APPLYING DECOMPOSITION METHODS TO SOLVE A STOCHASTIC AVAILABLE-TO-PROMISE PROBLEM

by

Arm Pangarad
A Dissertation
Submitted to the
Graduate Faculty
of
George Mason University
In Partial fulfillment of
The Requirements for the Degree
of
Doctor of Philosophy
Information Technology

Committee:

[illegible]	Dr. Karla L. Hoffman, Dissertation Director
[illegible]	Dr. Andrew G. Loerch, Committee Member
[illegible]	Dr. Chien-Yu Chen, Committee Member
[illegible]	Dr. Clifton D. Sutton, Committee Member
[illegible]	Dr. Daniel Menascé, Associate Dean for Research and Graduate Studies
[illegible]	Dr. Lloyd J. Griffiths, Dean, The Volgenau School of Information Technology and Engineering
Date: April 23, 2008	Spring Semester 2008 George Mason University Fairfax, VA

Applying Decomposition Methods to Solve a Stochastic Available-To-Promise Problem

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at George Mason University

By

Arm Pangarad
Masters of Business Administration
Southeastern University, 2001

Director: Karla L. Hoffman, Professor
Department of Systems Engineering and Operations Research

Spring Semester 2008
George Mason University
Fairfax, VA

DEDICATION

This dissertation is dedicated to my parents, Gen. Sopon Pangarad and Mrs. Popporn Pangarad, for their loves and supports.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Karla L. Hoffman, for her help, support and encouragement. I would like to thank Dr. Chien-Yu Chen who inspired me to do this dissertation. I also would like to thank my doctoral supervisory committee Dr. Andrew G. Loerch, Dr. Chien-Yu Chen and Dr. Clifton D. Sutton, for their useful comments and suggestions.

I would like to deeply thank my parents who always support and encourage me to complete my education.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

APPLYING DECOMPOSITION METHODS TO SOLVE A STOCHASTIC AVAILABLE-TO-PROMISE PROBLEM

Arm Pangarad, Ph.D.

George Mason University, 2008

Dissertation Director: Dr. Karla L. Hoffman

The available-to-promise (ATP) model is a mechanism that provides recommendations about when to accept customer orders that takes into account both product availability information, current customer orders and future orders in order to maximize overall profits. It becomes an important tool in a decision making process for manufacturing businesses. In this thesis, we present the stochastic available-to-promise problem which addresses the problem of needing to accept-or-reject in real-time orders for customizable computer configurations where the manufacturer cannot predict when the most profitable customers might arrive, but does have some probabilistic information about the likelihood of order arrivals and their requirements. Because the problem is stochastic, modeling all possible future scenarios results in an exponentially large problem. Even when one limits the total number of scenarios considered, solving the problem by off-the-shelf commercial solvers such as CPLEX results in computation times that are too large to be usable. We study the underlying structure of the model and propose decomposition

methods to solve it. We test both a Dantzig-Wolfe decomposition (“Column-generation approach) and a Bender’s decomposition (a row-oriented decomposition). We compare solution times of both methods to solution times of CPLEX.

1. Introduction

In this dissertation we discuss the available to promise problem. This problem is designed to determine whether to take a given order when it arrives or to reserve manufacturing resources in order to be able to handle future orders. These decisions are based on current and potential orders. An available to promise (ATP) model is a decision support tool used in manufacturing business that determines whether to accept or deny orders to maximize overall profit. We define classes of orders we use throughout our paper as follows.

	Current Orders	Future Orders
High Profit	class 1	class 2
Low Profit	class 3	class 4

Figure 1.1 Order classification

Consider the following example of a company that manufactures computers. Ideally the company would like to accept all profitable orders. But there are insufficient resources such as raw materials, hard disk drives, memory modules, motherboards, labors and production capacity to be able to accept all current and future orders. Thus, the company needs to choose which orders to accept and which to deny in order to maximize profit. How does one make this decision? Normally, one wants to choose orders that yield the higher net profit (class 1 and 2) and deny others that yield lower net profit (class 3 and 4). This approach makes sense as long as there are sufficient resources to manufacture all of class 1 and class 2 orders. Within the process of choosing orders to accept, we have a commitment to fulfill all requirements of those orders once they are accepted or promised. Such requirements are quantities, delivery dates and material compatibilities or supplier preferences. If the company cannot fulfill all of the requirements of the orders, then the company must refuse the orders. The issue becomes more complicated when one is attempting to guess when and what future orders might occur.

In an earlier work by [Ball et al. 2001] used a deterministic version of the ATP model to determine a batching interval, an inventory level and due date ranges in order to maximize the overall profit. Although the model itself can be solved by commercial solvers but there are some difficulties. Results from experimenting on the batching interval, showed that the optimization model yielded the best results (i.e. the highest profit) when the optimization had data over many days and could make decisions over a very long horizon. Thus, the optimization had more complete information and could determine a "best" strategy for the entire period. The downside of this approach was customers' satisfactions. Imagine that you are a customer, you place an order and then you have to wait 6 or 7 days before you get a response from the manufacturer as to whether your order is accepted or denied. Customers are unwilling to wait that long. On the other hand, responding to customers within 1 or 2 days adds significant risk to the manufacturer and can significantly reduce the manufacturers' profit.

To resolve the problem, Ball et al. chose to keep a certain amount of inventory in reserve for future orders and would deny orders if the inventory would be forced to fall below that level. The inventory reserve level is a predetermined inventory level which was set before running the optimization model. Its goal was to keep the resources from being consumed by current (possibly) less profitable orders. By staying at or above the inventory level, a set reserve is maintained for future, possibly more profitable orders. Whenever the inventory goes below the predetermined level, a penalty is incurred. The problem with this solution approach is that we do not know how much of our resources to reserve. If we reserve too much, profitability could be harmed when not as many higher profitable orders come in as expected. Another approach that Ball et al. considered was to experiment with reducing the acceptable due date range by fifty percent. This approach yielded slightly less profit than the base case. They concluded that the range of acceptable due dates does not have much effect on the profitability.

Are there other ways that can help us make a better decision? Ball et al. used a deterministic optimization model, where the coefficient in both the objective and the constraints for each stage are known with certainty. The disadvantage of using the deterministic model is that it does not consider events or contingencies that might occur in the future. Since the deterministic model does not consider future orders, it makes its decisions based on orders gathered over the past few days. Thus, the solution depends heavily on the length of the batch. An alternative approach would be to have the current orders compete against future orders that have a certain probability of occurring. This approach falls under the class of problems known as “stochastic programming problems”

[Birge and Louveaux 1997]. The stochastic programming differs from the deterministic programming in that it explicitly includes uncertainty in the model and weights these uncertain (future) events with an expectation estimate that they will occur. The optimization explicitly uses these probability estimates when determining the optimal solution. The particular form of stochastic programs that we will consider is that of finding an optimal solution to a linear program whose coefficients and constant terms are not known at the time the decision is made but whose probability distributions are known or have been estimated based on historical experience [Birge and Louveaux 1997].

There are two basic types of the stochastic programs, two-stage stochastic programs and multi-stage stochastic programs [Birge and Louveaux 1997]. In the case of the two-stage stochastic programs, the programs contain two stages of decision making, the first-stage decision and the second stage decision. The first-stage decision is made with information that is known with certainty. The second-stage decisions are those events that have some probability of occurring in the future. Therefore, at the time the first-stage decision is made, we do not know with certainty what is going to happen in the second-stage. An optimal solution in a maximization problem is found by maximizing the first-stage profit plus the expected value of all of the possible events that could impact the second-stage profit. Multi-stage stochastic programs are similar to two-stage programs except that there are more than two stages. The problem we discuss is a two-stage stochastic program where the first stage decision is to determine if we should take a given order now, and all future possible orders belong to the second-stage, each with a given expected likelihood of occurring.

ATP's Types

There are three types of ATP models: Push-based ATP, Pull-Based ATP and Push-Pull integrated ATP [Chen 2003].

Push-Based ATP

Push-Based ATP models are used to allocate finished products to locations where customers will be served. The Push-Based ATP maximizes a long-term profit. There are two alternative modeling strategies: (a) The model estimates what future orders are likely to occur using forecasting data and includes all such orders in the model as if they were guaranteed to occur, and then an optimization model determines which of these order to take; or (b) The model collects all possible future orders and then uses a stochastic optimization approach to choose among future orders. Stochastic models, because they include many scenarios, are more expensive to run (i.e. can take considerable time to determine the optimal solution) and are therefore more suitable for the planning phase of an operation rather than for real-time order taking.

Pull-Based ATP

[Chen 2003] stated that the Pull-Based ATP can be as simple as a database look up or it can be an optimization model that updates the production schedule whenever orders are received and determines the amount of resources that will now be used. One advantage of the database look up application is it can operate in real time. It can promise orders within seconds. It takes orders on a first-come first-serve basis based on the current availability of finished products. The Pull-Based ATP maximizes the short term profit because its resources or finished products are predetermined by the Push-Based ATP. Because it operates under a first-come first-served basis, small profit orders may consume all of the available resources and future more profitable orders may then be denied solely because they arrived later.

Push-Pull integrated ATP

The Push-Pull integrated ATP was developed in order to take into account advantages from the Push-Based ATP and the Pull-Based ATP. The Push-Pull integrated ATP maximizes a long term profit by using an optimization model. For example, it incorporates both high level information (finished product inventory information) and component level information (production scheduling and material flows information) into an optimization model as seen in [Ball et al. 2001] and [Chen 2003]. However, this model often becomes too large and too complicated to solve especially when stochastic scenario-based information is included in the model. Although a deterministic version [Ball et al. 2001] is solvable, it does not consider future orders. To resolve the problem of keeping resources for future more profitable orders, the authors employed an inventory reserve level and executed the model in batching intervals.

Research Problem

The ATP model in [Chen 2003] falls into the Push-Pull integrated ATP category. It is a two-stage stochastic mixed integer programming which the author found extremely difficult to solve when decision variables in both stages are solved to integer optimality. The model discussed in this paper was too large to be solved with off-the-shelf sophisticated optimization software such as (CPLEX). Solving the problem this size as it is posts a problem about computing resources such as a memory management. This could cause a machine to run out of memory as the size of the problem increases. We propose a use of decomposition methods approach to solve the problem efficiently. We choose to solve the first stage decisions to integer optimality but relax the second stage decisions to be non-integral. This idea makes sense because the second stage decisions are an expected value in the stochastic programming.

We start our research by simplifying the model to study its structure, and trying to solve this simplified model using decomposition methods. We believe that working with a smaller model with the same underlying structure as the original problem will help us gain a better understanding of the structure of the problem. It is also easier to measure performance of the decomposition methods. Once we identify an efficiency of the decomposition methods, we choose the decomposition method which is the most efficient to solve the original problem. Thus, this research studies the underlying structure and determines a decomposition approach that allows the solution of the entire large problem by iteratively solving simpler problems and conveying information obtained from the subproblems back to the master problem. This problem is interesting because the general structure relates to numerous manufacturing applications such as inventory and production scheduling. Many such optimization problems have large numbers of constraints and variables and are currently unsolvable unless decomposed. Specifically, the problems become unsolvable as one increases the number of scenarios to a realistic number. We believe that both Bender's decomposition (row generation) and Dantzig-Wolfe decomposition (column generation) may be used to solve the problem.

This research will examine both types of decomposition and compares the performance of each. Note that this dissertation does not consider the underlying distribution of the scenarios, we assume that they have already been generated and are correct. We leave as future research the question of whether scenarios can be generated on an "as needed" basis and whether such a generation scheme would help the optimization procedures. This dissertation focuses on the already completely-formulated stochastic optimization problem and whether decomposition can help solve this problem efficiently and provide a practical real-time decision making.

Literature review

Our research focuses on the efficient solution to the stochastic available to promise problem. We therefore provide a literature review of both the formulation alternatives and the solution techniques for solving this problem. We begin with a literature review of the ATP function and then prior solution approaches to this problem. In terms of solution approaches, we concentrate our attention on decomposition methods that have been used to solve these problems. This discussion includes literature on Bender's decomposition, Dantzig-Wolfe decomposition and Branch-and-Price algorithms. We will provide background of these techniques. We will show which decomposition methods are used in which applications, how suitable they are to our problem, and how efficient they are in solving these problems. The details of the decomposition algorithms will be discussed in Chapter 3.

ATP Models

We begin this literature review with that directly relating to the ATP function. [Chen 2003] presented a push-pull integrated stochastic available-to-promise model that is formulated as a stochastic mixed-integer programming problem. The model is quite complicated and could not be solved within a 48-hour time limit using the commercial optimization solver, CPLEX. The model could be solved with all components of the model included, only when the number of scenarios was limited to two scenarios. Additionally, the authors attempted to aggregate the daily data into weekly data to reduce the amount of data in the model. Even with this aggregation, the solver still could not solve the model when a maximum time limit of 48-hours was allowed.

Other researchers [Ball et al. 2005] and [Neumann et al. 2002] use similar approaches of aggregating the data or decreasing the components included in the models to solve their problems. [Ball et al. 2005] modified a mixed-integer programming based on [Ball et al. 2001] to support multiple levels of resource availability that varied over time. Since the original model contained several million variables and constraints, it could not be solved. Therefore, they decomposed the original model into a master problem and a collection of sub-problems. The master problem and the sub-problems represent the weekly ATP problem and the daily ATP problem respectively. Each model was solved iteratively. That is, the weekly ATP model was solved to get a promised quantity and completion week for each order based on weekly production capacities and availabilities. They then fed the result into the daily ATP model to determine the day on which the promised quantity would be produced. Since the master problem's output provides the promised quantity and completion week for each accepted orders, the sub-problems were solved on a week-by-week basis. The sub-problems were modified by eliminating the inventory constraints since they do not have any effect in a single week. However, the decomposition of the problem can not guarantee global optimality.

[Neumann, et al. 2002] developed an advanced production scheduling system for batch plants in the process industries. This modeling effort considers only the production portion of the supply chain. It consists of three phases, a long-term decision, a mid-term decision and a short-term decision. The ATP function is executed in the mid-term phase and provides quantities and due dates for accepted orders. The decision is based on weekly or monthly forecasts. The short-term phase or the detailed production scheduling uses the information from the mid-term phase to allocate resources over time to produce the finished products. Thus, this modeling approach can be viewed as a decomposition of the problem whereby one aggregates data at higher levels and disaggregates at lower levels. However, there is no looking ahead or looking back.

[B. Jeong et al. 2000] and [Ervolina et al. 2006] avoid the difficulty of solving mixed integer programs (MIP) by solving their problems as linear program relaxations. [B. Jeong et al. 2000] proposed an available-to-promise system for the TFT, LCD manufacturing company. Their model is a linear mixed-integer programming problem. They solved the model in two steps to avoid the difficulties of solving a mixed-integer problem. First, they solved the linear programming relaxation of the problem using the commercial solver, CPLEX. Then the solution is used as an upper bound for the material requirements and also used as an initial schedule in the second step. They then refine this schedule using a heuristic to find feasible schedules that do not overuse the capacity available.

[Chen 2003] previously stated that in a push-based ATP function, i.e. in the planning component of the ATP function, demand forecasts and manufacturing capacity are the two most important pieces of information for the determination of the allocation of the finished products to meet customers' demand. A pull-based ATP or an execution side of the ATP then uses the allocated supply to deal with order promising and order fulfillment. What if there are not so many orders as expected by the demand forecasts? There will be an excess of inventory and one will incur the cost of that inventory. Normally, the ATP models focus on how to best allocate resources to serve customers and maximize our profit, but do not take into account the issue of excess inventory. The literature that deals with such inventory issues is known as the Availability Management Process. Here one is concerned with how to best allocate resources that become excess inventory due to the low demand. [Ervolina et al. 2006] proposed the Availability Management Process which combined the ATP function and an available-to-sell (ATS) system so that excess inventory would be considered in the overall process. The Availability Management Process works by first using the ATP model to allocate resources. If low demand is realized, (i.e. the best solution from the ATP model results in excess inventory), then the ATS system tries to figure out how to best use that excess.

The ATS system is designed to find alternative production configurations that best consume the excess supply. Here, alternative products are considered that are capable of using the inventory excess. Such products are sellable, but may need to be sold at a lesser sales price than those provided to the customers who have specified specific needs and delivery dates. Since the Availability Management Process uses excess inventory to create *new* products, it does not have to make the "yes/no" decisions of the ATP function, and can therefore be approximated as a linear (rather than integer) optimization problem.

In each of the above cases, the authors decomposed the problem into stages. At the first stage, they determined the overall capacity and orders that were to be produced. This information was then used to determine the schedules in each time periods. Thus, each was decomposed into a two-step solution process. In the first step, much of the data was aggregated from daily to weekly information, thereby reducing the amount of data needed to solve the model. Once decisions were made on the aggregate level, those results permanently determined the possible outcomes of the second step – that of determining specific production schedules. Some authors used linear programming relaxations to solve the upper level problem while others used a mixed-integer model. At the second step, some used heuristics while others, [B. Jeong et al. 2000], used mixed-integer optimization. The only stochastic version of the ATP model, [Chen 2003] was modeled as a mixed-integer linear program but could not be solved with standard off-the-shelf codes. Even when attempting to aggregate some of the data, the problem remained intractable. Though the above-described decompositions help speed up the solving process, they could not guarantee a globally optimal solution since there was no look-ahead or look-back to these approaches. In the next section, we introduce other decomposition methods which have the potential to consider the entire problem simultaneously and thus, if solvable, would provide a globally optimal solution.

Solution Procedures

In this research, we will introduce decomposition methods which are well-known in solving large scale optimization problems. The decomposition methods we consider are Dantzig-Wolfe decomposition and Bender's decomposition. Each of these decomposition methods exploits the problem's structure to decompose the original problem into a master problem and sub-problems. The sub-problems provide information back to the master problem which is re-solved. The process iteratively moves from master problem to subproblem and back again, in order to provide a global solution to the overall problem. In this way, we do not permanently assume that an aggregation is correct without considering the impact of that aggregation on the subproblem results. In [Chen 2003], the author pointed out that the overall problem has a structure that can be exploited by such decomposition methods. Chapter 3 will discuss these decomposition alternatives.

In the remainder of this chapter, we will present background of the block structure of the generic stochastic problem and describe how these characteristics relate to the usual decomposition approaches of Dantzig-Wolfe and Bender's decompositions. The general block structure of the stochastic programs is known as the *L-shaped* block structure. We demonstrate a general two-stage stochastic programs formulation as

$$\begin{aligned} &Max && c^T x + E(q^T y) \\ &Subject\ to && Ax + Ty = b \\ & && x \geq 0, y \geq 0 \end{aligned}$$

where x are the first stage variables, and y are the second stage variables. The objective function is to maximize the first stage profit $c^T x$ plus an expected value of the second stage profit $E(q^T y)$. The constraint $Ax + Ty = b$ takes form of the *L-shaped* block structure in the figure 1.2.

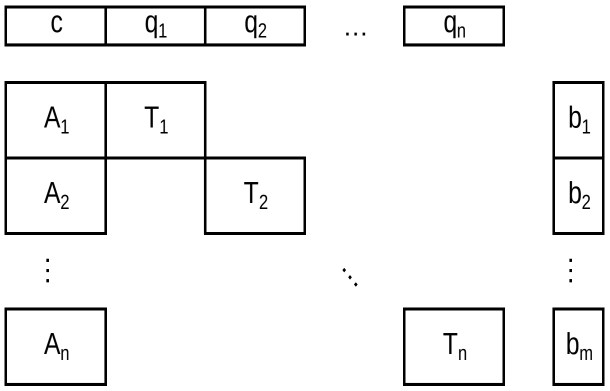

Figure 1.2 L-shaped block structure

The coefficient matrix A represents constraints in the first stage and the coefficient matrix T represents constraints in the second stage. The subscript 1..n represents the index number of the corresponding scenario that may occur in the second stage. Only one of the second stage events will occur and each has a given probability of occurring. We assume that we can provide a probability estimate for each scenario. The first stage decisions x are taken under uncertainty of future realizations of the second stage decision *y*. If one knew which event (scenario) would occur then it would be a straightforward optimization to determine the optimal course of action given the actions that were taken in stage one.

This problem is often referred to as the *recourse* problem in which the first stage decisions are chosen by taking their future effects into account. These future effects are measured by the recourse function, $E(q^T y)$, which computes the expected value of taking decision *x* given the expectations that might occur in stage *y*. A decomposition method is appropriate since the recourse function can be easily computed via linear programming for a fixed value of *x* and a given scenario. Thus, we create a master problem in the first stage decisions *x*, but we only evaluate the recourse function exactly as a sub-problem. The block structure of the stochastic programs has given rise to the name, "*L-shaped structure*" and can be used within a Bender's decomposition [Benders, 1962] and for stochastic optimization [Van Slyke and Wets 1969]. This method has been widely used in a variety of stochastic programming problems, see [Birge and Louveaux, 1997] for examples.

The Bender's decomposition method decomposes the problem into a master problem that determines the first stage decisions and sub-problems which determine the second stage decisions. Once the first stage decisions have been made, then this information is passed on to each of the sub-problems. The subproblems become independent problems and are easy to solve. The solution to the sub-problems provides feedback to the master problem, and requires that the master problem be re-solved. That is, the new information about the effects of setting the *x* (the first stage) variables provides a “cut” that forces the master problem to examine only solutions that will provide a better global bound. The algorithm iterates until it reaches the specified tolerance of the upper bound and lower bound gap.

