules. This should provide enough detail to accurately make claims based on the data of different schedules at different reference points.

The BaselineSave function of Visual Basic helps here. With it, the system saves the current starting and ending times. When appropriately saved into BaselineStart and BaselineFinish, or Start1 and Start2, at the right time, the system can create snapshots of the schedule during the entire scheduling and rescheduling process. Currently, the system has four snapshots of interest: a first schedule satisfying precedence relations, a first feasible schedule satisfying both precedence and resource constraints, a disturbed schedule, and a repaired schedule. Project allows up to twelve schedules on the Project Gantt chart, but only allows 10 date fields for saving. If absolutely necessary, the user could probably save more by putting the information in the Notes fields or other miscellaneous fields. When viewed concurrently the user should be able

to understand every step of the process, and retrieve the information necessary for data analysis.

Special care had to be taken to view the information. Milestones (any tasks with duration equal to zero) are handled a little differently than tasks with strictly positive duration. Also, there are different styles to illustrate how the schedule evolved over time. From work done in human factors and industrial psychology, we know that how the information is presented has direct bearing on how much of the information the user can absorb effectively. The system mainly uses primary colors to highlight differences. The times of the tasks during the baseline phase were colored grey. The times of the tasks during the initial feasible phase were yellow, and so on. In this manner, the user may look at the schedule during different phases by focusing on certain colors, or the change of each task by focusing on one group of colors.

4.3 System Components — Instance Creation

The test system consists of many Visual Basic subroutines. This section describes the subroutines that perform functions related to instance creation, as shown in Figure 4.4. Each subsection describes a subroutine's input, output, and purpose. Section 4.4 describes the subroutines related to schedule analysis. Section 4.5 describes the integration of the subroutines.

4.3.1 Reading In

Input: Text file of project description

Output: Microsoft Project project file

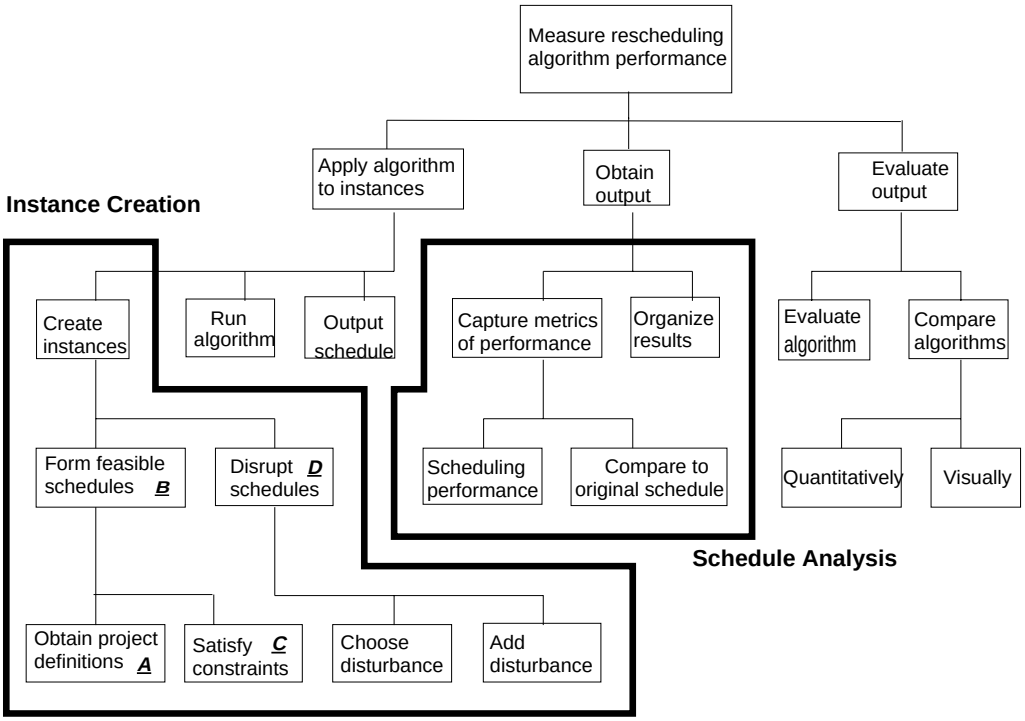

Figure 4.4: Functional Decomposition of User Functions

This subroutine, which corresponds to the function labeled A in Figure 4.4, translates a text file into a Microsoft Project project. Typical warehouses of data sets for project scheduling instances contain all of the necessary information about a project in a minimum amount of storage. For simple resource-constrained project scheduling problems, this entails the number of tasks and resources, the available capacity of each resource, the resource requirements of each task, and the precedence relations among tasks. This format, or some simple derivative of it, is what is used in J. H. Patterson's 110 test problems, a benchmark used by many researchers. The Reading In subroutine uses a derivative of this: For example, consider the text file in Figure 4.5. The first line describes the number of tasks and resources. The next line lists the capacities of each resource. Each following line describes a single task: its duration, resource usage, number of successors, and successors. Most projects can be described in this format. It does not allow, however, for financial considerations, multi-modes, or other temporal constraints. This is justifiable in the cases where monetary matters are dominated by the efficiency of the schedule, each task has only one mode (which is the maximum allowable by Microsoft Project) and the only temporal constraints are simple finish-start relations (which are the most common ones).

Given the text file's name, the subroutine reads the text file and uses Visual Basic commands to establish the resources, create tasks, define precedence constraints, and save this information as a Microsoft Project project file. The schedule at this point looks like Figure 4.6. Note that this schedule already satisfies the precedence constraints. However, the schedule may not satisfy the resource constraints.

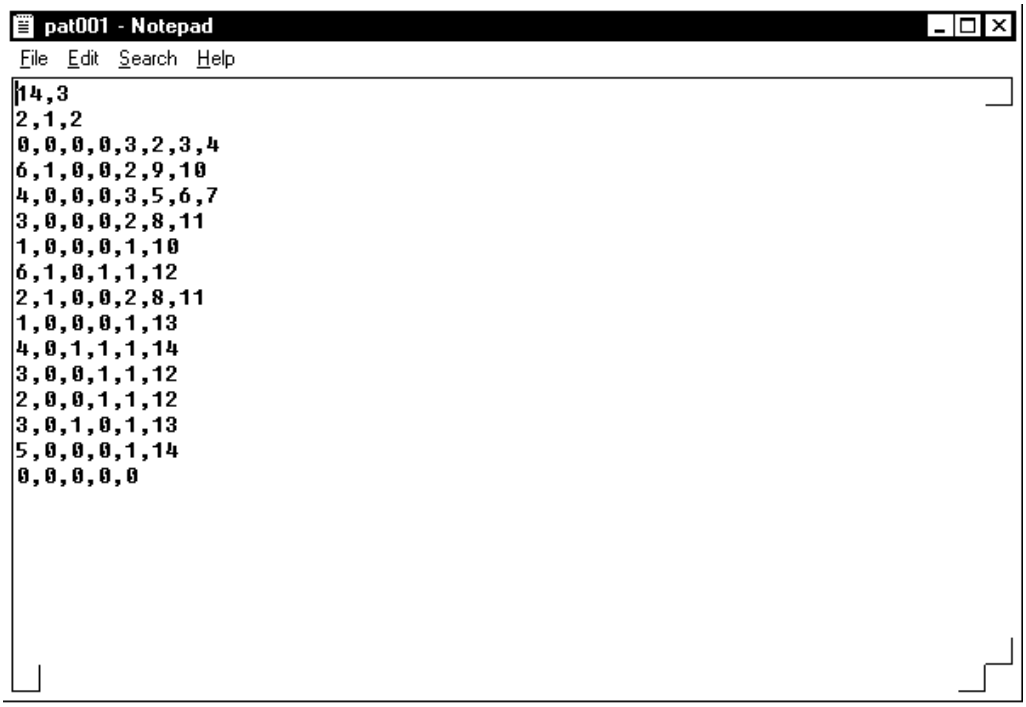

Figure 4.5: Plain Text Project File

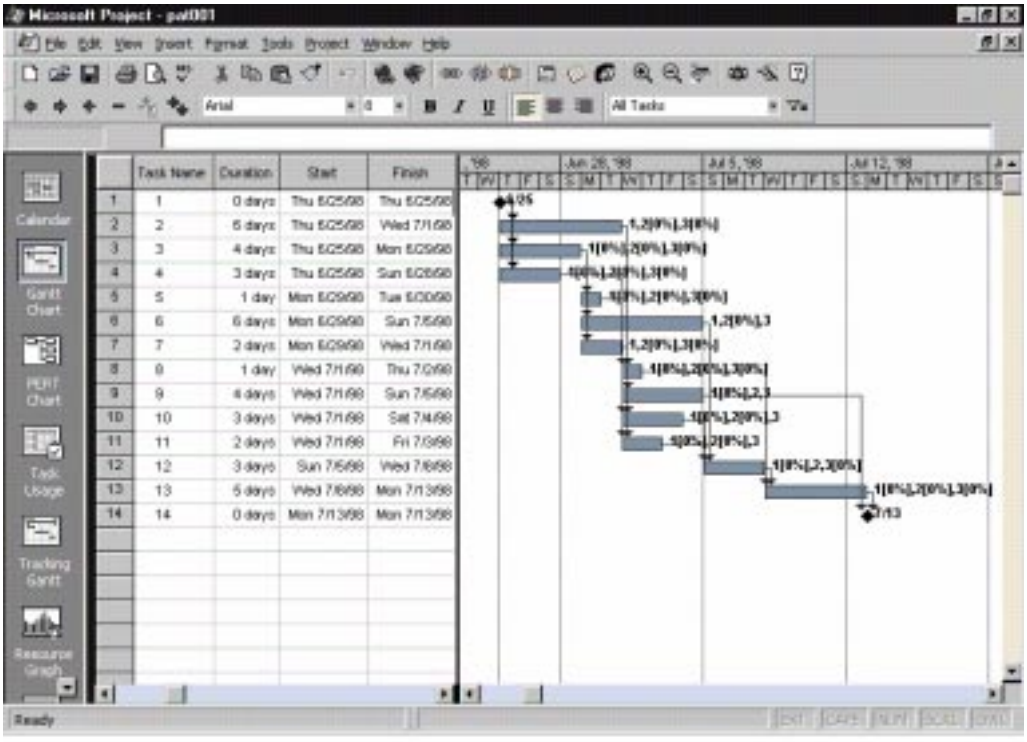

Figure 4.6: Initial Schedule Immediately After Reading

The subroutine has a few safeguards. If the text file does not exist in the location specified, or the information is not ordered correctly, an error message appears on the computer screen stating that an error has been encountered. Though not elegant, this is an attempt to alert the computer user that any project formulation that follows has the possibility of being corrupt. The system's error messages are as useful as possible to the user, and are not condescending or insulting to the user. These messages should aid in the learning of the user, not discourage him from using the system. These can be roughly labeled as "friendly error messages." These error messages appear on the screen and state in which subroutine the program was running when an error was found.

