Algorithmic Framework for Improving Heuristics in Stochastic, Stage-Wise Optimization Problems

A Thesis
Presented to
The Academic Faculty

by

Jaein Choi

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

School of Chemical and Biomolecular Engineering
Georgia Institute of Technology
November 2004

Algorithmic Framework for Improving Heuristics in Stochastic, Stage-Wise Optimization Problems

Approved by:

Jay H. Lee, Committee Chair

Shabbir Ahmed

Matthew J. Realff, Advisor

Hayriye Ayhan

Andreas S. Bommarius

Date Approved: 22 November 2004

To My Parents, Sunghyung Choi and Jueogsoon Goh,

and

To Wife, Eunmyung Hong and Daughter, Seoyoon Choi

With Love ...

ACKNOWLEDGEMENTS

As I stand at the threshold of earning my doctorate, I am overwhelmed when I recall all the people who have helped me get this far. Many people have been on my side, in different way, in this wonderful learning experience, both on a personal and a scientific level

First and foremost, I would like to thank my Ph.D. advisors, Professor Jay H. Lee and Professor Matthew J. Realff, for their tremendous support, guidance, and inspiration. Both of them are truly remarkable advisors who grant students a lot of freedom to explore new ideas, but at the same time interact closely with them. Prof. Lee was patient to make me a sincere researcher. Besides his excellent technical advice, his personal care for me and warmly hosted dinners at his home have made me forget that I have been away from my home. Energetic discussions with Prof. Realff always kept me refreshed and excited to think new ideas. With them, I have enjoyed numerous meetings we had every week for the last four years although it was the toughest time in my life. Most of all, thank for their cheerful optimism that has been very important in a few dark hours and for giving me the feeling that they trusted me. I look forward to continuing my association with them in the future.

I would like to thank my thesis committee members, Professors Andreas S. Bommarius, Shabbir Ahmed, and Hayriye Ayhan, for accepting the burden of sitting in my reading committee. I am honored that they were called to give the final word on my work.

I would like to thank my former advisor at KAIST, Professor Sunwon Park, for his early guidance in my M.S. research on process systems engineering area and for his kind advice both on my work and life.

I have three important elder brothers who have advised and helped me for long time since I have known them. Gwangsoo Kim, who inspired me to be a good student after I decided Chemical Engineering as my major in my 2nd year at KAIST in 1995. Our life-time friendship have continued so far wherever we have been. He was the one who taught me

'enthusiasm' and 'independence' in my work. Even after he moved to U.S. for this graduate study at MIT and to Canada for this research work in fuel cell area, I never really missed him because he managed somehow to make me feel his presence. Jongmin Lee, my senior, friend, and elder brother in Atlanta, who always has been around me during last four and a half years. Whenever I was in trouble with my work and personal matters, he was the first person who helped me and cheered me up. My family, including my wife and daughter, regards him as our family member in Atlanta. Sungyong Moon who encouraged me to study abroad before he started his Ph.D. study at Purdue University. I still remember a small lecture room on the 2nd floor of the department of ChE at KAIST where he gave me his GRE book and made me confident to decide to apply top engineering schools in U.S..

I would like to thank all members of the Lee group (ISSICL). Dr. Kangwook Lee for his kindness and for taking care of me, especially when I just joined the group in 2000. Andrew Dorsey, Yangdong Pan, for guiding me as senior students. Niket S. Kaisare for being an excellent office-mate and a nice friend. I shall not forget so many nights I studied and discussed with Niket and thank him for considering me a reliable gourmet when we had 'eating adventures'. And Thidarat Tosukhowong, Manish Gupta, Swathy Ramaswamy, Anshul Dubey, and Nikolaos Pratikakis for being responsible junior students to interact with. Over the past years, I have had great opportunities to get to know and to work with many visiting scholars. I also would like to thank them for their friendship, Prof. Dae R. Yang, Jochen Till, Hyungjin Park, Dr. Jongku Lee, Dr. Kyung Joo Mo, and Heejin Lim. Especially, I thank Dr. Jongku Lee for his advice on my future work after graduation. Based on his abundant industrial experience, he helped me draw an overview of my career paths as a process systems engineer.

In summer 2004, I had a great experience for working Owens Corning as an engineering intern. During the internship, I was exposed to interesting industrial problems relevant to my research meanwhile I also learned to work as a team with great people in MCO (Modeling, Control, and Optimiation) Group at Owens Corning. I would like to thank Dr. James Beilstein for supervising my work as a director of the group, and Dr. Chad Farschman and project manager Karthick Vaidyanathan for their guidance, valuable discussions, and for

advising me with very practical point of view on my project.

Besides the internship experience, I also had a great opportunity to carry on industrially oriented project with LG Chemical Co. Ltd., the company I will work for after graduation. For the project, I closely communicated with Dr. Hokyung Lee at LG Chem. and his insightful overview of the project helped me discover potential of my research in real-world.

The friendship among Korean students and postdocs in the department has been developed wonderfully since I started my graduate study in 2000. As the youngest one among them, I have greatly benefited from the association. I thank all of them for their love and care for each other - Seongho Park, Se-Young Yoon, Young-Soo Kim, Yeu Chun Kim, Jeongwoo Lee, Ingu Song, Dr. Ketack Kim, Dr. Jaewon Lee, Dr. Weontae Oh, and Dr. Jeonghyun Yeom.

Most important of all, I would like to express my deep gratitude to my family for being an unstinting source of support and encouragement. My parents have taught me the value of education and have worked very hard to provide me the very best of it. Even though I have been far away from my home, I could have felt their prayers and wishes, that always have been around and blessed me. My parents-in-law also have supported me and encouraged me since I got married. They have treated me as a real son and granted me unexpected gifts, sometimes, to cheer me up. My wife, Eunmyung Hong, has been the greatest help to me. When I proposed her in winter, 2000, my future was uncertain as a new graduate student. Regardless of my situation at that time, regardless of uncertain paths ahead of me, she accepted to be my life-time soul mate because she believed my sincere love to her and our bright future that we will make together. And eventually, we did it together as we believed. Last but not the least, I would like thank to my angelic daughter, Seoyoon Choi, born in July, 2002 as a precious gift to us from the God. Her existence was persistent motivation to me. As she grows up, she could cheer me up with her fabulous smiles and innocent words. I will try to spend more time with her from now on to fill up my absence with her as a busy graduate student.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

There have been many developments in methods for optimization under uncertainty recently. Among these developments, *Stochastic Programming* and *Stochastic Dynamic Programming* represent two major paradigms for rigorous optimization under uncertainty. On the other hand, various heuristics prevail in practical applications, production planning and scheduling, supply chain management, and engineering design, of the optimization under uncertainty. These two approaches have ever-conflicting advantages and disadvantages in their computational load and solution quality. Due to the computational infeasibility of the rigorous approach for many practical applications, problem-specific heuristics are prevalent, even though they cannot provide any guarantee on the solution's quality. It is our belief that for optimization under uncertainty, the needs of solution quality and computational feasibility should be integrated.

In this thesis, we try to find an answer to the question, 'What could lie between heuristic and rigorous approaches?'. We develop an algorithmic framework for improving solutions from heuristics without adding significant computation time. The proposed algorithmic framework tries to combine the advantages of heuristics and rigorous methods, in terms of computational feasibility and solution quality. The proposed framework consists of two major modules, *simulation* with heuristic policies and *optimization* with rigorous solution method, stochastic dynamic programming. The origin of the computational infeasibility of stochastic dynamic programming is huge solution space in which the optimal solution resides. If there is any systematic way to reduce the size of the solution space without eliminating the optimal solution, the problem could be solved more efficiently. But, in general, finding "a significantly reduced solution space" is as hard as solving the original problem. Indeed, many of the rigorous optimization solvers have been developed from the idea of successively reducing the solution space. (i.e. The branch and bound algorithm).

Three critical issues emerge in applying the algorithmic framework to a class of industrially motivated *stochastic, stage-wise optimization problems*[1], Traveling Salesman Problem(TSP), Resource-Constrained Project Scheduling Problem(RCPSP), Supply Chain Management(SCM) Problem: (i) high dimensional state space which is inevitably generated in stochastic dynamic programming for discrete system. (ii) complicated or unknown state transition rules in complicated problem structure. (iii) high dimensional action space due to necessity of simultaneous decisions in large systems. In this thesis, those issues are addressed with relevant examples and overcome by appropriate modifications of the general algorithmic framework.

The techniques for dealing with the critical issues can be integrated in the proposed algorithmic framework to tailor the algorithmic framework to a particular optimization problem under uncertainty.

[1]Parameters of the problems are uncertain and the decisions of the problems should be made in sequential manner along time or stage

CHAPTER I

INTRODUCTION

This dissertation is motivated by a need for a new computational framework for solving optimal stage-wise decision problems that involve significant amounts of uncertainties. Uncertainty is an integral part of almost every practical optimization problem found in planning, scheduling, supply chain management in process industries. The approach proposed in this thesis is geared towards such problems, which cannot be solved effectively by the currently existing optimization approaches like Stochastic Programming [48] and Stochastic Dynamic Programming [5, 7]. The proposed approach builds on some ideas developed in the Artificial Intelligence community, including those of 'simulation-based' optimization and Approximate Dynamic Programming (ADP). These methods were originally developed in the context of robot planning and game playing, and their direct applications to problems in the process industries are limited due to the differences in the problem formulation and size. In order for any new approach to be practical for such problems, it should address issues such as extremely high dimensional state and action spaces and a very large number of scenarios.

1.1 Motivation

Optimization approaches can broadly be classified into two categories, heuristics and rigorous methods. A heuristic method, tailored for a specific optimization problem, is often conceptually appealing and computationally efficient but sacrifices the solution quality. Rigorous methods include several general equation-based mathematical techniques such as 'Mathematical Programming' and 'Dynamic Programming'. These methods can guarantee the optimality of the solution in finite time for certain types of problems (e.g., P and NP-complete Problems)[31]. However, for large sized NP and NP-hard class of problems, they are computationally infeasible. Many practical problems are cast as integer optimization

problems that are intractable and are addressed with heuristics, which provide a solution but without any information on the solution's quality.

In the case of stochastic optimization, the mathematical programming based approaches are not efficient methods for handling significant uncertainty in the system due to the limited ways to represent the uncertainty in an equivalent deterministic form. The size of the equivalent deterministic form oftentimes increases exponentially with the dimension of the uncertain parameters because it is a superstructure of a sufficient number of samples of the uncertain parameters needed to capture the problem variability. In Stochastic Programming [48], the most practical way to reduce the size of the superstructure is to limit the number of stages, as evidenced by the predominance of the two-stage stochastic programming problems in the stochastic programming literature. In practice, this requires shortening the time horizon of the optimization, or making an unrealistic aggregation of future periods, leading to an unnatural, restrictive problem representation. Along with the stochastic programming, stochastic dynamic programming is a general and rigorous solution method for stochastic stage-wise optimization problems. However, we have not seen any application of the conventional stochastic dynamic programming [7] to real industrial problems since dynamic programming formulations of such problems result in extremely large state and / or action spaces, which are not amenable to numerical solution approaches like the value iteration or the policy iteration. This was recognized early and is referred to as the 'curse of dimensionality.'[7]

To overcome the curse of dimensionality, two alternative solution approaches, *Reinforcement Learning(RL)* [79] and *Neuro-Dynamic Programming(NDP)*[8], appeared in the middle of the 1980's. Both approaches are based on performing a large number of simulations and improves starting suboptimal solutions in an iterative manner. NDP is a suboptimal methods based on the evaluation and approximation of the optimal *value function*, through the use of simulation data and neural networks. In AI terms, NDP can be described as "learning how to make good decisions by observing the system's own behavior(simulation) and using built-in mechanisms for improving their actions though a reinforcement mechanism (iterative schemes for improving the quality of approximation of the optimal value function)."

In the DP's context, 'the curse of dimensionality' is resolved by identifying the working regions of the state space through simulations and approximating the value function in these regions through function approximation. Although it is a powerful method in general, in discrete systems, the function approximation can mislead decisions by extrapolating to regions of the state space with limited simulation data. To avoid excessive extrapolation(or interpolation) of the state space, the simulation and the Bellman iteration must be carried out in a careful manner to extract all necessary features of the original state space.

1.2 Issues in Practical DP Applications to Optimization Under Uncertainty

Central to the DP method is the 'Bellman Iteration' Equation, through which the cost-to-go value of every state in the state space is updated. In discrete systems, the computational load of the Bellman iteration is directly proportional to the number of states to be evaluated and the number of candidate actions for each state. The total number of discrete states increases exponentially with the state dimension. The stochastic, stage-wise optimization problems addressed in this thesis have the state and action variable dimensions that cannot be handled by the conventional value iteration. In addition, DP formulation is highly problem dependent and often requires careful defining of the core elements (states, actions, state transition rules, and cost functions). In this thesis, the following issues in applying *Approximate Dynamic Programming* (ADP) to practical optimization problems are addressed with relevant examples in the process industries.

1. **High dimensional discrete state space:** In DP, the state is defined as necessary and sufficient information for optimial decision making at each time step. To reduce unnecessary explosion of the state space, the state has to be defined as compact as possible while representing all the necessary information for the decision making. However, for the real-world optimization problems addressed in this thesis, a high dimensional state space is inevitable. Furthermore, the state has to contain the current best knowledge of the uncertainty (e.g., the conditional distribution of the uncertain

parameter vector) and the number of state variables needed to represent this information is at least as large as the number of the uncertainty sources. For instance, the state of the stochastic Resource Constrained Project Scheduling Problem (RCPSP), which will be discussed Chapter 4, consists of the state variables, information state variables, and resource state variables representing the status of the projects, the observed outcome of the current tasks in the projects, and the availability of the resources respectively. Furthermore, absolute time is added as a state variable in order to keep track of the time-dependent reward values of the projects. For a stochastic RCPSP with M projects and N resources, the state consists of $2M+N+1$ state variables including the state variables mentioned above. Hence, even for small number of projects and resources, the DP formulation results in high dimensional state space that makes the conventional DP approaches computationally infeasible.

2. **Complicated (or unknown) state transition:** In stochastic DP, expected cost-to-go values are calculated in the Bellman iteration. For a given state and an action, exact calculation of the expected cost-to-go requires identifying all possible next states and their realization probabilities. Analytic calculation of the all possible state transition probabilities is not practically feasible for a large size problem with complex interactions among states and actions. Furthermore, necessary inclusion of information state variables representing uncertainty makes the exact calculation of all possible state transitions more difficult because transitions of those information state variables are not controllable but autonomously evolved by underlying probability models. Hence, obtaining well approximated cost-to-go values while circumventing "awkwardness" of exact analytical state transition is an important issue to extend application area of the DP to the stochastic optimization problems addressed in this thesis.

3. **High dimensional action space:** Even if high dimensional state spaces can be handled effectively, the complicated high dimensional, decision structure of the real-world stochastic optimization problems limit the convergence of Bellman iteration

in finite time. For example, the Supply Chain Management problem discussed in Chapter 6 has a high dimensional discrete action space for simultaneous decisions of every material flow in the system. In discrete systems, although the number of possible actions is limited to a finite combination of all actions in the high dimensional action space, it is far beyond current computational capability to access all the actions.

In conclusion, the ADP approach can be applied to solve stochastic, stage-wise optimization problems. For successful application of the ADP, systematic solution approaches for circumventing, or overcoming, each issue have to be developed.

1.3 Outline of the Thesis

The rest of this thesis is organized to propose the algorithmic framework and to resolve the important issues with relevant applications. In Chapter 2, conventional optimization techniques, such as mathematical programming, stochastic programming, and stochastic dynamic programming, are reviewed. To motivate this work, advantages and disadvantages of the optimization techniques are addressed. Although the existing methodologies have been developed and applied to more realistic and challenging problems, those methodologies are not adequate to solve a class of stochastic, stage-wise optimization problems under significant uncertainty we are interested in.

In Chapter 3, we suggest an algorithmic framework for solving stochastic, stage-wise optimization problems. In the proposed method, DP is performed in a restricted state space of the states visited during various suboptimal simulations according to the state transition rules of the problem and heuristic policies. Therefore, the worst case of the proposed approach is the best suboptimal solution visited during simulations. We apply the proposed approach to a new deterministic TSP variant, which illustrates the important notions of optional and conditional tasks in planning and scheduling applications, to examine the degree of improvement that can be obtained by the method. For a small problem, we compare the quality of the solution with the globally optimal one. Then, the algorithmic framework developed for improving heuristics of a new version of deterministic TSP [27] is extended to stochastic case. To verify the algorithmic framework for the stochastic case, a new variant

of the stochastic TSP with an optional task, in which key parameters(cost matrix) of the problem change according to an underlying Markov model, is introduced in Section 3.3 as a prototypical stochastic optimization problem. The proposed algorithmic framework finds the approximate optimal cost-to-go only for the states visited during suboptimal simulation. The results show that the algorithmic framework also works efficiently for stochastic problems by improving the heuristic policies for making decisions for different realizations of the Markov chains.

In Chapter 4 and 5, we apply the proposed algorithmic framework to the stochastic Resource Constrained Project Scheduling Problems (sRCPSP) that have not been solved efficiently by existing optimization algorithms. In Chapter 4, we propose a novel way of addressing the uncertainties in the sRCPSP including the uncertainties in task durations and costs, as well as uncertainties in the results of tasks(success or failure) by using a discrete time Markov chain, which enables us to model probabilistic correlation of the uncertain parameters. The proposed approach is tested on a simplified version of sRCPSP that has a fairly complicated stochastic nature, with 1,214,693,756 possible parameter realizations(scenarios), and involves 5 projects and 17 tasks. As a result, an online policy is obtained, which can use the information states in real-time decision making and improve the heuristics rather than a fixed solution obtained by the previous MILP problem formulations.

In Chapter 5, the issue of "complicated state transition rules" is addressed with the sRPCPS. The bottleneck is overcome by adopting the idea of Q-Learning approach [86], which can be used when a model of the system is unavailable, to the proposed framework. The Q-Learning approach can be viewed as a simultaneous identification of the probabilistic state transition rule and the cost-to-go function. Hence, it removes the need to perform the analytical calculation of the transition rule, which can be painstakingly tedious. The stochastic simulation under the certain suboptimal law (heuristics) is used here to obtain the empirical state transition probabilities of the states in the restricted state space, which are defined as combinations of the conventional states and the corresponding actions, and initial cost-to-go values for the value(Q) iteration.