[Gondzio and Kouwenberg, 2000] used Bender’s decomposition to solve the asset-liability management model. Their model is up-to-that-date, the largest stochastic linear program ever solved. It contains almost 5 millions scenarios, 12.5 millions constraints and 25 millions variables. The algorithm solved the problem within 4 hours. Similarly, [Geoffrion and Graves, 1974] used Bender’s decomposition to solve the multi-commodity distribution systems. They solved problems having 11,854 constraints, 727 binary variables and 23,513 continuous variables. For each of these problems, a straight-forward formulation and solution via branch and bound would have been computationally infeasible. In both cases, the algorithm required very few cuts to solve the global problem. Similarly, [Hiller and Eckstein 1993] used Bender’s decomposition to solve a fixed income portfolio model. This was also a large stochastic problem. The authors showed that the structure of the problem had the dual block angular structure (*L-shaped*). They found that Bender’s algorithm generated more cuts as the number of scenarios grew.

Bender’s decomposition has been widely used in many applications. For examples, [Bahl and Zionts, 1987] used Bender’s decomposition to solve a minimization of a multi-item scheduling problem. The problem is a mixed-integer (0-1) programming, and the structure of the problem was essential to the use of Bender’s decomposition. Specifically, when the capacity constraint was removed, the problem could be solved individually. The master problem became a pure 0-1 integer and was solved using an integer programming approach. The subproblems could be solved as transportation problems. Since solving transportation problems is easy, one only needed a strategy to efficiently solve the master problem. The authors employed a strategy suggested by [Geoffrion and Graves, 1974] whereby an initial lower bound is set to minus infinity and an initial upper bound is set to infinity. Normally in Bender's algorithm, if the problem is a minimization, solving the master problem provides a lower bound and solving the subproblem provides an upper bound. Geoffrion and Graves suggested that the master problem need not be solved to optimality. Thus, the master problem is solved only to produce a feasible integer solution which an objective function value obtained is less than the upper bound provided by the subproblem. The Bender's algorithm iterates until the master problem can not find any feasible integer solution which the objective function value obtained is less than the current upper bound. Bahl and Zionts also suggested that a heuristic procedure may be used to find a good starting solution which would provide a good bound, thus should result in a faster convergence.

[Van Roy 1986] developed a cross-decomposition algorithm for capacitated facility location. The capacitated facility location problem was decomposed to a master problem and subproblems. The algorithm was based on the Bender's decomposition but used a more sophisticated way to generate cuts to the master problem. The method was to iteratively solve two types of the subproblems (primal subproblems and dual subproblems) before generating cuts to the master problem. The primal subproblems give dual information to be fixed in the dual subproblems. One iterates between these two problems until one obtains convergence. Then, this solution provides cuts to the master problem. They also noticed that the subproblems could have multiple dual solutions and that the duals chosen have significant effect on the speed of convergence of the overall algorithm. They therefore developed strong cuts based on the multiple dual information.

[Bloom 1983] used generalized Bender's decomposition to solve a model for planning least-cost investments in electricity generating capacity with probabilistic reliability constraints. The problem naturally decomposes into two parts, determining the optimal investments in new generating capacity and determining the operating cost and reliability of the generating system. The master problem, which is a linear program, is used to generate trial solutions for the optimal capacity expansion plan. The subproblems are used to determine the minimum cost of operation and the reliability of the trial system in each period of the planning horizon. Though the subproblems are nonlinear optimization problems, the dual of the problem can be solved quite easily by using the probabilistic simulation algorithm [Baleriaux et al. 1967 and Booth 1972]. Associated with the solution of the subproblems are sets of dual multipliers which are returned to the master problem. The algorithm iterates until an optimal capacity expansion plan is found. Other related literature in this area can be found in [Bloom et al. 1984], [Sherali and Staschus 1990] and [Hobbs and Ji, 1999]. The differences relate to how the subproblems are solved given the specific structure of their problem. Each handles the master problem by adding Bender cuts.

[Cai et al. 2001] used Bender’s decomposition to solve large nonconvex water resources management models. The nonconvex nonlinear programming problems are large and sparse. The existing solvers, e.g., MINOS5 and CONOPT2 can not solve problems of this size. When applying Bender’s decomposition, the master problem and subproblems are linear programs. The authors demonstrated that the use of Bender’s decomposition was much faster than using either MINOS5 or CONOPT2 directly to the overall problem. Other applications which involve nonlinear programming problems such as [Clasen 1984] used the same approach of using Bender’s decomposition to solve the chemical equilibrium problem. The problem was transformed and decomposed to linear master problem and subproblems. The Bender’s algorithm was solved iteratively and optimality was proven. [Armstrong and Willis 1977] developed a model for water planning that simultaneously considered both investment and allocation decisions. The model was a quadratic mixed-integer programming problem. The Bender’s decomposition was used to solve the problem and convergence was relative quick, i.e. required very few iterations of the master problem.

[Magnanti and Wong 1981] developed an approach to accelerate convergence of Bender’s Decomposition for solving a facility location which is categorized to a network optimization. The network optimization such as shortest route, transshipment etc. are known for their degeneracy. Degenerate solutions provided by subproblems pose an issue of how to select a good solution to construct a good cut. Magnanti and Wong solved another linear program to determine which solution to be chosen. This technique chose a strong or dominating cut. In their approach, the overall time required of the subproblems increased (since additional linear programs were required), but such additional effort was worthwhile since the number of Bender's cuts needed to solve the overall problem was radically reduced.

We have seen that the Bender's decomposition is suitable for problems with the *L-shaped* or *dual block angular structure*. In this dissertation we pose the question of can we apply Bender's decomposition when the problem does not have this classic *L-shaped block structure*. Specifically, the structure that our problem exhibits is the one shown in the figure 1.3 – a structure commonly called the *block angular structure*.

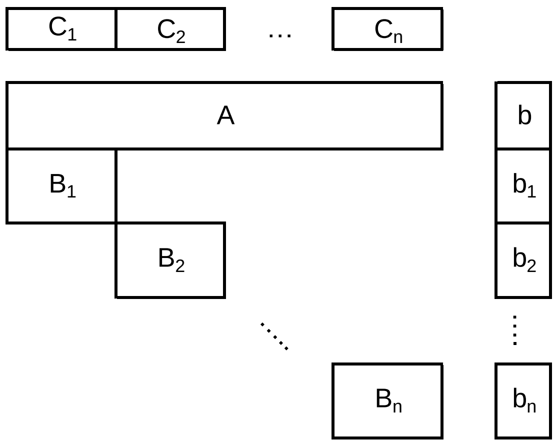

Figure 1.3 Block-angular structure

The *block-angular structure* is comprised of sets of constraints B_j, which are arranged in a diagonal shape and would form independent problems if the constraints in the *A-matrix* were ignored. The above structure can be formulated as

$$\begin{array}{ll} \textit{Max} & c^T x \\ \textit{Subject to} & Ax = b \end{array}$$

$$B_j x = b_j$$

$$x \geq 0$$

where we are maximizing $c^T x$ subject toe $Ax = b$ constraints and additional constraints $B_j x = b_j$. We can exploit this type of block structure by using the Dantzig-Wolfe (D-W) decomposition [Dantzig and Wolfe 1960], often referred to as a column-generation method. The D-W decomposition removes the $B_j x = b_j$ constraints from the original problem and creates a new problem which contains only the $Ax = b$ constraints. This problem will be called the master problem. The remaining $B_j x = b_j$ constraints create subproblems (or "pricing problems". [Barnhart et al. 1996] because these problems determine new columns with their respective prices that will be added to the master problem. Thus, this decomposition is similar to Benders decomposition and provides a mechanism to divide the work into a master problem and a number of subproblems.

The D-W algorithm begins by solving some collection of sub-problems to generate columns that provide the master problem with a set of initial columns. The master problem is solved with these columns that then provide dual information to the subproblems. This dual information is used to generate new columns to the master problem. For a maximization problem, we are looking for columns to the master problem that will have negative reduced costs (i.e. will improve the solution value of the master problem). We solve the master problem which provides us with new dual prices associated with the constraints $Ax = b$. These dual prices are used to determine if there is a sub-problem that provides a new column that will have negative reduced cost. The process continues until there are no columns that price out correctly, i.e. have negative reduced cost. This guarantees the optimal solution to the original problem (if the problem is linear) or provides the value of the linear programming relaxation to the master problem (if the problem is an integer linear-programming problem).

This approach removes the necessity of generating all possible columns "up front", when the number of such columns can easily be in the billions. Since only a small number of columns actually improve the linear programming solution, we improve the objective function iteratively and reduce the overall work required to solve the problem. The master problem with a limited number of columns is called a "restricted master problem" (RMP). Once a new column with negative reduced cost is found, it is added to the RMP. For mixed-integer programming (MIP) problems, the D-W decomposition may not give an optimal integer solution. In such cases, a branching procedure similar to the normal branch-and-bound approach of linear integer programming is needed to prove optimality. The algorithm which incorporates the column generation with the branch-and-bound is called the Branch-and-Price. We will discuss the details of the D-W decomposition and the Branch-and-Price algorithm in chapter 3.

Decomposition for the ATP problem

Next, we present literature in the ATP area which has applied the column generation and/or the Branch-and-Price algorithm to solve their problems. [Cheng et al. 2006] developed a model to determine product offerings (i.e. product substitution possibilities to meet demand forecasts) that might be used in an Assemble-To-Order environment. The model finds marketable product alternatives in a given product portfolio that best utilize available component supply. A column generation was used to solve this model. The problem was decomposed to a master problem and sub-problems. The master problem generates an optimal build plan for a recommended set of product configurations. The sub-problems use dual prices from the master problem to determine new configurations to be added to the existing set. The authors did not discuss the use of a branch and price routine. We must infer that they stopped when they had arrived at a feasible integer solution that was within some tolerance. This paper is the only one we could find that employed a column generation to solve a production problem.

On the other hand, column generation has been widely used in many other applications. For examples, [Desrosiers et al. 1999] solved airline crew scheduling problems using the column generation method. The airline crew scheduling involves determining a collection of crew schedules that minimize total cost to the airline while assuring that each crew is provided with a schedule that meets all of the labor rules. The problem is decomposed into a master problem and sub-problems. The master problem solves a linear relaxation of the generalized set partitioning problem. The subproblems generate new crew schedules with negative reduced cost to add to the master problem. A partial branching with a depth-first search was used to find an integer solution. The resulting integer solutions were found to be close enough to the linear program bound to stop with relatively reasonable solution times. Other set partitioning problems which used the column generation include the dynamic routing of independent vehicles by [Savelsbergh and Sol 1998], aircraft routing and scheduling by [Desaulniers et al. 1997] and generalized assignment problem by [Savelsbergh 1997], in cutting stock problems [Vanderbeck 2000] and [Vance, et al.], combinatorial auctions [De Vries et al. 2004], vehicle scheduling [Foster and Ryan 1976], political districting [Johnson 1998]. For an overview of such methods see [Barnhart, et al. 1999]

We found only one paper that described using column generation to solve a stochastic MIP problem. [Lulli and Sen 2004] used a branch-and-price algorithm to solve a multistage stochastic batch-sizing problem. The problem has a block angular structure as shown in figure 1.3. Subproblems were alternative scenarios. Each scenario could be considered independent if the linking constraint was removed (i.e the constraint that forces all scenarios' decisions with the same history until the t^{th} stage must result in the same decision until this stage). For this problem, the branch-and-price algorithm it was shown to have taken less time to solve compared to a commercial general purpose code, especially as the size of the problem increased. We will present the SATP problem's description and formulation in chapter 2.

2. Problem Formulation

In this chapter, we describe a full version of the SATP problem descriptions, notations and its formulation. Since the full version of the SATP could not be solved by commercial solvers, we have also created a simplified version of the SATP to work with. The simplified version of the SATP has the same structure as the full version one. We will study and experiment with the simplified version of the SATP because it is easier to gain insights of the problem structure with this version. Another benefit of using the simplified version of the SATP is it can be solved by off-the-shelf solvers. Thus, we can have a benchmark to compare against. Since the model is less complicated, it will not take long time to solve the problem even with a number of scenarios are included to the problem.

The full version of the SATP problem descriptions and notations:

We model the SATP problem as an optimization problem. The purpose of the model is to maximize overall profit. A time periods is usually equal to an 8 hour shift or a working day. We denote t_e the time period that has just ended when the model is about to be executed. A planning horizon of the model consists of T time periods. We assume the manufacturer offers multiple products of personal computers with different component options. Components are grouped into different component types according to their basic functional specifications, for example, 100 GB hard disk drive, 3 GHz Intel Pentium 4 CPU, 16x4x56 DVD-RW combo drive and so forth. All component types used in assembling finished products are grouped into set *M*. Some component types such as a DVD-RW at a certain specification may be supplied by multiple suppliers. The collection of suppliers *k* of type-*j* component are grouped into set in M_j. Customers can customize their orders to consist of a certain supplier *k* for component type-*j*, for example, they can specify that they want a Maxtor disk drive. They may also specify a group of suppliers for a certain component type-*j* and allow the manufacturer to choose among the available alternatives. The collections of customers' preferences are grouped into set M_j^i.

A customized bill-of-material (BOM) for the first stage decision, $fbom_{i,j}$, and for the second stage decision, $sbom_{i,j,s}$, specify the number of type-*j* component required in finished products for order *i*. We denote ω a production lead time and assume it to be constant. There are three types of flexibilities embedded in a customer order, a range of promised quantity, an allowed time window for promised due date and choices of suppliers for each component. The optimization model exploits those flexibilities to manage order promising and production planning as illustrated in figure 2.1.

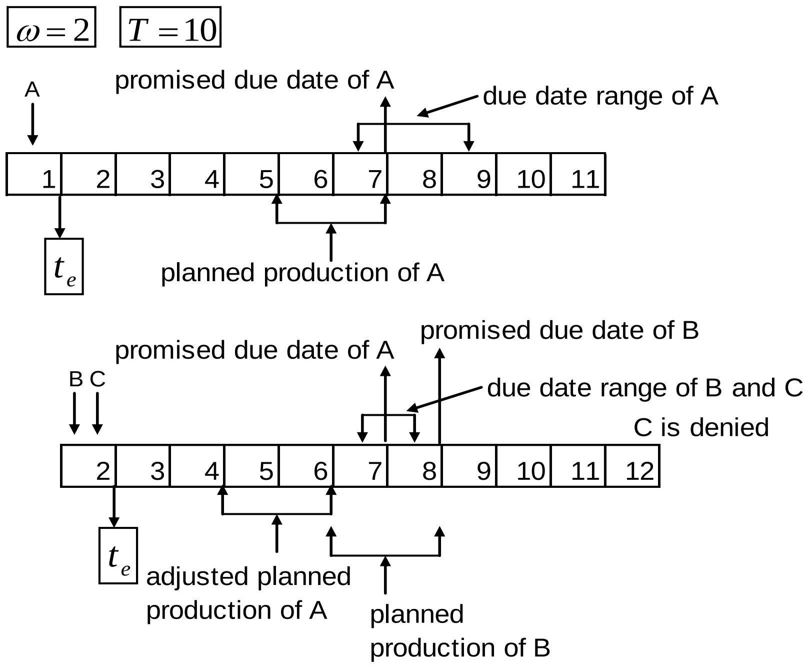

Figure 2.1 ATP function

We show an example of an ATP execution with a rolling horizon in figure 2.1. In this example, the planning horizon time T is 10 time periods. Since the current time period that has just ended is $t_e = 1$, the ATP planning horizon is time periods 2 through time period 11, and the production lead time is $\omega = 2$ time periods. During time period 1, there was an order "A" with due dates ranging from time period 7 to 9. We executed the ATP at the end of time period 1. The ATP decided to accept the order "A" with promised due date in time period 7 and scheduled the production at the beginning of time period 6 to finish at the end of time period 7. The delivery took place at the end of time period 7, thereby avoiding any finished product inventory cost. When there is no time between production completion and delivery, the process is called "just in time" production. Such production reduces both finished goods inventory and goods-in-process inventory costs. The figure below illustrates the next execution in the next time period. During time period 2, orders “B” and “C” came in with the same range of due dates (between time period 7 and 8). The ATP execution at the end of time period 2 decided to accept order “B” but deny order “C” due to production capacity limitations. It could not produce both orders at the same time. Order “C” is denied because order “B” yields more profit than order “C”. The ATP model needed to adjust its planned production of order “A” to accommodate the production of order “B”. We explain notations of an index set of time period we use in our model. We use the notation $t(T]$ to represent the set $\{t+1, t+2, ..., t+T\}$ and the notation $t[T]$ to represent the set $\{t, t+1, t+2, ..., t+T\}$.

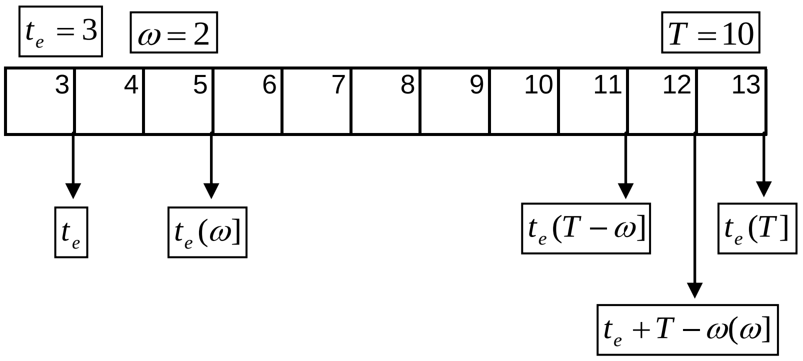

Figure 2.2 Example of index notation 1

Figure 2.2 shows examples of index set of time periods we use in our model. For example, $t_e(T-\omega]$ represents time periods between time period 4 and time period 11.

Because the beginning time period of $t_e(T-\omega]$ is $t_e+1=3+1=4$ and the ending time period is $t_e(T-\omega]=3+10-2=11$.

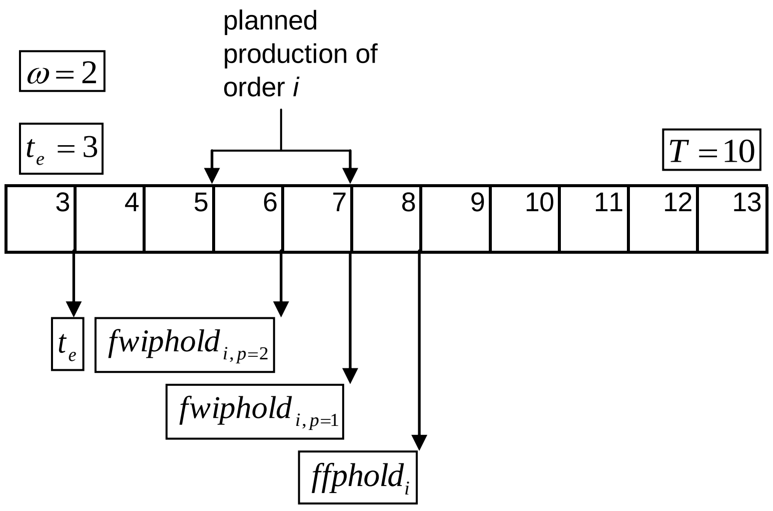

Figure 2.3 Example of index notation 2

There is a finished product holding cost per time period denoted by $ffphold_i$ for first stage decisions and $sfphold_{i,s}$ for second stage decisions. There is also a process holding cost denoted by $fwiphold_{i,p}$ for first stage decisions and $swiphold_{i,p,s}$ for second stage decisions. The finished product holding cost is incurred at the end of the time period as seen in figure 2.3. For example, if the procudtion of order *i* is finished at the end of time period 7, but the product is not delivered until the end of time period 8, then a finished product holding cost is incurred at the end of time period 8.

The work-in-process holding cost represents the holding cost of the unfinished product in stage *s* of the production. The index *p* denotes the number of additional time periods needed to complete the ω-time-period production process. For example, in figure 2.3, the planned production of order *i* is between time period 6 and 7. The production lead–time is $\omega = 2$, the production begins at the beginning of time period 6. At the end of time period 6, stage 1 of the production process has passed. During this time period, the work-in-process incurs a holding cost with index *p*=2 which means it needs an additional two time periods to complete the production. The work-in-process holding cost with index *p*=1 means it needs an additional one time period to complete the production. It will incur cost during the next time period 7. At this time the production process is in stage 2. Normally, a work-in-process holding cost is cheaper when the production process is in an early stage. This means the work in process holding cost for *p*=2 is cheaper than *p*=1. Because in the early stage of production we may be assembling only motherboards and their components, storage space required to store the unfinished products is small. Once the unfinished products continue to the next phase of assembly, they contain more parts and require more storage space.

The SATP Model Formulation and Description

Indices

i – orders

j – component types

k - suppliers

s – scenarios

t – time periods

p - additional time periods needed to complete the ω-time-period production

Sets

O - newly-arrived orders

$\hat{O}$ - previously-promised orders

O^s - future orders

T - time periods

W_i - set of delivery time window *t* for newly-arrived order *i*

W_i^s - set of delivery time window *t* for future order *i* in scenario *s*

S - scenarios

M – set of all component types used in assembling finished products

M_j - set of suppliers *k* of component type *j* also called component instance *(j,k)*

M_j^i - set of component instance *(j,k)* of order *i*

Data

$prob_s$ – probability associate to scenario *s*

$planrect_{j,k,t}$ – planned receipt of component type-*j* from supplier *k* in time period *t*, it is assumed to be known and can not be changed.