4.3.2 Creating

Input: Resource-infeasible Microsoft Project project file

Output: Resource-feasible Microsoft Project project file

This subroutine, which corresponds to the function labeled B in Figure 4.4, checks to see if any resources are overallocated. If so, the subroutine reschedules tasks so that no resource is overutilized in any time period. This resource-feasible schedule is the first milestone of the test system. An example of such a schedule is shown in Figure 4.7. Associated with each task are two bars. The bottom bar was the task in the initial schedule. The top bar is the task in the new schedule. For instance, note that forming the feasible schedule shifted Task 6 by two days.

The subroutine forms the resource-feasible schedule by a simple

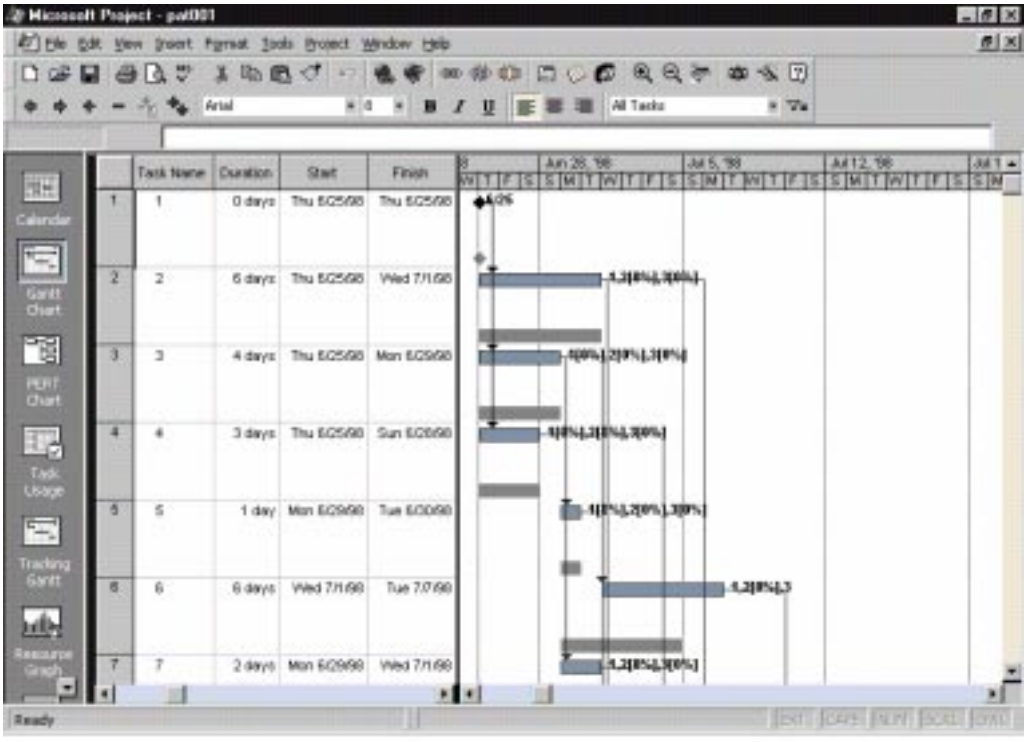

Figure 4.7: First Feasible Schedule

right-shift of the tasks. The subroutine chooses which tasks to shift simply by the order in which they are listed. As many tasks as possible remain where they are. Proceeding down the list, the first task listed that causes a resource to be overallocated and all other tasks involved in that resource overallocation are shifted, and this is continued until all overallocations are remedied. Of course, shifting successor tasks is necessary to satisfy the precedence constraints. Because it uses this simple heuristic, this subroutine obviously does not create "optimal" resource-feasible schedules. It is not clear how forming the initial feasible schedule affects testing rescheduling algorithms. In the literature, certain rescheduling algorithms and schemes have performed better than others based on the optimality of the initial feasible schedules before disruption. The method that forms the initial feasible schedule must be carefully considered. Of course, the user has the ability to change this method by changing a few lines of code in the program, which is not difficult.

4.3.3 Checking Feasibility

Input: Microsoft Project project file

Output: Statement of feasibility

This subroutine, which corresponds to the function labeled C in Figure 4.4, checks the feasibility of a schedule. It checks resource capacity constraints and task precedence constraints. This subroutine verifies both the initial schedule and the resource-feasible schedule. If either schedule is not feasible, an error message declares why the schedule failed.

4.3.4 Disrupting

Input: Feasible Microsoft Project project file

Output: Microsoft Project project file with disturbance

This subroutine, which corresponds to the function labeled D in Figure 4.4, adds a disturbance to a schedule. The interesting disruptions are the ones that are most likely to cause a feasible schedule to become infeasible.

In practice, there are a large number of disruptions that cause infeasibilities. Most of these are equivalent to adding a task to a project. This new task can be a real added task, model the effects of downtime for a damaged resource, or simulate the effects of a task taking longer than its expected duration. Other disturbances are rare, such as precedence relations changing, or do not cause infeasibilities, such as tasks being deleted from a schedule, although the latter does open up the opportunity to improve schedules through the use of rescheduling.

The specifics of how a disturbance is randomly created are important. For adding a task to a project's definition, they should deal with how the duration of the task is determined (based on existing tasks' durations), the predecessor-successor relationships, and the number of added tasks.

The subroutine can create three types of disturbances: delays, new tasks, and failures. The subroutine begins by randomly selecting which type of disturbance to create. Each type is equally likely.

If the disturbance is a delay, the subroutine randomly selects a task that has a positive duration. All such tasks are equally likely. Then the subroutine randomly chooses the delay from a uniform distribution. The minimum

possible value is zero. The maximum possible value is the task's current duration. It then extends the task's duration to include this delay.

If the disturbance is a new task, the subroutine adds a new task to the schedule. The subroutine randomly chooses a time t^* between the project's start time and finish time. The new task cannot start before t^*. For each resource, the subroutine randomly chooses the task's requirements. The minimum value is the smallest requirement of any existing task. The maximum value is the largest requirement of any existing task. The subroutine randomly chooses the new task's successors. Only tasks that have not yet started by t^* are candidates. The probability of a task being a successor equals the average number of successors divided by the number of tasks.

If the disturbance is a failure, the subroutine adds a new task to the project. The subroutine randomly chooses a time t^*. The new task must begin at t^*. The subroutine chooses the failure duration from a uniform distribution. The minimum possible value is the shortest task's duration. The maximum possible duration is the longest task's duration. The subroutine randomly chooses one resource, and the new task requires one unit of that resource. The new task has no successors.

Figure 4.8 shows an example of a disturbed schedule. Note that each task now has three bars. The top bar corresponds to the disturbed schedule and the two bottom bars correspond to the previous schedules. Figure 4.9 shows the delayed task, task number 13, whose duration is 0.27 days longer than originally anticipated. Note that the disturbed schedule does satisfy precedence constraints. Figure 4.10 shows the progress of the schedule by adding black horizontal bars for completed portions. This helps the user find

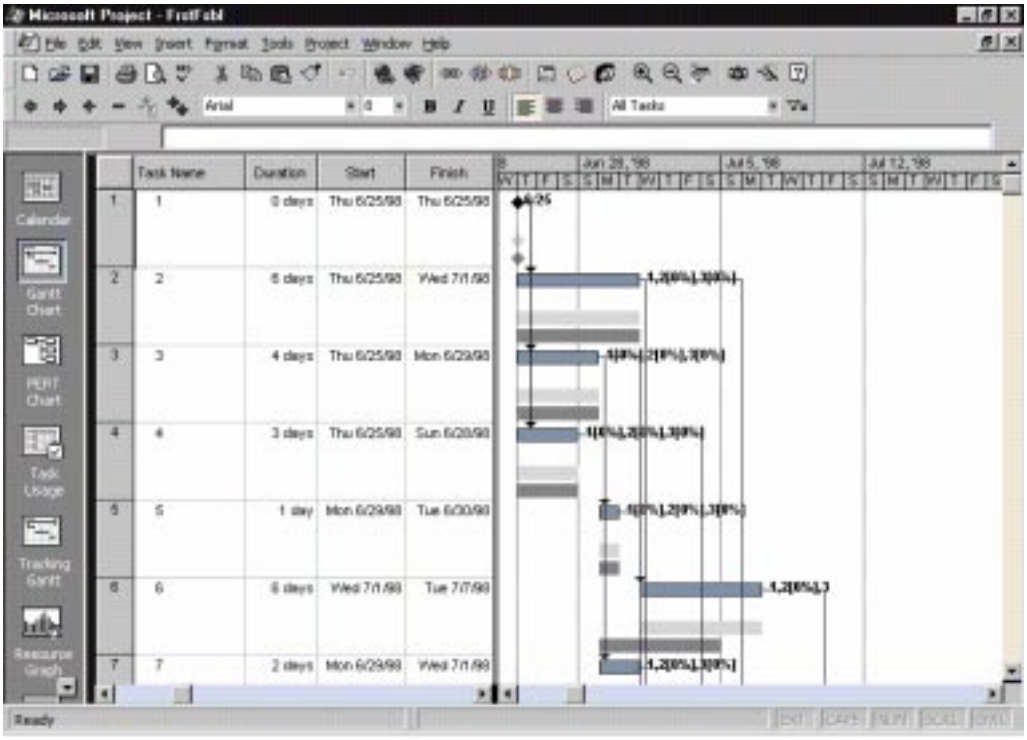

Figure 4.8: Disturbed Schedule

the disturbance time t^*.