In Chapter 6, the algorithmic framework is applied to supply chain management (SCM)

problem. We represent the uncertainty through Markov chains, and employ the proposed algorithmic framework which can generate a dynamic operating policy that incorporates information about the uncertainty in the problem at each time step. For the SCM problem, conventional stochastic DP is computationally infeasible due to the high dimensional action space as well as the high dimensional state space. The restricted state space is obtained by simulating various potential scenarios under centralized dynamic inventory and production policy generated by combining static inventory policy heuristics. Heavy computational load implied by the high dimensional action space is efficiently circumvented by introducing "implicit sub-action space" in which an action space is defined for each state with the heuristics used in simulation. The resulting DP policy responds to the time varying demand for products by stitching together decisions made by the heuristics and improves overall performance of the SC.

Finally, Chapter 7 summarizes the contributions of the proposed approaches developed in this thesis and recommends future work.

CHAPTER II

BACKGROUND

In this chapter, we give a broad overview of deterministic/stochastic optimization techniques as applied to problems in *process systems engineering* and *the operations research* area.

2.1 Conventional Deterministic Optimization Methods: Mathematical Programming

Over the last decade there have been considerable advances in mathematical programming techniques. For instance, the solution of mixed-integer nonlinear programming problems and the rigorous global optimization of nonlinear programs have become a reality [30]. There has also been a recent trend towards new logic-based formulations that can facilitate the modeling and solution of these problems [45, 52]. Finally, the development of modeling tools that can facilitate the formulation of optimization problems has also seen some great progress, as well as has the development of alternative solution strategies [15, 12]. In general, a mathematical programming model is represented in the following form:

$$\begin{aligned} &\min : Z = f(x,y) \\ &\text{s.t.} \quad h(x,y) = 0 \\ &\qquad g(x,y) \leq 0 \\ &x \in X \ , \ y \in (0,1) \end{aligned}$$

where $f(x,y)$ is the objective function (e.g. cost), $h(x,y) = 0$ are the equations that describe the performance of the system (mass and heat balances, design equations), and $g(x,y) \leq 0$ are inequalities that define the specifications or constraints for feasible choices. The variables x are continuous and generally correspond to the state or design variables, while y are the discrete variables, which are generally restricted to take 0-1 values to define the selection of an item or an action. Problem (MIP) corresponds to a mixed-integer nonlinear program

(MINLP) when any of the functions involved are nonlinear. If all the functions are linear, it corresponds to a mixed-integer linear program (MILP). If there are no 0-1 variables, it reduces to a nonlinear program (NLP) or a linear program (LP) depending on whether or not the functions are linear.

MILP solution methods rely largely on the simplex LP-based branch and bound approach consisting of a tree enumeration in which LP subproblems are solved at each node and are eliminated based on bounding properties. These methods are being improved through cutting plane techniques, which produce tighter lower bounds for the optimum. LP and MILP codes are widely available. The best known general solvers include CPLEX, OSL and XPRESS, all which have achieved impressive improvements in their capabilities for solving problems over the last decade. On the other hand, since MILP problems are NP-complete, it is always possible to run into time limitations when solving problems with a large number of 0-1 variables, especially if the integrality gap is large.

The solution of NLP problems relies either on the successive quadratic programming (SQP) algorithm, or on the reduced gradient method. Major codes include MINOS and CONOPT for the reduced gradient method, and OPT [82] for the SQP algorithm. These NLP methods are theoretically guaranteed to find the global optimum if the problem is convex (i.e. convex objective function and constraints). When the NLP is nonconvex, achieving a global optimum cannot guaranteed. One option is to try to convexify the problem, usually through exponential transformations, although the number of cases in which this possible is rather small. Alternatively, one could use rigorous global optimization methods, which over the last decade have made significant advances. These methods assume special structures such as bilinear, linear fractional and concave separable. Although this may appear to be quite restrictive, Smith and Pantelides [74] have shown that algebraic models are always reducible to these structures, provided they do not involve trigonometric functions. Computer codes for global optimization still remain in the academic domain, and the best known are BARON[68], and α-BB[3, 2]. It should also be noted that non-rigorous techniques such as simulated annealing and genetic algorithms, which have also become popular, do not make any assumption on the underlying function structure, but then they

cannot guarantee optimal solutions or bounds, at least not in finite time [47, 32]. Also, these methods do not formulate the problem as a mathematical program since they involve procedural search techniques that in turn require some type of discretization. Furthermore, violation of constraints is typically handled through ad-hoc penalty functions.

Major methods for MINLP problems include first Branch and Bound (BB), which is a direct extension of the linear case, except that NLP subproblems are solved at each node. Generalized Benders Decomposition (GBD) and Outer-Approximation (OA) are iterative methods that solve a sequence of NLP subproblems with all the 0-1 variables fixed, and MILP master problems that predict lower bounds and new values for the 0-1 variables. The difference between the GBD and OA methods lies in the definition of the MILP master problem: The OA method uses accumulated linearizations of the functions, while GBD uses accumulated Lagrangian functions parametric in the 0-1 variables. The LP/NLP based branch and bound essentially integrates both subproblems within one tree search, while the Extended Cutting Plane Method (ECP) does not solve the NLP subproblems, and relies exclusively on successive linearizations. All these methods assume convexity to guarantee convergence to the global optimum. The only commercial code for MINLP is DICOPT (OA-GAMS), although there are a number of academic versions (MINOPT , a-ECP).

In recent years a new trend that has emerged in the formulation and solution of discrete/continuous optimization problems through a model that is known as Generalized Disjunctive Programming (GDP) [52]. The basic idea in GDP models is to use boolean and continuous variables, and formulate the problem with an objective function subject to two or three types of constraints: (a) global inequalities that are independent of discrete decisions; (b) disjunctions that are conditional constraints involving an OR operator; and (c) pure logic constraints that involve only the boolean variables. More specifically the problem is given as follows:

$$\begin{aligned} &\min : Z = \sum_{k \in K} C_k + f(x, y) \\ &\text{s.t.} \quad g(x) \leq 0 \end{aligned}$$

$$
\begin{aligned}
& \vee_{j \in I_k} \begin{bmatrix} Y_{jk} \\ h_{jk}(x) \leq 0 \\ c_k = \gamma_{jk} \end{bmatrix} k \in K \\
& \Omega(Y) = \text{True} \\
& x \in X, Y_{j,k} \in \{\text{True,False}\}
\end{aligned} \tag{1}
$$

where x are continuous variables and y are boolean variables. The objective function involves the term $f(x)$ for the continuous variables (e.g. cost) and the charges c_k that depend on the discrete choices. The equalities/inequalities $g(x)$ must hold regardless of the discrete conditions, and $h_{ik}(x) = 0$ are conditional equations that must be satisfied when the corresponding boolean variable y_{ik} is True for the ith term of the kth disjunction. Also, the fixed charge c_k is assigned of the value g_{ik} for that same variable. Finally, the constraints $\Omega(y)$ involve logic propositions in terms of the boolean variables.

Problem (GDP) represents an extension of disjunctive programming, which in the past has been used as a framework for deriving cutting planes for the algebraic problem (MIP). It is interesting to note that any GDP problem can be reformulated as a MIP problem, and vice versa. It is more natural, however, to start with a GDP model, and reformulate it as a MIP problem. This is accomplished by reformulating the disjunctions using the convex hull transformation or with "big-M" constraints. The propositional logic statements are reformulated as linear inequalities. For the linear case of problem GDP, and when no logic constraints are involved, Beaumont[4] proposed a branch and bound method that does not rely on 0-1 variables and branches directly on the equations of the disjunctions. This method was shown to outperform the solution of the alternative algebraic MILP models. Raman and Grossmann[62] developed a branch and bound method for solving GDP problems in hybrid form; i.e. with disjunctions and mixed-integer constraints. For this they introduced the notion of "w-MIP representability" to denote those disjunctive constraints that can be transformed into mixed-integer form without loss in the quality of the relaxation. For the nonlinear case of problem (GDP), and for the case of process networks, Turkay and

Grossmann[81] proposed a logic-based Outer-Approximation algorithm. This algorithm is based on the idea of extending the Outer-Approximation algorithm by solving NLP subproblems in reduced space, in which constraints that do not apply in the disjunctions are disregarded. This way both the efficiency and robustness can be improved. In this method the MILP master problems correspond to the convex hull of the linearization of the nonlinear inequalities. Also, several NLP subproblems must be solved to initialize the master problem in order to cover all the terms in the disjunctions. Penalties can also be added to handle the effect of nonconvexities as in the method by Viswanathan and Grossmann[85]. This method has been implemented in the computer prototype LOGMIP, a GAMS-based computer code. Finally, it should be noted that a new method for solving GDP problems has been recently been reported by Lee and Grossmann[52]. These authors have developed reformulations and algorithms that rely on the convex hull of nonlinear convex inequalities. Although the mathematical programming approaches have progressed significantly to be applicable to various optimization problems in the process systems engineering, applications of the approaches are largely limited to deterministic problems. In next section, two major conventional techniques for solving optimization problems involving undertainties, stochastic programming and stochastic dynamic programming, are discussed.

2.2 Optimization Under Uncertainty

A large number of problems in production planning and scheduling, location, transportation, finance, and engineering design require that decisions be made in the presence of uncertainty. Uncertainty, for instance, governs the price of fuels, the availability of electricity, and the demand for products. From the very beginning of the application of optimization to these problems, it was recognized that analysis of natural and technological systems are almost always confronted with uncertainty. A key difficulty in optimization under uncertainty is in dealing with an uncertainty space that is usually huge and frequently leads to very large-scale optimization models. Decision-making under uncertainty is often further complicated by the presence of integer decision variables to model logical and other discrete decisions in a multi-period or multi-stage setting.

Approaches to optimization under uncertainty have followed a variety of modeling philosophies, including expectation minimization, minimization of deviations from goals, minimization of maximum costs, and optimization over soft constraints. Main purpose of this section is to give a broad overview of the main approaches to optimization under uncertainty: stochastic programming and stochastic dynamic programming.

2.2.1 Stochastic Programming

Under the standard two-stage stochastic programming paradigm, the decision variables of an optimization problem under uncertainty are partitioned into two sets. The first stage variables are those that have to be decided before the actual realization of the uncertain parameters. Subsequently, once the random events have presented themselves, further design or operational policy improvement can be made by selecting, at a certain cost, the values of the second-stage, or recourse, variables. Traditionally, the second-stage variables are interpreted as corrective measures or recourse against any infeasibility arising due to a particular realization of uncertainty. However, the second-stage problem may also be an operational level-decision problem following a first-stage plan and the uncertainty realization. Due to the uncertainty, the second-stage cost is a random variable. The objective is to choose the first-stage variables in a way that the sum of the first-stage costs and the expected value of the random second-stage costs is minimized. The concept of recourse has been applied to linear, integer, and nonlinear programming.

The two-stage formulation is readily extended to a multi-stage setting by modeling the uncertainty as a filtration process. Under discrete distributions, this reduces to a scenario tree of parameter realizations. Decomposition schemes that partition the time stage[10] as well as those that partition the scenario space[65] have been developed for multi-stage linear programs.

A standard formulation of the two-stage stochastic program is [48]:

$$\min \quad c^t x + E_{\omega \in \Omega}[Q(x, \omega)], \tag{2}$$

$$\text{s.t.} \quad x \in X$$

$$\text{with} \quad Q(x, \omega) = \min f(\omega)^t y, \tag{3}$$

$$\text{s.t.} \quad D(\omega)y \geq h(\omega) + T(\omega)x, \quad y \in Y,$$

where, $X \subseteq \Re^{n1}$ and $Y \subseteq \Re^{n2}$ are polyhedral sets. Here, $c \in \Re^{n1}$, ω is a random variable from a probability space (Ω, F, P) with $\Omega \subseteq Re^k$, $f : \Omega \rightarrow \Re^{n2}$, $h : \Omega \rightarrow \Re^{m2}$, $D : \Omega \rightarrow \Re^{m2 \times n2}$, $T : \Omega \rightarrow \Re^{m2 \times n1}$. Problem (2) with variables x represents the first stage, which needs to be decided prior to the realization of the uncertain parameters $\omega \in \Omega$. Problem (3) with variables y constitutes the second stage. Under the assumption of discrete distribution of the uncertain parameters, the problem can be equivalently formulated as a large-scale deterministic mathematical program, which can be solved using standard mathematical programming techniques addressed in the previous section 2.1. Convexity properties of the recourse function $Q(\cdot)$ [88] have been effectively used in decomposition-based solution strategies [11]. For continuous parameter distributions, these properties have been used to develop sampling-based decomposition and approximation schemes [40, 44] as well as gradient-based algorithms [73].

Recently, Schultz et al. [70] proposed a finite scheme for two-stage stochastic programs with discrete distributions and pure integer second-stage variables. For this problem, Schultz et al. observe that only integer values of the right-hand side parameters of the second-stage problem are relevant. This fact is used to identify a finite set in the space of the first stage variables containing the optimal solution. Schultz et al. propose the complete enumeration of this set to search for the optimal solution. This set may be very large and evaluation of each of its elements requires the solution of the second-stage integer subproblems. Thus, this approach is, in general computationally prohibitive. In most of the previous work, uncertain parameters are presented with discrete probability distributions. Except for simple cases that afford closed form solutions, sampling is required when dealing with continuous distributions of the problem parameters. Thus, convergence proofs for the resulting algorithm have to be probabilistic. For continuous distributions, Norkin et al. [55] developed a branch-and-bound algorithm that makes use of stochastic upper and lower bounds and proved almost sure convergence.

2.2.2 Stochastic Dynamic Programming

Dynamic Programming, as the name implies, is an approach developed to solve sequential, or multi-stage, decision problems. But, as we shall see, this approach is equally applicable for decision problems where sequential property is induced solely for the computational convenience. Unlike other branches of mathematical programming, one cannot talk about an algorithm that can solve all dynamic programming problems. For example, George Dantzig's Simplex Method can solve all linear programming problems. Dynamic programming, like the branch and bound approach, is a way of decomposing certain hard problems into equivalent formats that are more amenable to solution. Basically, what dynamic programming approach does is that it solves a multi-variable problem by solving a series of single variable problems. This is achieved by tandem projection onto the space of each of the variables. In other words, we project first onto a subset of the variables, then onto a subset of these, and so on.

The essence of dynamic programming is Richard Bellman's Principle of Optimality. This principle, even without rigorously defining the terms, is quite intuitive:
An optimal policy has the property that whatever the initial state and the initial decisions are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.
Since there is no limitation in extending the Principle of Optimality to stochastic systems, dynamic programming algorithm is not particularly limited to deterministic optimization. In stochastic systems, the general form of the Bellman equation is written as:

$$J^*(X(k)) = \max_{u(k)\in U} E\{\phi(X(k), u(k)) + \alpha J^*(X(k+1)|X(k), u(k))\} \tag{4}$$

where $\phi(X(k))$ is one stage cost for a state $X(k)$ and an action $u(k)$. Next state $X(k+1)$ is defined by a stochastic state transition equation f_I. α is a discounting factor ranged from 0 to 1.

$$X(k+1) = f_I(X(k), u(k)) \tag{5}$$

In value iteration, one starts with some arbitrary values for the value function. Then a transformation, derived from the Bellman optimality equation, is applied on the vector

successively till the vector starts approaching a fixed value. A general value iteration in stochastic Dynamic Programming is defined as following:

- Step 1: Set $i = 0$ and initialized $J^0(X)$ for all $X \in \chi$. Specify $\epsilon > 0$.
- Step 2: For each state X, compute:

$$J^{i+1}(x) = \min_{u(k) \in U} E\{\phi(X(k), u(k)) + \alpha J^i(X(k+1)|X(k), u(k))\} \tag{6}$$

- Step 3: If

$$\|(J^{i+1} - J^(i))\|_\infty < \epsilon(1-\alpha)/2\alpha, \tag{7}$$

  stop the iteration. Otherwise increase k by 1 and go back to Step 2.

Several important remarks are in order here.

- The max-norm of the difference $(J^{i+1} - J^i)$ decreases with every iteration. The reason for the use of the expression $\epsilon(1-\alpha)/2\alpha$ in Step 3 is explained in [34] with convergence of the value iteration. The condition in Step 3 ensures that when the algorithm terminates, the max norm of the differences between the cost-to-go values returned by the algorithm and the optimal cost-to-gos is ϵ.
- The algorithm's speed (which is inversely proportional to the number of iterations needed to terminate) can be increased by other methods such as *Gauss Siedel value iteration* and *relative value iteration for discounted reward*. Detail explanation of the methods are discussed in [61, 34].

CHAPTER III

ALGORITHMIC FRAMEWORK FOR IMPROVING HEURISTICS

3.1 Proposed Framework

Solving a stage-wise optimization problem requires finding the optimal trajectory of states in the state space. In general, the number of states visited by the optimal solution is very small compared to the total number of states in the entire state space. The main idea of the proposed algorithmic framework comes from following questions.

1. What if we solve DP in a restricted state space of the entire state space which includes all the states visited by the optimal solution?

2. How to find the restricted state space?

The answer for the first question is simply that we can find the optimal solution and the computational load of the DP will be significantly reduced. Therefore the second question is substantial: how to find this restricted state space. We can approximate this restricted state space using simulation, much as it is used in in RL or NDP. The idea may look same as NDP which is also based on the main DP framework with simulation and training. However, the major difference between the proposed approach and NDP is in the way the simulation data is used. NDP utilizes the simulation data for neural net training to map the input features to estimations of the cost-to-go function. Thus, it both extrapolates and interpolates over the examples. On the other hand, the simulation data is used to construct the restricted state space of the states in the proposed method by memorization of the states and connecting actions. This results in big differences in the performance of DP when the original state space is large. In particular, for the problem with a large state space, it may be impossible to cover the acceptable state space during the simulation. The extrapolated approximation generated by NDP may then cause the solution to deteriorate.