$prodcap_t$ – production capacity in time period *t*

$undrutilp_t$ – penalty associate to capacity under-utilization in time period *t*

$utiltrgt_t$ – desired production utilization target in time period t

flb_i – minimum quantity required for order i for first stage variables

fub_i – maximum quantity required for order i for first stage variables

$fbom_{i,j}$ – bill-of-materials specify number of type-j components required in a finished product for order i for first stage variables

$frev_i$ – revenue per unit from delivering order i for first stage variables

$fdeny_i$ – denial penalty cost for order i for first stage variables

$fmatc_{j,k}$ – material cost per unit of component type-j from supplier k for first stage variables

$ffphold_i$ – holding cost of finished product per time period for order i for first stage variables

$fwiphold_{i,p}$ – holding cost of work-in-progress for order i for first stage variables, where the index p denotes the number of additional time periods needed to complete the ω-time-period production process

$fmatinvh_{j,k}$ – holding cost per time period of material inventory of component type-j from supplier k for first stage variables

$slb_{i,s}$ – minimum quantity required for order i in scenario s for second stage variables

$sub_{i,s}$ – maximum quantity required for order i in scenario s for second stage variables

$sbom_{i,j,s}$ – bill-of-materials specify number of type-j components required in a finished product for order i in scenario s for second stage variables

$srev_{i,s}$ – revenue per unit from delivering order *i* in scenario *s* for second stage variables

$sdeny_{i,s}$ – denial penalty cost for order *i* in scenario *s* for second stage variables

$smatc_{j,k,s}$ – material cost per unit of component type-*j* from supplier *k* in scenario *s* for second stage variables

$sfphold_{i,s}$ – holding cost of finished product per time period for order *i* in scenario *s* for second stage variables

$swiphold_{i,p,s}$ – holding cost of work-in-progress for order *i* in scenario *s* for second stage variables, where the index *p* denotes the number of additional time periods needed to complete the ω-time-period production process

$smatinvh_{j,k,s}$ – holding cost per time period of material inventory of component type-*j* from supplier *k* in scenario *s* for second stage variables

Variables

$FQ_{i,t}$ – first stage quantity delivered for order *i* in time *t*, ≥ 0

FZ_{i} – first stage order fulfillment indicator (1, if a new order *i* is promised, 0 otherwise)

$FX_{i,j,k,t}$ – first stage quantity of component instance *(j,k)* use in the following ω-time-period production for order *i* in time period *t*, ≥ 0

$FFP_{i,t}$ – first stage finished-product inventory for order *i* at the end of time period *t*, ≥ 0

$FP_{i,t}$ – first stage quantity produced for order *i* in time period *t* (note that the entire production process actually starts in the time period $t-\omega$), ≥ 0

$FD_{i,t}$ – first stage delivery time period indicator (1, if a new order *i* is promised a delivery time period *t*, 0 otherwise)

$SQ_{i,t,s}$ – second stage quantity delivered for order *i* in time *t* in scenario s, ≥ 0

$SZ_{i,s}$ – second stage order fulfillment indicator (1, if a new order *i* is promised, 0 otherwise) in scenario *s*

$SX_{i,j,k,t,s}$ – second stage quantity of component instance *(j,k)* use in the following ω-time-period production for order *i* in time period *t* in scenario s, ≥ 0

$SFP_{i,t,s}$ – second stage finished-product inventory for order *i* at the end of time period *t* in scenario s, ≥ 0

$SP_{i,t,s}$ – second stage quantity produced for order *i* in time period *t* in scenario *s* (note that the entire production process actually starts in the time period $t-\omega$), ≥ 0

$SD_{i,t,s}$ – second stage delivery time period indicator (1, if a new order *i* is promised a delivery time period *t* in scenario s, 0 otherwise)

$SR_{j,k,t,s}$ – second stage material inventory of component instance *(j,k)* at the end of time period *t* in scenario s, ≥ 0

$SY_{t,s}$ – second stage capacity utilization indicator (1, if the specific capacity utilization is met in time period *t* in scenario s, 0 otherwise)

The SATP model consists of:

- An objective function which maximizes a total expected profit.
- Order promising and order fulfillment constraints.
- Finished product production and inventory constraints.
- Production capacity constraints.
- Material requirements constraints.
- Raw material inventory constraints.

Maximize

$$\sum_{t=t_e+1}^{t_e+T} \sum_{i\in O\cup\hat{O}} frev_i \cdot FQ_{i,t} - \sum_{i\in O} fdeny_i \cdot (1-FZ_i) - \sum_{t=t_e+1}^{t_e+T} \sum_{i\in O\cup\hat{O}} \sum_{j\in M} \sum_{k\in M_j} fmatc_{j,k} \cdot FX_{i,j,k,t}$$

$$- \sum_{t=t_e+1}^{t_e+T} \sum_{i\in O\cup\hat{O}} ffphold_i \cdot FFP_{i,t} - \sum_{t=t_e+1}^{t_e+T-1} \sum_{i\in O\cup\hat{O}} \sum_{p=1}^{\min(\omega,t_e+T-t)} fwiphold_i \cdot FP_{i,(t+p)}$$

$$+ \sum_s prob_s \cdot \Bigg[\sum_{t=t_e+1}^{t_e+T} \sum_{i\in O^s} sfrev_{i,s} \cdot SQ_{i,t,s} - \sum_{i\in O^s} sdeny_{i,s} \cdot (1-SZ_{i,s})$$

$$- \sum_{t=t_e+1}^{t_e+T} \sum_{i\in O^s} \sum_{j\in M} \sum_{k\in M_j} smatc_{j,k,s} \cdot SX_{i,j,k,t,s} - \sum_{t=t_e+1}^{t_e+T} \sum_{i\in O^s} sfphold_{i,s} \cdot SFP_{i,t,s}$$

$$- \sum_{t=t_e+1}^{t_e+T} \sum_{j\in M} \sum_{k\in M_j} smatinvh_{j,k,s} \cdot SR_{j,k,t,s} - \sum_{t=t_e+1}^{t_e+T-1} \sum_{i\in O\cup\hat{O}} \sum_{p=1}^{\min(\omega,t_e+T-t)} swiphold_{i,p,s} \cdot SP_{i,(t+p),s}$$

$$- \sum_{t=t_e+1}^{t_e+T} undrutilp_{t-t_e} \cdot (1-SY_{t,s}) \Bigg]$$

Our objective function maximizes the total expected profit by subtracting material costs, finished product holding costs and work in process holding costs from revenues and adds the expected profit (revenue minus costs) from future orders. A denying penalty is incurred and is based on the long-term impact of denying a particular order plus the short term impact of not receiving the revenue from the current sale. The denying penalty can be express by $\alpha_i + \beta_i * rev_i * lb_i$ where α_i is a long term impact value, β_i is a short term impact value multiply by revenue of order *i* and a lower bound of promised quantity to represent the lowest estimate of the lost sale.

Subject to

Order promising and order fulfillment:

$$\sum_{t \in W_i} FD_{i,t} = FZ_i \qquad \forall i \in O \qquad (2.1)$$

$$\sum_{t \notin W_t} FD_{i,t} \leq 0 \qquad \forall i \in O \qquad (2.2)$$

$$FQ_{i,t} \geq flb_i \cdot FD_{i,t} \qquad \forall i \in O, t \in t_e(T] \qquad (2.3)$$

$$FQ_{i,t} \leq fub_i \cdot FD_{i,t} \qquad \forall i \in O, t \in t_e(T] \qquad (2.4)$$

$$\sum_{t \in W_s^i} SD_{i,t,s} = SZ_{i,s} \qquad \forall i \in O^s, s \in S \qquad (2.5)$$

$$\sum_{t \notin W_t} SD_{i,t,s} \leq 0 \qquad \forall i \in O^s, s \in S \qquad (2.6)$$

$$SQ_{i,t,s} \geq slb_i \cdot SD_{i,t,s} \qquad \forall i \in O^s, s \in S, t \in t_e(T] \qquad (2.7)$$

$$SQ_{i,t,s} \leq sub_i \cdot SD_{i,t,s} \qquad \forall i \in O^s, s \in S, t \in t_e(T] \qquad (2.8)$$

$$FQ_{i,\hat{t}} = f\hat{q}_i \qquad \forall i \in \hat{O} \qquad (2.9)$$

$$FQ_{i,t} = 0 \qquad \forall i \in \hat{O}, t \neq \hat{t}, t \in t_e(T] \qquad (2.10)$$

The model determines whether to accept new orders in constraint (2.1). If the order is accepted then a due date and promised quantity will be determined by constraint (2.2)-(2.4). Since the SATP model will be executed repeatedly over a planning horizon, any previously promised orders from a previous execution are kept in the current execution as shown in constraint (2.9). Let $f\hat{q}_i$ be a previously promised quantity for order i and $\hat{t}$ be a previously promised due date or time period.

Finished product production and inventory:

$$FFP_{i,(t=t_e)} = 0 \qquad \forall i \in O \qquad (2.11)$$

$$SFP_{i,(t=t_e),s} = 0 \qquad \forall i \in O^s, s \in S \qquad (2.12)$$

$$FFP_{i,(t=t_e)} = f\hat{p}_{i,(t=t_e)} \qquad \forall i \in \hat{O} \qquad (2.13)$$

$$FP_{i,t} = 0 \qquad \forall i \in O, t \in t_e(\omega] \qquad (2.14)$$

$$SP_{i,t,s} = 0 \qquad \forall i \in O^s, s \in S, t \in t_e(\omega] \qquad (2.15)$$

$$FP_{i,t} = f\hat{p}_{i,t} \qquad \forall i \in \hat{O}, t \in t_e(\omega] \qquad (2.16)$$

$$FFP_{i,(t-1)} + FP_{i,t} = FFP_{i,t} + FQ_{i,t} \qquad \forall i \in O \cup \hat{O}, t \in t_e(T] \qquad (2.17)$$

$$SFP_{i,(t-1),s} + SP_{i,t,s} = SFP_{i,t,s} + SQ_{i,t,s} \qquad \forall i \in O^s, s \in S, t \in t_e(T] \qquad (2.18)$$

$$FFP_{i,(t+T)} = 0 \qquad \forall i \in O \cup \hat{O} \qquad (2.19)$$

$$SFP_{i,(t+T),s} = 0 \qquad \forall i \in O^s, s \in S \qquad (2.20)$$

We specify the initial finished product inventory to be zero for new order and equal to ending inventory from previous execution for promised orders by constraints

(2.11)-(2.13). A production quantity from time period $t_e + 1$ to $t_e + \omega$ for new orders, if accepted, is zero because the production has not completed, this is done by constraint (2.14)-(2.15). On the other hand, a production quantity from time period $t_e + 1$ to $t_e + \omega$ is equal to previously promised production quantity that is still in the production process and scheduled to be completed in those time periods. It is carried over from previous execution for promised orders. This is done by constraint (2.16). An inventory balancing is done by constraints (2.17)-(2.18). An ending inventory is zero in constraints (2.19)-(2.20).

Production capacity:

$$\sum_{i \in O \cup \hat{O}} FP_{i,t} + \sum_{i \in O^s} SP_{i,t,s} \geq utiltrgt_{t-t_e} \cdot prodcap_t \cdot SY_{t,s} \qquad \forall s \in S, t \in t_e(T] \qquad (2.21)$$

$$\sum_{i \in O \cup \hat{O}} FP_{i,t} + \sum_{i \in O^s} SP_{i,t,s} \leq prodcap_t \qquad \forall s \in S, t \in t_e(T] \qquad (2.22)$$

A production capacity in time period *t* denoted by $prodcap_t$ must meet a utilization target which is expressed in percentage of the production capacity, denoted by $utiltrgt_{t-t_e}$. Otherwise an under utilization penalty denoted by $undrutilp_{t-t_e}$ will incur in the objective function. This is done by constraint (2.21). The production capacity in each time period *t* and in each scenario *s* cannot exceed the capacity limit. This is done by constraint (2.22).

Material requirements:

$$\sum_{k \in M_j^i} FX_{i,j,k,t} = fbom_{i,j} \cdot FP_{i,(t+\omega)} \qquad \forall j \in M, i \in O \cup \hat{O}, t \in t_e(T-\omega] \qquad (2.23)$$

$$\sum_{k \in M_j \backslash M_j^i} FX_{i,j,k,t} = 0 \qquad \forall j \in M, i \in O \cup \hat{O}, t \in t_e(T-\omega] \qquad (2.24)$$

$$\sum_{k \in M_j} FX_{i,j,k,t} = 0 \qquad \forall j \in M, i \in O \cup \hat{O}, t \in t_e(T-\omega(\omega] \qquad (2.25)$$

$$\sum_{k \in M_j^i} SX_{i,j,k,t,s} = sbom_{i,j} \cdot SP_{i,(t+\omega),s} \qquad \forall j \in M, i \in O^s, s \in S, t \in t_e(T-\omega] \qquad (2.26)$$

$$\sum_{k \in M_j \backslash M_j^i} SX_{i,j,k,t,s} = 0 \qquad \forall j \in M, i \in O^s, s \in S, t \in t_e(T-\omega] \qquad (2.27)$$

$$\sum_{k \in M_j} SX_{i,j,k,t,s} = 0 \qquad \forall j \in M, i \in O^s, s \in S, t \in t_e(T-\omega(\omega] \qquad (2.28)$$

Materials required to produce a finished product must be equal to a specification according to bill of material. This is done by constraint (2.23). Customers can specify their preference of suppliers for a certain kind of component by constraint (2.24). The constraint forces components supplied by suppliers that are not in customers' preference set M_j^i to be zero. The model does not allow a production process with ω production lead time to cross over from the current planning horizon to the next one by using constraint (2.25).

Raw material inventory:

$$SR_{j,k,(t=t_e),s} = \hat{r}_{j,k,(t=t_e)} \qquad \forall k \in M_j, j \in M, s \in S \qquad (2.29)$$

$$SR_{j,k,(t-1),s} + planrect_{j,k,t} = SR_{j,k,t,s} + \sum_{i \in O \cup \hat{O}} FX_{i,j,k,t} + \sum_{i \in O^s} SX_{i,j,k,t,s}$$

$$\forall k \in M_j, j \in M, s \in S, t \in t_e(T] \qquad (2.30)$$

Raw material inventory is considered only for future orders because every time the model is executed, we are in time period t_e, so there is no inventory at the present time for future orders. The raw material inventory involves planned receipt of raw materials and the number of components used to assemble finished products from the promised orders, new orders and future orders. The raw material inventory balancing constraint is (2.29). Initial raw material inventory for each scenario is the ending inventory of raw material from the last execution. This is done through constraint (2.30).

Simplified SATP Problem Formulation and Description:

Since the original problem could not be solved to optimality, we created a simpler problem that has the same underlying structure. The simplified problem can be solved to optimality by current commercial solvers. We will use the result from CPLEX as a benchmark for performance of the decomposition algorithms. Using the simplified version of the SATP, it is also easier to gain insights of the problem structure. Another advantage of using the simplified version of the SATP problem is we can incorporate as many scenarios as possible into the model to study the performance of the decomposition methods. This contribution could compensate for the lack of the multitude of constraints that the original problem possesses. The following is the simplified version of the SATP problem.

Indices

i – orders

s – scenarios

t –delivery time window for order *i*

Sets

O - newly-arrived orders

O^s - future orders

T - time periods

S - scenarios

W_i - set of delivery time window *t* for newly-arrived order *i*

W_i^s - set of delivery time window *t* for future order *i* in scenario *s*

Data

flb_i – lower bound on delivered quantity for order *i* for first stage variables

fub_i – upper bound on delivered quantity for order *i* for first stage variables

$frev_i$ – revenue from order *i* if it is accepted for first stage variables

$fhold_i$ – inventory holding cost per unit of product per time period for first stage variables

$fprodc_i$ - production cost per unit of product for first stage variables

$prob_s$ – probability associated to each scenario *s*

$capacity_t$ – production capacity in time *t*

$slb_{i,s}$ – lower bound on delivered quantity for order *i* in scenario *s* for second stage variables

$sub_{i,s}$ – upper bound on delivered quantity for order *i* in scenario *s* for second stage variables

$srev_{i,s}$ – revenue from order *i* if it is accepted in scenario *s* for second stage variables

$shold_{i,s}$ – inventory holding cost per unit of product per time period in scenario *s* for second stage variables

$sprodc_{i,s}$ - production cost per unit of product in scenario *s* for second stage variables

Variables

$FQ_{i,t}$ – delivered quantity for order *i* in time period *t*, >=0

$FD_{i,t}$ – delivered date for order *i* in time *t*, 1 if time t is selected to be a delivered date for order *i*, 0 otherwise, {0,1}

$FP_{i,t}$ – production quantity for order *i* in time *t*, >=0

$FINV_{i,t}$ – inventory level of order *i* in time *t*, >=0

$SQ_{i,t,s}$ – delivered quantity for order *i* in time period *t* in scenario *s*, >=0

$SD_{i,t,s}$ – delivered date for order *i* in time *t* in scenario *s*, 1 if time *t* is selected to be a delivered date for order *i* in scenario *s*, 0 otherwise, {0,1}

$SP_{i,t,s}$ – production quantity for order *i* in time *t* in scenario *s*, >=0

$SINV_{i,t,s}$ - inventory level of order *i* in time *t* in scenario *s*, >=0

Maximize

$$\sum_{t\in T}\sum_{i\in O} frev_i \cdot FQ_{i,t} - \sum_{t\in T}\sum_{i\in O} fprodc_i \cdot FP_{i,t} - \sum_{t\in T}\sum_{i\in O} fhold_i \cdot FINV_{i,t} +$$
$$\sum_{s\in S} prob_s \cdot (\sum_{t\in T}\sum_{i\in O^s} srev_{i,s} \cdot SQ_{i,t,s} - \sum_{t\in T}\sum_{i\in O^s} sprodc_{i,s} \cdot SP_{i,t,s} - \sum_{t\in T}\sum_{i\in O^s} shold_{i,s} \cdot SINV_{i,t,s})$$

Subject to

$$\sum_{i\in O} FP_{i,t} + \sum_{i\in O^s} SP_{i,t,s} \le capacity_t \qquad \forall t\in T, s\in S \qquad (2.31)$$

$$FQ_{i,t} \ge flb_i \cdot FD_{i,t} \qquad \forall i\in O, t\in T \qquad (2.32)$$

$$FQ_{i,t} \le fub_i \cdot FD_{i,t} \qquad \forall i\in O, t\in T \qquad (2.33)$$

$$\sum_{t\in W_i} FD_{i,t} \le 1 \qquad \forall i\in O \qquad (2.34)$$

$$\sum_{t\notin W_t} FD_{i,t} = 0 \qquad \forall i\in O \qquad (2.35)$$

$$FINV_{i,(t-1)} + FP_{i,t} = FINV_{i,t} + FQ_{i,t} \qquad \forall i\in O, t\in T \qquad (2.36)$$

$$FINV_{i,t=0} = 0 \qquad \forall i\in O, t\in T \qquad (2.37)$$

$$SQ_{i,t,s} \geq slb_{i,s} \cdot SD_{i,t,s} \qquad \forall i \in O^s, t \in T, s \in S \quad (2.38)$$

$$SQ_{i,t,s} \leq sub_{i,s} \cdot SD_{i,t,s} \qquad \forall i \in O^s, t \in T, s \in S \quad (2.39)$$

$$\sum_{t \in W_i^s} SD_{i,t,s} \leq 1 \qquad \forall i \in O^s, s \in S \quad (2.40)$$

$$\sum_{t \notin W_t^s} SD_{i,t,s} = 0 \qquad \forall i \in O^s, s \in S \quad (2.41)$$

$$SINV_{i,(t-1),s} + SP_{i,t,s} = SINV_{i,t,s} + SQ_{i,t,s} \qquad \forall i \in O^s, t \in T, s \in S \quad (2.42)$$

$$SINV_{i,t=0,s} = 0 \qquad \forall i \in O^s, t \in T, s \in S \quad (2.43)$$

Non negativity

$FQ_{i,t} \geq 0$

$SQ_{i,t,s} \geq 0$

$FP_{i,t} \geq 0$

$SP_{i,t,s} \geq 0$

$FINV_{i,t} \geq 0$

$SINV_{i,t,s} \geq 0$

Binary

$FD_{i,t}\{0,1\}$

$SD_{i,t,s}\{0,1\}$

We assume a manufacturer offers one product, i.e. a PC with fixed configuration. The model represents a high-level planning production that disregards several components within PCs such as monitors, hard disk drives, optical drives, etc. We assume an order represents the numbers of PCs required. Our purpose is to learn what decompositions work for this structure. For simplicity, we will not run our experiment as a simulation model in which it will be run repeatedly over a time horizon. Instead, we will run the model once at the end of the business day. Therefore, a batch length is one time period. Order promising decisions will be made for orders collected during the business day by taking into account future orders for each scenario. The model offers two types of flexibilities, due date range and promised quantity range. It can be seen that the structure of the simplified problem is the same as the full version of the problem.