4.3.5 Writing Out

Input: Microsoft Project project file with disturbance

Output: Text file

This is approximately the opposite of the Reading In subroutine. This routine translates a Microsoft Project project into a plain text file that a rescheduling algorithm can use to create a new, repaired schedule.

A significant difference between this subroutine and the Reading In subroutine is that this text file must contain the schedule (the task start times)

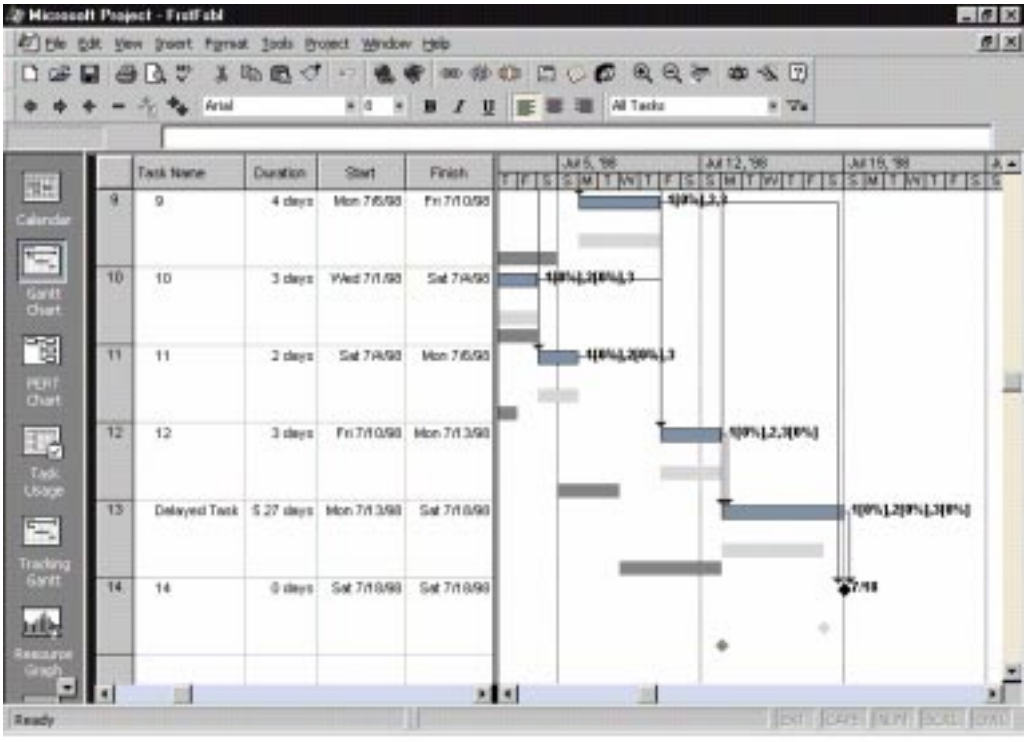

Figure 4.9: Disturbed Schedule

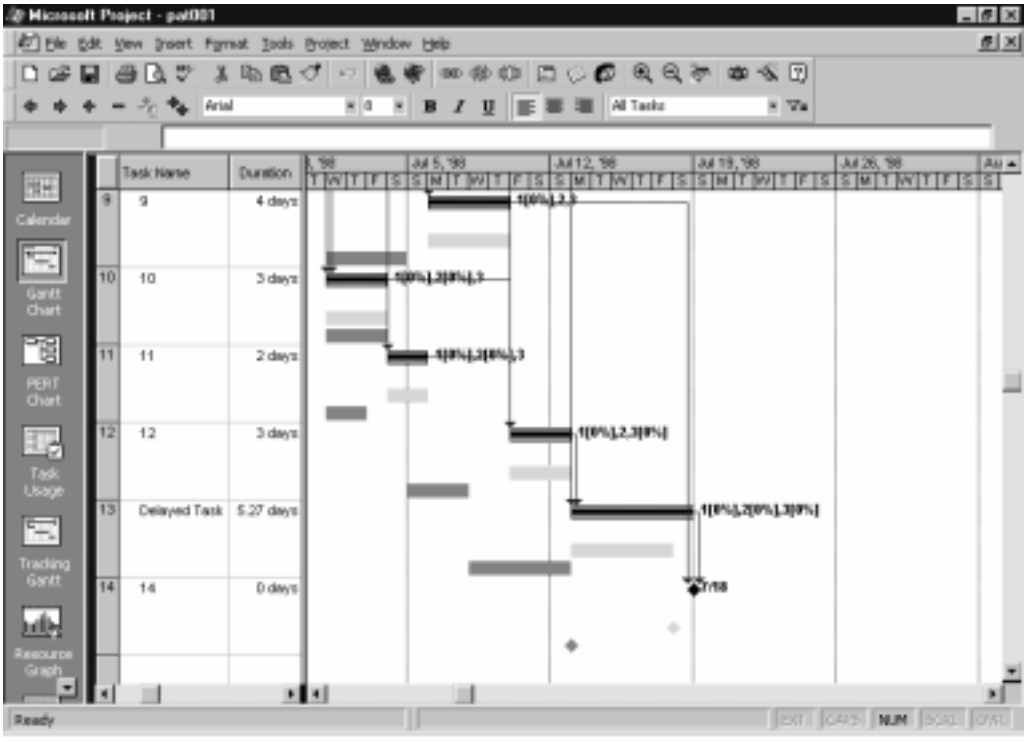

Figure 4.10: Disturbed Schedule with Progress Lines

and not just the project specifications. When designing this subroutine, we faced a choice between two approaches.

How Microsoft Project interacts with the outside rescheduling algorithms during this middle step of the testing process is important. To this point, only the project definitions have been captured in the initial text file. The actual schedule, including starting and ending dates, are still in Microsoft Project format. Hence, we are posed with the two contenders with how to deal with this quandary.

Should the subroutine combine all the information about the project and the schedule and create one large text file, or should it create a file that contains the schedule information and describes how the original project has changed? With the original project information, this file would depict the complete new scenario. Each alternative has its advantages and disadvantages.

- If the project is large, the information about schedules can be quite substantial. Duplicating this data in the new text file may pose a burden to the storage capacity of some environments. For this reason, it is preferable to have a separate, smaller file solely for the schedule.

- During the testing process, it may be necessary to test many instances of problems, with many possible disturbances. It is not clear, however, which alternative requires less time. Creating the larger, single file increases the amount of information that needs to be transmitted. On the other hand, if the separate file is used, then an additional file must be opened and processed to gather the complete information.

- According to Figure 4.3, this file is the system's link to researchers. Their

rescheduling program uses as an input this file(s) and then creates a file (or files) for the system to analyze. A single file is a simpler interface.

It therefore appears that the two methods are quite similar in overall goodness. It is imperative to always keep in mind the system users when designing the system. "Know thy user." For this reason, the subroutine creates the single larger file that contains both the project definition and the schedule information. Note that this decision is an example of the spiral development process, since we're doing concept selection during subsystem design.

The researcher does not care about the names of the tasks or the baseline history. Therefore, there is no need to communicate such information through this intermediate text file. However, the researcher cares very much about the project definition and the disturbed schedule. The researcher also needs to know the time t^* when the disruption occurs. This signals the breakpoint between the fixed part of the schedule and the tasks that can be moved. An example of the text file is given in Figure 4.11. The first line in the text file gives the name of the project and the second line gives the creation date of the project. The third and fourth lines are the disturbance time t^* and the rescheduling algorithm's name, respectively. The remainder of the file is similar to the original text file, except that it changes some values for some tasks to show the disturbance. Note that the duration of task 13 is now 5.27 rather than 5 days. Also, each task has a start time and constraint type, given after the list of successors. The number 2 represents a "Must Start On" constraint, 0 represents an "As Soon As Possible" constraint, and 4 represents a "Start No Earlier Than" constraint.

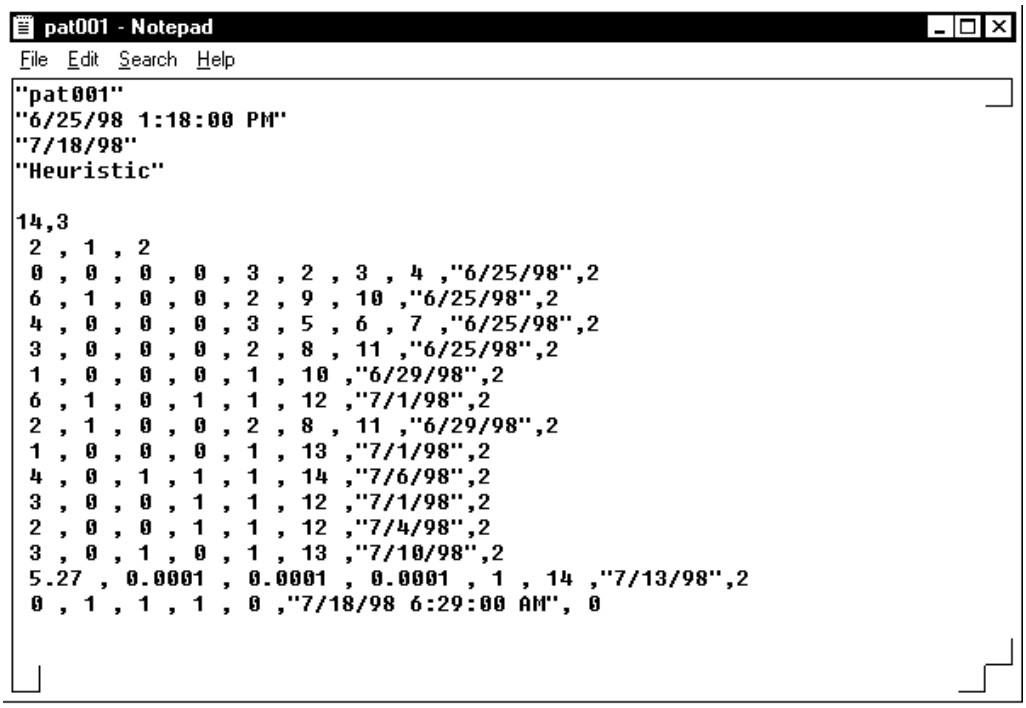

Figure 4.11: Plain Text Output After Disturbance

4.4 System Components — Schedule Analysis

The second major function of the test system is to analyze repaired schedules. This can be done once the instances from the first major function as described above are repaired by a researcher's rescheduling algorithm and output into appropriately formatted plain text files. Schedule Analysis performs everything in the middle large box of Figure 4.4.