In the proposed method, DP is performed in the restricted state space of the states visited by different suboptimal simulations according to the state transition rules of the problem. Therefore, the worst case of the proposed approach can be ending up with the best of the suboptimal solution visited by the simulations. The the key procedures of the proposed approach is illustrated in following figure 1.

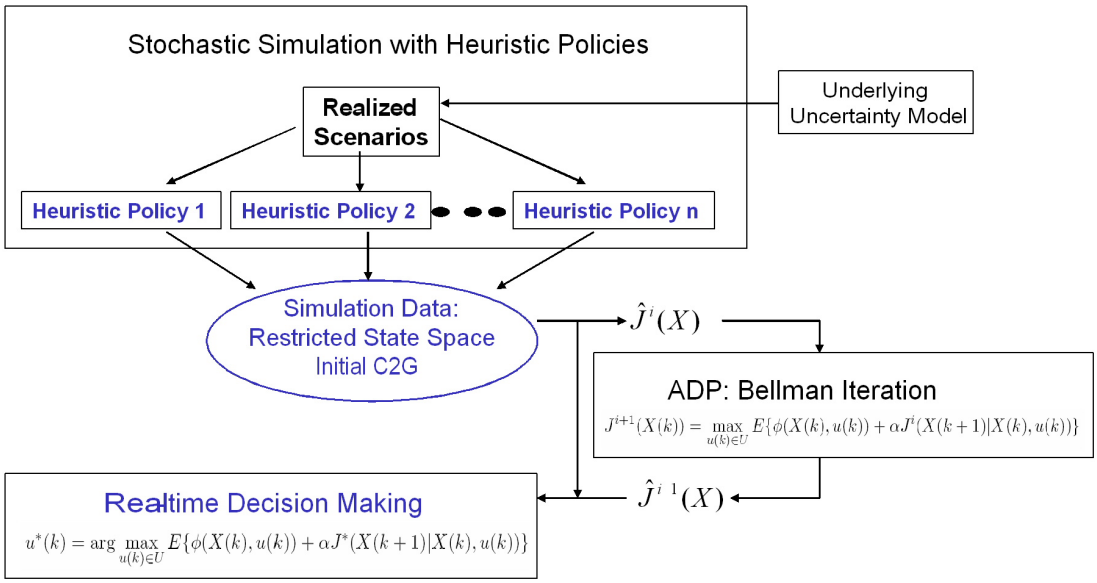

Figure 1: The Proposed Approach: Stochastic DP in the restricted state space

Generalized steps for applying the algorithmic framework is summarized as following:

1. Step 1 : Stochastic simulation with heuristic policies.
2. Step 2 : Identification of the restricted state space visited by heuristics and the initial cost-to-go approximation
3. Step 3 : Bellman iteration in a heuristically restricted state space
4. Step 4 : Real-time policy evaluation

A detailed description of the algorithm follows.

3.1.1 Simulation of Heuristic Policies

The purpose of the simulation is two-fold. First, the simulation is performed in order to obtain a meaningful set of the states within which the DP is to be performed. Obtaining a reasonably sized subset containing trajectories of good policies is critical for solving the problem because DP over the entire state space is computationally infeasible for the given problem. For the simulation, a large number of uncertain parameter realization sets are generated by the underlying Markov chains. Each realization set represents one scenario out of the enormous number of possible scenarios. Several different heuristics are applied for each realization and a set(trajectory) of visited states(as defined in 4.3.1) is obtained from each heuristic. Because each heuristic works in a different way, there can be several different state trajectories even for the same scenario. Those different state trajectories will be combined in the state space by the later step, Bellman Iteration, of the algorithmic framework. The heuristic policies applied for this problem will be described in Section 4.5.

Second, the simulation provides initial 'cost-to-go' values, which can be used in the Bellman iteration step, for the states. According to the definition of the 'cost-to-go'(4.3.4), a 'cost-to-go' value is calculated for each state in the state trajectories obtained by the simulation of the heuristic policies. The same state in different state trajectories can have different estimates of its 'cost-to-go' values according to which heuristic is used. For example, every state trajectory starts with the unique initial state, shown equation (77), but different heuristic policies may give different average values of the reward and cost. In the Bellman iteration step, one(lowest or average among the heuristics tried.) can be assigned as the initial estimate of the cost-to-go for each state.

3.1.2 Cost-to-Go calculation for the restricted state space

The total number of state trajectories obtained by the simulation of the heuristic policies(30) is $\nu \times n$, where ν is the number of realizations and n is the number of heuristics tried in the simulation. The subset should consist of all non-redundant states in the $\nu \times n$ trajectories. This step requires substantial computation. If one state appears μ times in the set of trajectories, all the realized cost values obtained from the trajectories are added and divided

by μ for the initial 'cost-to-go' value calculation. For example, the cost-to-go value for the initial state is chosen as the mean value of the total rewards minus the total costs over all the simulations. The initial guess for the 'cost-to-go' values obtained in the previous step are used as $\hat{J}^0$ to initialize the Bellman Iteration.

3.1.3 Bellman iteration

we iterate the following equation (8) for each state $X(k)$ in the subset, until $\hat{J}^i$ meets a certain convergence criteria, i.e. $||\frac{J^{i+1}-J^i}{J^i}||_\infty < 0.01$:

$$\hat{J}^{i+1}(X(k)) = \min_{u(k)} E\{\phi(X(k), u(k)) + \alpha \hat{J}^i(X(k+1)|X(k), u(k))\} \tag{8}$$

In the above, $\phi(X(k), u(k))$ represents the cost incurred by the decision $u(k)$ for the state $X(k)$. For each state in the subset, we can identify all the possible decisions, $u(k)$. Once we know the possible decisions, the expected cost can be calculated for each of the possible decisions using the conditional probability. For each one of these decisions, the possible next states and their transition probabilities are obtained analytically according to the state transition rules and the given conditional probabilities. After a sufficient number of iterations of equation (65), the converged cost-to-go $J^*(X(k))$ is obtained for every state in the subset.

- **Cost-to-Go Approximation for Partially Connected States**

  In the Bellman Iteration equation of (65), the calculation of $E\{\phi(X(k), u(k))\}$ is described. However, the exact calculation of $E\{\hat{J}^i(X(k+1)|X(k), u(k))\}$ is not possible because there is no guarantee that the subset is closed, i.e., for any state in the subset, all possible next states are in the subset as well. The subset may be open for the following two reasons:

  1. **A finite number of heuristic policies, which do not cover the entire decision space, are applied in simulation to form the subset.**

     The states in the subset are not arbitrarily chosen. A set of reasonable heuristics are implemented in simulation to collect all the visited states. Hence, in doing so, many state transitions, possible with certain decisions not covered by the

heuristics, may never have occurred during the simulation. The states involved in the unrealized transitions would not have been included in the subset. Indeed, our intention was to reduce drastically the number of states we must examine.

2. **Only a finite number of realizations are simulated**

   If the number of possible scenarios are very large, it is unlikely that one can realize every possible scenario in simulation because some scenarios have a very low probability of occurring.

The states not included in the subset due to 1 and 2 have to be distinguished and dealt with differently in the Bellman iteration. In the case of 1, a group of possible next states associated with decisions not covered by the chosen heuristics, are not visited at all in the simulation.

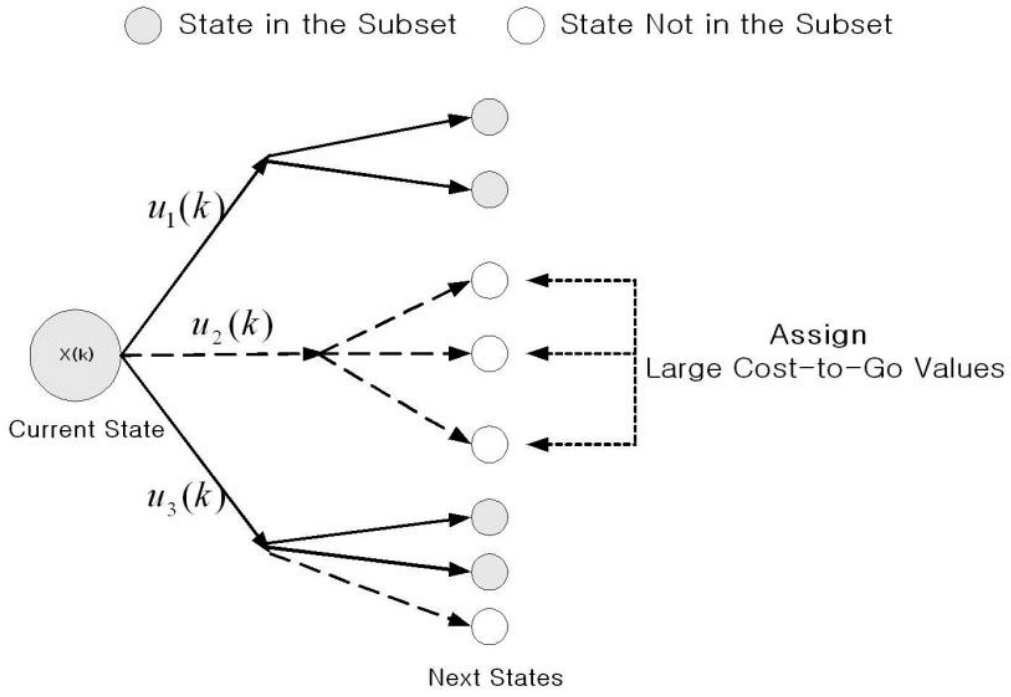

Figure 2: Cost-to-Go Approximation Type 1

In Figure 2, the decision $u_2(k)$ has never been made during the heuristic simulation. To confine the decision to those leading to a state in the subset, we propose to prevent

the unseen decision by assigning a large cost-to-go values to those states. The large cost-to-go value will act as a barrier for the decision and one of the other decisions will be chosen in the minimization step of the Bellman Iteration. This implies that the large cost-to-go values will not be propagated to current or previous states. This cost-to-go approximation is based on the assumption that a reasonable number of good heuristics have been tried and all good decisions have been covered.

For reason 2, some of the next possible states associated with a simulated decision may be absent in the subset In Figure 2, a state linked from the decision $u_3(k)$ is not in the subset because the state transition is not only governed by the decision but by random factors as well. Theoretically, all the possible states under the tried heuristic policies can be included in the subset by performing a 'sufficient' number of realizations. However, for a problem with an enormous number of scenarios, this may not be feasible. Thus, an approximation strategy is necessary to deal with this inevitable absence of some states in the subset. If a state is not in the subset due to the reason 2, it implies that the probability of transition to the state is comparatively small. Thus, we suggest that those states can be ignored and the state transition probabilities for the rest of the states are normalized accordingly as shown Figure 3.

With the proposed approximation methods, the Bellman Iteration gives 'converged' cost-to-go values rather than 'optimal' cost-to-go values. The issue of the open subset is re-examined in the next step of real-time decision making.

3.1.4 Real-time decision making

The 'converged' cost-to-go values obtained in the previous step are used for real-time decision making as follows. If the 'converged' cost-to-go, $\hat{J}^*$, is the optimal cost-to-go, the following decision $u^*(k)$ also will also be optimal according to the 'Principle of Optimality'

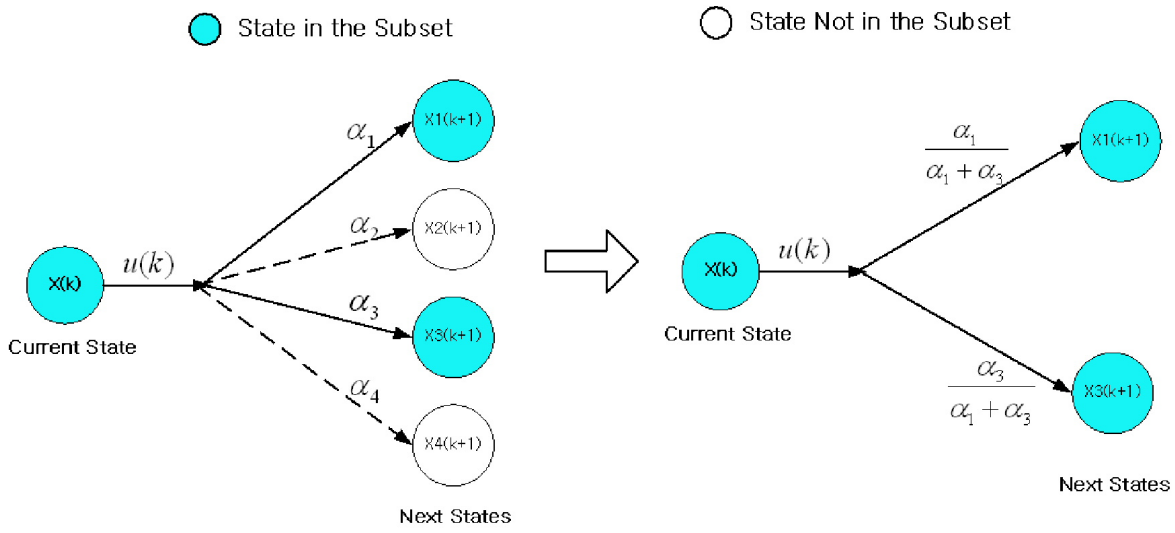

Figure 3: Cost-to-Go Approximation Type 2

of DP.

$$u^*(k) = \arg\min_{u(k)} E\{\phi(X(k), u(k)) + \alpha \hat{J}^*(X(k+1)|X(k), u(k))\} \tag{9}$$

However, the decision may be suboptimal because the converged cost-to-go, $\hat{J}^*$, is obtained by the approximation procedure described in section 3.1.2. Furthermore, the real-time decision making equation of (9) is not valid for every situation because the random factors may take the system outside the previously experienced subset. The real-time decision has to be robust for any possible realization, some of which may lead a state trajectory outside the subset, for which the cost-to-go is not available. In this work, we use the following two approaches to real-time decision making.

- Method 1 : Real-time decision making with a cost-to-go barrier

  A fixed high cost-to-go value is assigned to all states outside the subset, thereby making a decision leading to a state outside the subset highly unlikely. This approach is basically the same as the approximation method developed for the Bellman Iteration step.

- Method 2 : Real-time decision making with a guiding heuristic

  In this approach, we allow the state to step outside the subset. We use a heuristic

policy whenever a state outside the subset is encountered. The best among the tried heuristic policies, in terms of the mean value of the reward can be used for this. Once the state comes back into the subset, the decision making is switched to the minimization of the cost-to-go, as shown in Figure 26.

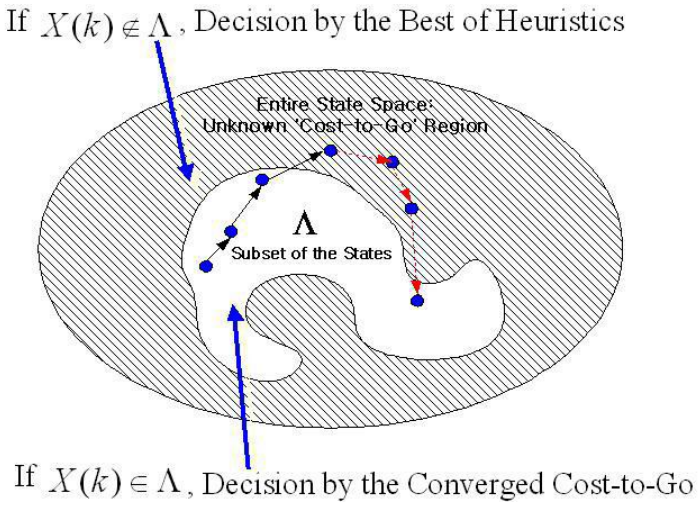

Figure 4: Real-time Decision Making with A Guiding Heuristic

3.1.5 Generalization of the algorithmic framework in discrete state space

The algorithm framework thus embodies a general idea of combining heuristics over the restricted state space by rigorous solution method (as pictorialized in Figure 5, dynamic programming). The approach can be applied to any problem formulated as a dynamic program, provided that there are reasonable heuristics available for simulation.

Furthermore, the application is also not limited by whether the problem is deterministic or stochastic, as applications of the basis rigorous solution method, DP, is flexible and accommodates both. The algorithmic framework is particularly designed to solve stochastic optimization problems with discrete state space and for those class of problems, we can state several features of the approach.

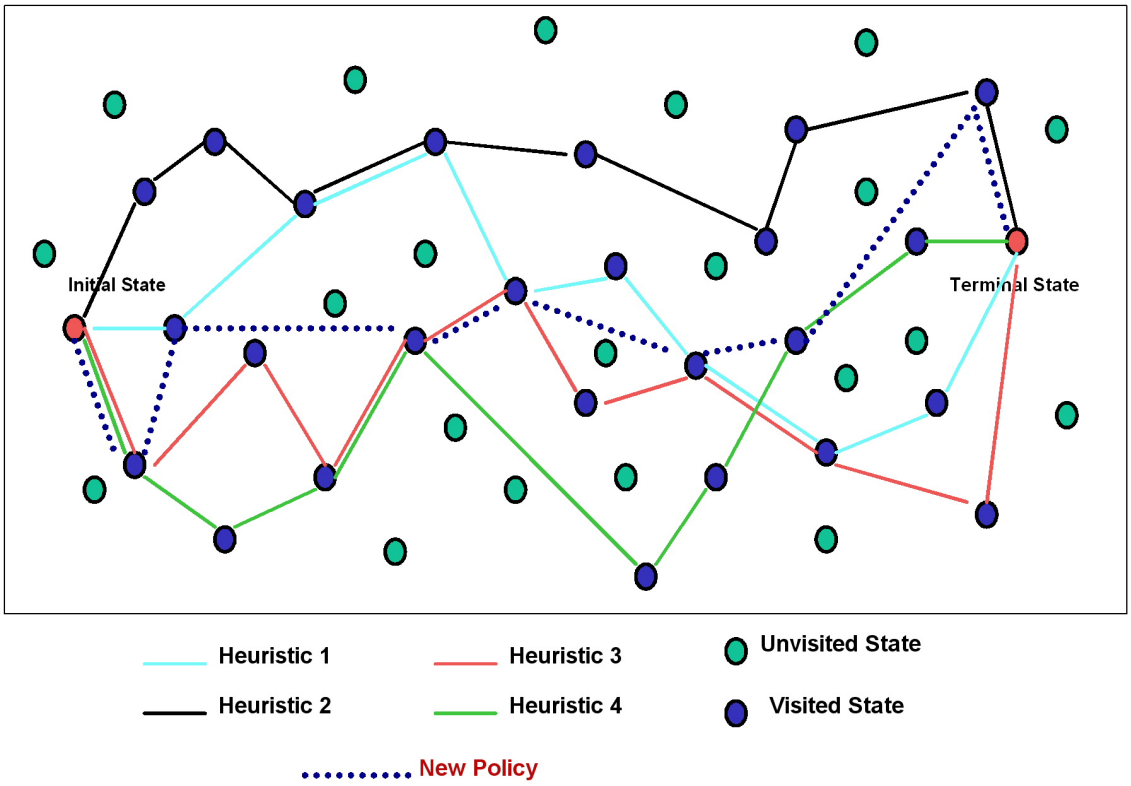

Figure 5: Combining heuristics over state space

1. The state trajectories obtained by the heuristics guarantee that a feasible solution exists for a set of experienced realizations.