Our objective function is to maximize an overall profit by subtracting a production cost and a holding cost term from a revenue term. Constraint (2.31) ensures that production in the first-stage and the second-stage do not exceed a production capacity. Constraints (2.32) and (2.33) define a lower bound and an upper bound of quantity of product to be delivered for an accepted order. Constraints (2.34) and (2.35) ensure that each order within an allowed window is accepted only once. Those that are not in the allowed window are not accepted. Constraint (2.36)-(2.37) keeps track of inventory, production and promised quantity in time t for the first-stage. Variables in constraints (2.32)-(2.37) are the first-stage variables. Constraints (2.38)-(2.39) define a lower bound and an upper bound of quantity of product to be delivered for the future order. Constraints (2.40)-(2.41) ensure that each future order within an allowed window is accepted only once. Those that are not in the allowed window are not accepted. Constraint (2.42)-(2.43) keeps track of inventory, production and promised quantity in time t for the second-stage. Variables in constraints (2.38)-(2.43) are the second-stage variables.

3. Comparing Decomposition Methods

In this chapter we discuss the details of decomposition methods which were introduced in chapter 1. Details of the *Dantzig-Wolfe (D-W) decomposition* or *column generation* and the *Bender's decomposition* will be discussed. We then demonstrate how to apply both Bender's decomposition and the D-W decomposition methods to the simplified version of the SATP problem. The decomposed problem formulations corresponding to each decomposition method will be presented. Branching rules and branching algorithms will be discussed after the discussion of the D-W decomposition to complete the *Branch-and-Price* algorithm. The last section will discuss about a selection of the decomposition methods to solve our problem based on the underlying structure of the SATP problem.

We begin by referring to a *block-angular* structure in figure 1.3 to demonstrate the structure of the simplified version of the SATP problem. From now on, every time we say "the SATP problem", we refer to "the simplified version of the SATP problem". In the block-angular structure, the constraints in each block *i* of $B_i x = b_i$ appear to be independent from each other. However, the constraints $Ax = b$ bind all $B_i x = b_i$ together into one large problem. If the constraints $Ax = b$ are removed, the problem can be decomposed into several independent sub-problems which are smaller and easier to solve. The constraints $Ax = b$ correspond to constraints (2.31) in the SATP problem. These constraints link all other constraints together because they contain both the first-stage and the second-stage variables, *FP* and *SP* respectively. The constraints $B_i x = b_i$ where i=1 correspond to constraints (2.32)-(2.37). These constraints represent the first-stage decision because they contain only the first-stage variables (*FP, FD, FQ* and *FINV*). The constraints $B_i x = b_i$ where i>1 correspond to constraints (2.38)-(2.43). These constraints represent the second-stage decision because they contain the second-stage variables (*SP, SD, SQ* and *SINV*). These variables are separated by scenarios, for example, $SD_{i,t,1}$ and other second-stage variables with the scenario index s=1 are in a block where i=2. The other second-stage variables with the scenario index s=2 appear in a block where i=3 and so forth.

Dantzig-Wolfe Decomposition

In Chapter 1, we introduced the D-W decomposition method. In this section, we will discuss details of the algorithm as it applies to the SATP problem. The SATP problem is decomposed into a master problem and sub-problems. The master problem is constructed from constraints 2.31. The rest of the constraints create the sub-problems. From [Dantzig and Wolfe 1960], the master problem contains a number of columns or extreme points. Each column is associated with a variable. An optimal solution to the master problem can be represented by a linear convex combination of those columns. We perform the D-W decomposition in two steps or phases. This is the same as the two-phase method employed by the simplex method. Phase_I identifies if the SATP problem is feasible. It initializes and populates feasible columns to the master problem to begin

Phase_II. If the sub-problems cannot provide feasible columns to the master problem, the SATP problem is infeasible. The differences of the master problem in Phase_I and Phase_II are the objective function and constraints. Figure 3.1 shows the master problem phase_I formulation.

Min $\quad Excess$

Subject to
$$\sum_k \sum_i FP^*_{i,t,k} \cdot \alpha_k + \sum_{k[s]} \sum_i SP^*_{s,i,t,k[s]} \cdot \alpha_{s,k[s]} - Excess \le capacity_t \quad \forall s,t \qquad (3.1)$$

$$\sum_k \alpha_k = 1 \qquad (3.2)$$

$$\sum_{k[s]} \alpha_{s,k[s]} = 1 \quad \forall s \qquad (3.3)$$

$$\alpha_k, \alpha_{s,k[s]}, Excess \ge 0 \qquad (3.4)$$

Figure 3.1 Master problem Phase_I formulation

As stated before, the purpose of Phase_I is to initialize and populate feasible columns to the master problem. The objective function of the master problem Phase_I is to minimize a variable, labeled *Excess*. Notice that $FP^*_{i,t,k}$ and $SP^*_{s,i,t,k[s]}$ with a superscript * mean they are parameters not variables, sent to the master problem Phase_I as columns. An index *k* means a number of the first-stage columns added to the master problem. An index *k*[*s*] means a number of the second-stage columns added to the master problem. The second-stage column itself is indexed by a scenario. Each scenario corresponds to each of the second-stage subproblems. Each subproblem provides one column to the master problem per iteration. Therefore, there are multiple columns added to the master problem per iteration.

An α_k is a variable associates to the first-stage column while an $\alpha_{s,k[s]}$ is a variable associates to the second-stage column in each scenario. Constraint 3.1 is a production capacity which is removed from the SATP problem. Notice that the *Excess* variable is a slack variable to constraint 3.1. Constraints 3.2 and 3.3 are linear convex combination of columns. Figure 3.2 shows the first-stage subproblems Phase_I formulation.

Max $$-\sum_i \sum_t \left(\sum_s cap\_pr_{s,t} \right) \cdot FP_{i,t} - fconvex\_pr$$

Subject to

$$FQ_{i,t} \geq flb_{i,t} \cdot FD_{i,t} \qquad \forall i,t \qquad (3.5)$$

$$FQ_{i,t} \leq fub_{i,t} \cdot FD_{i,t} \qquad \forall i,t \qquad (3.6)$$

$$\sum_{t \in W_i} FD_{i,t} \leq 1 \qquad \forall i \qquad (3.7)$$

$$FD_{i,t \notin W_i} = 0 \qquad \forall i, t \notin W_i \qquad (3.8)$$

$$FINV_{i,(t-1)} + FP_{i,t} = FINV_{i,t} + FQ_{i,t} \qquad \forall i,t \qquad (3.9)$$

$$FINV_{i,t=0} = 0 \qquad \forall i \qquad (3.10)$$

Figure 3.2 First-stage subproblem Phase_I formulation

An objective function of the first-stage sub-problem Phase_I consists of dual prices associate to the constraint 3.1, $cap\_pr_{s,t}$, and a dual price associate to the constraint 3.2, $fconvex\_pr$. This constraint is the convexity constraint that says that only possible stage-1 schedule can be chosen. Constraints 3.5 to 3.10 are the same as the ones in the SATP problem. Figure 3.3 shows the second-stage subproblems Phase_I formulation.

Max $$-\left(\sum_i \sum_t cap\_pr_{s,t} \cdot SP_{s,i,t}\right) - sconvex\_pr_s$$

Subject to
$$SQ_{s,i,t} \geq slb_{s,i,t} \cdot SD_{s,i,t} \qquad \forall s,i,t \qquad (3.11)$$

$$SQ_{s,i,t} \leq sub_{s,i,t} \cdot SD_{s,i,t} \qquad \forall s,i,t \qquad (3.12)$$

$$\sum_{t \in W_i^s} SDs,i,t \leq 1 \qquad \forall s,i \qquad (3.13)$$

$$SD_{s,i,t \notin W_i^s} = 0 \qquad \forall s,i,t \notin W_i^s \qquad (3.14)$$

$$SINV_{s,i,(t-1)} + SP_{s,i,t} = SINV_{s,i,t} + SQ_{s,i,t} \qquad \forall s,i,t \qquad (3.15)$$

$$SINV_{s,i,t=0} = 0 \qquad \forall s,i \qquad (3.16)$$

Figure 3.3 Second-stage subproblems Phase_I formulation

An objective function of the second-stage subproblems Phase_I consists of the same dual prices associate to the constraint 3.1, $cap\_pr_{s,t}$ as appear in the first-stage subproblems Phase_I, and the dual prices associate to the constraints 3.3, $sconvex\_pr_s$, where similarly, constraints 3.3 indicate that only one possible schedule for each scenario can be chosen. Constraints (3.11) to (3.16) are the same as the ones in the SATP problem.

Phase_I begins by initializing $cap\_pr_{s,t} = 0$ and $fconvex\_pr = -1$, $sconvex\_pr_s = 1$ then solves subproblems as a MIP. Solutions from the subproblems constitute columns that will be added to the master problem. One column consists of two parts, an objective function coefficient and constraint coefficients. First we compute an objective function coefficient of a column that will be added to the master problem. The objective function coefficient of the added column is a net profit which the current solution produces. The coefficient for the first-stage decision is computed via the following formula. It is indexed by a column index *k*.

$$F\_net\_profit_k = \sum_i \sum_t frev_i \cdot FQ_{i,t} - \sum_i \sum_t fprodc_i \cdot FP_{i,t} - \sum_i \sum_t fhold_i \cdot FINV_{i,t}$$

The coefficient for the second-stage decision is computed via the following formula. It is indexed by scenario *s* and column index *k*[*s*].

$$S\_net\_profit_{s,k[s]} =$$

$$prob_s \cdot \left(\sum_i \sum_t srev_{s,i} \cdot SQ_{s,i,t} - \sum_i \sum_t sprodc_{s,i} \cdot SP_{s,i,t} - \sum_i \sum_t shold_{s,i} \cdot SINV_{s,i,t} \right)$$

Note that the objective function coefficient is not a reduced cost of the column. The reduced cost of the column is determined by the objective function value of the subproblem. We formulate the subproblem as a maximization so that the positive reduced cost or objective function value will price out favorably. Once the objective function coefficients of the columns were computed then the constraint coefficients are constructed to constitute a complete column. The coefficient of constraint 3.1 are $FP^*_{i,t,k}$ and $SP^*_{s,i,t,k[s]}$ which are the solutions from the subproblems. The coefficient of constraint 3.2 and 3.3 are 1s. The complete first-stage column and second-stage column are displayed in figure 3.4.

$$\begin{array}{cc} \text{first-stage column} & \text{second-stage column} \\ \begin{bmatrix} F\_net\_profit_k \\ FP_{i,t,k} \\ 1 \end{bmatrix} & \begin{bmatrix} S\_net\_profit_{s,k[s]} \\ SP_{s,i,t,k[s]} \\ 1 \end{bmatrix} \end{array}$$

Figure 3.4 Coefficients of columns added to master problem

The subproblems are solved to optimality to guarantee an optimal solution to the SATP problem. This means that when the solution procedures ends, there are no columns with positive reduced cost, i.e. the best column that each of the subproblems can produce given the current dual prices has are non-positive. If the reduced cost determined by the subproblem is positive, the column in figure 3.4 is added to master problem. The master problem is solved as an L.P. relaxation. Dual prices of constraints

3.1, 3.2 and 3.3 are updated in the subproblems' objective function. The algorithm repeats by solving the subproblems. There are two possible results at the end of Phase_I.. When the procedure ends, either *Excess=0,* and we can continue to Phase II or *Excess>0,* and the overall problem is infeasible.

Before Phase_II can begin, the master problem and the subproblems need to be modified. Their objective functions need to be changed from Phase_I to the following objective functions. The new master problem's objective function is:

$$Max \quad \sum_{k} F\_net\_profit_{k} \cdot \alpha_{k} + \sum_{s}\sum_{k[s]} S\_net\_profit_{s,k[s]} \cdot \alpha_{s,k[s]}$$

The new subproblems' objective function for the first-stage is:

$$Max \quad \sum_{i}\sum_{t} frev_{i} \cdot FQ_{i,t} - \sum_{i}\sum_{t}\left(fprodc_{i} + \sum_{s} cap\_pr_{s,t} \right) \cdot FP_{i,t}$$
$$- \sum_{i}\sum_{t} fholdi \cdot FINV_{i,t} - fconvex\_pr$$

The new subproblems' objective function for the second-stage is:

$$Max \quad \left(prob_{s} \cdot \left(\sum_{i}\sum_{t} srev_{s,i} \cdot SQ_{s,i,t} - \sum_{i}\sum_{t} sprodc_{s,i} \cdot SP_{s,i,t} - \sum_{i}\sum_{t} shold_{s,i} \cdot SINV_{s,i,t} \right)\right)$$
$$- \sum_{i}\sum_{t} cap\_pr_{s,t} \cdot SP_{s,i,t} - sconvex\_pr_{s}$$

Everything else remains the same as in Phase I. The constraints of the subproblems in Phase_II are the same as Phase_I. The only changes are the objective functions. The constraint 3.1 in the master problem is modified too. The *Excess* variable is taken out of the constraint 3.1. The constraint 3.1 is:

$$\sum_{k}\sum_{i} FP^{*}{}_{i,t,k} \cdot \alpha_k + \sum_{k[s]}\sum_{i} SP^{*}{}_{s,i,t,k[s]} \cdot \alpha_{s,k[s]} \le capacity_t \qquad \forall s,t$$

Other constraints in the master problem are the same as Phase_I. All columns that were added to the master problem in Phase_I have been carried to Phase_II.

Phase_II begins by solving the master problem. Solving this problem provides new dual prices to the subproblems and the solution algorithm continues as in Phase I. Phase_II terminates when no columns with positive reduced cost can be found by the subproblems. Hence, D-W decomposition has found a linear-programming relaxation to the SATP problem. But, the D-W solution is in terms of α_k and $\alpha_{s,k[s]}$. We translate that back to our original variables of the first stage, by calculating:

$$FP^{*}_{i,t} = \sum_{k} FP^{*}_{i,t,k} \cdot \alpha_k \quad (3.16)$$

first-stage column **second-stage column**

$$\begin{bmatrix} FP^*_{i,t,k} \\ FD^*_{i,t,k} \\ FQ^*_{i,t,k} \\ FINV^*_{i,t,k} \end{bmatrix} \qquad \begin{bmatrix} SP^*_{s,i,t,k[s]} \\ SD^*_{s,i,t,k[s]} \\ SQ^*_{s,i,t,k[s]} \\ SINV^*_{s,i,t,k[s]} \end{bmatrix}$$

Figure 3.5 Columns created from solving subproblems

Figure 3.5 shows what each column generated by subproblems consists of. An optimal solution to the SATP problem is computed by taking the values of α_k , and $\alpha_{s,k[s]}$ and using these to translate our solution back to the original variables described for the SATP problem.

Optimal first-stage solution **Optimal second-stage solution**

$$\begin{bmatrix} FP^*_{i,t} \\ FD^*_{i,t} \\ FQ^*_{i,t} \\ FINV^*_{i,t} \end{bmatrix} = \sum_k \begin{bmatrix} FP^*_{i,t,k} \\ FD^*_{i,t,k} \\ FQ^*_{i,t,k} \\ FINV^*_{i,t,k} \end{bmatrix} \cdot \alpha_k \qquad \begin{bmatrix} SP^*_{s,i,t} \\ SD^*_{s,i,t} \\ SQ^*_{s,i,t} \\ SINV^*_{s,i,t} \end{bmatrix} = \sum_{k[s]} \begin{bmatrix} SP^*_{s,i,t,k[s]} \\ SD^*_{s,i,t,k[s]} \\ SQ^*_{s,i,t,k[s]} \\ SINV^*_{s,i,t,k[s]} \end{bmatrix} \cdot \alpha_{s,k[s]}$$

Figure 3.6 Computation of the optimal solution

But, this linear combination may not result in an integer solution where each *FD* and *SD* variable takes on the value either 0 or 1. We therefore need to develop a branching procedure that will allow us to find the integer optimal solution. This branching procedure needs to be consistent with the overall master problem, so that the same technology can be used on every branch of the branch-and-bound tree.

Specifically, we need to create a dichotomy that cuts off the current solution and allows our process to proceed. That is, we must be able to find columns with positive reduced cost exist. If such columns exist, we re-solve the master problem and the column generation process continues until there are no columns with positive reduced cost can be found. This is to ensure that every column with positive reduced cost is accounted for at each node of the branching tree to guarantee optimality to the problem. The algorithm that performs column generation at each node of the branching tree is called the *Branch-and-Price* algorithm [Barnhart et al. 1996].

Next we describe our branching rules and its algorithm. Once a relaxation of the SATP problem is solved to optimality, variables *FDs* and *SDs* are checked for integrality. The first ones checked are the *FDs*. If all *FDs* are integer then the *SDs* are checked. A reason we select *FDs* to branch on first is because, when all *FD* variables are integer, we can terminate with a feasible, although possibly non-optimal, solution. If we stop at this point, we may have overestimated the second-stage decision, but we still have both a bound and a feasible solution. One can argue: Why waste computational time on assuring *integral* optimality to the second-stage decisions when these decisions are stochastic in nature? If we stop at the end of this phase, we are allowing a linear combination of alternative decisions in the second phase, which may be a good thing to do given that we are less sure of which of the scenarios might eventuate.

We have therefore decided to *only* branch on first-stage variables. Our next branching rule is we select among variables *FDs* which is the closest to 0.5 to branch on. Once we have selected the variable *FD*, the next rule is to always branch on the one-branch since we are maximizing net profit and we would like to accept orders. Once we have selected a variable *FD* and forced it to one by fixing its lower bound to one on the "one-branch" (or to zero on the "zero branch"), we next delete columns in the master problem which contain the selected variable with value equals to zero (one). This is done by fixing the variables α_k to zero (one) in the master problem. Then the column generation repeats by re-solving the master problem from Phase_II. By changing the variables' bounds in variable fixing and column deletion, both the subproblems' structure and the master problem's structure are not affected. After the column generation stops, the algorithm creates a new node. We summarize the *Branch-and-Price* algorithm in term of activities performed in each node. The algorithm is initialized by solving the SATP problem at the root node. Set a current best bound to zero. Then activities performed in each node are:

1. Check if the problem is infeasible.
    a. If so, clear all columns in the master problem. Start solving the problem from Phase_I. Once the problem is solved, check for infeasibility again. If the problem is still infeasible, the current node is pruned by infeasibility. Go to step 5. If it is feasible, go to step 2.
    b. If it is feasible, proceed to step 2.
2. Check if the objective function value is greater than the current bound.

    a. If so, go to step 3.
    b. If not, attempt to generate columns that may improve the bound. If no columns exist, then the current node is pruned by bound. Go to step 5.

3. Looks for fractional variable *FDs*.
    a. If there is none, make sure that there are no columns that could improve the current solution. If not, then the current node is pruned by optimality. In this case, we store the incumbent solution and update the current bound to the current objective function value. Go to step 5.
    b. If there are fractional variable *FDs*, go to step 4.
4. Perform a variable selection of *FDs* then check how many times the current node is visited.
    a. If it is the first time then create a new node and branch on the one-branch. Perform a variable fixing and a column deletion. Resolve the problem by starting from Phase_II. Go to step 1.
    b. If it is the second time then create a new node and branch on the zero-branch. The variable fixing and the column deletion of the search on the zero-branch are opposite to that of the one-branch. Perform a variable fixing and a column deletion. Resolve the problem by starting from Phase_I. Go to step 1.
    c. If it is the third time then go back to the parent node. Then repeat step 4 without performing a variable selection.

5. If any of three kinds of pruning occurs, go back to the parent node. Then go to step 4 without performing a variable selection.

The algorithm searches the branching tree until all nodes are pruned. The current incumbent solution is reported as the best feasible integer solution to the SATP problem. As we said before, the reported solution overestimates the optimal solution of the problem but we believe the solution obtained is good enough for these first-stage decisions. We conclude the *Branch-and-Price* algorithm at this point. We describe the *Bender's decomposition* algorithm applied to the SATP problem in the next section.

Bender's decomposition

In this section, we describe a problem formulation how Bender's decomposition can be applied to the SATP problem. As stated in chapter 1, in Bender's decomposition, the problem is decomposed into a master problem and subproblems. The master problem determines the first-stage decision. The subproblems then use that solution to determine the second-stage decision. We first present the master problem and the subproblem formulations, we then describe a solution approach. The master problem formulation is shown in figure 3.7.

$$Max \quad \sum_i \sum_t frev_i \cdot FQ_{i,t} - \sum_i \sum_t fprodc_i \cdot FP_{i,t} - \sum_i \sum_t fhold_i \cdot FINV_{i,t} + Min\_Stage2$$

$$Subject\ to \quad \sum_i FP_{i,t} \le capacity_t \qquad \forall t \qquad (3.17)$$

$$FQ_{i,t} \ge flb_i \cdot FD_{i,t} \qquad \forall i,t \qquad (3.18)$$

$$FQ_{i,t} \le fub_i \cdot FD_{i,t} \qquad \forall i,t \qquad (3.19)$$

$$\sum_{t \in W_i} FD_{i,t} \le 1 \qquad \forall i \qquad (3.20)$$

$$FD_{i,t} = 0 \qquad \forall i,t \notin W_i \qquad (3.21)$$

$$FINV_{i,t=0} = 0 \qquad \forall i \qquad (3.22)$$

$$FINV_{i,(t-1)} + FP_{i,t} = FINV_{i,t} + FQ_{i,t} \qquad \forall i,t \qquad (3.23)$$

$$Min\_Stage2 \le \sum_s \sum_t cap\_pr_{s,t,k} \cdot \left(capacity_t - \sum_i FP_{i,t} \right) +$$

$$\sum_s \sum_i spromise\_pr_{s,i,k} + \sum_s \sum_i \sum_t sd\_bnd\_pr_{s,i,t,k} \quad \forall k \qquad (3.24)$$

Figure 3.7 Bender's master problem formulation

The master problem has three new things which make it differ from a normal first-stage subproblem. There are new components to the objective function, and we have added constraint set 3.17 and constraint set 3.24. The objective function of the master problem has a new term, *Min_Stage*2, in addition to the normal objective function components of the first-stage decision. The *Min_Stage*2 is a variable which represents an upper bound of the second-stage decision. This variable is used to measure a gap between the upper bound of the second-stage decision determined by the first-stage decision and the second-stage decision determined by the subproblems. The Bender's algorithm terminates when the gap between these two objectives is sufficiently small (i.e. smaller than the preset tolerance). The *Min_Stage*2 variable also appears in constraint 3.24 which is constructed from dual prices of the subproblems. Details of constructing the constraint 3.24 will be discussed later. The constraint 3.24 is the optimality cut of the Bender's decomposition [Birge and Louveaux, 1997]. One such constraint is added to the master problem per iteration. Its purpose is to use information from the subproblems to adjust the first-stage decision. Finally, constraint 3.17 has been changed so that it now includes only the first-stage decisions.