4.4.1 Reading Back in a Schedule

Input: Text file

Output: Microsoft Project project file

This subroutine is similar in principle to the reading in subroutine from Instance Creation. However, this subroutine needs to address a few additional matters. The subroutine must identify the rescheduling algorithm's name. In addition to reading in the project specification data, the subroutine must also read in the schedule, including start times. Not only does this subroutine call the feasibility checker to check the schedule, but it includes checks to ensure that the repaired schedule satisfies the project constraints. See the repaired schedule in Figure 4.12. Associated with each task are four bars. The top bar is the user's repaired schedule. The bottom three bars for each task are the three previous schedules as in the disturbed schedule.

This subroutine must understand a repaired schedule that has preempted tasks. Preemption may be necessary to repair a schedule, especially when a breakdown has occurred. The rescheduling algorithm may preempt any task. However, the system requires that the repaired schedule follow a specified

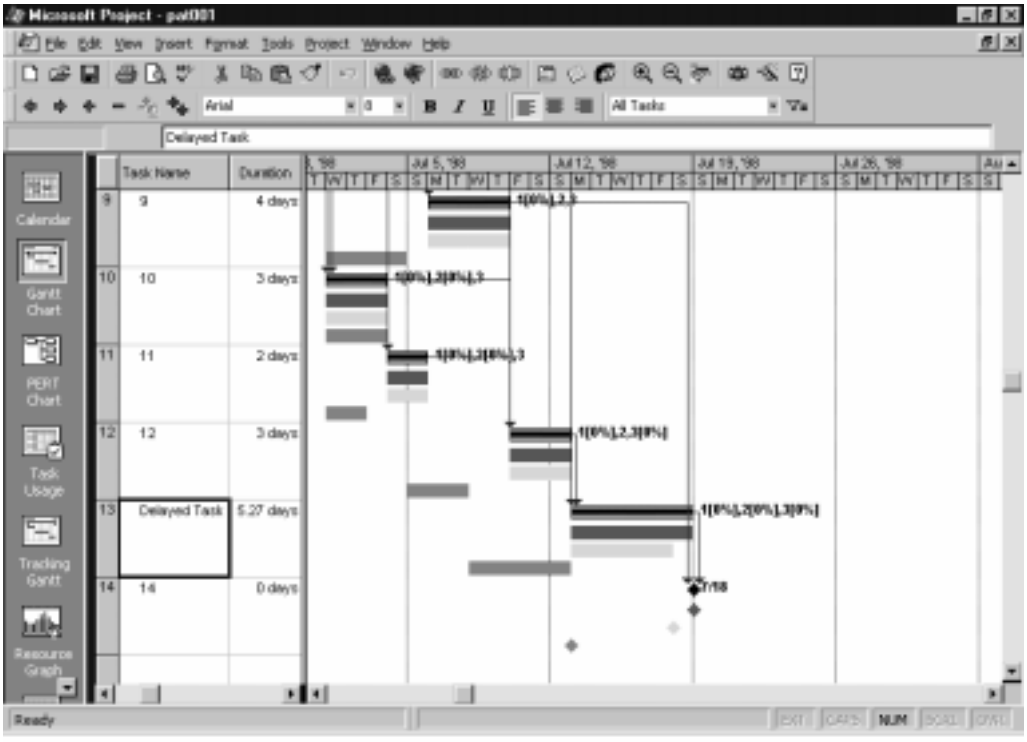

Figure 4.12: Repaired Schedule

format so that it can decipher the new schedule. Any task in the disrupted schedule may be preempted at most once and only if it has not completed by the disturbance time. Breakdown tasks cannot be preempted. The order of the original tasks listed in the disrupted schedule must remain the same. The first new task after the original list must be the second half of the first preempted task. The second additional task must be the second half of the second preempted task. And so forth.

A preempted task's sole successor must be its second half, whose successors are the preempted task's original successors. The resource requirements per time period remain the same. The sum of the durations must equal the original duration. A preempted task counts once as a task being rescheduled, and its disruption time equals the time between the preempted task's completion and the beginning of the second half.

4.4.2 Analyzing

Input: Microsoft Project project file

Output: Text file and Microsoft Excel workbook

This subroutine performs the calculations to compare the repaired schedule to the disrupted schedule. The information comes primarily from the different baselines and interim schedules created along the way. The system captures simple measures of scheduling (makespan) and stability (how much has the schedule changed from before to after the disruption because of repair). The subroutine creates a plain text file of performance, as in Figure 4.13. For each task, the disruption time is calculated as given in Section 3.4. For most

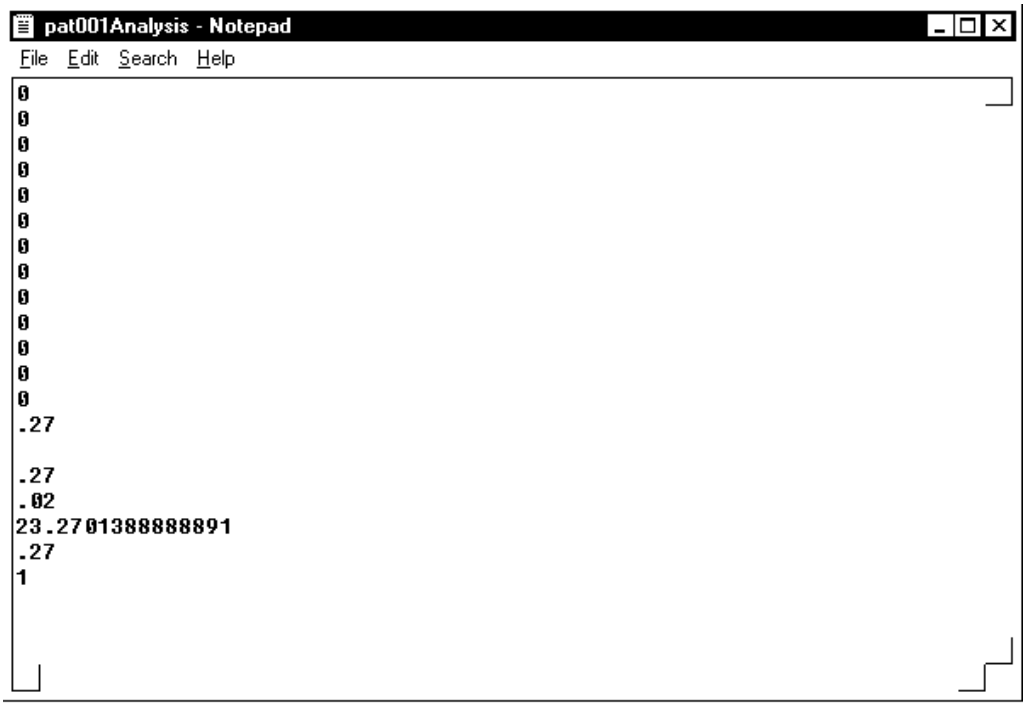

Figure 4.13: Plain Text Output After Analysis

non-preempted tasks, this will be the difference between the task's top two bars. For instance, the disruption time for task 14 is 0.27 days because its predecessor (Task 13) is delayed. The last five lines in the text file are the total disruption time which equals the sum of the disruption times for all tasks, the mean disruption time which is the average disruption time per task, the project makespan which is the difference between the project start and project finish, the largest disruption time for a single task, and the number of tasks rescheduled. In addition, the subroutine adds the metrics to a Microsoft Excel spreadsheet, as shown in Figure 4.14, where each of the first three sheets displays in tabular form a particular metric for each algorithm on each instance.

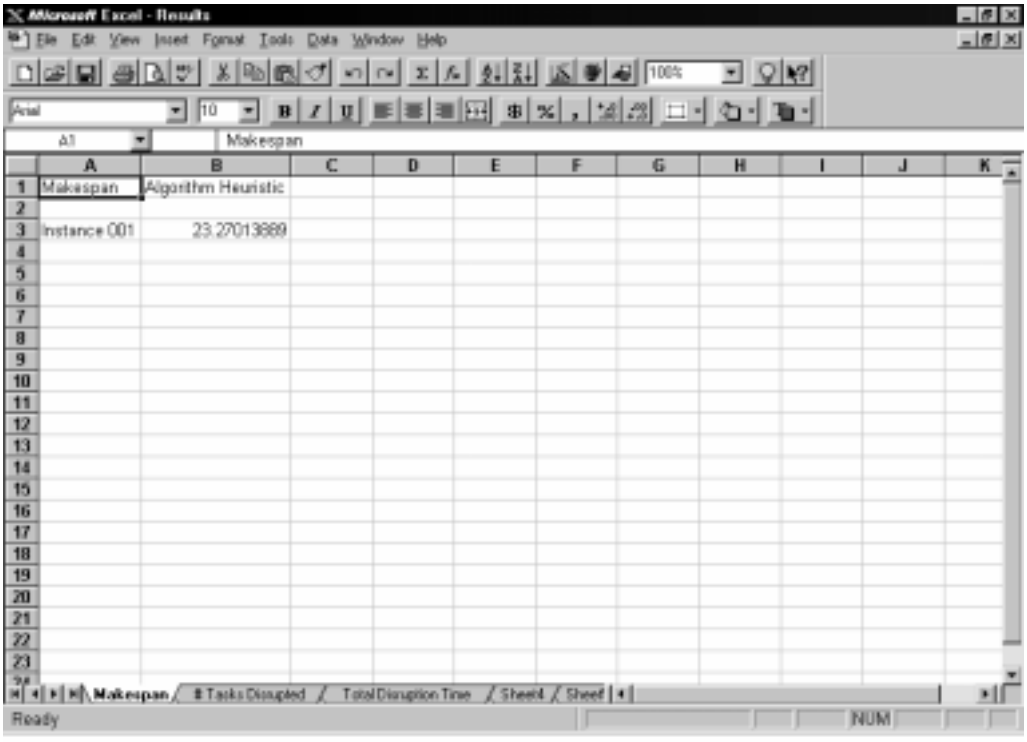

Figure 4.14: Spreadsheet of Results

4.5 System Integration

Once I had individual subroutines running correctly, I needed to make sure that they worked well together. This marked a transition in the process from subsystem design to system integration. This section describes the Instance Creation routine, the Schedule Analysis routine, the Main Menu, and the implementation.