2. If no states can be connected from different heuristics, the procedure will be no better than the original heuristics.

3. There is a chance of finding an improved solution by connecting states in the restricted state space found by different heuristics.

4. If the state space includes all of the states of the global optimum solution, eventually DP will lead to it – often in dramatically reduced computational time.

In later parts of this thesis, the proposed algorithmic framework is tailored for various applications in the class of stage-wise optimization problems.

3.2 Application to Deterministic Traveling Salesman Problem

3.2.1 Introduction

To verify the efficacy of the proposed algorithmic framework in the deterministic case(the stochastic case is studied in section 3.3, a new deterministic combinatorial optimization problem, a discount coupon traveling salesman problem(TSP), is introduced. The problem is interesting in itself given its higher combinatorial complexity compared to the original TSP. For the discount coupon TSP, conventional rigorous optimization problem formulations such as a dynamic program (DP) and mixed integer linear program (MILP) can be used to find a global optimum solution. However, these methods are limited to small size problems due to the rapid growth of the solution space that has to be searched with the size of the problem. Furthermore, the problem also embodies important notions, *optional* or *conditional tasks*, involved in many practical scheduling and planning problems. Here optional tasks refer to those that the scheduler has the option of determining whether and when to perform. Conditional tasks refer to those that must be performed if certain conditional requirements are met. For example, cleaning process units to remediate fouling may be related to an observation of the state of the unit or a measurement of batch quality. Additional measurements of current process conditions or batch properties could be made to enable better downstream processing and batch-to-batch control. Furthermore, measuring the properties of a batch to reveal a processing problem or an opportunity to reduce the batch times at the early stage of the batch processes are more difficult(require more cost or time) but the benefit of the investigation is larger because there are more stages over which the benefit can accrue.

Scheduling and planning problems with these types of tasks have additional complexities and some novel features with respect to traditional problem definitions and formulations. First, the task network structure is no longer fixed; it can be changed by additional tasks that may not be under the control of the planner. Second, the batch task parameters may assume values that depend on the performance of the tasks. For example, cleaning a reactor may increase the product yield. Third, the decision of whether to perform these types of tasks

is often based on information about the process state directly gathered from the process at the time of decision. Fourth, often only partial information to support the decision is available, thus what information is known at the time must be explicitly represented. The evolution of the *information state* is usually coupled with the performance of certain optional tasks. For example, optional tasks such as measurements, may not change the state of the process itself, but improve the future information available for decision-making. These issues regarding information for decision making will be elucidated in section 3.3, which treats of stochastic version of TSP.

The next section outlines the deterministic TSP variant with a discount coupon purchasing option. In section 3.2.3, four different solution methods including the proposed algorithmic framework will be introduced with a small size 10-city illustrative example. In section 3.2.3.4, we conduct a moderate scale numerical study for a 50-city problem, which is computationally infeasible with the conventional MILP and DP based solution approaches. Finally, section 3.2.3.5 summarizes our major results.

3.2.2 Deterministic Version of Traveling Salesman Problem with an Optional Task

The traveling salesmen problem(TSP) is one of a class of seemingly simple problems in combinatorial optimization with a structure that makes them very difficult to solve. It is representative of a large number of important scientific and engineering problems[51]. The TSP has been studied in the Chemical Engineering area [57, 56, 33] because of its relationship to batch scheduling problems. The class of multi-products batch scheduling problems can be characterized as a TSP because transition costs(time) are incurred by changing raw or intermediate materials which depend on a particular transition. Therefore, several scheduling problems of interest to chemical engineers have been formulated as TSPs or close variants. For example, no-wait Flowshop problem can be transformed into a TSP [57], resource constrained sequencing problem can be reduced to a resource constrained TSP [56] and parallel flowshop problem also can be transform into a constrained TSP [33]. In this section, we introduce a new type of deterministic TSP with optional tasks, which frequently arise in realistic scheduling problems.

3.2.2.1 Problem Description

To make the idea of optional tasks more concrete, we have chosen to take a classic problem in combinatorial optimization and modify it to contain the optional task structure. The problem belongs to the class of NP-complete optimization problems [31], for which no algorithm with computation time that scales as a polynomial in the size of the problem is known. The optional task TSP maintains the same basic structure of the problem, each city being visited exactly once per tour, but modifies the cost of travel and fixes the starting city. It is assumed that when a salesman reaches a city, a coupon may be purchased that will lower the "distance" or, more abstractly, the cost of traveling between the cities he has not yet visited. The discount is not applied uniformly to the costs of travel and hence the salesman may bias his tour to reach certain cities early to take advantage of discounts on other potential legs of the journey. The coupon is to be purchased exactly once or not at all during the tour, and its cost decreases with the number of cities remaining to be visited. The decisions that the salesman has to make are both the order in which to visit the cities from the given starting city and the location at which to buy the coupon. The introduction of the coupon adds a new dimension to the classic TSP. It disrupts the original problem structure. For example, to represent the problem as an integer program requires not just capturing the binary decisions of the connectivity between cities, but also the relative location of the city within the tour with respect to the coupon purchase.

3.2.3 Illustrative Example : Deterministic TSP with a discount coupon

Consider a small size(10-city) TSP with the option of buying a coupon. The effect of purchasing the coupon can be conceptualized as switching the cost matrix as shown in Figure 6. In this example, the coupon prices were chosen by drawing 10 random numbers from a uniform distribution from 0 to 120 and assigning them to stages 1 through 10 in order of decreasing value. The discount factor for each cost element was also drawn from a uniform random distribution between 0 to 0.8.

For the original TSP, without the optional task, a large number of efficient MILP solution

Original Cost Matrix

	1	2	3	4	5	6	7	8	9	10
1	0	89.5167	91.1407	90.7272	81.0302	103.4314	90.4617	71.3898	101.4641	49.0764
2	89.5167	0	73.7626	55.9636	92.8644	58.7311	111.1011	71.4369	83.3804	55.3383
3	91.1407	73.7626	0	66.6533	70.6356	61.1500	71.8187	92.9357	44.3465	78.3892
4	90.7272	55.9636	66.6533	0	110.7573	54.7909	86.7378	53.5823	76.5848	76.2541
5	81.0302	92.8644	70.6356	110.7573	0	76.1813	48.5833	59.0675	43.2402	49.5003
6	103.4314	58.7311	61.1500	54.7909	76.1813	0	56.3654	99.1209	87.2816	40.2269
7	90.4617	111.1011	71 .8187	86.7378	48.5833	56.3654	0	98.5178	77.8292	53.9221
8	71.3898	71.4369	92.9357	53.5823	59.0675	99.1209	98.5178	0	95.2300	36.2608
9	101.4641	83.3804	44.3465	76.5848	43.2402	87.2816	77.8292	95.2300	0	44.4793
10	49.0764	55.3383	78.3892	76.2541	49.5003	40.2269	53.9221	36.2608	44.4793	0

Coupon

Discounted Cost Matrix

	1	2	3	4	5	6	7	8	9	10
1	0	89.1499	52.3945	41.1297	48.6407	78.7415	67.6102	60.7255	74.0502	37.9582
2	89.1499	0	63.0154	46.9704	70.1262	35.9146	106.7932	54.8685	82.7426	29.0940
3	52.3945	63.0154	0	32.9931	68.6228	52.8924	57.6815	90.1603	18.4157	43.3183
4	41.1297	46.9704	32.9931	0	86.9778	39.2158	42.9363	53.3371	75.6485	36.5709
5	48.6407	70.1262	68.6228	86.9978	0	69.2815	21.8564	29.1561	17.7382	35.7358
6	78.7415	35.9146	52.8924	39.2158	69.2815	0	46.6727	90.5364	57.3791	34.7358
7	67.6102	106.7932	57.6815	42.9363	21.8564	46.6727	0	70.1939	41.0522	28.8477
8	60.7255	54.8685	90.1603	53.3371	29.1561	90.5364	70.1939	0	84.4898	20.5878
9	74.0502	82.7426	18.4157	75.6485	17.7382	57.3791	41.0522	84.4898	0	41.1814
10	37.9582	29.0940	43.3183	36.5709	35.6364	34.7358	28.8477	20.5878	41.1814	0

Coupon Prices for Each Stage

1	2	3	4	5	6	7	8	9	10
119.7351	104.1289	84.7613	73.0508	28.4684	23.8392	14.1330	11.8603	8.5224	2.8820

Figure 6: Cost Parameters for the Illustrative Example

algorithms and heuristics have been developed [51]. With the optional task however these methods may not apply, at least not directly. In this paper, four different solution methods will be introduced, including our novel approach based on combining heuristics within dynamic programming. Each approach represents a different level of compromise between the accuracy of solution and computational complexity.

3.2.3.1 Dynamic Programming

Dynamic Programming is a technique that can be used to solve optimization problems with a certain multi-stage structure. Dynamic programming obtains solutions by working stage by stage, usually backward from the last stage to the first stage, thus breaking up a large, unwieldy problem into a series of smaller, more tractable problems. The original TSP has been formulated as a dynamic programming problem [89] [7], and we modify this for our

particular variant.

Definition of State The *state*, denoted by X_t, consists of three pieces of information: the first two are the current city, i, and the set of cities already visited before the current stage t, which is denoted by S_t. These two are the states used for the original TSP. The additional state information is a binary variable,γ_t, indicating whether or not the coupon has been purchased. It takes the value of 1 if the coupon has already been purchased before stage t, and 0 if not. This will be termed the coupon status. Hence, the state for our problem is:

$$X_t = (i, S_t, \gamma_t) \tag{10}$$

Once the state is defined, the DP recursion can be formulated using the following equations.

$$f_t(X_t) = \min_{j \notin S_t, j \neq 1, \delta_t \in \{0,1\}} \{g_t(X_t, j, \delta_t) + f_{t+1}(X_{t+1})\} \quad \text{for} \quad t = \{1, 2, ..., N\} \tag{11}$$

$$f_{N+1}(X_{N+1}) = 0 \quad \forall X_{N+1} \tag{12}$$

$$X_{t+1} = (j, S_t \cup j, \gamma_t + \delta_t), \text{s.t. } \gamma_t + \delta_t \leq 1 \tag{13}$$

where $t = 1, 2, ..., N$ for N-city TSP and $f_t(X_t)$ represents the minimum cost that must be incurred to complete a tour if the $t-1$ cities in the set S_t have been visited, city i was the last city visited, and the coupon has been purchased already if $\gamma_t = 1$ (or not purchased if $\gamma_t = 0$). δ_t is introduced to represent the decision of purchasing the coupon at stage t. According to the problem definition in 3.2.2.1, the salesman can buy the coupon only once throughout the tour; therefore δ_t is constrained to be 0 if γ_t is 1. γ_{t+1} can be expressed as $\gamma_t + \delta_t$. In the equation (11), the current stage cost, g_t, is calculated by following equations:

$$g_t = c_{ij}^O; \text{ if } \gamma_t = 0 \text{ and } \delta_t = 0,$$
$$g_t = c_{ij}^D + P_t; \text{ if } \gamma_t = 0 \text{ and } \delta_t = 1,$$
$$g_t = c_{ij}^D; \text{ if } \gamma_t = 1 \text{ and } \delta_t = 0$$

where c_{ij}^O is the cost of traveling from the city i to j before buying the coupon, c_{ij}^D is the cost for doing the same once the coupon has been purchased, and P_t is the coupon price at

stage t.

The DP approach checks all the feasible state transitions between stages to find the minimum cost-to-go at each stage. The graphical illustration of this DP approach is shown in Figure 7 in which the dotted lines represent feasible state transitions.

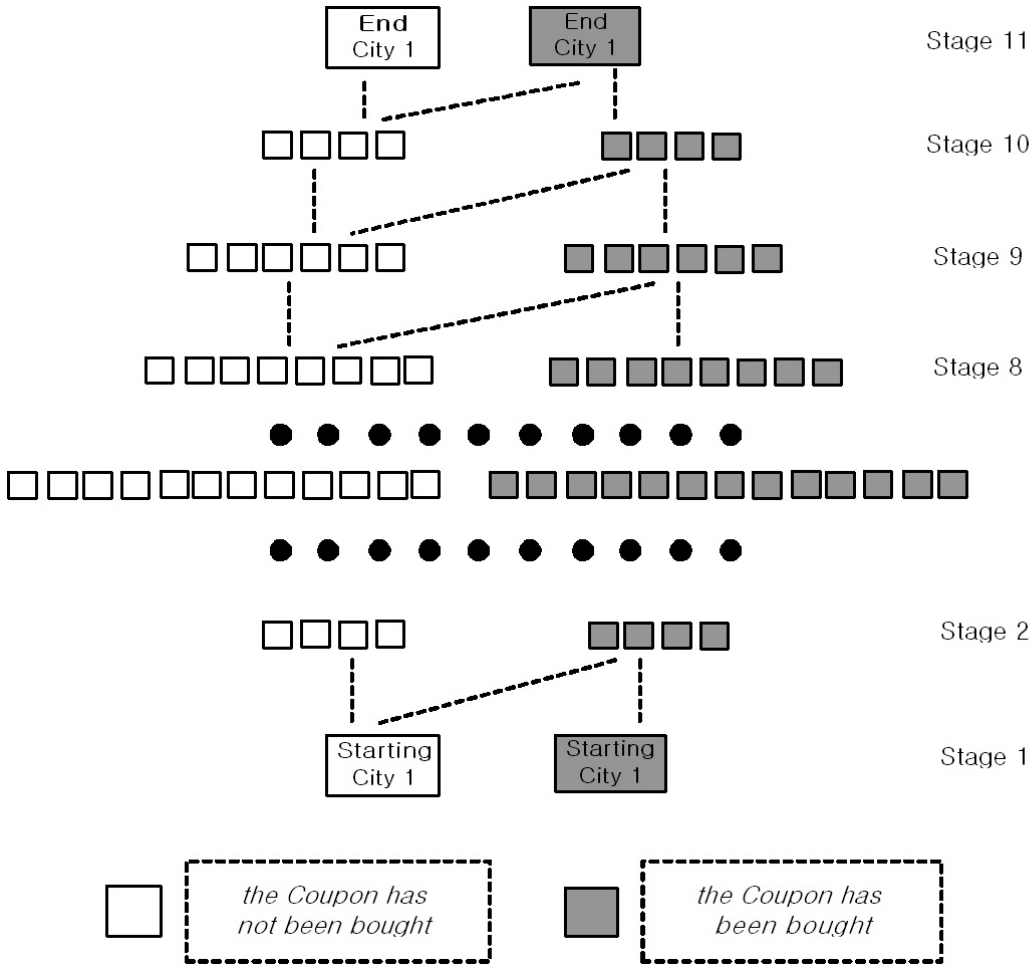

Figure 7: Dynamic Programming for the Illustrative Example

Computational Load of Dynamic Programming The computational load of this DP algorithm is directly dependent on the size of state space. Since one must always start from city 1 with or without buying the coupon, there are 2 states for the first stage. Also, since one must end at city 1 and one has the option of not buying the ticket at all, there are two possible states for the last stage.

From stage 2 to N, the number of possible states for stage t, (NP_t), can be calculated by the following equation :

$$NP_t = \frac{2(N-1)!}{(N-t)!(t-2)!} \tag{14}$$

where, N = the number of cities , $t = \{2, 3, ..., N\}$ is the stage number. Because the DP solves the problem though stage-wise recursion, the number of comparisons at each stage t is given by the multiplication of the number of states at stage t and the number of states at stage $t+1$. Considering the state transition rules (for example, one cannot "unbuy" the previously bought coupon) and using the equations in (14), the total number of comparisons required to solve this problem by DP can be calculated. There are NP_t and NP_{t-1} states at stage t and $t-1$ respectively, therefore without considering the state transition rule, there can be $NP_t \cdot NP_{t-1}$ cost-go-values incurred by connecting states at t stage and states at $t-1$ stage. The state transitions from "not bought" states at stage t to "bought" states at stage $t-1$ are infeasible. Therefore, at stage t, $\frac{3}{4}NP_t \cdot NP_{t-1}$ comparisons of cost-to-go are required.

For this example(10-city), the number of possible states is 4612 and the number of comparisons is 2,779,974 calculated by $\sum_{t=2}^{N}(\frac{3}{4}NP_t \cdot NP_{t-1})$. On the other hand, to solve this problem with explicit enumeration requires in the worst case $(N-1)!\ln((N-1)!) = 4,645,527$ comparisons. Despite the superiority of DP to the explicit enumeration, it is limited to fairly small TSPs. For example for a 50-city TSP, the total number of states goes up to 2.76×10^{16}. The computational load scales exponentially with the number of cities and the approach becomes quickly intractable. The solution obtained by using the DP approach for the given example is listed and compared with the solutions from the other methods in subsection 3.2.3.5.

3.2.3.2 MILP Formulation

The MILP formulations for the original TSP have been developed by adding constraints for eliminating subtours in the assignment problem [51]. Unfortunately, these classical MILP formulations are not directly applicable to our variant of the TSP. The solution of our variant of the TSP is an incomplete tour if we consider it as a TSP with $2N$ cities (N cities

with original cost matrix and N cities with discounted cost matrix). The compact subtour elimination constraints cannot be used to develop a MILP model for the given problem.