Next we present the subproblem that must be solved to determine the dual prices that will be used to generate the Bender's cut. Specifically, the subproblem determines the second-stage decision given the first-stage decision in the previous solution to the master problem. Figure 3.8 shows a formulation of the Bender's subproblem.

$$Max \quad \sum_s prob_s \cdot \left(\sum_i \sum_t srev_{s,i} \cdot SQ_{s,i,t} - \sum_i \sum_t sprodc_{s,i} \cdot SP_{s,i,t} - \sum_i \sum_t shold_{s,i} \cdot SINV_{s,i,t} \right)$$

$$Subject\ to \quad \sum_i FP^*_{i,t} + \sum_i SP_{s,i,t} \le capacity_t \qquad \forall s,t \qquad (3.25)$$

$$SQ_{s,i,t} \ge slb_{s,i} \cdot SD_{s,i,t} \qquad \forall s,i,t \qquad (3.26)$$

$$SQ_{s,i,t} \le sub_{s,i} \cdot SD_{s,i,t} \qquad \forall s,i,t \qquad (3.27)$$

$$\sum_{t \in W_i^s} SD_{s,i,t} \le 1 \qquad \forall s,i \qquad (3.28)$$

$$SDs,i,t = 0 \qquad \forall s,i,t \notin W_i^s \qquad (3.29)$$

$$SINV_{s,i,t=0} = 0 \qquad \forall s,i \qquad (3.30)$$

$$SINV_{s,i,(t-1)} + SP_{s,i,t} = SINV_{s,i,t} + SQ_{s,i,t} \qquad \forall s,i,t \qquad (3.31)$$

$$0 \le SD_{s,i,t} \le 1 \qquad \forall s,i,t \qquad (3.32)$$

Figure 3.8 Bender’s subproblems formulation

In our computational study, presented in Chapter 4, we solve this subproblem only once per iteration. Multiple subproblems could be used to generate multiple of constraints 3.24 per iteration but we found that we were obtaining relatively tight cuts with only one cut per iteration. Note that, except for constraint set 3.25 and 3.32, the Bender’s subproblem is the same as the second-stage formulation of the simplified version of the SATP problem. The constraint 3.25 is a capacity constraint similar to constraint 2.31 with the first-stage decision having already been made. FP^* denotes the value of the first-stage decision. This information is therefore fixed when determining the second-stage decision. We added the constraint 3.32 which is a lower bound and an upper bound of the variable *SD* to the subproblem. This constraint set provides dual prices needed to construct the Bender's cuts. If we do not change the variable *SD*'s upper bound to the constraint set 3.32, the information for constructing the Bender's cuts is not complete.

Next we describe the Bender's decomposition algorithm. The algorithm begins by setting the first-stage decision $FP^*_{i,t} = 0$. We solve a relaxation of this subproblem and obtain dual prices. Figure 3.9 exhibits the dual prices of the constraints in the subproblem which are used to construct the Bender's cuts.

Constraints	**Dual prices**
$\sum_i FP^*_{i,t} + \sum_i SP_{s,i,t} \le capacity_t$	$cap\_pr_{s,t}$
$\sum_{t \in W_i^s} SD_{s,i,t} \le 1$	$promise\_pr_{s,i}$
$0 \le SD_{s,i,t} \le 1$	$sd\_bnd\_pr_{s,i,t}$

Figure 3.9 Dual prices associated to constraints in the Bender's subproblem

The dual prices obtained from constraint set 3.25 $(cap\_pr_{s,t})$ are the value of having additional capacity for the second-stage manufacturing process. The dual prices of constraint set 3.28 $(promise\_pr_{s,i})$ provide us with an additional profit of selling more of a promised order. The last information is the dual prices obtained from the constraint set 3.32 $sd\_bnd\_pr_{s,i,t}$. The constraint set is the variable *SD*'s upper bound. The dual prices imply that if the *SD*'s upper bound were greater than one, the second-stage profit would increase. Because the variable *SD* relates to the upper bound and the lower bound of the delivered quantity in the constraint 3.26 and 3.27. Dual prices of other constraints (3.26, 3.27, 3.29, 3.30 and 3.31) can not be used to construct the Bender's cuts because a construction of the Bender's cuts requires the dual prices of the constraints time a constant or a right-hand side [Birge and Louveaux, 1997]. Those constraints do not have constant terms or the constant terms are zero so their dual prices do not play any roles in the Bender's cuts. Figure 3.10 shows how the constraint 3.24 or the Bender's cuts are constructed. Note that the constant term of the capacity constraint in the subproblem (3.25), the first-stage decision is fixed but it becomes a variable *FP* when it is in the Bender's cuts or constraint 3.24 in the master problem. This is because the algorithm needs to adjust the first-stage decision bases on the information obtained from the subproblem in each iteration.

Constraints	**Dual prices**	**Constant term**
$\sum_i FP^*_{i,t} + \sum_i SP_{s,i,t} \le capacity_t$	$cap\_pr_{s,t}$	$(cap\_pr_{s,t} - \sum_i FP^*_{i,t})$
$\sum_{t \in W_i^s} SD_{s,i,t} \le 1$	$promise\_pr_{s,i}$	1
$0 \le SD_{s,i,t} \le 1$	$sd\_bnd\_pr_{s,i,t}$	1

Constraint 3.24

$$Min\_Stage2 \le \sum_s \sum_t cap\_pr_{s,t,k} \cdot \left(capacity_t - \sum_i FP_{i,t} \right) + \sum_s \sum_i (spromise\_pr_{s,i,k} \cdot 1) + \sum_s \sum_i \sum_t (sd\_bnd\_pr_{s,i,t,k} \cdot 1) \qquad \forall k \qquad (3.24)$$

Figure 3.10 Construction of the Bender's cuts

We then feed the information from the subproblem to the master problem through the constraint 3.24 constructed from the dual prices as shown in figure 3.7 and 3.10. Then the master problem is solved as a MIP problem. The first-stage decision (variable *FP*'s value) is sent to the subproblem. Then the algorithm repeats until a gap between *Min_Stage*2 and the subproblem objective function value is within a preset tolerance. When this tolerance is met, the algorithm terminates. The following is a summary of the Bender's decomposition algorithm. The Bender's algorithm begins with initialization of parameters. We preset the first-stage decision $FP^* = 0$, *Min_Stage*2 = *infinity* and a gap *tolerance* = 0.001. The gap is defined by a difference between *Min_Stage*2 and the subproblem objective function value. Then the algorithm proceeds as follows.

1. Solve a subproblem. Then check if the gap is less than the *tolerance*.
    a. If so, exit. The problem is optimal.
    b. If it is not, go to step 2.
2. Get dual prices from constraint 3.25, 3.28 and reduced cost of the variable *SD*.
3. Use the dual prices to construct constraint 3.24.
4. Solve a master problem.
5. Update the first-stage decision $FP^* = FP^k$ then go to step 1.

Bender's Decomposition versus Dantzig-Wolfe:

How does structure work for each?

Next we discuss about how each decomposition method uses the structure of the problem for the SATP problem. Recall that the SATP problem has a *block-angular structure* as described in Chapter 1, figure 1.3. We can consider this problem to have an *L-shaped structure,* with constraint 2.31 being the linking constraint that has both the first-stage and the second-stage variables since they share production capacity. Figure 3.11 shows how the *L-shaped structure* in constraint 2.31 is arranged. This example has two scenarios and two time periods per scenario.

$$\begin{array}{llll}
\sum_i FP_{i,t=1} & +\sum_i SP_{s=1,i,t=1} & & \le capacity_{t=1} \\
\sum_i FP_{i,t=2} & +\sum_i SP_{s=1,i,t=2} & & \le capacity_{t=2} \\
\sum_i FP_{i,t=1} & & +\sum_i SP_{s=2,i,t=1} & \le capacity_{t=1} \\
\sum_i FP_{i,t=2} & & +\sum_i SP_{s=2,i,t=2} & \le capacity_{t=2}
\end{array} \qquad \forall s,t$$

Figure 3.11 L-shaped structure in constraint 2.31

First, we analyze how Bender's decomposition uses the problem structure when it is applied to solve the SATP problem. The Bender's decomposition creates a master problem and a subproblem. The master problem contains the first-stage variables and the subproblem contains the second-stage variables. The master problem provides the first-stage decision to the subproblem. Once the first-stage decision has been made, it is fixed in constraint 3.25 in the subproblem. The constraint 3.25 then looks like figure 3.12.

$$\begin{array}{lll}
\sum_i SP_{s=1,i,t=1} & & \le capacity_{t=1} - \sum_i FP^*_{i,t=1} \\
\sum_i SP_{s=1,i,t=2} & & \le capacity_{t=2} - \sum_i FP^*_{i,t=2} \\
& \sum_i SP_{s=2,i,t=1} & \le capacity_{t=1} - \sum_i FP^*_{i,t=1} \\
& \sum_i SP_{s=2,i,t=2} & \le capacity_{t=2} - \sum_i FP^*_{i,t=2}
\end{array} \qquad \forall s,t$$

Figure 3.12 Structure of constraint 3.25

We see that the variables *SP* in each scenario are independent of each other, since constraint 3.25 remains valid for all inequalities in each scenario and each time period. The first-stage variables *FP* interact with the second-stage variables *SP* separately for each scenario. What is more to that is once the first-stage decisions have been made and fixed, the capacity constraint (2.31) is no longer a linking constraint which binds all the subproblems together. The second-stage subproblems can be separated from each other. This means that one can solve the subproblems for each scenario separately, making the Bender's subproblems to the SATP problem relatively easy to solve. Figure 3.13 shows the structure of the SATP problem before and after the Bender's decomposition is applied to solve it. The structure after applying the Bender's decomposition (on the right of figure 3.13) is the structure of the subproblems.

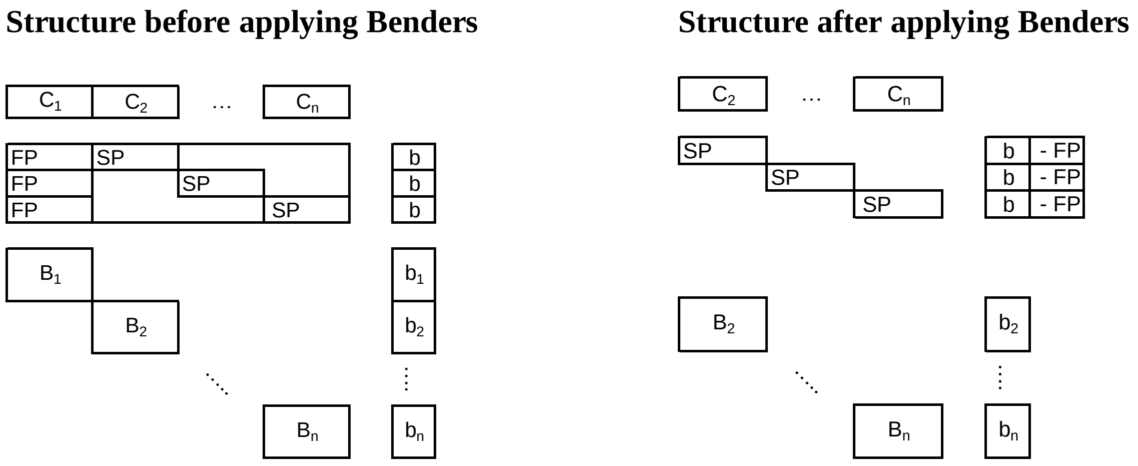

Figure 3.13 Simplified SATP Problem's structure

However, since the subproblem must be solved as a linear programming problem in order to obtain the requisite dual prices. The algorithm only uses approximate solutions to the integer programming subproblem as input to the master problem. Also, for the master problem of Bender's decomposition, one must solve an integer programming problem that may become more difficult as one adds multiple cuts that can be dense cuts with no given structure to exploit. Thus, the solutions we obtain from this method will have integer to the first-stage variables and, possibly fractional solutions to the second-stage variables. We think that this is reasonable since we are less definite about the eventual outcomes of future decisions and the solution we obtain is likely to be "good enough". For this reason, we believe that Bender's decomposition may be still suitable to solve the SATP problem.

Next we analyze whether the *Branch-and-Price* algorithm is suitable to solve the SATP problem. We know that the D-W decomposition creates a master problem from constraint 2.31. Without constraint 2.31 the subproblems are independent of each other.

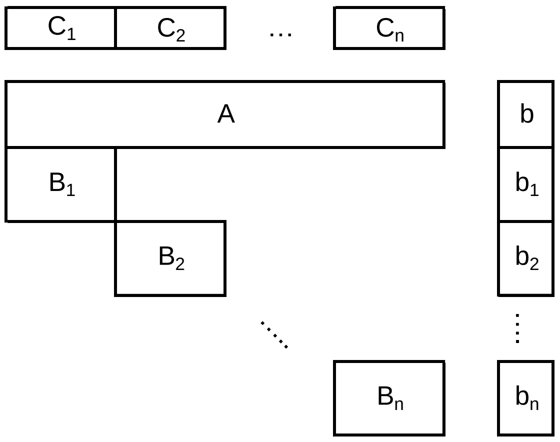

Figure 1.3 Block-angular structure

Figure 1.3 illustrates how each subproblem forms a small blocks B_j. The first-stage decision form a single subproblem B_1 consisting of constraints 2.32 through 2.37. Constraint sets 2.38 to 2.43 form each of the second stage alternatives $B_2 ... B_n$, and represent each possible second-stage decision. Thus, this structure makes it possible to decompose the problem into a Branch-and-Price algorithm. To be consistent with the Benders; approach, we will not enforce second-stage decision variables to take on integer values. In Chapter 4, we will compare how well Bender's decomposition and the Dantzig-Wolfe decompositions work on the very special problem-structure that occurs in the SATP problem. Numerical results and conclusions will be presented in chapter 4.

4. Numerical Results and Conclusions

In this chapter, we conduct experiments on both the Branch-and-Price algorithm and the Bender's algorithm. There are two experiments we do, one with the simplified SATP problem, the other with the original SATP problem. We do the experiment with the simplified SATP problem to test an efficiency of the decomposition methods. We solve the simplified SATP problem using the Branch-and-Price algorithm and the Bender's algorithm. We then compare numerical results of both algorithms to the base case which is solving the SATP problem without applying decomposition methods. Analyses of results and performance of the decomposition algorithms are discussed. Then we choose the decomposition method the better of the two methods, and perform a second set of experiments using the original SATP problem. In the second set of experiments, we conduct sensitivity analyses to investigate the robustness of the decomposition method. Conclusions and future research directions are given at the end of the chapter.

Simplified SATP Problem's Experiment Setup

An objective of this experiment is to choose a decomposition method which will solve the SATP problem in a time capable of being used by those accepting production orders (i.e. in a real-time situation). We used the same data set from Toshiba notebook company as used in [Chen 2003]. We aggregated some data to be compatible with the simplified SATP problem. There are 21 time periods in a planning horizon. We execute the model at the end of each time period so a batching interval is one time period. An order represents finished products e.g. preconfigured PCs ordered by a customer. There are two types of flexibilities that customers provide to manufacturers, the due date flexibility and the quantity flexibility. We assume that there is no production lead time which means the finished product is available to be delivered in the same time period as the order is accepted. Pseudo-orders are randomly generated for use as future orders. There are 25 scenarios; each one is associated with a given probability of occurring. The probability distribution is uniformly distributed. An average number of future orders per scenario is 50 orders. An average number of current orders arriving in each time period is ten orders. To test the decomposition algorithms' performance, we varied the number of arriving orders while holding the number of scenarios constant to see what effect the number of arriving orders have on the solution procedure. Similarly, we varied the number of scenarios while holding the arriving orders constant to test the effect that problem size has on the ability of each solution procedure to solve the problem. The experiment was run on a notebook with a processor speed 1.83 GHz and 2 GB of RAM. We use the commercial solver, CPLEX 11.0, as the engine for both decomposition methods as well as for the solution of the problem directly.

Analyses of Results of Simplified SAPT Problem

In the first experiment, we would like to see whether the number of arriving orders or the first-stage decision in each time period affects an objective function value. We vary a number of arriving orders between 20 and 120 while holding a number of scenarios at 7. The production capacity was kept at 100 percent.

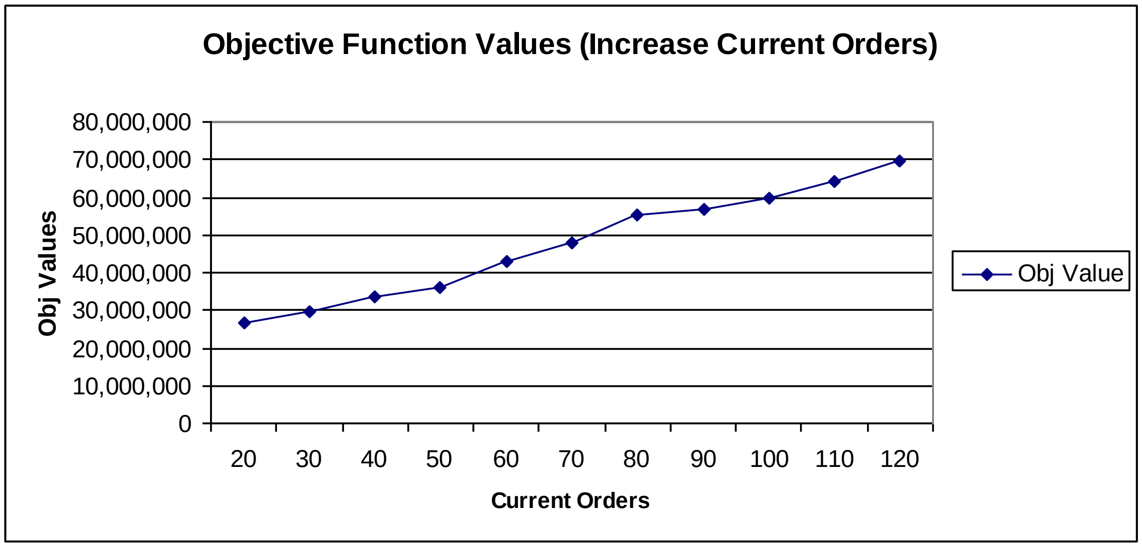

Figure 4.1 Objective values (vary current orders at 100% capacity)

Figure 4.1 shows that the greater the number of arriving orders, the greater the net profit. This can be explained by our having more orders to choose among and we therefore are capable of making better decisions. The result is consistent with the idea of extending the batching interval in [Chen 2003]. Furthermore, the result indicates that all orders are accepted even in the case of 120 orders (recall that the average of the arriving order is ten orders per time period). This means the manufacturer's production capacity is sufficient to accept all arriving orders. If this is the case, the manufacturer's decision is to always accept arriving orders and to manage planned production schedules to meet customers' due date range. Next we compare solve times of the two decomposition algorithms for this base case.

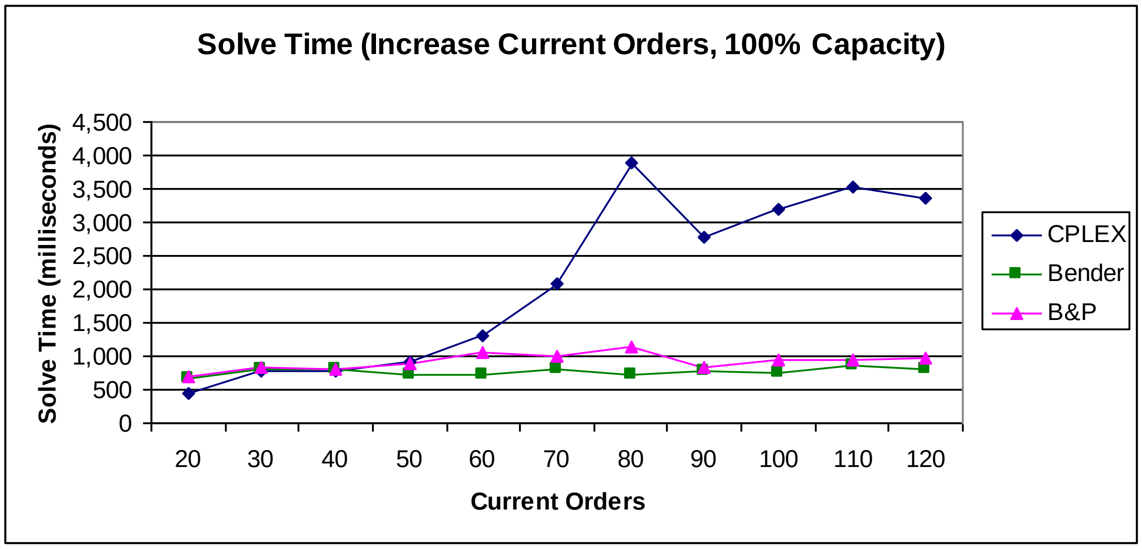

Figure 4.2 Solve time (vary current orders at 100% capacity)

Figure 4.2 shows the solve times of the decomposition algorithms compared to solving the SATP problem without applying the decomposition algorithms. It is obvious that the decomposition algorithms perform better than the base case as the number of current orders increase. We can see that the Bender's decomposition performs slightly better than the Branch-and-Price algorithm. The result also shows that the number of current orders has little effect on the performance of either decomposition algorithm. It does, however, affect the solution times of CPLEX to solve base case without decomposition. As we collect more current orders, the CPLEX solution times increase dramatically. For this case, both decomposition algorithms worked well with a slight advantage toward Bender's decomposition.