4.5.1 Instance Creation

Input: Text files of project descriptions

Output: Microsoft Project project files and text files of disrupted schedules

The Instance Creation routine, which corresponds to the functions in the left heavy outline in Figure 4.4, creates a set of rescheduling problem instances. First, the user specifies a set of project descriptions. Then, for each project description, the routine performs the Reading In, Creating, Checking Feasibility, Disrupting, and Writing Out subroutines, which correspond to the functions labeled A through D in Figure 4.4. These create, for each project, a Microsoft Project file and a text file with the disrupted schedule. After the routine finishes, the user has a set of rescheduling problem instances.

4.5.2 Schedule Analysis

Input: Microsoft Project project files and text files with repaired schedules

Output: A Microsoft Excel spreadsheet and text files with results

The Schedule Analysis routine, which corresponds to the functions in the right heavy outline in Figure 4.4, measures the performance of a rescheduling algorithm. First, the user specifies a set of repaired schedules and an Excel spreadsheet for the results. For each repaired schedule, this routine performs the Reading Back In and Analyzing subroutines, which correspond to the functions on the large middle branch of Figure 4.4. These evaluate the repaired schedule, add the results to the spreadsheet, and create a text file with the results. All of the results are stored in the same spreadsheet. An example is shown in Figure 4.14.

4.5.3 Main Menu

The Main Menu is the primary user interface to the system. When the user starts the system by running the appropriate macro, the main menu appears. From there, the user can choose to run the Instance Creation routine, run the Schedule Analysis routine, or exit the program. The Main Menu will perform the selected routine.

4.5.4 System Implementation

All of the subroutines are coded in Visual Basic and associated with the Microsoft Project project called "Macro." Figure 4.15 illustrates the calling hierarchy of these subroutines. Under this schema, Macro must be open to run any of the subroutines, and hence at least one project (Macro itself) must be open in Microsoft Project to run any subroutine.

The user starts the test system by running the fsb_main subroutine associated with the Microsoft Project Macro. This calls the appropriate

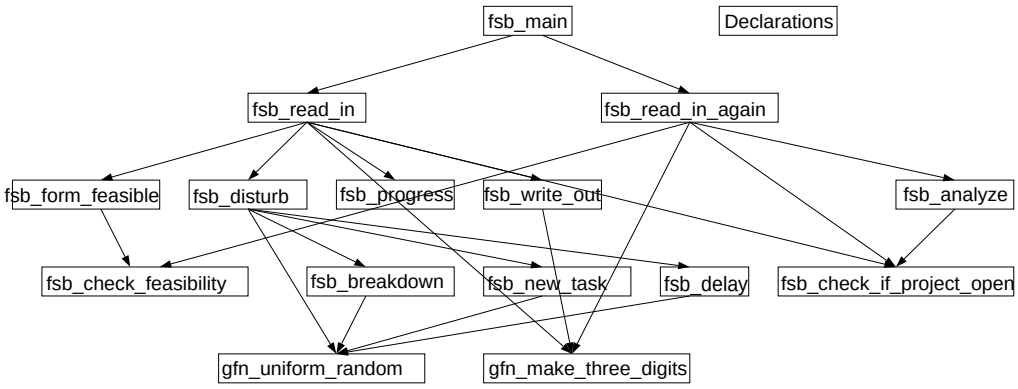

Figure 4.15: Subroutine Calls

routines. Each routine calls appropriate subroutines as in Figure 4.15. These function calls are not apparent to the user.

Although the system includes many subroutines, the four most important ones are analyze, disturb, form feasible, and read in. Read In Again and Write Out are very similar to these critical subroutines. The remaining subroutines can be viewed as subordinates to these subroutines, such as the individual disturbance subroutines, or generic helper functions, such as selecting randomly from a uniform distribution. Significant modifications would require changing these critical routines.

The system does not do everything necessary to test rescheduling algorithms. As discussed in Section 4.2.1, the system does not include the rescheduling algorithms. Thus, it does not create repaired schedules. Although the system does evaluate repaired schedules, it does not evaluate or compare algorithms. Because this task is so complex, deductive, and subjective, it is best left to the user. In essence, it collects data but does not draw conclusions. Figure 4.4 illustrates this: the functions Run algorithm, Output schedule, and

Evaluate output are those that the test system leaves for the user.

4.6 System Testing

I tested the components individually to ensure that they performed the small piece of the puzzle that they were supposed to. I noted the data inputs and outputs and made refinements until they performed correctly. Once the components performed correctly on their own, the system as a whole required testing. This is a critical step in the systems engineering process, because it is usually the system which must meet certain objectives, and not individual subsystems. There are many terms for this transition in focus, such as unit testing to integration testing [War95], and realization to systems integration [Aus97, page 48].

4.6.1 Prototypes

The system was too complex to design and develop at once. Therefore, prototypes were constructed. I made incremental changes along the way and saved each version that worked. Thus, attempts to enhance the system did not cause problems if they failed, because the previous version still existed. Although this approach was somewhat tedious, it worked.

An initial prototype of the system, as illustrated in Figure 4.3, consisted of independent subroutines that could read in an initial project definition, create a feasible schedule, create a random disturbance, repair the schedule, and record performance measures. To test this prototype, the system created instances using James H. Patterson's benchmark problems. The text files were manually changed to simulate the algorithm repairing the schedule. This shed

light on all the steps necessary for the automated system. In practice, prototypes are frequently developed before the full-scale working end-product. They can be produced quicker and at less cost than the full-scale product, and often give information that changes the end-product.

4.6.2 Complete Test

I performed many tests on the system. I manually changed many tasks in many instances and saved these to many spreadsheets, sometimes combining results into a large spreadsheet, and sometimes keeping results separate. In addition to my tests, Dr. Jeffrey Herrmann tested the system.

Using the Instance Creation routine, he created 20 rescheduling problem instances. To repair the schedules, he used two algorithms. The first, called "level," performed simple right-shifting to satisfy resource constraints. The second, called "string," shifted tasks so that no two tasks executed simultaneously. He created the appropriate text files for all forty solutions. Using the Schedule Analysis routine, he created the text files with the results and an Excel spreadsheet with the results. Using this spreadsheet, he calculated the average makespan, number of tasks disrupted, and total disruption time. With these results, he concluded that the level algorithm was a better algorithm than the string algorithm.

4.6.3 Results

The results are promising. The system achieved what it was supposed to: the testing and comparison of rescheduling algorithms' schedules. It creates instances from project definitions. It allows the user to use any rescheduling

algorithm. It measures the repaired schedules and allows the user to compare algorithms. Although not perfect, the system is good. There are, however, areas in which it could improve.

4.6.4 Needs for Future

There is room for improvement for the system. The system could automate some of the analysis statistics. The system could create instances that have more than one disturbance and instances that have more than one mode per task. The system could explicitly include financial aspects. It could accept other input formats besides Patterson's, or simply accept a first feasible schedule. The system could also include some well-known pre-packaged algorithms. Some of these advancements are the responsibility of the designer, whereas others that are more custom-oriented could be performed by the user. Including documentation and preparing for maintainability of the system throughout its life cycle allow, facilitate, and encourage this.

4.7 System Documentation

Engineers do more than simply calculate numbers and solve formulas. They must convincingly communicate and express their ideas, opinions, and results. In fact, about half of many engineers' work days are spent writing. This is especially true and important for systems engineers, whose primary responsibility is often the communication between groups of engineers. For large systems, documentation is needed to ensure that designers meet requirements and deadlines, builders understand the designers and build the system correctly, and that users learn how to use the system developed by the

designers. The technical manuals alone for the B-1 Bomber in 1986 had over one million pages [Cha96, page 288]. The major sources of documentation for this test system are this written thesis, the user's manual in the appendix, and the help file.

4.7.1 Thesis

This thesis provides a detailed background and formulation of both the project scheduling and rescheduling problems. It defines the requirements necessary in a test system, and how the system was designed and developed. This is most beneficial to users who want to develop their own system or enhance this one.

4.7.2 User's Manual

The user's manual in the appendix is for users who want to use, not redesign, this system. It provides step-by-step instructions, along with helpful tips, as well as an example of the system in use.

4.7.3 Help File

The computer code itself is documented and prepared for maintainability through naming conventions, global routines, and limited hardcoding, as already described. For further help in using Microsoft Project or Microsoft Visual Basic for Applications, there is extensive documentation and help menus, both within the programs as well as in most libraries and book stores. This was a prime reason why these programs were chosen for the system's technology.

4.8 Summary

This chapter described the system that we designed and implemented. This ranged from the requirements development, through exploring different concepts and tradeoffs, and on through subsystems development and systems integration. Finally, this chapter discussed the testing of the system and describes sources of documentation.

Chapter 5

Summary and Conclusions

This thesis presented and described a system to test rescheduling algorithms. From our initial work with the Patuxent River Naval Air Station, we learned the importance of rescheduling. Rescheduling is imperative in any project or job shop environment with uncertainty. This includes all practical project and job shop environments. Rescheduling, and in particular selecting a good rescheduling algorithm, has the potential to improve the productivity, profits, and customer relations of any business.