A MILP model for the given problem can be derived by modifying the assignment problem (see Figure 8). For a N-city problem, N cities are assigned to N+1 slots. The sequence of the cities assigned to N+1 slots must be optimized to minimize the total travel cost. Because of the starting and the ending city constraints of the problem statement, city 1 is assigned to slot 1 and N+1, although a slight modification allows the starting city to be left unspecified. The binary variables, X_{ij} are introduced to represent the assignment from city i to slot j.

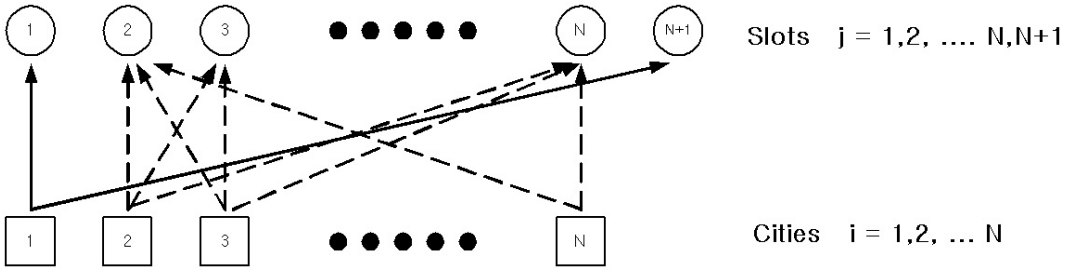

Figure 8: Assignment Problem from N cities to N+1 slots

Constraints (15)-(17) are the assignment constraints for the problem.

$$\sum_i X_{ij} = 1 \quad \text{for} \quad j = \{2, 3, ...N\} \tag{15}$$

$$\sum_j X_{ij} = 1, \quad i = \{2, 3, ..., N\} \tag{16}$$

$$X_{11} = 1, \quad X_{1N+1} = 1 \tag{17}$$

The traveling costs are incurred from the assigned cities in every pair of adjacent slots according to the 'coupon status' at slot j denoted by γ_j as we denoted γ_t in DP formulation. The transition cost from slot j to $j+1$ can be expressed by the following constraints:

$$C_j \geq c^D_{ik}\gamma_j + c^O_{ik}(1-\gamma_j) - (2 - X_{ij} - X_{kj'})M \tag{18}$$

$$i, k = \{1, 2, ..., N\} \quad \text{and} \quad j = \{1, 2, ..., N\} \quad , j' = j + 1$$

where, C_j is the transition cost from slot j to $j+1$ and γ_j the coupon status at slot j

M is a big "M" value.

The minimum value of C_j satisfying the constraint (18) represents cost incurred between slot j and $j+1$ according to the assignment decisions X_{ij}, $X_{kj'}$ and the coupon status γ_j. For example, when $X_{ij} = 1$ and $X_{kj'} = 1$, city i is assigned to slot j and city k is assigned to slot $j+1$, which means the segment tour from city i to k is chosen. Hence the transition cost from slot j to $j+1$, C_j, is bounded by $C_j \geq c^D_{ik}\gamma_j + c^O_{ik}(1-\gamma_j)$, and according to the coupon status, γ_j, the active bound for C_j is decided. For a reasonable relaxation, the maximum cost element in the original cost matrix and the discounted matrix can be used for M.

From the problem definition in 3.2.2.1, the salesman can buy the coupon only once during his tour and the coupon cannot be "unbought" once purchased. The following constraints ensure this:

$$\gamma_j \leq \gamma_{j'}, \quad j' = j+1 \tag{19}$$
$$j = \{1, 2, ..., N-1\}$$

Then, the coupon price is calculated by constraints (20)-(21).

$$Z \geq \gamma_1 P_1 \tag{20}$$

$$Z \geq (\gamma_{j'} - \gamma_j) P_{j'} \tag{21}$$
$$j = 1, 2, ..., N-1 \;\; j' = j+1$$

where, P_j : given coupon price at stage j, for $j = \{1, 2, ..., N\}$
Z : optimum coupon price(decision variable)

The objective of this problem is to assign the cities and coupon buying status to the slots to minimize the total cost incurred by the assignments. This objective can be simply formulated by the equation (22) under the constraints of (15)-(21).

$$\min_{C_j, z} \sum_j C_j + Z \tag{22}$$
$$j = \{1, 2, ..., N\}$$

The proposed MILP model is well defined for the problem and can guarantee the optimal solution of the problem. But the number of constraints and the number of binary variables

of the model can become very large even for a small size TSP. For an N-city TSP with the additional coupon buying option, the number of binary variables is $N(N+2)$ and the number of constraints is of order N^3 because of the constraints (18). Furthermore, the "big-M" relaxation introduced in the constraints (18) increases the integrality gap of the LP relaxation leading to poor computational performance. We cannot tighten the value of 'big-M' because any pair of cities can be assigned to any pair of slots, hence the largest cost element could occur at any slot j.

3.2.3.3 Heuristics

For the original TSP, many heuristics have been developed to find suboptimal but 'good' solutions in reasonable time for large N ($> 10^6$) TSPs. For our TSP example, we consider two heuristics for finding suboptimal solutions. The main idea behind the heuristics is to solve a TSP and then modify the solution using a shortest path problem to reflect the change in the cost matrix that occurs after buying the coupon.

Heuristic 1 This heuristic can be described by the following procedure and Figure 9.

1. Solve the TSP with the original cost matrix and the fixed starting city to obtain the optimal tour, set $i = 1$
2. For the option of purchasing the coupon at the ith city in the optimal tour, follow the obtained optimal tour until the ith city
3. Solve the shortest path problem with the discounted cost matrix for the rest of the tour after the ith city
4. $i = i + 1$, while $i \leq N$, repeat from 2.

Heuristic 1 determines the first part of the tour from the optimal tour obtained with the original cost matrix. The tour after purchasing the coupon is found by solving a shortest path problem through the rest of the cities. *Heuristic 1* generates N different suboptimal solutions with N different coupon buying locations for the N city problem.

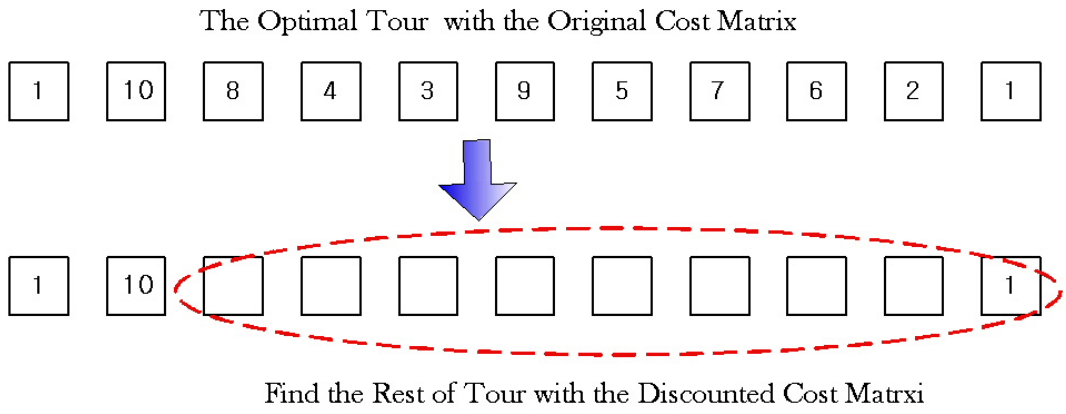

Figure 9: Pictorial Illustration of Heuristic 1

Another N suboptimal solution can be obtained by inverting the optimal tour with the original cost matrix and proceeding as before. As a result, for a N cities problem, the *heuristic 1* generates $2N$ suboptimal solutions.

Heuristic 2 This heuristic follows the same idea as in *Heuristic 1* but we reverse the sequence. First, we determine the optimal tour for the regular TSP with the discounted cost matrix. In this case, the tour obtained with the discounted cost matrix gives the second part of the suboptimal tour because the discounted cost matrix is in effect after buying the coupon. The the following procedure for *Heuristic 2* is also displayed pictorially in Figure 10.

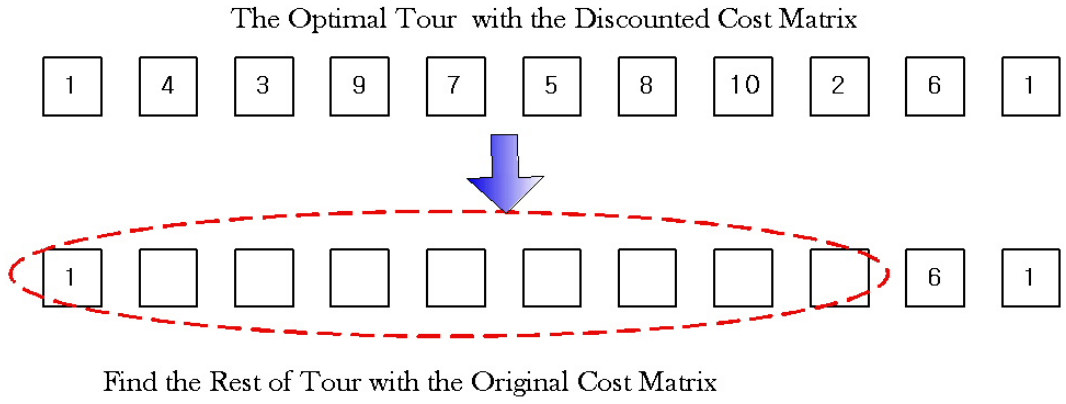

Figure 10: Pictorial Illustration of Heuristic 2

1. Solve the TSP with the discounted cost matrix and the fixed starting city to obtain the optimal tour, set $i = N$.

2. For the option of buying the coupon at the ith city, follow the obtained optimal tour from the ith city to the end

3. Solve the shortest path problem with the original cost matrix for the first part of the tour before the ith city

4. $i = i - 1$, while $i \geq 1$, repeat from 2.

As in *heuristic 1*, for an N city TSP, $2N$ suboptimal solutions can be obtained from *Heuristic 2*.

Quality of the Solution and the Computational Load of the Heuristic 1 and 2 The difficulty of solving the given problem arises from buying coupon, which must be considered simultaneously with the sequence of cities. This major difficulty is eliminated in the above heuristics because the coupon buying decision is treated as an iterative routine from the first city to the last city after separating the problem into a TSP and a shortest path problem, the latter of which is solvable in polynomial time. The elimination of the major difficulty is an advantage in terms of computational load but a disadvantage in terms of the quality of solution. The computational loads of these heuristics are relatively small compared to those of the DP approach and MILP formulation. They require solving an original TSP and a shortest path problem for each coupon buying stage. The heuristics still give a high quality solution, although the problem is separated into 2 parts, *a priori*, and the optimal solution is found for each part of the problem. Furthermore, it also checks every possible coupon buying stage from 1 to N. Hence the best solution among the $4N$ suboptimal solutions obtained by the heuristics can be regarded as a 'good' suboptimal solution.

3.2.3.4 DP in the Subset of the States Visited by the Heuristics

DP is shown in 3.2.3.1 as a solution method that can guarantee the global optimum for this type of problem. But DP is often not applicable to practical problems due to the exponential growth of the state space. In theory, the portion of the state space that has to be visited by an "intelligent" algorithm consists of just N states, those on the optimal path, a vanishingly small fraction of the overall state space. Identifying this restricted state space, without searching the state space, clearly must be as intractable as solving the original problem. However, finding some small regions of the state space that might contain the optimal (or very good suboptimal) subset and then searching them rigorously, using DP, could prove to be a tractable approach for large problems. The idea is to use heuristics to identify the relevant regions of the states and use DP to "patch" these states together. The proposed method in this section describes how to obtain this *relevant subset of the states* and find an optimal path of states within the subset. When there are several reasonable heuristics and an appropriate state definition and DP formulation for a certain optimization problem, the heuristic solutions can be translated to a set, or trajectories, of states defined in DP formulation. The same state can be visited by different suboptimal(heuristic) solutions and this will happen more frequently when heuristics that exploit the problem structure in a similar manner are used. The key idea of our method comes from hypothesizing that the states visited by several reasonable heuristics represent a 'good' subset of the state space, within which search for a 'good' solution can be conducted. The DP in the restricted state space follows the same algorithms as the full DP, except some of the states in adjacent stages cannot be connected.

DP in the Subset of the States for the TSP example The generalized proposed method described in 3.2.3.4 can be tailored for the TSP example by using DP formulation(3.2.3.1) and heuristics(3.2.3.3) for the problem. In subsection 3.2.3.3, two heuristics for the given problem were developed based on the idea of first finding good lows and then modifying the before or after coupon purchase. For many well-known types of combinatorial optimization problems, a large number of heuristics can be and have been developed. Often, certain

heuristics can be modified by changing a parametric description of them. For example, the 'Nearest Neighborhood' search [89], a well known greedy algorithm for TSP, can be parameterized by an explicit description of the neighborhood operator. The N city problem has N stages, thus from one suboptimal solution, N visited states are obtained. Since $4N$ suboptimal solutions can be obtained from the two heuristics we introduced earlier, at most $4N$ states are visited by these heuristics at each stage. The reduction of the search space is often dramatic, thus making the approach feasible for even very large problems. For the given 10 city illustrative example, the size of this subset of the states(101 states) is much smaller than the entire state space (4612 states), which is used for the DP in subsection 3.2.3.1 as shown in Figure 11. Figure 11 also shows the number of feasible state transitions is also much smaller than that of original state space. The reduction of the state space to the subset visited by the two heuristics enables a considerable reduction in the computational load compared to the original DP.

Guideline for Expanding the Subset For the given example, states in the subset is only 2%(101 to 4612) of the original states. Although those states in the subset are "good states", it is unlikely that it contains all of the states belong to the global optimal solution trajectory. The possibility of obtaining a better solution by performing DP within the subset of the states can be increased by expanding the subset so that it includes more "good states". The extreme case of this expansion would be the original state space, in which the DP method can guarantee the global optimal solution but computationally intractable. Therefore, an important issue is how to expand the subset intelligently to increase the possibility of improving the suboptimal solution, while keeping the size of the subset relatively small compared to the original state space. Moreover, for the given example, just adding states to the subset in a random manner does not help to improve the solution unless the added states are connected to the states in the subset according to the state transition rules. Hence, the added states must be feasible in terms of the state transition as well as 'good' to

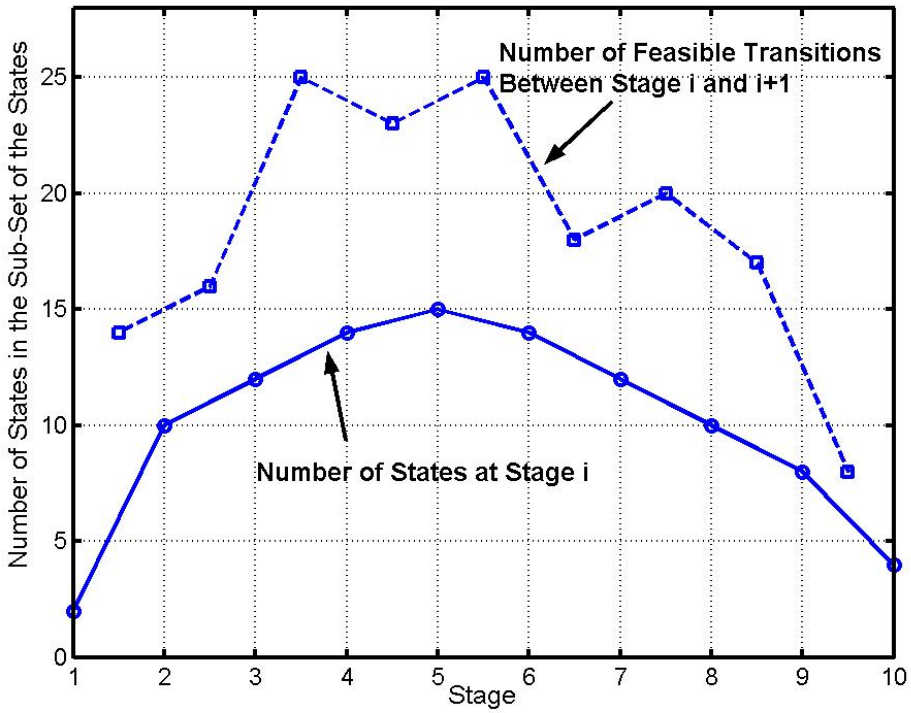

Figure 11: Number of States and Feasible State Transitions in the Subset of the States in the Illustrative Example

improve the suboptimal solution. We suggest following subset expansion guideline for the example.

1. Set $i = 2$
2. Set $j = 1$
3. Switch the ith city with $(i + j)th$ city in the suboptimal tour obtained by DP in the subset of the states and add the corresponding new states obtained by the switching
4. $j = j + 1$, while $i + j \leq N$, repeat from step 3
5. $i = i + 1$, while $i \leq N - 1$, repeat from step 2

Using the above guideline, we can add at most $2\sum_{k=1}^{N-2} k = N^2 - 3N + 2$ new states. As a result, the expanded subset is still small enough to make the DP tractable for a very large

N. And all of the added states are feasible in terms of state transition because they are generated by switching the suboptimal route. In this particular case, the expansion produced no improvement over the DP in the original subset. In general there is no systematic way to produce polynomially bounded expansions of the space that can guarantee improvement.

3.2.3.5 Comparison of Solutions

For the illustrative example, the 4 different solution methods have been tested. Among these 4 solution methods, the *DP* and *MILP* approaches can guarantee the global optimum solution. On the other hand, the other two methods, the *heuristics* and *DP in the subset of the states* are computationally more tractable, even though they cannot guarantee the global optimum solution. The comparison of the solutions by the four methods must be based on two points, the degree of optimality of the solutions and the computational time used to obtain the solutions. However, comparing the MILP method with the other methods is not appropriate because different languages were used to pose and solve the proposed MILP from the other solution methods. The proposed MILP is solved by using *CPLEX 7.0* in *GAMS* [15] and *MATLAB* is used for the other solution methods. Generally, the speed of computation based on *MATLAB* is much slower than that with GAMS. For this reason, in Table 3.3.5.1, we compare the computational times for the three solutions obtained by *MATLAB* only. However, it should be noted that the computational time of solving the proposed MILP with 0.01 error bound on a same machine is 1015.0 seconds. As a part of the two heuristics, one must solve the original TSPs without the coupon buying option. For this purpose, a TSP solver is coded in *MATLAB* with the *simulated annealing algorithm* [1].