In the next experiment, we fix the number of current orders at 50 and vary the number of scenarios from 1 to 25 scenarios. Production capacity is kept at 100 percent. Increasing the number of scenarios increases the size of the problem because adding a scenario is equal to adding an additional subproblem to each decomposition method. In this experiment, we study how the size of the problem affects the performance of the decomposition algorithms. We show results of objective function values and solve times.

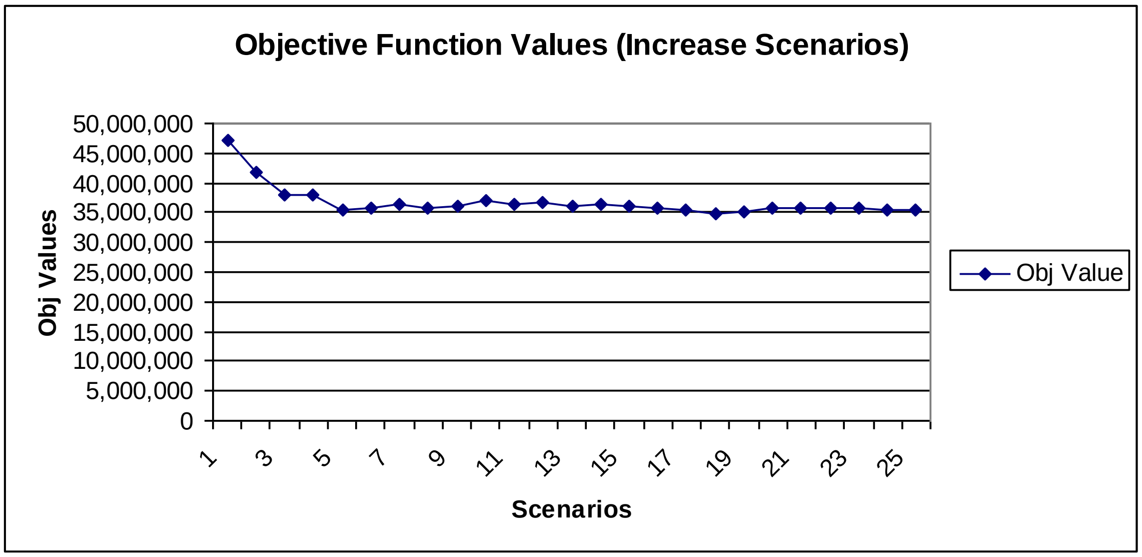

Figure 4.3 Objective values (vary scenario at 100% capacity)

It is interesting to see that the objective function values decreases as the number of scenarios increases as seen in figure 4.3. Having only one scenario is equivalent to having no future orders. Since there is only one scenario, the probability of it occurring is 1 and the problem becomes a deterministic, rather than a stochastic optimization problem. The objective function value is largest at this point. Let us now explain why the objective function values decreases as the number of scenarios increases. We need to look at an expected profit of the second-stage and an actual profit of each scenario to see why such result happened. Figure 4.4 shows the expected profit of the second-stage and the actual profit of each scenario. Note that, in this experiment, the production capacity is sufficient to accept all arriving orders plus the future orders. Therefore, the actual profit of each scenario is sum of the profit of orders in each scenario. We can see that the profit for scenario one is greater than all other scenarios. That is the reason why the objective function value is the largest when solving the problem with one scenario. Once other scenarios are included to the problem, their smaller profits decrease the expected second-stage profit until it reaches a point where the change in profit is relatively small from one run to another. Thus, the observed result was based on the data used. Had the profit of the orders in scenario one been small relative to the profit in other scenarios, we would expect that the second-stage profit would increase the overall profit.

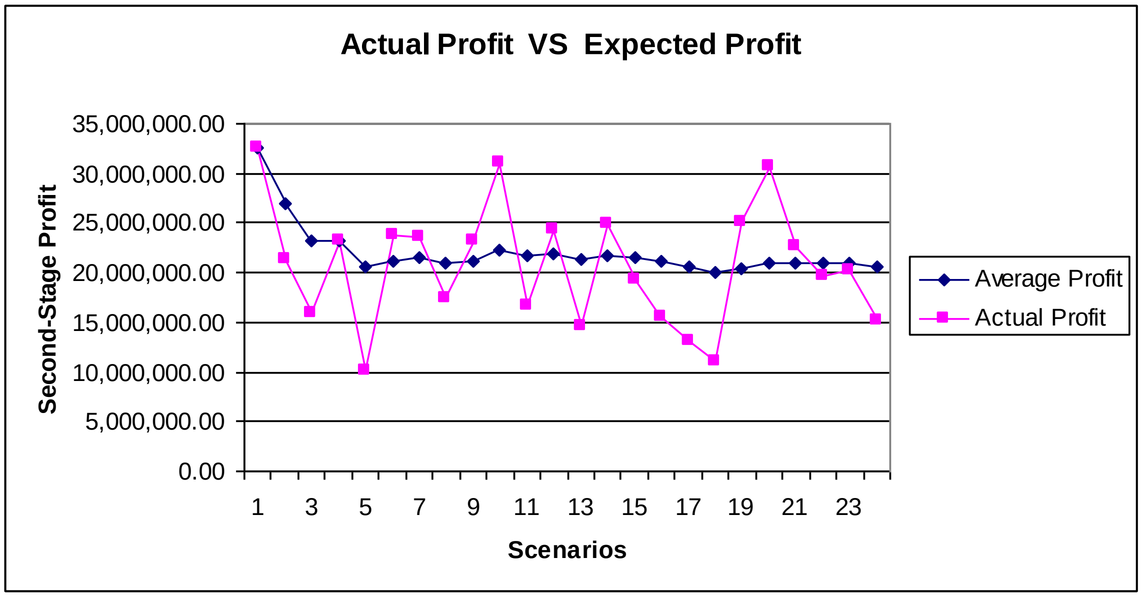

Figure 4.4 Actual profit vs Expected profit

Next we compare solve times of the decomposition algorithms to the base case. The result in figure 4.5 confirms our hypothesis that the decomposition algorithms can solve the problem faster than the base case. We can see from figure 4.5 that the decomposition algorithms perform better than the base case when there are 5 or more scenarios in the problem. Both decomposition algorithms have very similar solution times. The Bender’s performance is slightly better than the Branch-and-Price. We draw a conclusion from this experiment that the decomposition algorithms outperform the base case in terms of overall solution time.

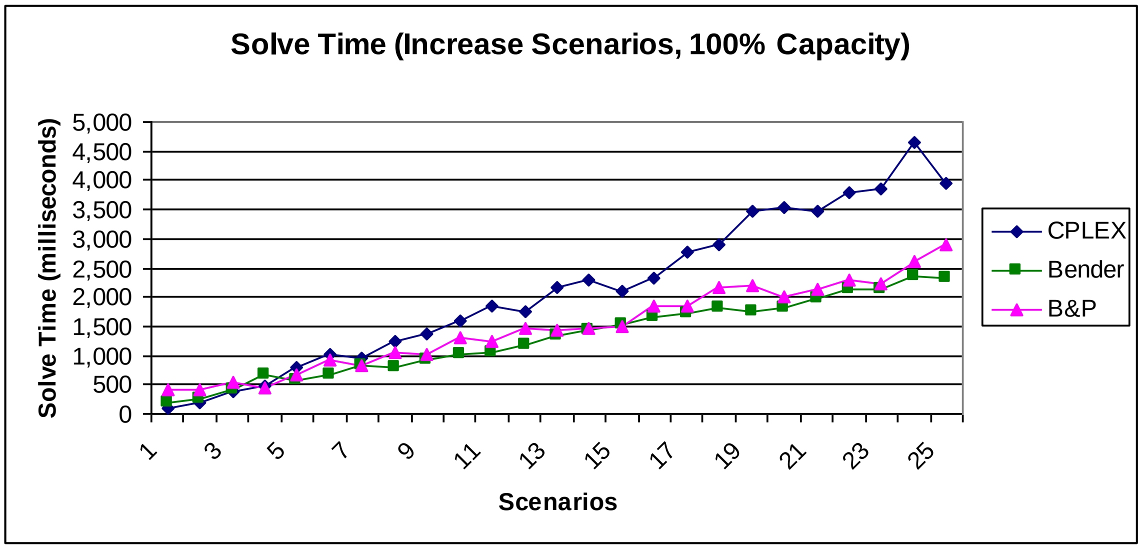

Figure 4.5 Solve time (vary scenario at 100% capacity)

All results so far have been based on having sufficient capacity to accept all orders and the algorithm had only to determine the *timing* of the productions. With this capacity setting, the Branch-and-Price algorithm was capable of finding integer optimal solutions at a root node so there was no need to branch. Similarly, the Bender's decomposition adds only one cut to find an optimal solution to the simplified SATP problem. The capacity assumption is only likely to be true at the beginning of a planning horizon when there are no previously promised orders. To simulate the case where there are some previously promised orders that consume some of the available production capacity, we reduce the production capacity by 50 percent and perform the same set of experiments to see how it affects the performance of the decomposition algorithms.

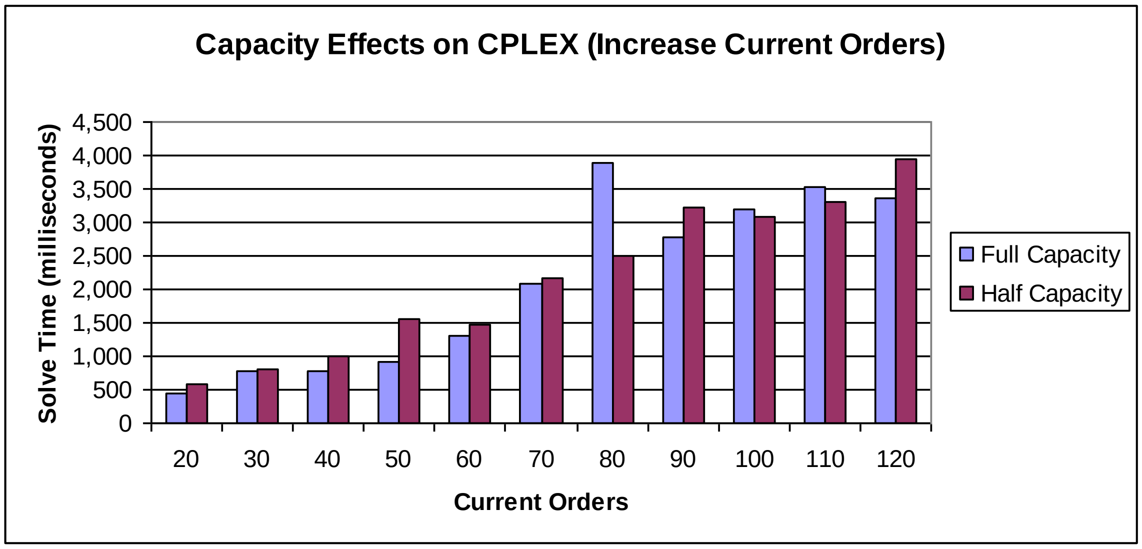

Figure 4.6 **Capacity effects on CPLEX (vary current orders)**

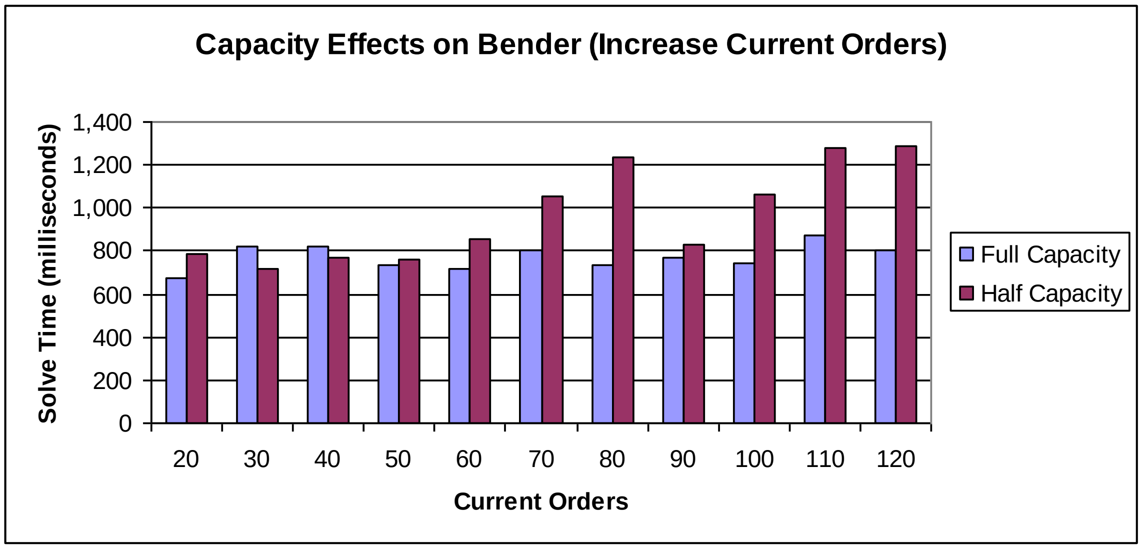

Figure 4.7 **Capacity effects on Bender (vary current orders)**

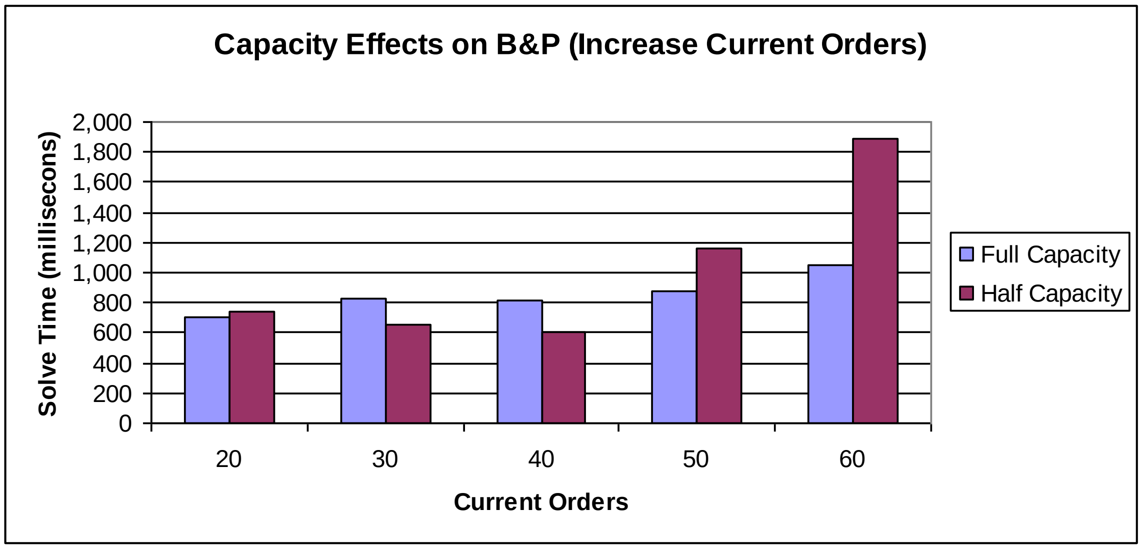

Figure 4.8 Capacity effects on Branch-and-Price (vary current orders)

Figure 4.6 shows that, for a general purpose solver, such as CPLEX, there is little change in the required time to solve the problem when there is a reduction in a production capacity. The CPLEX solve times of the SATP problem with a 50 percent reduction of the production capacity were not very significantly different from those when one had full capacity, and seemed to be slightly faster when there were more current orders. Results of decomposition algorithms are different. Figure 4.7 and 4.8 show that the reduction in production capacity affects performance of both decomposition algorithms. The solve times of both decomposition algorithms increase as the production capacity decreases. The Branch-and-Price algorithm is affected most. The problem with 70 arriving orders solved in 19,605 milliseconds and 37,301 milliseconds for the problem with 80 arriving orders. The greater the number of arriving orders, the longer the Branch-and-Price algorithm takes to prove optimality. We did not test the Branch-and-

Price algorithm for greater than 80 arriving orders because we conclude that without a change in the way in which the Branch-and-Price algorithm solves these problems, the solution times are too long to be practical for real world application. We believe the reason that our Branch-and-Price code did not perform better was that we did not optimize the mechanism for handling the branching tree search. In the previous experiment when the production capacity is sufficient to accept all arriving orders, the Branch-and-Price algorithm found an optimal integer solution at the root node. In this case, the approach's solution time is better than the CPLEX's.

There are a number of ways to improve the overall performance of the Branch-and-Price code. One could develop a heuristic to be used at the top of the tree, thereby allowing most nodes to be fathomed and might indicate the nodes that are most likely to provide good feasible solutions. In addition, more testing needs to be done to determine if a better branching scheme exists that reduces the number of fractional first-stage variables. We believe that further research is likely to improve the overall solution times of the Branch-and-Price procedure. Theoretically, it is the most suitable algorithm given the underlying problem structure of this problem.

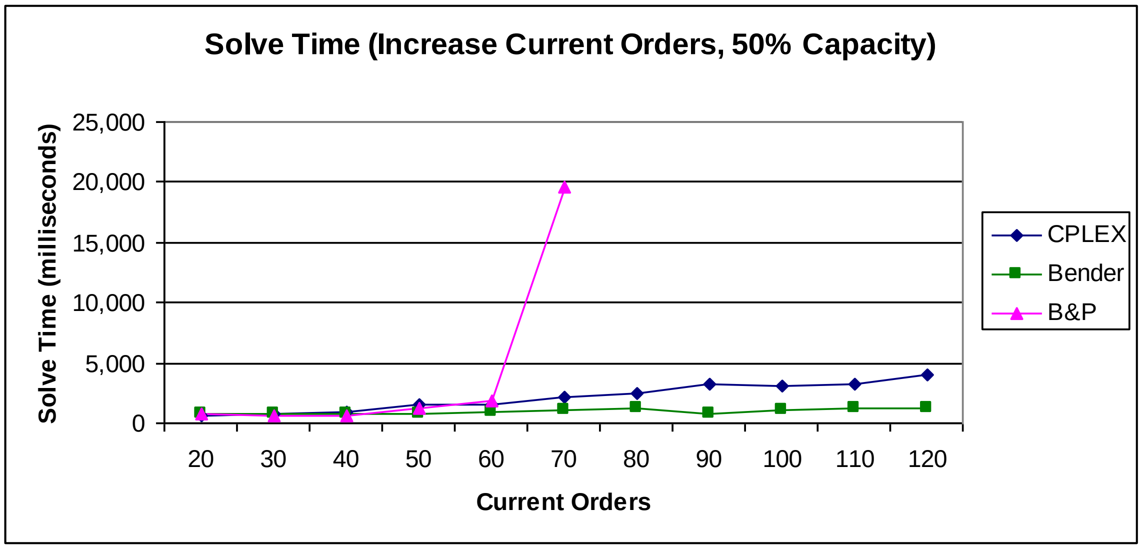

Figure 4.9 Solve time (vary current orders at 50% capacity)

Although the reduction of the production capacity affects the performance of each the decomposition algorithms, Bender’s decomposition is still better than solving the problem using CPLEX directly, as seen in figure 4.9. Even though we solve each of iteration of the master problem as a mixed-integer programming problem, using CPLEX directly, the number of times these MIPs need to be solved is not so large that the overall time becomes excessive. This highlights the fact that CPLEX is much more efficient at performing tree-search than our Branch-and-Price code. We think that more work in this research area is needed.

Finally, we perform another experiment where we fix the number of arriving orders at 50 orders and vary the number of scenarios between 1 and 25 scenarios. We then run the experiment at 100 percent and 50 percent of the production capacity. We wish to test whether the decomposition algorithms solution times are impacted when the problem's size gets larger with limited production capacity.