Projects can be decomposed into smaller tasks, each of which must be allocated resources at certain times. Teams seldom follow initial schedules throughout the life of a project because of unanticipated contingencies. Project managers must then reschedule: find new start and end times for tasks which have not yet finished in order to create a new feasible schedule. There are many algorithms in the operations management literature. However, there is little literature relevant to quantitatively comparing different rescheduling algorithms. This test system addresses this point.

I designed and developed a test system to evaluate rescheduling algorithms. I developed this system applying knowledge from systems

engineering such as financial as well as technical issues, the life cycle, and proper documentation. The system pays special attention to the anticipated requirements of its users in an attempt to facilitate using the system. The system uses Microsoft applications to perform two major functions. First, it creates rescheduling instances given project definitions. Second, after the user applies a rescheduling algorithm, the system evaluates the repaired schedules. The user can analyze the results, which are stored in a spreadsheet, and draw conclusions about the performance of rescheduling algorithms. In general, the system automates the tedious part of testing algorithms while giving the user as much flexibility as possible. For instance, the only constraint on the rescheduling algorithm is the requirement to read and write specially-formatted text files.

In addition, the system defines a methodology for testing algorithms: create a number of instances, apply different algorithms to these instances, and measure their performance. The system uses well-defined metrics of performance and captures elements of scheduling performance and rescheduling performance. The system provides a concrete framework of how to compare different approaches systematically.

There are many areas for improvement and future study. There are an almost limitless number of additional options that the user may desire from the system. The high quality of maintainability stressed throughout this thesis facilitates the user's changes. Specifically, if the user wants to identify the best algorithm, the user will need to compare algorithms on all three performance measures. Thus, the user may want to define a new performance measure that is a weighted combination of the three existing measures. In addition, the user

may want to visualize the tradeoffs that occur among the performance measures. The system could be extended to support these functions. Future study could also involve a similar design for job shop rescheduling algorithms.

Appendix A

User's Manual for Gerber Rescheduling Program

This appendix is intended to help the user use the test system. The user should familiarize himself with the rest of this thesis to learn the theoretical background for the scheduling and rescheduling problem. This appendix is specific to the Gerber Rescheduling Program using Microsoft Project, Microsoft Excel, and Microsoft Visual Basic for Applications.

Open the Microsoft Project project file Macro. Be sure to Enable Macros so that you can access the program. From the Tools menu, select Macro, then Macros, then run Macro.mpp!fsb_main. This is the main user interface of the system and should be the only subroutine you need to directly access. If you want to extend the capabilities or customize the system, you may later consider accessing other individual subroutines.

A dialog box will appear asking you which function you would like to perform, as in Figure A.1. Enter either 1, 2, or 3, followed by Return or pressing the OK button. See the appropriate sections in this appendix for either creating instances or evaluating repaired schedules. This appendix concludes with your responsibility as a rescheduler in this system, and other warnings and suggestions.

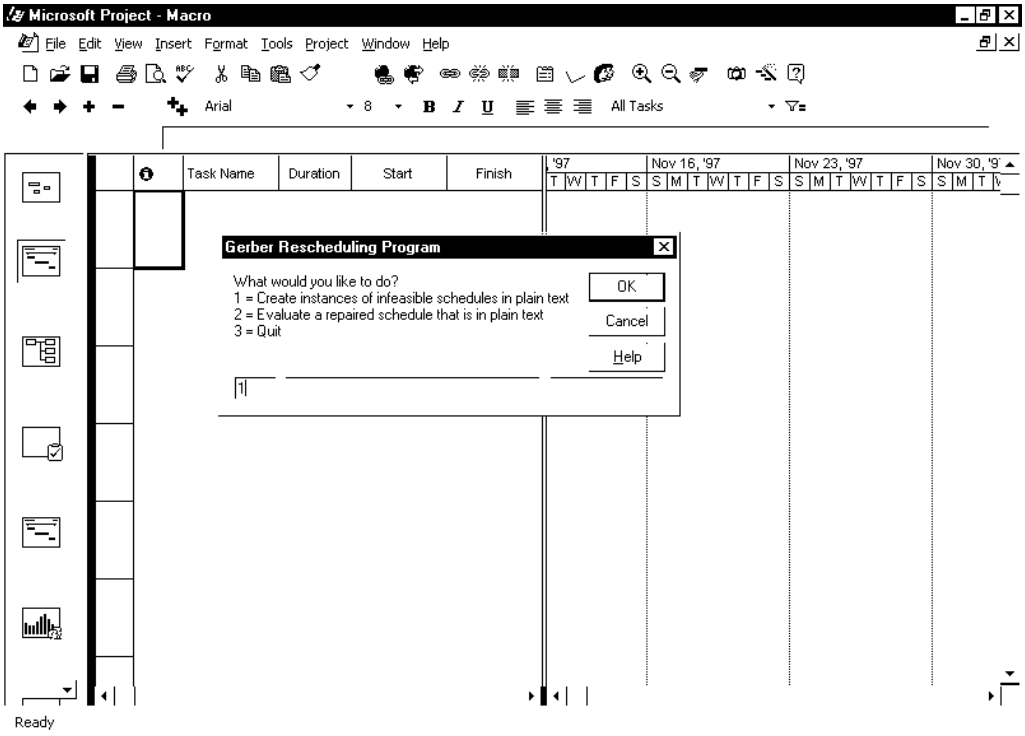

Figure A.1: Main Menu's Dialog Box

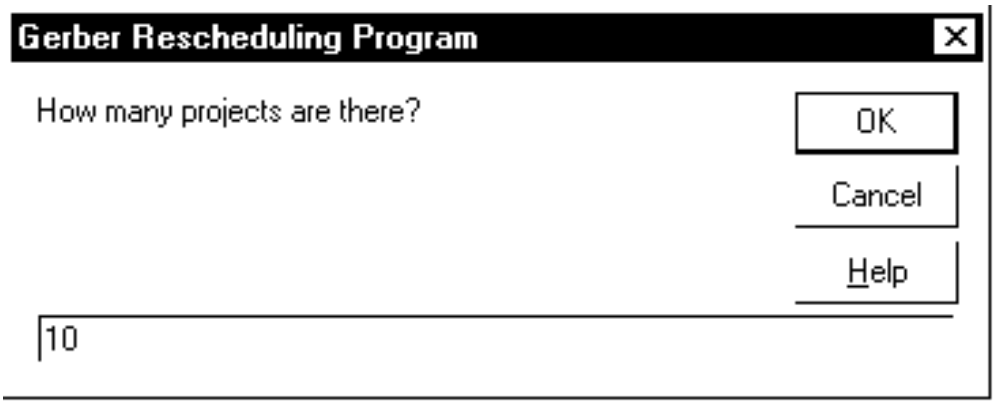

Figure A.2: Number of Instances

A.1 Instance Creation

Once you have entered 1 from the main menu, another dialog box, as shown in Figure A.2, will appear asking for the number of projects. Enter the number of instances you want to create, followed by Return. Then, for the next dialog box, enter the directory of project specifications, as in Figure A.3. For example, if you're accessing files in the top level from the A drive, enter "A:\". The next two dialog boxes will ask for the prefix of the files (Figure A.4) and the first number (Figure A.5). For example, all of the files distributed with this program are named "pat" followed by a three digit number, with the ".txt" extension. You may use or create other project specifications, but they must follow the naming convention of $prefixXXX.txt$, where $prefix$ is the same for all files and XXX is a three digit number, 000 through 999 inclusive.

After checking to ensure that the system correctly understands your input as in Figure A.6, enter the folder where you want to save the instances (Figure A.7). Each instance will have both a text file and a Microsoft Project