Table 1: Solution Comparison

	DP Solution	The Best Heuristic	DP in the SSS^{+}
Total Traveling Cost	416.34	432.54	422.78
Calculation Time (Sec.)*	30546.8	8.80	8.8+1.8=10.6

$^{+}$ Subset of the States
* On a Pentium III at 800 MHz: 512MB RAM

A greedy heuristic for the original symmetric TSP, the Nearest Neighborhood Search [89], which can find the globally optimal or nearly optimal solution for relatively small size(less than 100 cities) instances of TSP is used to provide a good initial solution for the

simulated annealing algorithm. The *simulated annealing algorithm* TSP solver starts its stochastic cooling from a temperature of 60 until it cools down to 5 with a reduction rate of 0.99. For most TSPs with less than 50 cities, the TSP solver can find the global optimal solution owing to the good initial solution and the high temperature reduction rate(0.99). As we expected, the DP method finds the global optimal solution and the other methods result in suboptimal solutions. Table 3.3.5.1 highlights the efficiency of the proposed method for performing DP within the subset of the states visited by the heuristics. The additional computational time for performing DP within the visited set is trivial because of the dramatically reduced state space. At the expense of small additional calculation time on top of the heuristics, we can obtain a significantly improved solution.

Furthermore, we can see that the solution from the *DP in the subset of the states* approach is close to the global optimum in this particular example. To measure the quality of each solution by different solution method, exhaustive enumerations are performed. The best 5 feasible solutions obtained by exhaustive enumerations of all feasible solutions are shown in Table 2. It turned out that the solution by the method is the second best solution.

Table 2: Comparison of Solutions Between the Best Heuristic Solution and the Optimal Solution

	Solution Method	Traveling Cost	Route
The Global Opt.	DP or MILP	416.34	1-10-8-4-***2-6-7-5-9-3-1***
2nd Ranked Soln	*DP in the SSS*	422.78	1-10-8-2-***6-7-5-9-3-4-1***
3rd Ranked Soln	Enum^{+}	427.84	1-10-8-4-6-***7-5-9-3-4-1***
4th Ranked Soln	Enum^{+}	431.23	1-8-2-6-***10-7-5-9-3-4-1***
5th Ranked Soln	Heuristics*	432.54	1-10-6-2-***8-5-7-9-3-4-1***

Italic Bold represents discounted tour

$^{+}$ Solution method: Enumeration

* Solution Method: Best among all the solutions by the heuristics

3.2.4 Statistical Analysis of Larger Deterministic TSPs with a Discount Coupon

In section 3.2.3, we alluded to the computational limitation of the DP and MILP methods in solving the new TSPs of large sizes. For larger problems, the two *heuristics* and *DP in*

the restricted state space are the only computationally tractable methods. A specialized solver for the particular MILP may make the solution of large-size problems feasible, but developing this was beyond the scope of this research. In this section, the performance of the proposed method is tested for randomly generated 50 city TSPs with the discount coupon purchasing option.

Random Parameter Generation In the 50 city example, there are 4500 total parameters involved with the original and discount cost matrices and coupon prices. All of these parameters are generated in a random manner using the following procedure:

- Cost Elements in the Original Cost Matrix : Uniform random variables ranging from 24 to 80.
- Discount Factors for the Cost Elements : Uniform random variables ranging from 0.3 to 1.0.
- Coupon Prices at the 50 Stages : Uniform random variables sorted in decreasing order, ranging from 0 to 850.

Elements of the discounted cost matrix are obtained by multiplying the discount factors to the corresponding cost elements of the original cost matrix. According to the random number generation routine for the discount factors, the maximum discount rate can be 70% of the original cost.

3.2.4.1 Statistics of Improvement

100 sets of parameters were generated by the random parameter generation routine and the corresponding 50 city TSPs are solved by the proposed method. To implement the proposed method, 100 TSPs are solved by the two heuristics and for each case the subsets of the states visited by the heuristics are extracted. For each subset of the states, DP is performed and the solution of the DP is compared to the best heuristic solution. Figure 12 shows the number of improved cases and the amount of improvement in the solution.

Out of the 100 cases, 35 cases showed improvement from the best solutions found by the

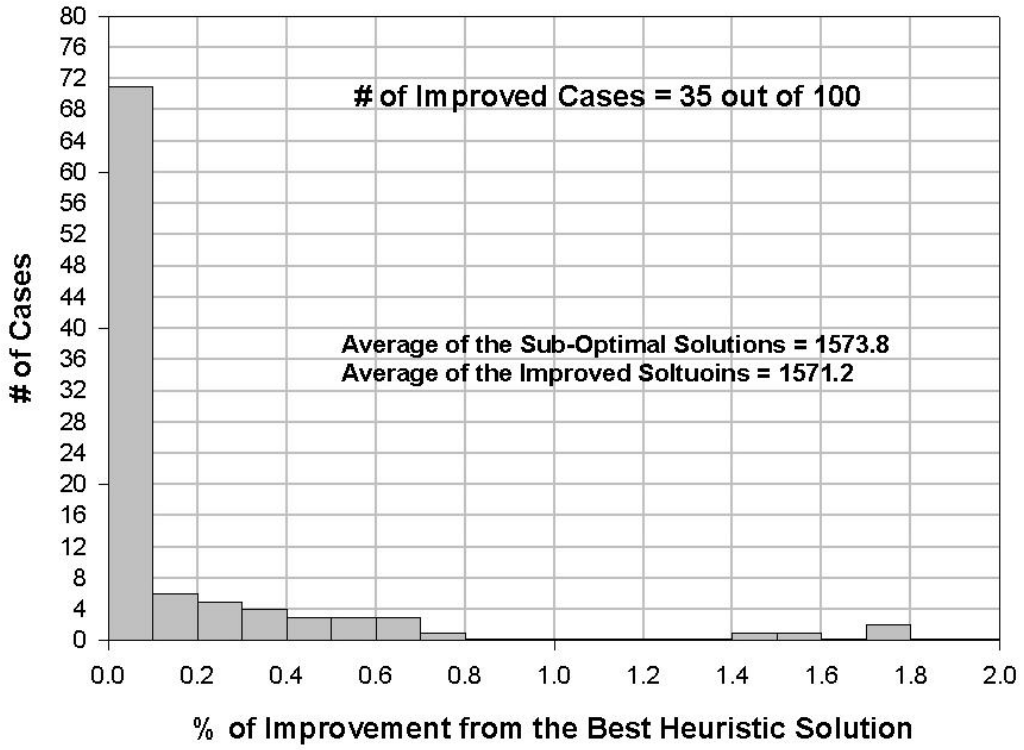

Figure 12: Statistical Improvement of the Solutions by the Proposed Method(Rigorous Original TSP Solver)

heuristics. The occurrence of a small number of improved cases and low percentage improvement may be due to two reasons. First, the size of the subset is too small compared to the entire state space. The average number of states in the subset used by the proposed method is only 3662. The number of states in the original state space is 2.76×10^{26}. Hence, the proportion($\frac{3662}{2.76 \times 10^{26}}$) of the states in the subset is miniscule. Of course, this proportion can be expanded by employing more heuristics, possibly with better results. Second, although we cannot verify the optimality gap between the heuristic solution and the global optimal solution for the larger examples, the heuristic solutions may be optimal or nearly optimal in many cases and hence no significant improvement may be possible.

3.2.4.2 The Effect of the Suboptimal TSP Solver

The computational time of the proposed method can be attributed to running to the heuristics necessary to obtain the subset of the states. The biggest computational load of the

heuristics is in solving N-city symmetric TSPs because the time required for solving the shortest path problem is trivial compared to that for TSP. As mentioned in 3.2.3.5, for rigorous calculation, *simulated annealing algorithm* was used for the TSP solver implemented in the heuristic method. The accuracy of TSP solver can be relaxed by decreasing the temperature reduction factor(hastening the cooling process) of the annealing process. The complete relaxation of the TSP solver corresponds to obtaining the solution without any annealing process by setting the temperature reduction factor as zero. In the case of complete relaxation, the solution is same as the initial starting point of the simulated annealing, the solution of the Nearest Neighborhood(NNH) Search[89]. When the NNH search is used as a TSP solver, the computational time for the proposed method can be dramatically reduced by sacrificing the quality of the suboptimal solution.

Solver Rigorousness VS. The Improvement The same 100 TSPs with a discount coupon option introduced in 3.2.4.1 are solved by the proposed method with a relaxed heuristic method that performs only the NNH search to obtain the solution of the subproblem. The Figure 13 shows the number of improved cases and the amount of improvement from the best heuristic solution by the proposed method.

The Figure 13 shows almost same trend of histogram as the Figure 12. Specifically, in cases of the largest improvement, bigger than 1.4%, it shows exactly the same trend for the same sets of the parameters. From the above results, we see that the proposed method has a similar effectiveness for improving solutions from different heuristics. It implies the proposed method can be applied to cases with worse heuristics to achieve a certain amount of improvement.

3.2.5 Conclusions

The algorithmic framework is applied for improving solutions from applying heuristics for deterministic combinatorial optimization problems. The key idea of the proposed method is to perform DP in a subset of the states visited by the heuristics. To test the proposed

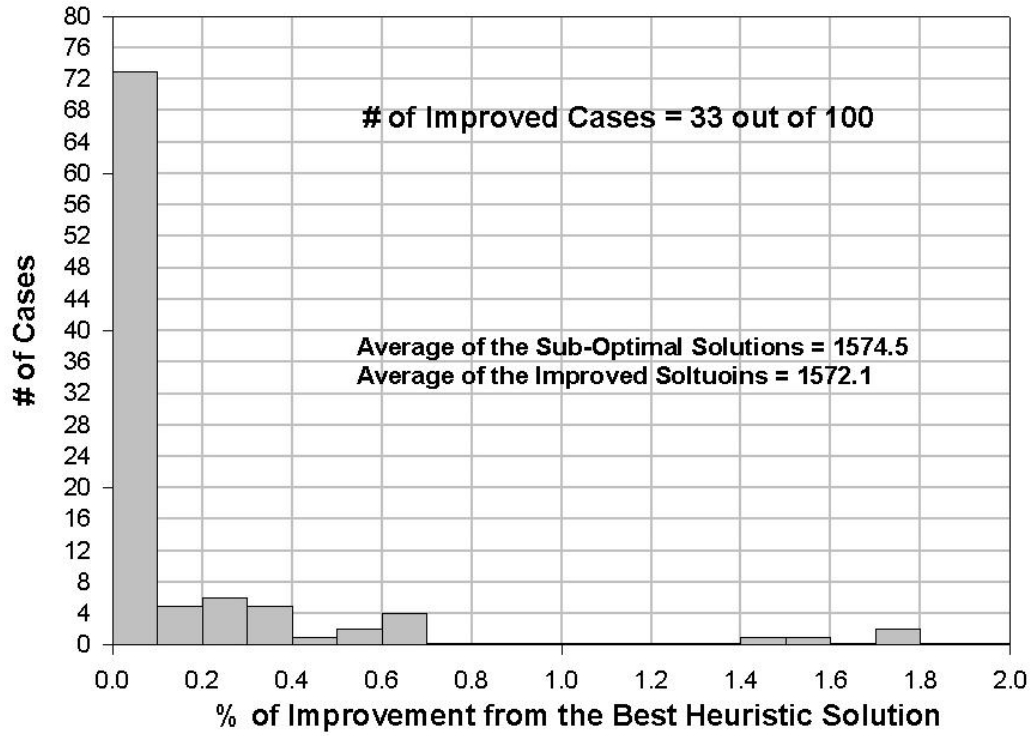

Figure 13: Statistical Improvement of the Solutions by the Proposed Method(Pure Heuristic Original TSP Solver

mathematical framework, a new variant of the deterministic TSP was introduced. This variant includes an optional task that changes the problem cost structure. A new variant of the deterministic TSP included the option of switching the cost matrix to a cheaper one for a price. Four different solution methods, DP, MILP, heuristics, and DP in the subset of the states were applied to this problem. The performance of these 4 solution methods was tested for a 10-city illustrative example. Among the 4 solution methods, DP in the subset of the states showed significant advantages in terms of computational time and solution quality. The performance of the proposed method was also tested for larger examples of the TSP variant, which are computationally intractable with the other methods. The proposed method showed good performance in these problems.

3.3 Application to Stochastic Traveling Salesman Problem

3.3.1 Introduction

Planning and scheduling problems for chemical production systems are a major focus of study as companies seek to lower operating costs with minimal capital investment. A significant challenge is to represent and account for the diverse sources of uncertainty that arise as the scope and complexity of the problem expand [71]. These uncertainties include processing time variations, rush orders, failed batches, equipment breakdowns, market trends and every problem will have its own unique set of uncertainties. The intractability of the general problem has led to the formulation and solution of deterministic scheduling and planning problems in chemical production systems [60, 41, 42, 63, 49, 72]. The progress in solving problems that involve uncertainty, [76, 84, 39, 59], has been limited to synthesizing solutions that are robust to the uncertainty rather than reactive to the changing conditions as they are realized, with the exception of [76]. There is little work on how to systematically use the information gained during the execution of the partial schedule proactively to influence future scheduling decisions based on revised information, [26]. Therefore, developing a systematic way to model uncertainties in the process and applying it to find the optimal solution, is one of the most challenging issues in the scheduling and planning area. One type of uncertainty is within the process itself, such as the quality of a batch at an intermediate stage. In this case, additional measurements of current process conditions or batch properties could be made to enable better downstream processing and batch-to-batch control. These measurements are labelled optional tasks [27] and may trigger new processing tasks to be performed on batches that do not meet specifications. Scheduling or planning problems involving these optional tasks and stochastic parameters require the solution of decision problems that have significant combinatorial complexity, layering the decisions about whether and when to perform the optional tasks on top of other decisions. Furthermore, the dynamically evolving nature of information for decision making makes the problem multi-stage in nature, as the new information state can be used to revise existing scheduling or planning decisions.

The purpose of this work is three-fold. First, we extend the algorithmic framework for

improving heuristic solutions developed and verified for the deterministic case in section 3.2 to the stochastic case. Second, we introduce a new version of TSP, stochastic TSP with an optional task(investigation) for reducing uncertainties. Several variants of the original deterministic TSP have been studied in the chemical engineering area and related to certain batch scheduling problems [57, 56, 33]. The original TSP itself represents the parallel flowshop scheduling problem because the scheduling problem can be transformed into an extension of the original TSP, the generalized TSP(GTSP)[33], which can be transformed back into the original TSP [54]. Hence, in this study, we add a stochastic component and an optional task to the original deterministic (symmetric) TSP to make it representative of scheduling problem with uncertainties. Third, a *discrete-time Markov process* is introduced as a way to model uncertainties in key parameters. Besides developing an efficient solution method, developing proper ways to represent uncertainties in the formulation of optimization problems is also very important for practical applications. In previous literature uncertainties in scheduling problems were introduced in two different ways, with scenario based representation [84, 75] and with probability distribution functions [39, 59, 76]. In both approaches, solution methods were based on MILP(or MINLP) formulations for deterministic equivalent or stochastic models of the problems. However, these formulations have some inherent limitation for solving complex stochastic scheduling problems because it only considers a "snapshot" of uncertain parameters by means of their expected values. Even with the reactive scheduling framework in which the expected values can be updated, this inherent limitation cannot be removed. Some recent literature [90, 50] point to the fact that Markov process is an attractive alternative for representing uncertainties in planning, scheduling and supply chain problems. Suppose the uncertain parameters are changing at each time unit according to some underlying probability distribution, unknown to the decision maker. Some of uncertain parameters are strongly correlated(i.e. processing time and processing cost) so that they vary together as a set.

With the Markov process representation, DP, which theoretically guarantees the optimal solution, is the natural solution method for the problem since the use of Markov process automatically implies that the problem has stage-wise characteristics.

The progression of the work is as follows. Section 3.3.2 will present the details of the stochastic TSP with an optional task. Section 3.3.3 contains the possible solution methods for the given problem, stochastic DP, suboptimal heuristic policies, and the proposed method, stochastic DP in the subset of the states. Section 3.3.4 will verify the efficacy of the proposed method with an illustrative example. This is followed by some concluding remarks and future works in section 3.3.5.

3.3.2 Stochastic Version of TSP with an Optional Task

In the past decade, the stochastic version TSP has been introduced by modeling each cost element as a random variable [64, 58]. In this work, we address a new version of stochastic TSP in which several cost modes, set of the cost elements, are changing stochastically according to a *Discrete-Time Markov Process*. A new feature, an optional task [27], is introduced to the new version of stochastic TSP to represent an opportunity to investigate the identity of the current cost mode.

3.3.2.1 Problem Description

A salesman is assigned to travel a set of N cities H times. If the cost matrix of the problem is deterministic, he need to follow the same route obtained by the deterministic cost matrix at every tour to minimize the total traveling cost for H tours. Instead, suppose that the cost matrix evolves tour to tour according to a given Markov process. Suppose there are M cost matrices with $N \times N$ cost elements(for a N-city TSP). Each matrix represents one possible *cost mode*, in which the salesman must find the optimal tour, which can be different for different cost modes. At the end of each tour or stage, one of the M modules is chosen according to the Markov process, which is unknown to the salesman. The transition probabilities from a mode i to j, P_{ij}, thus form an $M \times M$ transition matrix, which describes the governing dynamics of the cost mode change. We make two further refinements to this model:

- **Unobserved Stochastic Process** : The salesman does not know which cost mode he will experience on any given tour. However, he is informed of the cost matrix that governs his first tour.

- **Cost Mode Investigation** : Before starting a tour, the salesman has the option to determine the current cost mode by paying an investigation fee, β.