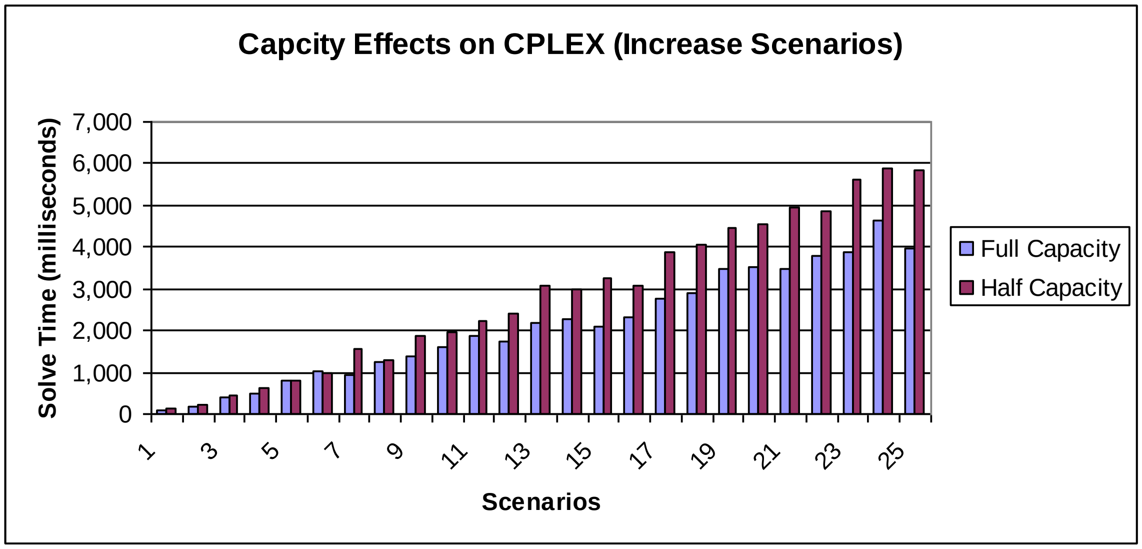

Figure 4.10 Capacity effects on CPLEX (vary scenario)

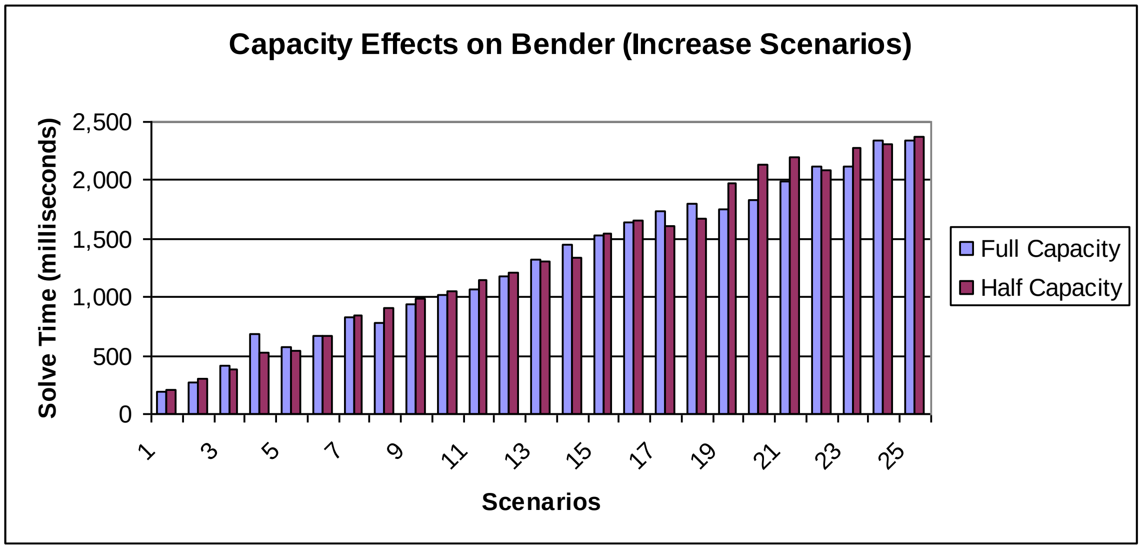

Figure 4.11 Capacity effects on Bender (vary scenario)

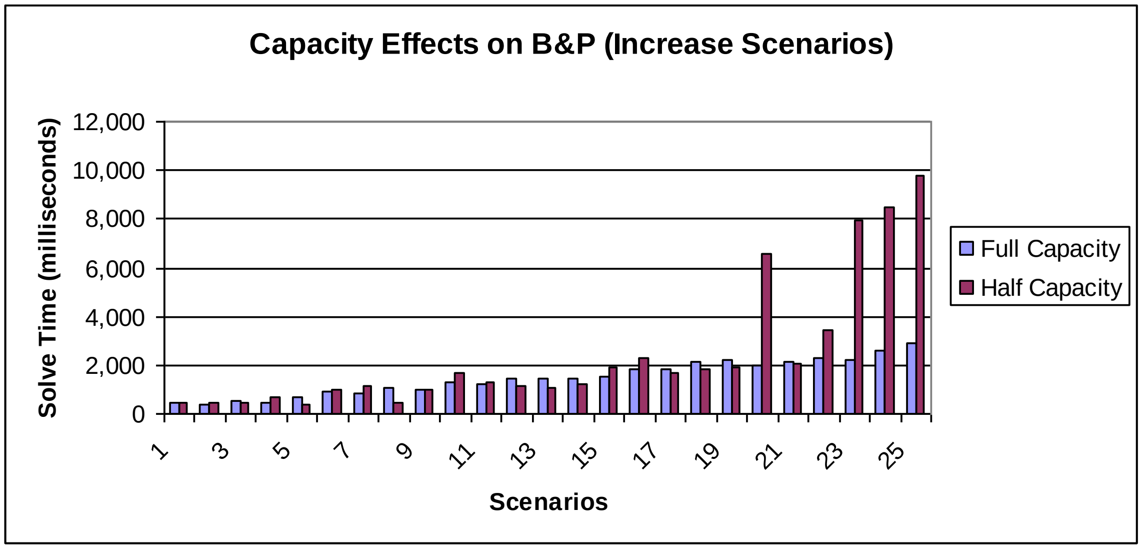

Figure 4.12 Capacity effects on Branch-and-Price (vary scenario)

Figure 4.10 to 4.12 show that all three algorithms are affected by the reduction of the production capacity. The solve times of the CPLEX and the Bender's decomposition at 50 percent production capacity increase proportionally to the full production capacity. The solve times of the Branch-and-Price algorithm at 50 percent production capacity does not increase proportionally to the full production capacity. An anomaly occurs when the problem has 21, 23, 24 and 25 scenarios. The solve times of those runs increase dramatically. This is because, in these instances, branching was needed to find an optimal integer solution. Again, this highlights the fact that further research is needed to determine how to improve the tree-search for this approach.

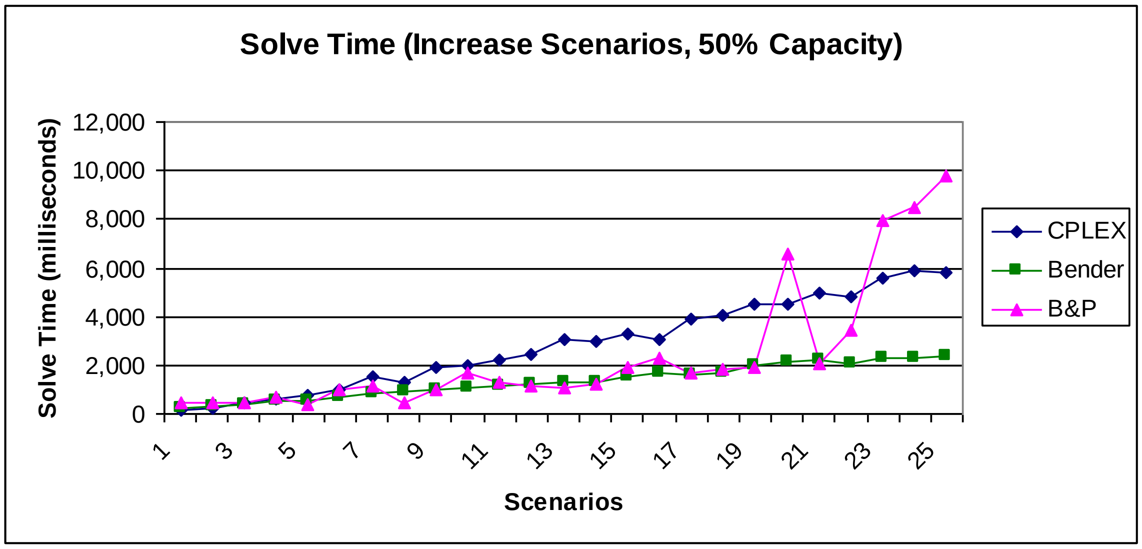

Figure 4.13 Solve time (vary scenario at 50% capacity)

Figure 4.13 shows that at half of full production capacity, as the problem size increases, the Bender's decomposition provides the best solution times. Although the Branch-and-Price is very close in solution times whenever there was little or no tree-search. Overall, Bender's decomposition has proven to be relatively robust under all conditions.

Original SATP Problem's Experiment Setup

We have shown the numerical results of the decomposition methods when solving the simplified SATP problem. The Bender's decomposition has proven to be the best overall. We will now test its effectiveness against the CPLEX in solving the original SATP problem. We will not test the Branch-and-Price algorithm in this case because it has been shown that its solution times were the worst when it needs to search the branching tree. Our prior empirical results when testing the simplified SATP problem demonstrated the effects of problem's size (i.e. the number of scenarios), the number of newly arrived orders, and the limitation of a production capacity on solvability and computation time. Our set of experiments will test (a) how well Bender's decomposition solves the original SATP problem, and (b) whether perturbations in either the profit array (in terms of both current and future orders) –or altering the probability that a future scenario will occur will impact our ability to solve the problem or impact the time required to solve the problem. Our previous experiments showed that the order in which the scenarios are presented may impact the objective function value when one limits the total number of scenarios provided to the optimizer. Thus, we will permute the order and rerun those experiments to better understand this phenomenon. All of these experiments will be tested using the original SATP problem using Bender's method and CPLEX. We include the CPLEX times as a mechanism from which to compare our decomposition methods. In each of these tests, we allow all of the second-stage decisions to be non-integral. We think that this makes sense since we do not know which may occur and each such scenario is weighted by its likelihood of occurring. We would like to to these comparisons using an average of 50 future orders in each scenario. However, CPLEX ran out of memory when we attempted the base-case. We found that CPLEX was capable of solving the base-case when we used an average of 40 future orders. Thus, all of the sensitivity analysis will be limited to this number of future orders.

Analyses of Results of Original SATP Problem

The first set of the experiments tests the effect of perturbing the profit array. As our test set, we solve 25 instances where each instance contains 40 previously promised orders, 10 newly arrived orders and 25 scenarios with an average of 40 future orders in each scenario. The profits of all orders of each problem are randomly perturbed by + or - 5%. Results are provided in figure 4.14 and 4.15.

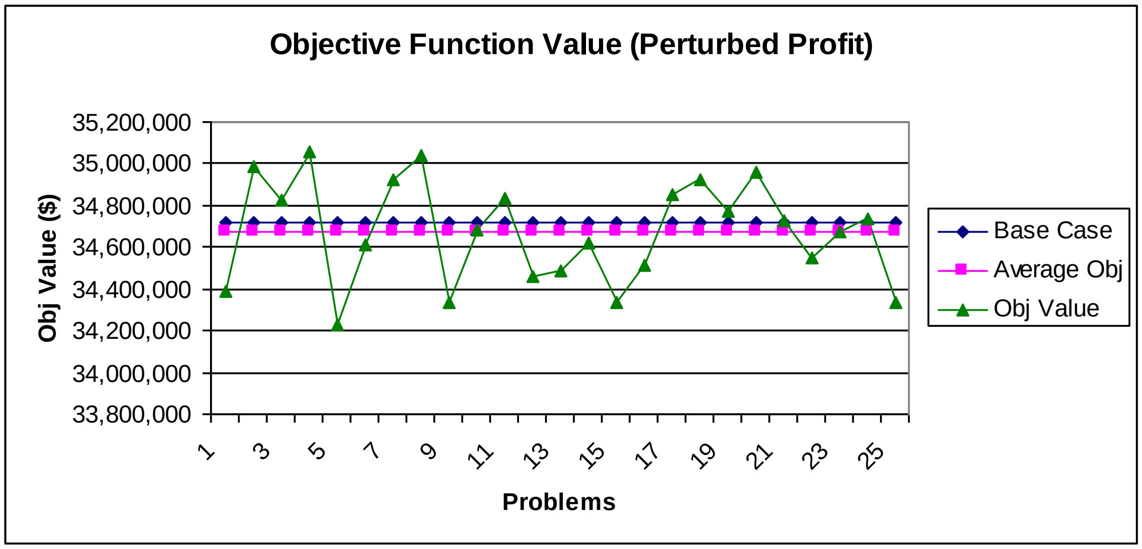

Figure 4.14 Original SATP objective values (perturbed profit)

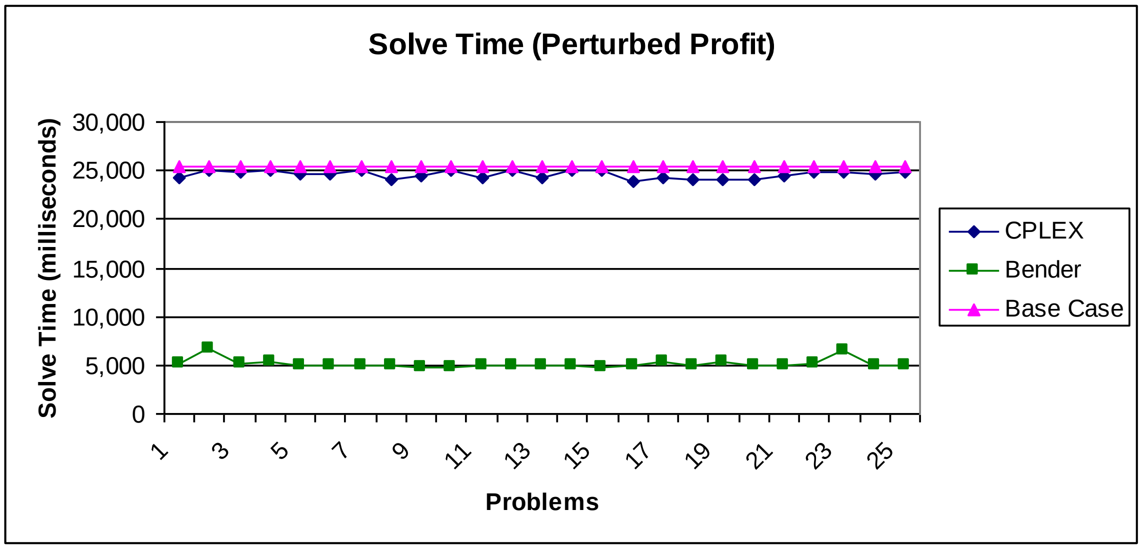

Figure 4.15 Original SATP solve time (perturbed profit)

Figure 4.14 shows that the perturbed problems' objective function values varies by plus or minus 2%. An average objective function value of the perturbed problems is slightly less than the base case by less than one percent. When looking at solve times, we see that the Bender's solve times, shown in Figure 4.15, are about one fifth of the base case's solve time in all 25 runs. The solve times of the perturbed problem solved by CPLEX is slightly less than the base case. The Benders performance has proved to be better than the CPLEX in solving the original SATP problem in this case.

The second set of the experiments tests the effect that perturbations in the likelihood that future scenarios occur have on the overall performance of the Bender's algorithm. The probability of each scenario is randomly perturbed by plus or minus 5% from its original value. The data about orders and number of scenarios are the same as in the first set of the experiments. The experiment was run 25 times. Results are shown in figure 4.16 and 4.17.

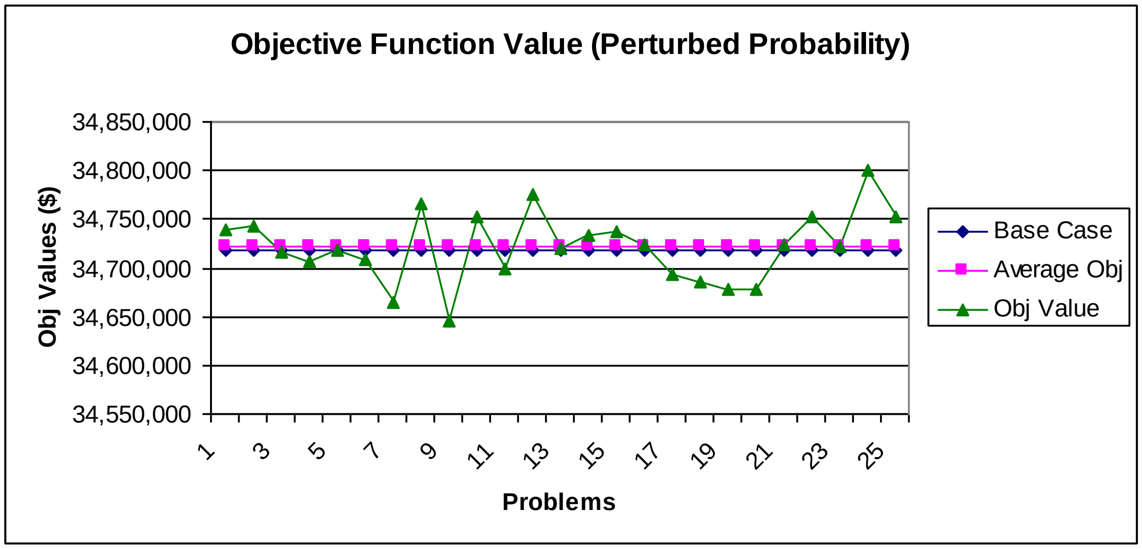

Figure 4.16 Original SATP objective values (perturbed probability)

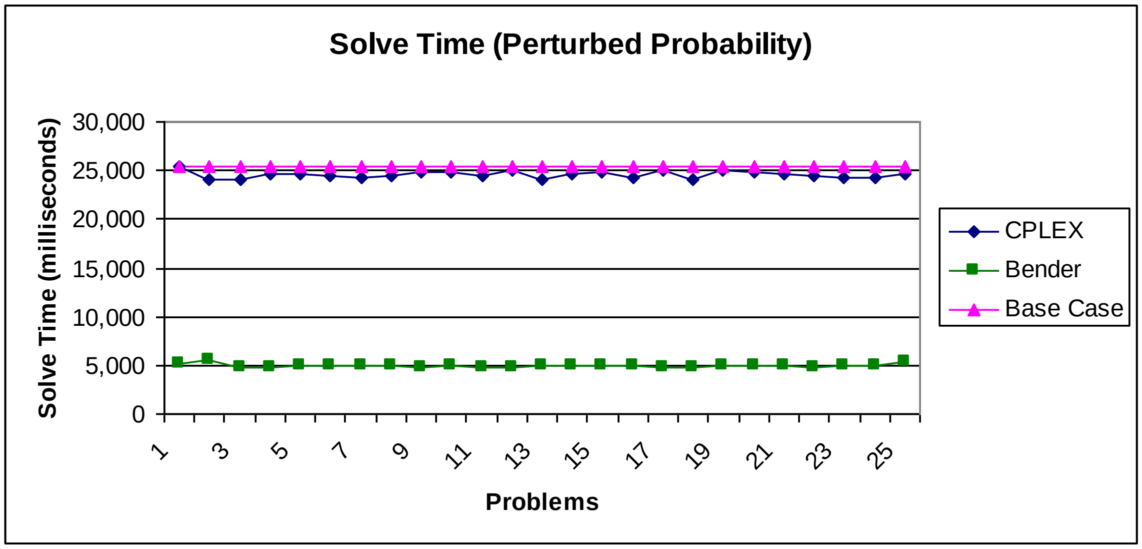

Figure 4.17 Original SATP solve time (perturbed probability)

Figure 4.16 shows that the objective function values vary less than one percent from the base case. An average of the perturbed objective function values is almost the same as the base case. Solve times are shown in figure 4.17. We can see that the Bender's solve times (perturbed problems) are still the best compare to both the CPLEX solve times of the perturbed problems and the base case. The Bender's solve times are about one fifth of the base case's solve time which are about the same as in the perturbed profit case.

The last set of the experiments tests an effect of an order of future scenarios on solve times of the Bender's algorithm. In the experiment with the simplified SATP, we tested the effect of the problem's size by adding to the problem one scenario per run. The order of the added scenarios is fixed. We noticed that as a number of scenarios increases, objective function values decrease as shown in figure 4.3. As we investigated further, we realized that the first few scenarios provided the best profit, so there was little improvement when many of alternative scenarios were added. Since the simplified SATP has the same structure as the original SATP and we used the same data to run the experiment so we could expect the same phenomenon to occur with the original SATP. If we change the order of the scenarios and do the same experiment (adding one scenario per run), we may see different results from figure 4.3. That does not matter because the objective function value will converge to one value after we have added all 25 scenarios to the problem regardless the order of the scenarios added. We will prove this by running the last experiment on 25 problems. Each problem contains 25 scenarios. The order of the scenarios in the problems is permuted. Results are shown in figure 4.18 and 4.19.