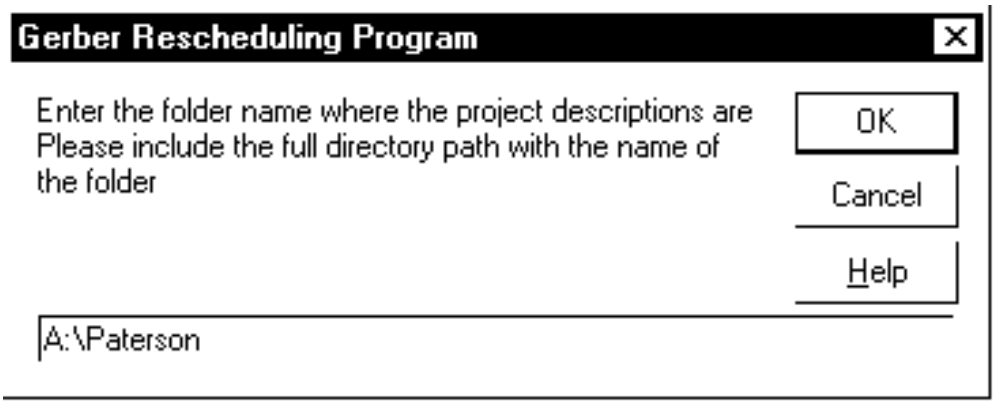

Figure A.3: Source File for Project Specifications

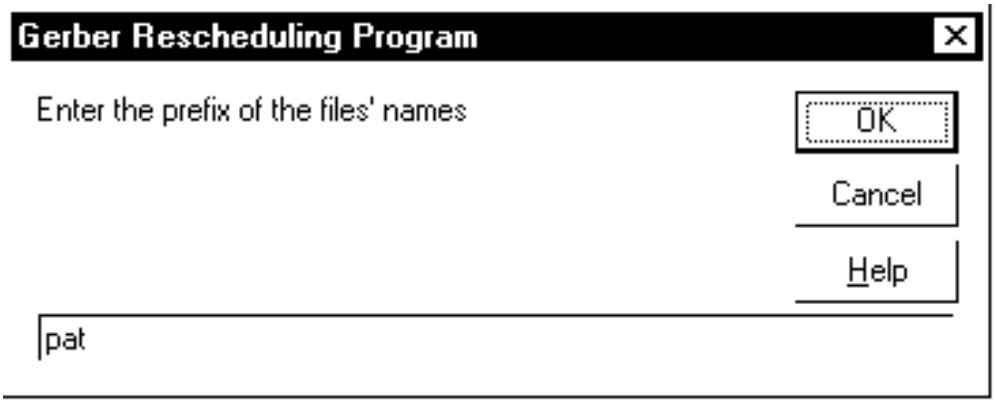

Figure A.4: Prefix of Source File Names

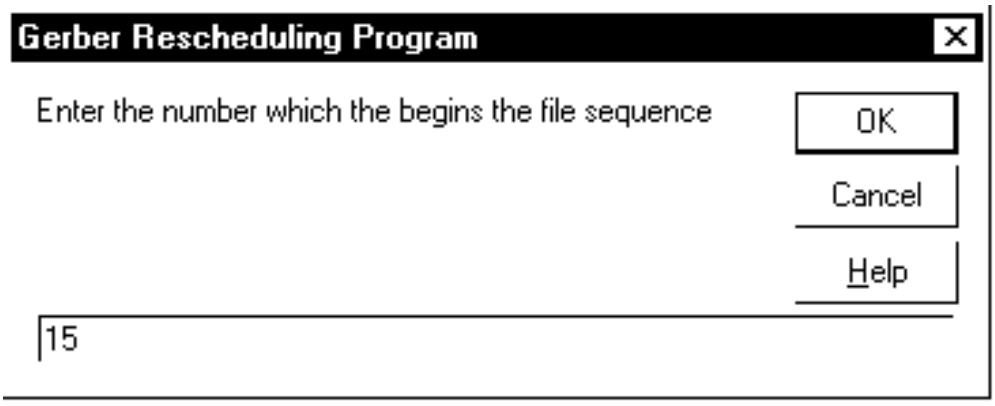

Figure A.5: Sequence Beginning for Project Specifications

project file, so be sure that the folder has adequate space. Finally, enter your rescheduling algorithm's name as in Figure A.8. The system will use this to track the instances through the rescheduling process and record information in the spreadsheet for results. Although you should use a descriptive name in order to identify your algorithm, the name itself is immaterial.

The system should then automatically perform the rest of the functions. The computer screen will show the various steps of the process in quick succession. Typical instances for small problems such a Patterson's take 25–30 seconds on a 100 MHz machine. Slower machines or accessing floppy rather than hard disks will increase this time. The program should conclude with an exit message.

A.2 Schedule Analysis

Once you have entered 2 from the main menu, the program will have a series of dialog boxes similar to the Create Instances function. Enter the number of repaired schedules you want to evaluate, their location, the prefix of the name

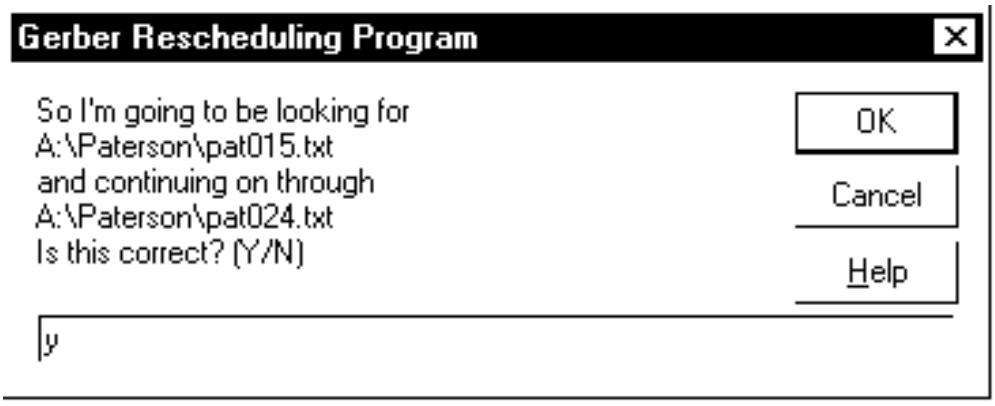

Figure A.6: Checking the Input

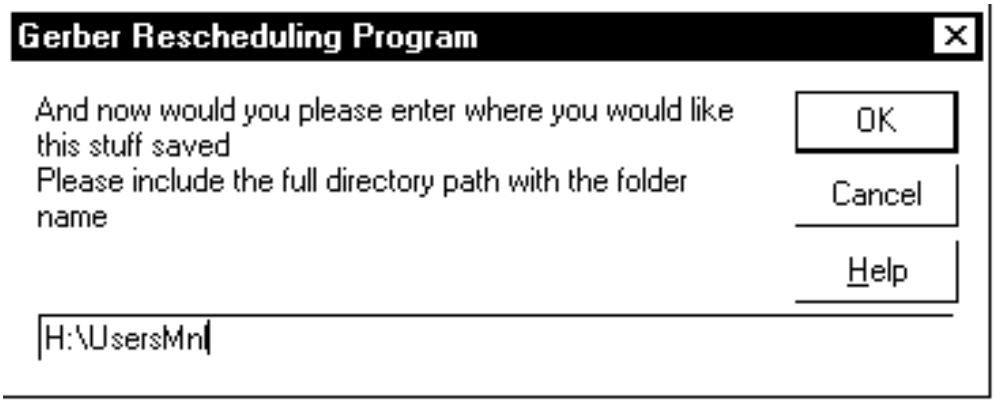

Figure A.7: Destination Directory

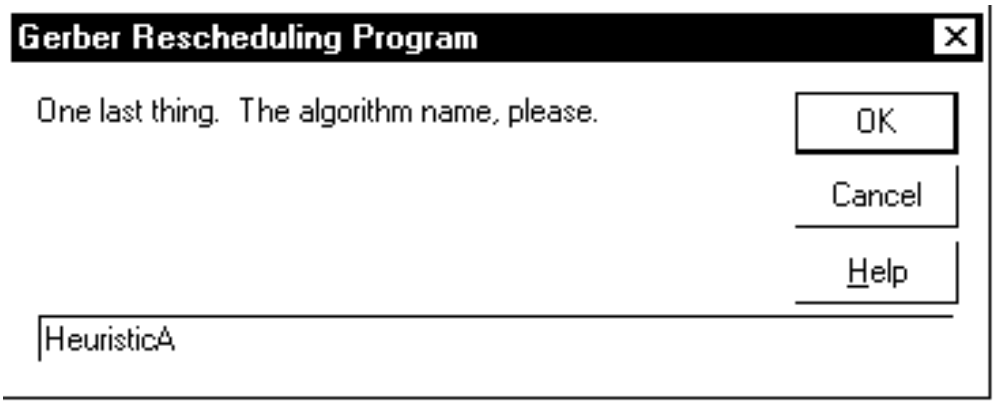

Figure A.8: Algorithm Name

and the first number in the series. If you want the results compiled into a larger spreadsheet, enter its name and location. Otherwise, type "n" for the dialog box in Figure A.9 and then enter the name and location of a new spreadsheet to create as shown in Figure A.10.

A.3 Rescheduling

Your job as the rescheduler is to repair an infeasible schedule. The plain text files containing the infeasible schedule must be altered. It is a good idea to copy these into a separate directory to avoid losing or overwriting them. The schedule can be changed by either changing the starting times for the tasks, or by preempting tasks.

To change the starting time, simply replace the current starting time with the new starting time, listed with both date and time, in the format $mm/dd/yy\ hh:nn:ss\ AM$ where mm is the month, dd is the day, yy is the year, and the time is given with either the AM or PM designation and hh hours, nn minutes, and ss seconds.

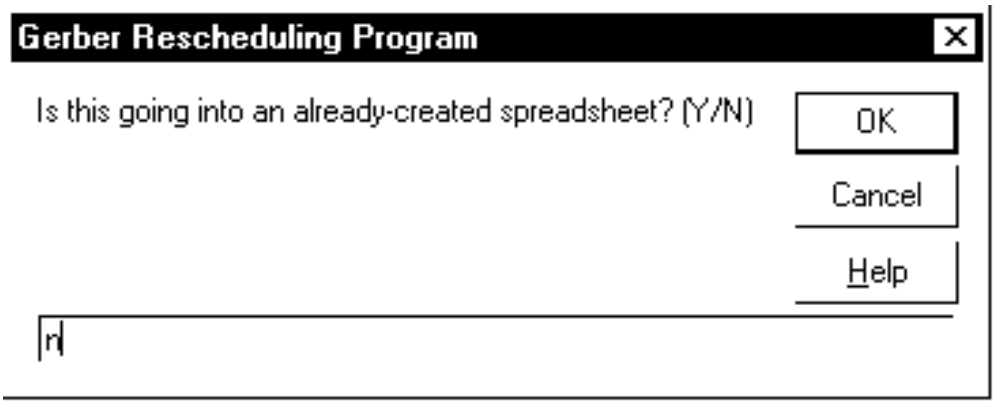

Figure A.9: Compilation of Results

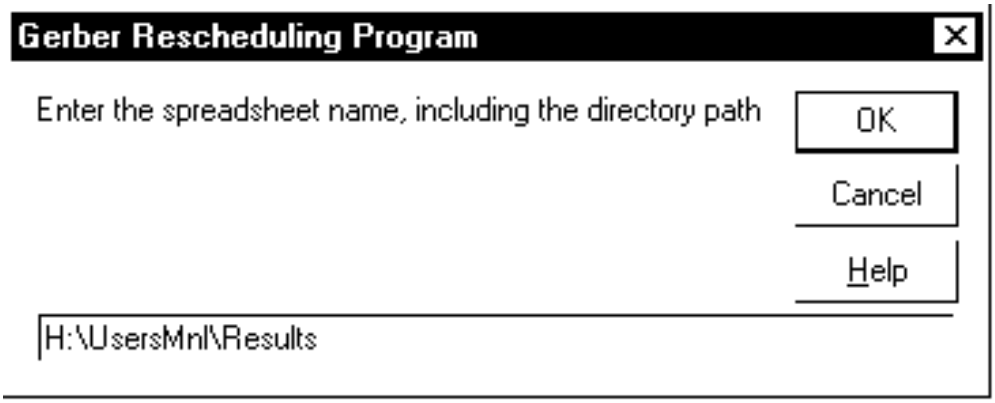

Figure A.10: Spreadsheet Name

You must be careful when preempting tasks. You cannot preempt a failure or breakdown task. The preempted tasks must be listed sequentially, with the first preempted task having its continuation immediately after the original list end, the second preempted task having its continuation immediately next and so on. A task may be preempted at most once. The total of the durations for the two parts must sum to the original duration. The resource usage per time period must be the same for both the preempted and continued portions. The continued portion must begin no earlier than the end of the preempted's portion finish. Finally, the precedence relations must be correct. The predecessors of the preempted portion remain as they were. Its only successor, however, is now the continued portion. The continued portion has as its sole predecessor the preempted portion, and as its successors the successors of the original task.