The investigation option complicates the decision problem, affecting the choice of optimal tours at subsequent stages. Therefore, to minimize the total cost of traveling over a certain number of tours, the tours before which the investigation is to be performed become important decisions. Frequent investigation will enable an accurate decision for the current tour but the total cost may be increased by the high investigation fees. On the other hand, too rare investigation may increase the total cost, because of the inaccurate decisions due to increased uncertainty.

3.3.3 Solution Methods for The Stochastic TSP

For the given problem, stochastic dynamic programming [7, 8] is an exact solution method, which can guarantee the optimal expected cost over a given horizon. However, stochastic DP requires significant computation to obtain the optimal cost-to-go value for each state because the dimension of the state increases due to the necessary information state introduced by the uncertainty of the system. The Bellman iteration(cost iteration) has an exponential complexity in the number of states. In this section, we develop suboptimal policies of high computational efficiency as well as the stochastic DP for the given problem. The role of suboptimal policies corresponds to the "heuristics" for the deterministic TSP developed in our previous work [27].

3.3.3.1 Stochastic Dynamic Programming

To develop an appropriate stochastic DP formulation for the given problem, all the necessary information of the problem must be represented explicitly in the state. The key information is the conditional probability of each cost mode at each tour. The information state $\hat{x}(k)$ is defined as the *conditional* probability of the 'cost mode' at tour k before the decision.

$$\hat{x}(k) = \begin{bmatrix} Pr\{CM_1\} \\ Pr\{CM_2\} \\ \vdots \\ Pr\{CM_M\} \end{bmatrix} \tag{23}$$

where $Pr\{CM_i\}$ denotes the conditional probability of realizing 'cost mode' i. For the state $\hat{x}(k)$ defined in (23), the state transition rules can be derived from the transition matrix of the Markov chain. With the investigation at step k, we set the investigation indicator $\delta(k) = 1$ and the information state for the tour decision changes to $\hat{z}(k)$, which represents the exact knowledge of the cost mode at tour k be the tour, as a benefit of the investigation. If the current cost mode is C_i,

$$\hat{z}(k) = e_i \tag{24}$$

where, e_i is $M \times 1$ elementary vector with all zero elements except 1 in ith position. For example, if the cost mode is 3 at tour k, the information state is reset to $[\ 0\ \ 0\ 1 \cdots\ 0\ ]^T$. If the particular realization of the cost mode at the time of investigation is i, then the next state $\hat{x}(k+1)$ is calculated by following Markov transition equation.

$$\hat{x}(k+1) = P^T \hat{z}(k) \tag{25}$$

On the other hand, without investigation at time $k(\delta(k) = 0)$,

$$\hat{z}(k) = \hat{x}(k) = P^T \hat{x}(k-1) \tag{26}$$

This is the information state propagates by the same transition rule of the equation (81) The overall procedures of the stochastic DP are summarized in the following Figure 14.

Simulation and the First Cost-to-Go Approximation The first step is realizing the random cost mode for simulation purposes. The realization of the random cost mode is

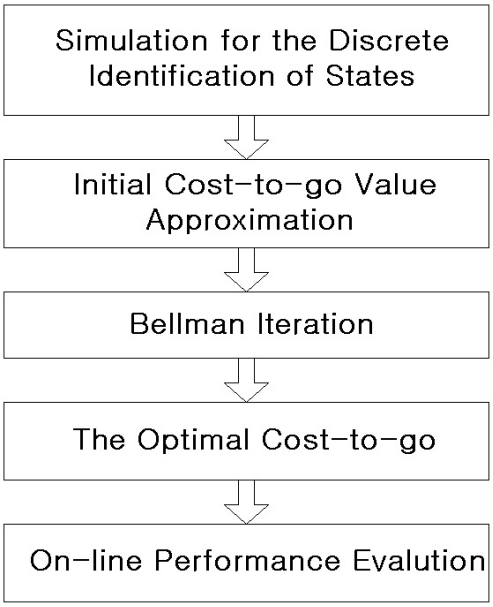

Figure 14: Overall Procedures of Formulating, Solving and Testing the Stochastic DP

started by choosing an arbitrary cost mode at time 0 and then evolving the cost mode according to the state transition probability matrix P. For sufficiently long cost mode sequences, the overall portion of each cost mode should be same according to the limiting probability of P, denoted by P^{∞}. However, for different realizations the cost mode sequences are different. The next step of the stochastic DP is to identify all the possible discrete values of the state. The identification can be performed by the simulation of a suboptimal policy *designed to visit all the states*. As results of the simulation of ν realizations of the cost modes, we can obtain following data sets, S^{data} and C^{data} :

$$S^{data} = \begin{bmatrix} \hat{x}^{data}(1) \\ \hat{x}^{data}(2) \\ \vdots \\ \hat{x}^{data}(\nu) \end{bmatrix} \quad , \quad C^{data} = \begin{bmatrix} \phi^{data}(1) \\ \phi^{data}(2) \\ \vdots \\ \phi^{data}(\nu) \end{bmatrix} \tag{27}$$

where, S^{data} is the set of visited states($\hat{x}^{data}(k)$) by the simulation and C^{data} contains the corresponding traveling costs($\phi^{data}(k)$) for the states in S^{data}. Once S^{data} and C^{data} are obtained, we can calculate the cost-to-go values for all the states in, S^{data} by following the state trajectories and summing the Cost-to-Go over an approximation horizon of H with

the discounting factor $\alpha < 1$.

$$\hat{J}(\hat{x}^{data}(k)) = \sum_{j=0}^{H} \alpha^j \phi^{data}(k+j) \tag{28}$$

Then, we have the approximated cost-to-go set, J^{data}.

$$J^{data} = \begin{bmatrix} \hat{J}(\hat{x}^{data}(1)) \\ \hat{J}(\hat{x}^{data}(2)) \\ \vdots \\ \hat{J}(\hat{x}^{data}((\nu - H)) \end{bmatrix} \tag{29}$$

Because of the large value of ν needed to cover the entire set of possible states, the same state can be visited in many stages by the simulation. After eliminating redundant states in the set S^{data}, the following set of states, S can be obtained.

$$S = \begin{bmatrix} \hat{x}(1) \\ \hat{x}(2) \\ \vdots \\ \hat{x}(n) \end{bmatrix} \tag{30}$$

Let $L(i)$ be the number of $\hat{x}^{data}(k)$ satisfying the condition, $\hat{x}^{data}(k) = \hat{x}(i)$, where, k is the index of set S^{data} and i is the index of set S. Then the expected cost-to-go for the state $\hat{x}(i)$ is obtained by averaging these data.

$$\hat{J}(\hat{x}(i)) = \frac{1}{L(i)} \{ \sum_{\ell=1}^{L(i)} \hat{J(i)}(\hat{x}^{data}(\ell, i)) \} \tag{31}$$

Where $\hat{x}^{data}(\ell, i)$ represents the ℓth data point of $\hat{x}^{data}(k) = \hat{x}(i)$. As a results of the above calculation, we obtain the first approximation of the cost-to-go values, which will be used as initial values in the Bellman iteration, for the states $\hat{x}(i)$ in the set S.

$$J = \begin{bmatrix} \hat{J}(\hat{x}(1)) \\ \hat{J}(\hat{x}(2)) \\ \vdots \\ \hat{J}(\hat{x}(n)) \end{bmatrix} \tag{32}$$

Bellman Iteration The Bellman equation for the given state $\hat{x}(k)$ can be formulated as:

$$J^*(\hat{x}(k)) = \min_{\delta(k)\in[0,1]} E\{\phi(\hat{z}(k)) + \beta\delta(k) + \alpha J^*(\hat{x}(k+1))|\hat{x}(k)\} \tag{33}$$

$J^*(\hat{x}(k))$ is the optimal cost-to-go for the state $\hat{x}(k)$ and $\phi(\hat{z}(k))$ is the cost of the tour k, which will be chosen based on $\hat{z}(k)$ obtained after the investigation decision. Based on the above equation, we propose the following Bellman iteration scheme.

1. Set $\hat{J}^1 = \hat{J}(\hat{x}(k))$, for $k = 1, 2, ..., n$, where $\hat{J}(k)$ is in the set $\jmath$ in the equation (32)

2. Repeat following equation (34) for each $\hat{x}(k)$ in S, until $\hat{J}^i$ is converged (i.e. $\|\hat{J}^{i+1}(\hat{x}(k)) - \hat{J}^i(\hat{x}(k))\| < \varepsilon$), for $k = 1, 2, ..., n$

$$\hat{J}^{i+1}(\hat{x}(k)) = \min_{\delta(k)^i\in[0,1]} E\{\phi(\hat{z}(k)) + \beta\delta(k)^i + \alpha\hat{J}^i(\hat{x}(k+1))|\hat{x}(k)\} \tag{34}$$

Although the cost-to-go update equation in (34) looks simple, the update is quite subtle. The detailed calculation procedures for the equation (34) can be derived using the properties of expectation operator E and conditional probability.

- **Detail Calculation Procedures to Obtain $\hat{J}^{i+1}(\hat{x}(k))$**

1. For $\delta^i(k) = 0$

$$\{\hat{J}^{i+1}(\hat{x}(k))|\delta^i(k) = 0\} = E\{\phi(\hat{z}(k)) + \alpha\hat{J}^i(\hat{x}(k+1))|\hat{x}(k)\} \tag{35}$$

   - Calculating $E\{\phi(\hat{z}(k))|\hat{x}(k)\}$: The current tour must be obtained for the given condition $\hat{x}(k)$. With $\delta^i(k) = 0$, $\hat{z}(k) = \hat{x}(k)$ and the expected cost matrix($\hat{C}$) can be calculated by the following equation.

$$\hat{C} = \sum_{i=1}^{M} \hat{x}_i(k) C_i \tag{36}$$

     Then, the optimal tour($tour^*(k)$) for the state $\hat{x}(k)$ can be obtained by solving a single TSP with the $\hat{C}$ obtained from the equation (36). The expected current cost can be calculated with the given conditional probabilities, $\hat{x}(k)$, of the cost modes.

$$E\{\phi(\hat{z}(k))|\hat{x}(k)\} = \sum_{i=1}^{M} \hat{x}_i(k)\{\phi(tour_k^*)|C_i\} \tag{37}$$

which means C_i is realized with probability $\hat{x}_i(k)$.

- Calculating $E\{\alpha\hat{J}^i(\hat{x}(k+1))|\hat{x}(k)\}$:

  $E\{\alpha\hat{J}^i(\hat{x}(k+1))|\hat{x}(k)\} = \alpha\hat{J}^i(\hat{x}(k+1))$, because the approximate cost-to-go term $\hat{J}^i(\hat{x}(k+1))$ is a constant value for a given $\hat{x}(k+1)$. Hence, we can take the cost-to-go term out of the expectation summation. When $\delta^i(k) = 0$. The next state, $\hat{x}(k+1)$ is calculated by the state transition rule in the equation (81).

  $$\hat{x}(k+1) = P^T\hat{x}(k)$$

  Find the next state in the set S, $\hat{x}(l) = \hat{x}(k+1)$ for $\hat{x}(l) \in S$. Then,

  $$\hat{J}^i(\hat{x}(k+1)) = \hat{J}(\hat{x}(l)) \tag{38}$$

  Combining the equation (37) and (38), we can calculate $\hat{J}^{i+1}_{\delta^i(k)=0}(\hat{x}(k))$.

2. **For** $\delta^i(k) = 1$

$$\{\hat{J}^{i+1}(\hat{x}(k))|\delta^i(k) = 1\} = E\{\phi(\hat{z}(k)) + \beta + \alpha\hat{J}^i(\hat{x}(k+1))|\hat{x}(k)\} \tag{39}$$

- **Calculating** $E\{\phi(\hat{z}(k))|\hat{x}(k)\}$:

  After an investigation, $\delta^i(k) = 1$, the state $\hat{z}(k)$ can be one of e_ℓ for $\ell = 1, 2, ..., M$. Because $\hat{x}(k)$ is the probability vector of the cost modes before the investigation, the probability of cost mode ℓ after the investigation($Pr(\hat{z}(k) = e_\ell)$) is given by $\hat{x}_\ell(k)$. Then, the expected current cost for the given $\hat{x}(k)$ is,

  $$E\{\phi(\hat{z}(k))|\hat{x}(k)\} = \sum_{\ell=1}^{M}\hat{x}_\ell(k)\phi(e_\ell) \tag{40}$$

  where $\phi(e_\ell)$ represents the tour cost for the cost mode ℓ.

- **Calculating** $E\{\alpha\hat{J}^i(\hat{x}(k+1))|\hat{x}(k)\}$:

  Once the investigation is performed, $\hat{z}(k)$ becomes one of the e_ℓs with probability $\hat{x}_\ell(k)$. Therefore, $\hat{x}(k+1) = P^Te_\ell$ with the probability $\hat{x}_\ell(k)$.

  $$\alpha E\{\hat{J}^i(\hat{x}(k+1))|\hat{x}(k)\} = \alpha\sum_{\ell=1}^{M}\hat{x}_\ell(k)\hat{J}^i(P^Te_\ell) \tag{41}$$

In the above equation, it is obvious that $P^T e_i \in S$ because all e_ℓs are visited by the selected suboptimal policy through the large number of cost mode realizations and all of their next states, $P^T e_\ell$ are also visited by the suboptimal policy. Combining the equation (40) and (41), we can calculate $\hat{J}^{i+1}_{\delta^i(k)=1}(\hat{x}(k))$.

3. Decision for $\hat{J}^{i+1}(\hat{x})$:

   With the results of step (1) and (2), the equation (34) simply becomes as following,

$$\hat{J}^{i+1}(\hat{x}(k)) = \min\{\hat{J}^{i+1}_{\delta^i(k)=0}(\hat{x}(k)), \hat{J}^{i}_{\delta^i(k)=1}(\hat{x}(k))\} \tag{42}$$

real-time Performance Evaluation The off-line obtained optimal cost-to-go J^* can be used for real-time decision making. Here we evaluated the performance of the resulting policy for different sets of cost mode realization though stochastic simulation.
The policy can be described as follows.

1. At the time k, solve,

$$J^*(\hat{z}(k)) = \min_{\delta(k)\in[0,1]} E\{\phi(\hat{z}(k)) + \beta\delta(k) + \alpha J^*(\hat{x}(k+1))|\hat{x}(k)\} \tag{43}$$

   where, J^* is the optimal cost-to-go from the Bellman iteration in 3.3.3.1.

   (a) Calculate,

$$J^*_{\delta(k)=0}(\hat{x}(k)) = E\{\phi(\hat{z}(k)) + \alpha\hat{J}^*(\hat{x}(k+1))|\hat{x}(k)\} \tag{44}$$

   as derived for $\hat{J}^{i+1}_{\delta(k)=0}(\hat{x}(k))$ as in the equation (35).

   (b) Calculate,

$$J^*_{\delta(k)=1}(\hat{x}(k)) = E\{\phi(\hat{z}(k)) + \beta + \alpha\hat{J}^*(\hat{x}(k+1))|\hat{x}(k)\} \tag{45}$$

   as derived for $\hat{J}^{i+1}_{\delta(k)=1}(\hat{x}(k))$ in the equation (39).

   (c) Compare $J^*_{\delta(k)=1}$ and $J^*_{\delta=0}$ and choose $\delta(k)$

2. Depending on $\delta(k)$, obtain $\hat{z}(k)$ and solve a deterministic TSP with the expected cost matrix conditioned by $\hat{x}(k)$ to determine the current tour.

3. Evaluate the real cost and store.

4. Update $\hat{x}(k+1)$ from $\hat{z}(k)$ according to the state transition rules in the section 4.5.

5. $k = k+1$ and repeat from step (1)

The real-time performance of the optimal policy with the optimal cost-to-go J^* should be robust for any set of cost mode realizations because it is obtained by considering the underlying stochastic characteristic of the problem.

3.3.3.2 Suboptimal Policies

Two suboptimal policies have been developed for the given problem. One is 'no investigation policy' which repeats a same traveling route for every tour. The other is an 'investigation policy' based on a threshold on uncertainty in the cost mode.

No Investigation Policy The first suboptimal policy repeats the same traveling route optimal in the sense of the mean traveling cost. One property of a Markov chain where every state is reachable and there are no attractor states is the existence of limiting probability distribution over the states P^{∞}. P^{∞} can be found by calculating the steady state of the transition equation. P^{∞} represents the long-run distribution of the cost modes. Thus, if the salesman follows the tour($mtour$) obtained from the mean cost matrix with the limiting probability, his long-run average performance without investigation could be optimized. When there are M cost modes, the optimal tour($mtour$) can be determined by the deterministic optimization with the 'mean cost matrix($\overline{C}$)', which is

$$\overline{C} = \sum_{i=1}^{M} P_i^{\infty} C_i \tag{46}$$

and

$$mtour = \arg\left(\min_{mtour} \parallel (\phi|\overline{C}) \parallel \right) \tag{47}$$

where, C_i is the cost matrix for cost mode i. Although this policy seems reasonable, it is far from being the optimal policy because the salesman cannot realize the potential benefit of accurate information provided by the investigation opportunity.

Investigation Policies Without investigation, the salesman's knowledge of which cost mode he will encounter ($\hat{x}(k)$) eventually converges to the limiting probability, P^{∞}. When $\hat{x}(k)$ is close to P^{∞}, the salesman has only limited information about the cost mode because of the diluted probabilities of the cost modes in $\hat{x}(k)$. To decide the proper investigation frequency, we define following variable, $\gamma(k)$ as an approximate indicator of the uncertainty.

$$\gamma(k) = \|\hat{x}(k)\|_{\infty} \tag{48}$$

The maximum value of $\gamma(k)$ is 1 if the salesman executes the investigation option at step k and it decreases as the salesman proceeds from tour to tour without investigation.

- **Investigation Criteria** : The salesman decides to investigate when $\gamma(k)$ is less than a certain constant θ.

$$\delta(k) = 1 \quad , \text{if} \;\; \gamma(k) < \theta \tag{49}$$

In equation (49), if the value of θ is close to 1, the salesman investigate very frequently. Thus, suboptimal policies can be generated by varying the parameter θ leading to different frequencies of investigation.