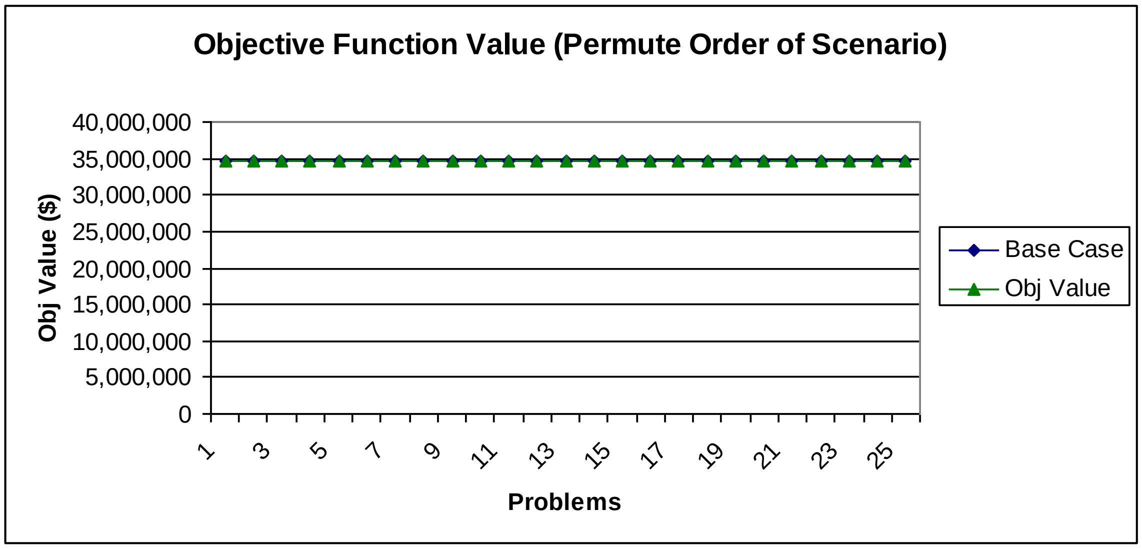

Figure 4.18 Original SATP objective values (permute order of scenario)

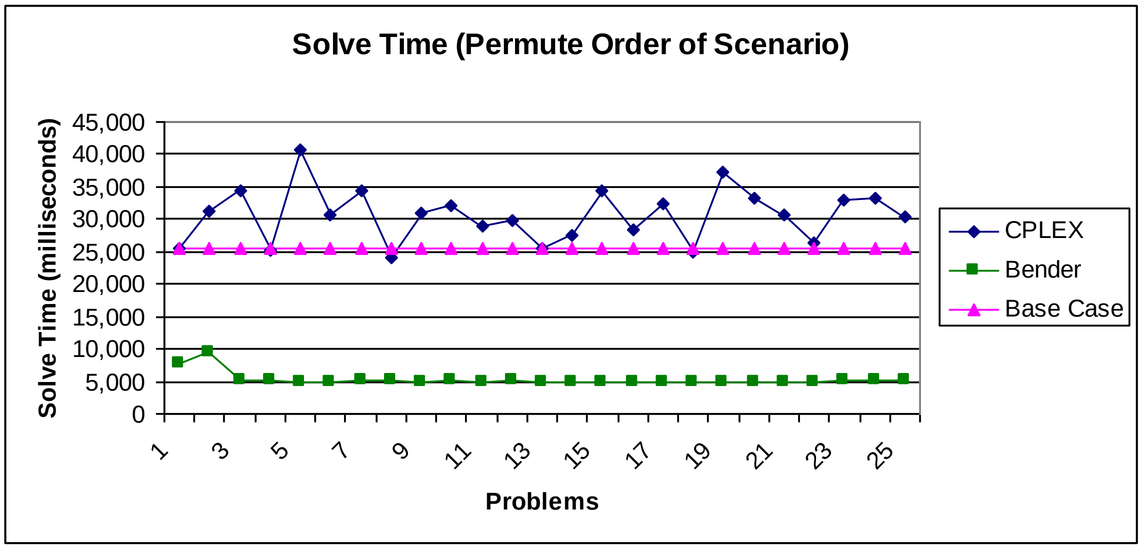

Figure 4.19 Original SATP solve time (permute order of scenario)

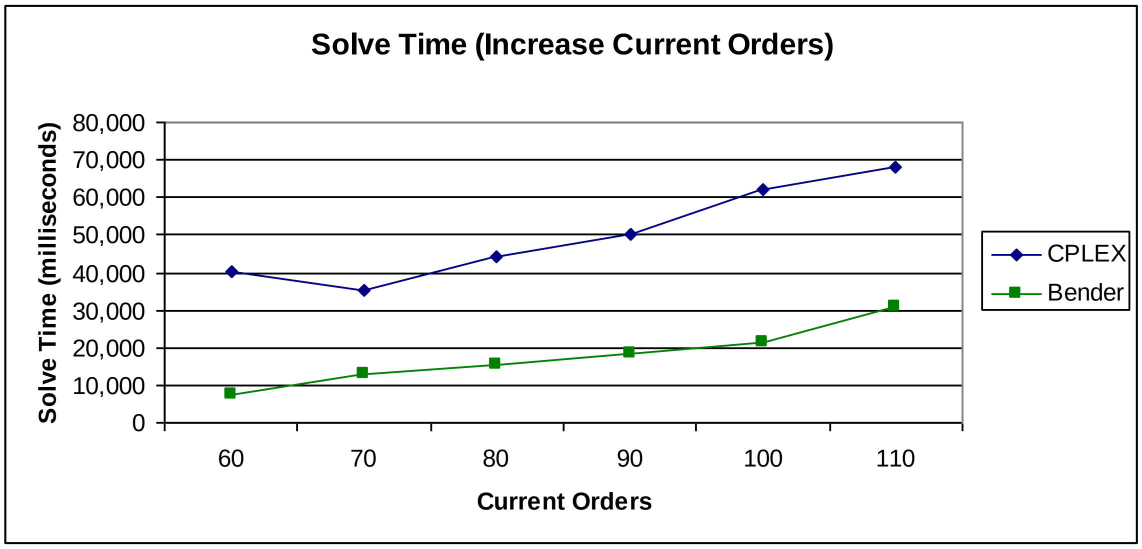

Figure 4.20 Original SATP solve time (vary current orders)

The result in figure 4.18 confirms our hypothesis about the objective function values are the same regardless the order of the scenarios. Solve times of the Bender's algorithm has proven to be the best again as shown in figure 4.19. The Bender's solve times are about one fifth of the base case's solve time. The results in figure 4.19 indicate that the order of the scenario may have an effect to the CPLEX's solve times because the solve times of 20 instances out of 25 instances are greater than the base case's solve time. All three sets of experiments show that the Bender's performance was not affected by any cases tested. We would like to make sure that the Bender's algorithm can cope up with extreme cases that may occur. We ran an experiment by fixing the number of scenarios at 25 scenarios and a number of previously promised orders at 40 orders. We then increased a number of newly arrived orders so that a number of combined current orders (previously promised orders + newly arrived orders) range between 60 and 110 orders.

This situation may happen when we run the optimization on a moving time horizon as in [Chen 2003]. Results in figure 4.20, show that the Bender's method has outperformed the CPLEX in all cases. We can conclude that the Bender's method outperforms the CPLEX in solving the original SATP problem in every case. We now provide our conclusions and avenues for future research.

Conclusions

We conducted two experiments to test the performance of the Branch-and-Price algorithm and Bender's decomposition. The first one was performed on the simplified SATP problem. The second one was performed on the original SATP problem. The first experiment compared solution times for the simplified SATP problem using two decomposition methods and compared these times to solving the problem without using the decomposition methods. The experiment tested the decomposition algorithms' performance by varying a number of problem characteristics. We tested how production capacity impacted solution times. We also varied the number of arriving orders while fixing the number of scenarios, and then varied the number of scenarios while fixing the number of arriving orders. All data was based on the data Chen [2003] collected from the Toshiba notebook company.

Results of the full production capacity case showed that both decomposition algorithms are more suitable to solve the SATP problem than attempting to solve the problem using a state-of-the-art general-purpose optimization package, such as CPLEX. Bender's decomposition solution times were a bit faster than the Branch-and-Price's solution times. There were significant differences in the solution time of both algorithms when the production capacity was reduced to a half. This case simulates the situation where there are previously promised orders in the system when the current decisions need to be made. The Branch-and-Price algorithm performed poorly compared to the others in this case due to the fact that the method, as currently implemented, required substantial tree-search. The poor performance of the Branch-and-Price happens *only* when the algorithm needs to do branching to find an optimal integer solution. Thus, improving the search-tree component, or enhancing the code with a heuristic solution, may improve the overall performance substantially. Since the problem has a block-angular structure, and the Branch-and-Price is ideally suited for such structures, we believe that further work on this approach is merited.

The Bender's decomposition performed well in every case. This algorithm is easy to implement since we can allow a general-purpose code to handle the IP calculations directly. However, if as the problem grows, the overall solution times for each optimization also increases, one might find that there is a maximum size problem that Bender's decomposition can handle. We designed our experiment to overestimate the data to make sure that the decomposition algorithms can perform better than solving the problem directly with the CPLEX code. An average of arriving orders, for example, is 10 orders per time period. We tested the algorithms by varying the arriving orders between 20 and 120 orders. Results showed that even in rare cases such as the 120 arriving orders in one time period, the Bender's decomposition was the fastest. As the size of the problem grows, the Bender's performance was the best overall in both a full production capacity case and a half production capacity case. Thus we conclude base on the results that the Bender's decomposition is the best algorithm to solve the simplified SATP problem.

We then performed the second experiment on the original SATP problem. We focused the experiment on testing the robustness of Bender's decomposition method. The original SATP problem is much more complicated than the simplified problem. We knew from the first experiment that the Bender's algorithm was the best overall. We therefore used this decomposition to test if perturbations in the data might have an effect on our above conclusions. We again tested Benders decomposition against a standard, state-of-the-art commercial code, CPLEX. In the original data set, there are 40 previously promised orders, 10 newly arrived orders and an average of 50 future orders in each scenario. Since the CPLEX code was incapable of handling this large problem – it ran out of memory – we reduced the number of average future orders to 40 for each scenario. There are 25 scenarios in each problem. We used this test set to see if (a) perturbing the objective function, or (b) perturbing the order of the scenarios, or (c) perturbing the probabilities associated with each scenario would impact either solution ability or computational effort. In each of these tests, Bender's decomposition could obtain the correct solution in times significantly faster than the CPLEX times. We conclude that the Bender's decomposition is the best method to solve the SATP problem.

The idea behind using stochastic optimization rather than a deterministic approach is that it captures more of the variability in the estimates. However, if it is very expensive to generate a multitude of scenarios that capture this information, or if it is expensive to solve once one has obtained these scenarios (because of the number of scenarios generated), one must reconsider the most appropriate way to apply this technology. We suggest that one should consider generating a small collection of the most likely scenarios, together with their probabilities and then aggregate less-likely scenarios into a few representative scenarios with their appropriate probabilities. In this way, one can capture most of the relevant information through sampling while maintaining a tractable, smaller model. The computational burden will be greatly decreased with little likelihood of obtaining incorrect consequences. One could also include in the modeling some measure of variability and therefore obtain a result that "hedges" against a less-likely but far more costly consequence. We have shown that the computation time is sensitive to the number of scenarios generated, so one should take this into account in the modeling stage.

Future research directions

There are a number of avenues yet to explore. Firstly, one needs to design a branch-and-price algorithm that obtains good feasible integer solutions at the top of the tree and a better branching scheme within the tree so that solution times are reduced. Obtaining heuristics for this problem should not be hard since one can easily pick high probability scenarios as one's choice to obtain feasible solutions.

One can also examine if adding many columns (rather than a single column) at each iteration of the branch-and-price method would significantly improve the computations, or at least, make the linear programming solution more integer: One might also examine if there are strong cuts that can be added to the problem that will enhance the solution times without destroying the structure of the problem.

As stated above, including a sampling approach that captures much of the variability in the data without including *every* possible scenario would enhance the overall usability of the stochastic programming approach. In addition, one might wish to consider how sensitive the recommendations (solutions) of the model are to perturbations in the probability estimates. If the model is particularly sensitive to the perturbations, then one might want to include some variance and covariance measures to obtain robust solutions.

Finally, one needs to consider what happens to each method when more side constraints are added to the general model. Do we lose the efficiency of the Bender's method when the subproblems are more difficult to solve? Is the branch-and-price method more suitable in this case? One could also tighten the model formulation and reduce a number of zero-one integer variables, FZ_i and $SZ_{s,i}$. In the original SATP model formulation, constraint 2.1 and 2.5 could be changed to $\sum_{t \in W_i} FD_{i,t} \leq 1$ and $\sum_{t \in W_s^i} SD_{i,t,s} \leq 1$ respectively. Thus, we eliminate those two integer variables from the constraints and the objective function. Those variables could be replaced by $\sum_{t \in W_i} FD_{i,t}$ and $\sum_{t \in W_s^i} SD_{i,t,s}$ in the objective function.

We believe that this is a very rich area for research. In the past, operations research has been primarily focused on planning models. With current computing manpower, and the strengthening of our optimization tools, we are now set to take on the harder, but far more important area of real-time decision making. This dissertation has begun to address that problem for a specific manufacturing example.

LIST OF REFERENCES

LIST OF REFERENCES

Adams, W. P., & Sherali, H. D. (1986). A Tight Linearization and an Algorithm for Zero-One Quadratic Programming Problems. *Management Science, 32*(10), 1274-1290.

Ahuja, R. K., Magnanti, T. L., & Orlin, J. B. (1993). *Network Flows Theory, Algorithms, and Applications*: Prentice-Hall, Inc.

Anthony, G. G., Shapiro, J. F., & Wolsey, L. A. (1972). Relaxation Methods for Pure and Mixed Integer Programming Problems. *Management Science, 18*(5), 229-239.

Armstrong, R. D., & Willis, C. E. (1977). Simultaneous Investment and Allocation Decisions Applied to Water Planning. *Management Science, 23*(10), 1080-1088.

Bahl, H. C., & Zionts, S. (1987). Multi-Item Scheduling by Benders' Decomposition. *The Journal of the Operational Research Society, 38*(12), 1141-1148.

Ball, M. O., Chen, C.-Y., & Zhao, Z.-Y. (2001). Quantity and Due Date Quoting Available to Promise. *Information systems Frontiers, 3*(4), 477-488.

Ball, M. O., Kotake, M., & Zhao, Z.-Y. (2005). Optimization-Based Available-To-Promise with Multi-Stage Resource Availability. *Annals of Operations Research, 135*, 65-85.

Barnhart, C., Johnson, E. L., Nemhauser, G. L., Savelsbergh, M., & Vance, P. H. (1996). Branch-and-Price: Column Generation for Solving Huge Integer Programs. *Operations Research, 46*(3), 316-328.

Barnhart, C., Johnson, E. L., Nemhauser, G. L., & vance, P. H. (1997). Airline Crew Scheduling: A New Formulation and Decomposition Algorithm. *Operations Research, 45*(2), 188-200.

Birge, J. R., & Louveaux, F. (1997). *Introduction to Stochastic Programming*: Springer-Verlag New York, Inc.

Birge, J. R., & Takriti, S. (2000). Lagrangian Solution Techniques and Bounds for Loosely coupled Mixed-Integer Stochastic Programs. *Operations Research, 48*(1), 91-98.

Bloom, J. A. (1983). Solving an Electricity Generating Capacity Expansion Planning Problem by Generalized Benders' Decomposition. *Operations Research, 31*(1), 84-100.

Cai, X., McKinney, D. C., Lasdon, L. S., & Jr., D. W. W. (2001). Solving Large Nonconvex Water Resources Management Models Using Generalized Benders Decomposition. *Operations Research, 49*(2), 235-245.

Chen, C.-Y. (2003). *Optimization-based available-to-promise (ATP).* Unpublished PhD Dissertation, UNIVERSITY OF MARYLAND, COLLEGE PARK.

Cheng, F., Ettl, M., Huang, P., & Sourirajan, K. (2006). *Product Offering Conditioning in Assemble-To-Order Supply Chains* (Research Report): IBM.

Clasen, R. J. (1984). The Solution of the Chemical Equilibrium Programming Problem with Generalized Benders Decomposition. *Operations Research, 32*(1), 70-79.

Cordeau, J.-F., Soumis, F., & Desrosiers, J. (2001). Simultaneous Assingment of Locomotives and Cars to Passenger Trains. *Operations Research, 49*(4), 531-548.

Dantzig, G. B., & Thapa, M. N. (2003). *Linear Programming 2: Theory and Extensions*: Springer Series in Operations Research.

Dantzig, G. B., & Wolfe, P. (1961). The Decomposition Algorithm for Linear Programs. *Econometrica, 29*(4), 767-778.

Desrochers, M., Desrosiers, J., & Solomon, M. (1992). A New Optimization Algorithm For The Vehicle routing Problem With Time Windows. *Operations Research, 40*(2), 342-354.

Ervolina, T. R., Ettl, M., Huang, P., Sourirajan, K., & Lin, G. Y. (2006). *Supply and Demand Synchronization in Assemble-To-Order Supply Chains*.

Geoffrion, A. M., & G.W.Graves. (1980). Multicommodity Distribution System Design by Benders Decomposition Part II. *Management Science, 26*(10), 1070-1071.

Geoffrion, A. M., & G.W.Graves. (1974). Multicommodity distribution System Design by Benders Decomposition. *Management Science, 20*(5), 822-844.

Gondzio, J., & Kouwenberg, R. (2001). High-Performance Computing for Asset-Liability Management. *Operations Research, 49*(6), 879-891.

Hane, C. A., Barnhart, C., & H.Vance, P. (1998). Using Branch-and-Price-and-Cut to Solve Origin-Destination Integer Multicommodity Flow Problems. *Operations Research, 48*(2), 318-326.

Hiller, R. S., & Eckstein, J. (1993). Stochastic Dedication: Designing Fixed Income Portfolios Using Massively Parallel Benders Decomposition. *Management Science, 39*(11), 1422-1438.

Hobbs, B. F., & Ji, Y. (1999). Stochastic Programming-Based Bounding of Expected Production Costs for Multiarea Electric Power Systems. *Operations Research, 47*(6), 836-848.

Holmberg, K., & Yuan, D. (2000). A Lagrangian Heuristic Based Branch-and-Bound Approach for the Capacitated Network Design Problem. *Operations Research, 48*(3), 461-481.

Jeong, B., Sim, S., Jeong, H., & Kim, S. (2002). An available-to-promise system for TFT LCD manufacturing in supply chain. *Computers and Industrial Engineering, 43*(1-2), 191-212.

Kolbin, V. (1977). *Stochastic Programming (Theory and Decision Library)* (I. P. Grigoryev, Trans. 1st ed.): Springer.

LAU, H. C., & LIM, M. K. (2002). *Multi-Period Multi-Dimensional Knapsack Problem and Its Application to Available-To-Promise.* National University of Singapore.

Magnanti, T. L., Mirchandani, P., & Vachani, R. (1995). Modeling and Solving the Two-Facility capacitated Network Loading Problem. *Operations Research, 43*(1), 142-157.

Magnanti, T. L., & Wong, R. T. (1981). Accelerating Benders Decomposition: Algorithmic Enhancement and Model Selection Criteria. *Operations Research, 29*(3), 464-484.

Mehrotra, A., Johnson, E. L., & Nemhauser, G. L. (1998). An Optimization Based Heuristic for Political Districting. *Management Science*, *44*(8), 1100-1114.

Nash, S. G., & Sofer, A. (1996). *Linear and Nonlinear Programming*: McGraw-Hill.

Neumann, K., Schwindt, C., & Trautmann, N. (2002). Advanced production scheduling for batch plants in process industries. *OR Spectrum*, *24*, 251-279.

Roy, T. J. V. (1986). A Cross Decompostion Algorithm for Capacitated Facility Location. *Operations Research, 34*(1), 145-163.

Savelsbergh, M. (1997). A Branch-And-Price Algorithm for the Generalized Assignment Problem. *Operations Research, 45*(6), 831-841.

Sen, S., & Lulli, G. (2004). Branch-and-Price Algorithm for Multistage Stochastic Integer Programming with Application to Stochastic Batch-Sizing Problems. *Management Science, 50*(6), 786-796.

Sherali, H. D., & Adams, W. P. (1984). A Decomposition Algorithm for a Discrete Location-Allocation Problem. *Operations Research, 32*(4), 878-900.

Sherali, H. D., & Staschus, K. (1990). A Two-Phase Decomposition Approach for Electric Utility Capacity Expansion Planning Including Nondispatchable Technologies. *Operations Research, 38*(5), 773-791.

Slyke, R. M. V., & Wets, R. (1969). L-Shaped Linear Programs with Applications to Optimal Control and Stochastic Programming. *SIAM Journal on Applied Mathematics, 17*(4), 638-663.

Sung, C. S., & Hong, J. M. (1999). Branch-and-Price Algorithm for a Multicast Routing Problem. *The Journal of the Operational Research Society, 50*(11), 1168-1175.

Vanderbeck, F. (2001). A Nested Decomposition Approach to a Three-Stage, Two-Dimensional Cutting-Stock Problem. *Management Science, 47*(6), 864-879.

Vanderbeck, F. (2000). Exact Algorithm for Minimising the Number of Setups in the One-Dimensional cutting Stock Problem. *Operations Research, 48*(6), 915-926.

Wolsey, L. A. (1998). *Integer Programming*: Wiley-Interscience Series in Discrete Mathematics and Optimization.

CURRICULUM VITAE

Arm Pangarad received his Bachelor of Science from Chulachomklao Royal Military academy, Thailand in 1997 and received his Master of Business Administration from Southeastern University, Washington D.C. in 2001. His research interests are in supply chain, logistics and decomposition methods.