Be sure to keep the header information as it is. The file name, creation date, disturbance time, and algorithm name are all essential for bookkeeping.

A.4 Warnings and Suggestions

You should be able to use most of this program without any problems. The system was designed for you, the user, and assumes you are a competent computer user and rescheduler. Most of the instructions should be easy to understand. When errors do occur, the dialog boxes should guide you in the right direction. However, there will be times when more serious problems arise. A few suggestions should limit these.

If a serious error such as an inability to access Excel occurs, quit the test program and run fsb_main again.

If an error occurs and the dialog box doesn't give a cause, you have the option to run the program line-by-line as in a debugger fashion. You can then see which line produces an error. This is especially useful for debugging during either customizing or extending the capabilities of the program.

The program overwrites the interim schedules in Microsoft Project. Once a schedule is repaired, it cannot be repaired again. To run more than one algorithm on an instance, be sure to copy instances and files to multiple destination folders.

BIBLIOGRAPHY

[Aus97] Mark Austin. *Design Projects*, volume I of *ENSE 623: Systems Engineering*. Institute for Systems Research, University of Maryland, 1 September 1997. austin@isr.umd.edu.

[BM96] R. Belz and P. Mertens. Combining knowledge-based systems and simulation to solve rescheduling problems. *Decision Support Systems*, 17(2):141–157, May 21 1996.

[Boc90] Fayez F. Boctor. Some efficient multi-heuristic procedures for resource-constrained project scheduling. *European Journal of Operational Research*, 49(1):3–13, November 6 1990.

[CBW92] William Luther Chapman, A. Terry Bahill, and A. Wayne Wymore. The system design process. In *Engineering Modeling and Design*, Systems engineering series, chapter 2, pages 9–32. CRC Press, Inc., Boca Raton, Fla., 1992.

[CDM79] Harlan Crowder, Ron S. Dembo, and John M. Mulvey. On reporting computational experiments with mathematical software. *ACM Transactions on Mathematical Software*, 5(2):193–203, June 1979.

[Cha96] Alphonse Chapanis. *Human Factors in Systems Engineering*. Wiley Series in Systems Engineering. John Wiley & Sons, Inc., New York, 1996. A Wiley-Interscience Publication.

[CK88] David P. Christy and John J. Kanet. Open order rescheduling in job shops with demand uncertainty: A simulation study. *Decision Sciences*, 19(4):801–818, Fall 1988.

[Cor95] Microsoft Corporation. *Microsoft^R Visual Basic^R*. Microsoft Corporation, United States of America, 1995. Language Reference. Programming System for Windows^R Version 4.0.

[Cor96] Microsoft Corporation. *Microsoft^R Project / Visual Basic^R Reference*. Microsoft Press, Redmond, Washington, microsoft^r professional edition, 1996. Official Reference to Visual Basic for Applications Keywords in Microsoft Project.

[Dav73] Edward W. Davis. Project scheduling under resource constraints — historical review and categorization of procedures. *AIIE Transactions*, 5(4):297–313, December 1973. American Institute of Industrial Engineers Transactions.

[DH97] Erik L. Demeulemeester and Willy S. Herroelen. A branch-and-bound procedure for the generalized resource-constrained project scheduling problem. *Operations Research*, 45(2):201–212, March-April 1997.

[dWB92] D. de Werra and J. Blazewicz. Some preemptive open shop scheduling problems with a renewable or a nonrenewable resource. *Discrete Applied Mathematics*, 35(3):205–219, March 6 1992.

[GS85] B. L. Golden and W. R. Stewart. Empirical analysis of heuristics. In Eugene L. Lawler, Lenstra, Rinnooy Kan, and Shmoys, editors, *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, Wiley-Interscience Series in Discrete Mathematics and Optimization, chapter 7, pages 208–214. John Wiley & Sons, Inc., Chichester [West Sussex]; New York, repr. with corrections edition, 1985. A Wiley-Interscience publication.

[Her72] Willy S. Herroelen. Resource-constrained project scheduling — the state of the art. *Operational Research Quarterly*, 23(3):261–275, 1972.

[IEZ93] Oya Içmeli, S. Selcuk Erengüc, and Christopher J. Zappe. Project scheduling problems: A survey. *International Journal of Operations & Production Management*, 13(11):80–91, November 1993.

[JE97] A. K. Jain and H. A. Elmaraghy. Production scheduling/rescheduling in flexible manufacturing. *International journal of production research*, 35(1):281–309, January 1997.

[KD82] I. Kurtulus and E. W. Davis. Multi-project scheduling: Categorization of heuristic rules performance. *Management Science*, 28(2):161–172, February 1982.

[Kra85] Ira H. Krakow. *Project Management with the IBM PC Using Microsoft Project, Harvard Project Manager, VisiSchedule, and Project Scheduler*, chapter 1–4, pages 1–76. Brady Communications Company, Inc., Bowie, MD 20715, 1985.

[LK91] Averill M. Law and W. David Kelton. *Simulation modeling and analysis*, chapter 10,12–13, pages 582–611,656–735. McGraw-Hill Series in Industrial Engineering and Management. McGraw-Hill, Inc., New York, second edition, 1991.

[LW93] R. K-Y. Li and R. J. Willis. Resource constrained scheduling within fixed project durations. *Journal of the Operational Research Society*, 44(1):71–80, January 1993.

[Off97] Business Development Office. Naval air warfare center — nawcad. Web Site, November 24 1997. http://www.nawcad.navy.mil/.

[ÖU95] Linet Özdamar and Gündüz Ulusoy. A survey on the resource-constrained project scheduling problem. *IIE Transactions*, 27(5):574–586, October 1995. Institute of Industrial Engineers Transactions.

[Pat84] James H. Patterson. A comparison of exact approaches for solving the multiple constrained resource, project scheduling problem. *Management Science*, 30(7):854–867, July 1984.

[PHY+92] C. J. Paul, L. E. Holloway, D. Yan, J. K. Strosnider, and B. H. Krogh. An intelligent reactive monitoring and scheduling system. *IEEE Control Systems magazine*, 12(3):78–86, June 1992.

[PS94] A. Alan B. Pritsker and Kent Snyder. Simulation for planning & scheduling. *APICS - The Performance Advantage*, pages 36–41, August 1994. American Production and Inventory Control Society.

[PVTM96] Tim Pyron, Kathryne Valentine, Pam Toliver, and Laura Monsen. *Using Microsoft^R Project for Windows^R 95.* Que^R Corporation, Indianapolis, Indiana, second edition, 1996. Special Edition.

[Sad94] Norman Sadeh. Micro-opportunistic scheduling: The micro-boss factory scheduler. In Monte Zweben and Mark S. Fox, editors, *Intelligent scheduling.* Morgan Kaufmann Publishers, San Francisco, Calif., 1994. Carnegie Mellon University.

[SC93] V. Suresh and Dipak Chaudhuri. Dynamic scheduling — a survey of research. *International Journal of Production Economics*, 32(1):53–63, August 1 1993.

[SDK78] Joel P. Stinson, Edward W. Davis, and Basheer M. Khumawala. Multiple resource-constrained scheduling using branch and bound. *AIIE Transactions*, 10(3):252–259, September 1978. American Institute of Industrial Engineers Transactions.

[SOS95] Norman Sadeh, Shinichi Otsuka, and Robert Schnelbach. Predictive and reactive scheduling with the micro-boss production scheduling and control system. In Michael Fisher and Richard Owens, editors, *Executable Modal and Temporal Logics: IJCAI '93 Workshop on Knowledge-Based Production Planning, Scheduling and Control,*

Chambery, France, August 28, 1993, Berlin; New York, 1995. Springer. International Joint Conference on Artificial Intelligence.

[SV93] M. Grazia Speranza and Carlo Vercellis. Hierarchical models for multi-project planning and scheduling. *European Journal of Operational Research*, 64(2):312–325, January 22 1993.

[Tea96] InfoLink Web Development Team. Patuxent river naval air station. Web Site, July 1996. http://www.nawcad.navy.mil/pax/.

[TKE94] Ayşegül Toker, Suna Kondakci, and Nesim Erkip. Job shop scheduling under a non-renewable resource constraint. *Journal of the Operational Research Society*, 45(8):942–947, August 1994.

[TXF97] Chen Tsiushuang, Qi Xiangtong, and Tu Fengsheng. Single machine scheduling to minimize weighted earliness subject to maximum tardiness. *Computers & Operations Research*, 24(2):147–152, February 1997.

[War95] Mark Warhol. *The Art of Programming with Visual Basic^{TM}*. John Wiley & Sons, Inc., New York, 1995.

[Wil85] R. J. Willis. Critical path analysis and resource constrained project scheduling — theory and practice. *European Journal of Operational Research*, 21:149–155, 1985.

[Woo93] Bruce M. Woodworth. A statistical evaluation of the impact of limited resources on project scheduling. *Cost Engineering*, 35(2):25–28,31–32, February 1993. The American Association of Cost Engineers.

[YS93] Kum-Khiong Yang and Chee-Chuong Sum. A comparison of resource allocation and activity scheduling rules in a dynamic multi-project environment. *Journal of Operations Management*, 11(2):207–218, June 1993.

[ZDDD93] Monte Zweben, Eugene Davis, Brian Daun, and Michael J. Deale. Scheduling and rescheduling with iterative repair. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(6):1588–1596, November/December 1993.