- **Tour Decision** : For given state $\hat{x}(k)$, the expected cost matrix($\hat{C}$) and the optimal tour($tour^*(k)$) for the expected cost matrix is calculated by following equation.

$$\hat{C}(k) = \sum_{i=1}^{M} \hat{z}_i(k) C_i \tag{50}$$

$$tour^*(k) = \arg\left(\min \parallel (\phi|\hat{C}(k)) \parallel\right) \tag{51}$$

The equations (50) and (51) reflect the benefit of investigation in the decision of the tour at time k because, with the state transition rules in (24) and (81), if the investigation is performed at time k, the $\hat{C}$ in (50) becomes exactly same as the cost matrix of the particular realization of the cost mode at time k. Nevertheless, we have not found a systematic way to determine the optimal value of θ. In addition, it is unlikely that a rule of this form is optimal.

Investigation Policy for Entire State Identification As mentioned in 3.3.3.1, to start Bellman iteration for the stochastic DP, the entire state must be given with an initial cost-to-go value for each state. The suboptimal policy proposed in section 3.3.3.2 can be modified to visit all the possible states by changing the investigation criteria.

- **Investigation Criteria** : The salesman decides to investigate when $\hat{x}(k)$ becomes same as the limiting probability of the state transition probability matrix, P^{∞}.(within some small tolerance), i.e.

$$\delta(k) = 1 \quad \text{, if } \ |\hat{x}(k) - P^{\infty}| < \epsilon \tag{52}$$

If the investigation is performed at time k when $\hat{x}(k) \simeq P^{\infty}$, the state $\hat{x}(k)$ is reset to $\hat{z}(k)$ according to the equation (24) which is then propagated again until it reaches P^{∞}. Hence, using the investigation criteria in (52), this suboptimal policy can visit all of the accessible states over the course of many simulations.

3.3.4 Stochastic DP in the Subset of the States

The idea of finding solution in the subset of the state applied for the deterministic TSP can be extended to replace the full Stochastic DP derived in 3.3.3.1. From the simulation results of reasonable suboptimal policies, the 'good' states can be identified and patched to obtain a subset of the states that can be searched rigorously in reasonable time. The overall idea of the proposed method is shown in Figure 15.

The proposed method can be derived from the modification of the stochastic DP method shown in 3.3.3.1. The three important modifications can be summarized as follows:

1. *Simulation of Heuristics and Subset Identification*:

   Instead of the suboptimal policy shown in 3.3.3.2, the suboptimal policies shown in 3.3.3.2 are used for simulation to find 'good' states with different value of the investigation criteria, θ. The 'good' states are found by evaluating different suboptimal policies in terms of the total cost of tours over a number of stages. The first cost-to-go approximation procedure for the proposed method is exactly same as shown in 3.3.3.1

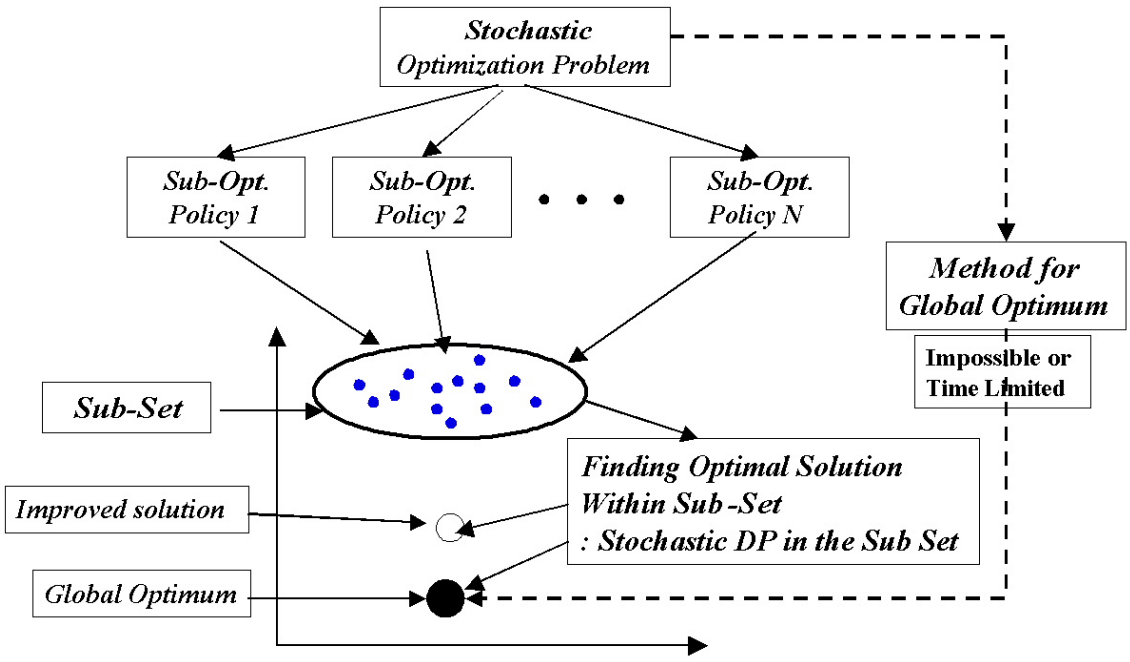

Figure 15: The Proposed Approach : Stochastic DP in the restricted state space of the States

except the S^{data} and C^{data} in the equation (27) consist of the visited states and their current costs by the selected suboptimal policies.

2. *Bellman Iteration for Disconnected States*:

   In the subset of the states, for some $\hat{x}(k)$, it is possible that the next state of $\hat{x}(k)$ is not in the set of the state S because the subset of the states are extracted from the selected suboptimal policies. In this case, the selected suboptimal policies execute the investigation option for the state following $\hat{x}(k)$. As a result of this investigation, the intermediate state following $\hat{z}(k)$ is set to e_i, therefore $\hat{x}(k+1)$ obtained by the state transition rule for the case $\delta^i(k) = 0$ will never appear in S. Therefore, for those states, the investigations should be performed to approximate the cost-to-go inside the subset. To avoid any state transition to the states outside the subset, we can assign large cost-to-gos for all of the states outside the subset as pictorially described in Figure 16.

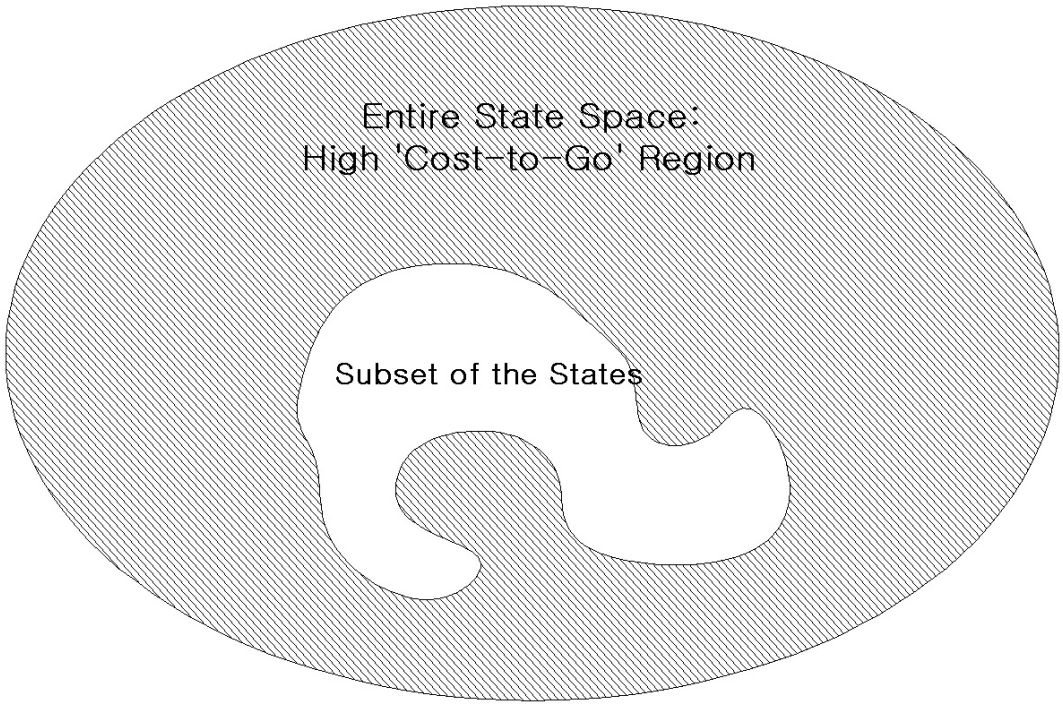

Figure 16: Cost-to-Go for Unexplored Region(Outside the Subset) in the State Space

3. *Cost-to-Go Barrier for the Real-time Performance Evaluation*:

   Before the real-time performance evaluation procedure shown in 3.3.3.1 is executed, high values of cost-to-go must be assigned to the unexplored states as described in Figure 16. This leads to a high cost barrier to prevent visiting unexplored states during the real-time decision making.

The reduction of the number of states by the proposed method can dramatically decrease the computational time for the Bellman iteration, which is the major computational load of the stochastic DP.

3.3.5 Illustrative Example : Stochastic TSP with An Investigation Option

The proposed method was verified for 2 different stochastic TSP examples, small(5 cost modes, 5 cities) and larger(20 cost modes, 5 cities) size of Stochastic TSPs. According to the definition of the state in (23), the dimension of state is same as the number of cost

modes. Hence, although the number of cities in both examples is 5, the complexity of the larger one(20 cost modes) is much higher than that of small one due to large state space. The choice of a very small TSP was made to avoid large computational costs for each step and would not affect the overall conclusions with the stochastic part of the problem.

3.3.5.1 Stochastic TSP Example 1 : 5 Cost Modes, 5 Cities

Obviously, the first example is a very simple but we choose this as we wanted to compare the solutions obtained by the proposed method with the optimal solution. The 5 symmetric cost matrices that represent corresponding cost modes consists of cost elements generated by realizing uniformly distributed random variables ranged from 10 to 70 as shown in equation (53)-(57).

$$\text{Cost Mode 1} = \begin{bmatrix} 0 & 29 & 45 & 16 & 38 \\ 29 & 0 & 25 & 20 & 13 \\ 45 & 25 & 0 & 13 & 21 \\ 16 & 20 & 13 & 0 & 46 \\ 38 & 13 & 21 & 46 & 0 \end{bmatrix} \tag{53}$$

$$\text{Cost Mode 2} = \begin{bmatrix} 0 & 34 & 11 & 41 & 31 \\ 34 & 0 & 21 & 23 & 25 \\ 11 & 21 & 0 & 33 & 24 \\ 41 & 23 & 33 & 0 & 29 \\ 31 & 25 & 24 & 29 & 0 \end{bmatrix} \tag{54}$$

$$\text{Cost Mode 3} = \begin{bmatrix} 0 & 22 & 16 & 24 & 19 \\ 22 & 0 & 38 & 50 & 18 \\ 16 & 38 & 0 & 26 & 25 \\ 24 & 50 & 26 & 0 & 15 \\ 19 & 18 & 25 & 15 & 0 \end{bmatrix} \tag{55}$$

$$\text{Cost Mode 4} = \begin{bmatrix} 0 & 19 & 38 & 30 & 25 \\ 19 & 0 & 26 & 18 & 37 \\ 38 & 26 & 0 & 56 & 43 \\ 30 & 18 & 56 & 0 & 25 \\ 25 & 37 & 43 & 25 & 0 \end{bmatrix} \tag{56}$$

$$\text{Cost Mode 5} = \begin{bmatrix} 0 & 22 & 30 & 16 & 19 \\ 22 & 0 & 43 & 32 & 23 \\ 30 & 43 & 0 & 31 & 44 \\ 16 & 32 & 31 & 0 & 65 \\ 19 & 23 & 44 & 65 & 0 \end{bmatrix} \tag{57}$$

Another important parameter, the transition probability matrix P of the underlying Markov chain, is given by the following 5 by 5 matrix in Table 3.

Table 3: Cost Mode Transition Probability Matrix for the Illustrative Example

0.8071	0.0147	0.0608	0.0640	0.0534
0.0043	0.5891	0.2241	0.0348	0.1477
0.0425	0.1353	0.7359	0.0154	0.0709
0.0745	0.4210	0.0151	0.0482	0.4412

The corresponding limiting probability, P^{∞}, of P is $\begin{bmatrix} 0.1125 & 0.2326 & 0.2321 & 0.3157 & 0.1069 \end{bmatrix}$ and the investigation cost β is given as 60. For simulation of the suboptimal policies, 10,000 cost mode sequences are generated according to the underlying Markov chain. Figure 17 shows the performance of several suboptimal policies with different values of θ. The suboptimal policies inside the shaded area of Figure 17 are used for extracting the subset of the states.

The stochastic DP using the entire state space was executed and the total number of states for the given problem turned out to be 315. The subset of the states contains 46 elements determined by the proposed method.

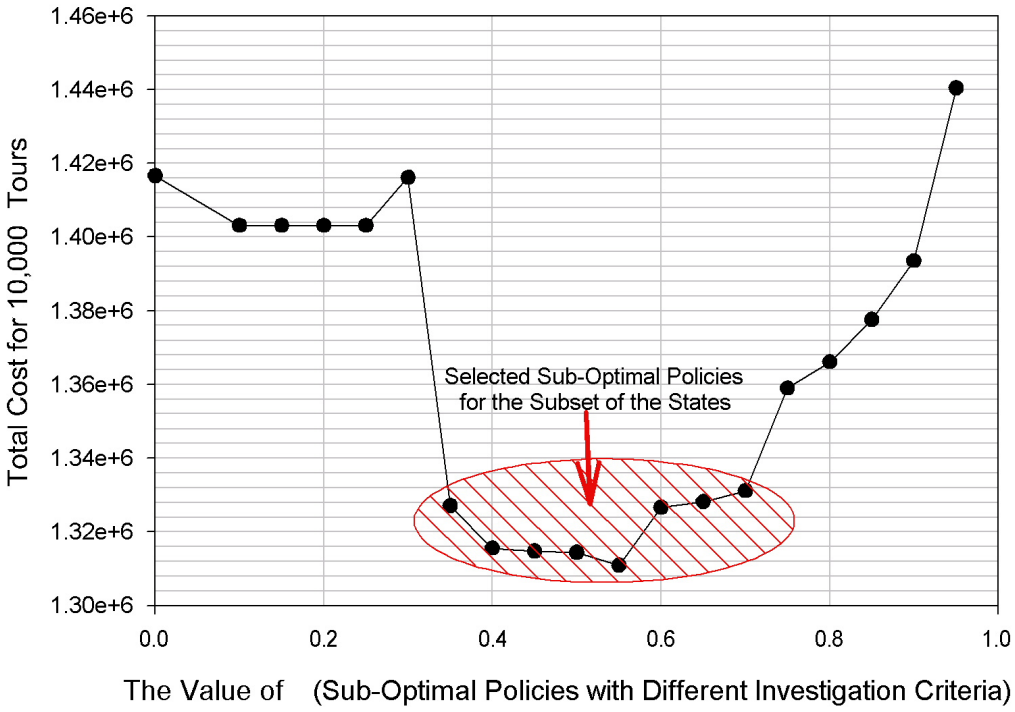

Figure 17: Example 1: Simulation Results of the suboptimal Policies

The total cost of 10,000 tours for the first realization of the cost mode sequences are calculated by the real-time performance evaluation using the 2 different optimal cost-to-go values obtained by the stochastic DP in the entire space and just in the subset of the states. The optimal solution by the stochastic DP in the entire space is 1303591 and the solution obtained by the proposed method is 1306110 which is a 65.6% improvement of the best of the suboptimal solutions, 1310927. To verify the robustness of the cost-to-go obtained by the proposed methods, the real-time performance evaluation was performed for different sets of 10,000 cost realizations. The results of this policy evaluation verify the policy obtained by the proposed method is robust with respect to different cost modes realizations and not just for the realization set used for cost-to-go construction shown in Table 1.

3.3.5.2 Stochastic TSP Example 2 : 20 Cost Modes, 5 Cities

A larger example with 20 cost modes is introduced in this section. Although the number of cities(5) in this example is same as the previous one, the complexity of the problem is

Table 4: Example 1: Comparison of the Solutions by 3 Different Methods for Different Sets of Realizations(The values are the total costs for 10,000 tours)

Realization Set #	1 $^{+}$	2	3	4	5
Full DP *	1303591	1302743	1292996	1296222	1297076
DP in the Subset *	1306110	1303299	1296291	1297953	1300475
The Best of Heuristics	1310927	1311197	1304445	1305716	1307001
% of Improvement $^{++}$	65.60	93.42	71.20	81.70	65.76

* 315 States : Computational Time for BI = 12834 seconds
** 46 States : Computational Time for BI = 485 seconds
for $\varepsilon < 0.01$, where $\|\hat{J}^{i+1}(\hat{x}(k)) - \hat{J}^{i}(\hat{x}(k))\| = \varepsilon$
on a Pentium III at 800 MHz: 512MB RAM
$^{+}$ Realization Used for Cost-to-Go Construction.
$^{++}$ The amount of improvement from the best of the heuristic solutions.

increased dramatically due to larger number of cost modes. All parameters of the problem(20, 5 by 5 cost matrices, a 20 by 20 state transition matrix and a investigation cost) will be supplied by the authors upon request. For simulation of the suboptimal policies developed in 4.5, 20,000 cost mode sequences are generated according to the underlying Markov chain. Figure 18 shows the performance of optimal policies with different values of investigation criteria θ.

As the proposed method is applied to the previous example, the suboptimal policies inside the shaded area of Figure 18 are used to extracting the subset of the states. The total number of states in the entire state space of the problem turned out to be 1748. On the other hand, the subset of the states contains 176 states. The computational results for the problem is summarized in following table 2.

The computational results shown in table 2 imply that the proposed method is efficient in finding solutions within 0.5% of optimality in computation times much reduced from the full stochastic DP, and also robust with respect to different cost mode realizations.

3.3.6 Conclusions

Planning and scheduling problems under uncertainty are a challenging class of stochastic optimization problems. Finding reasonable ways to represent the uncertainty is crucial, particularly when the decision involves actions whose sole purpose is to reduce uncertainty

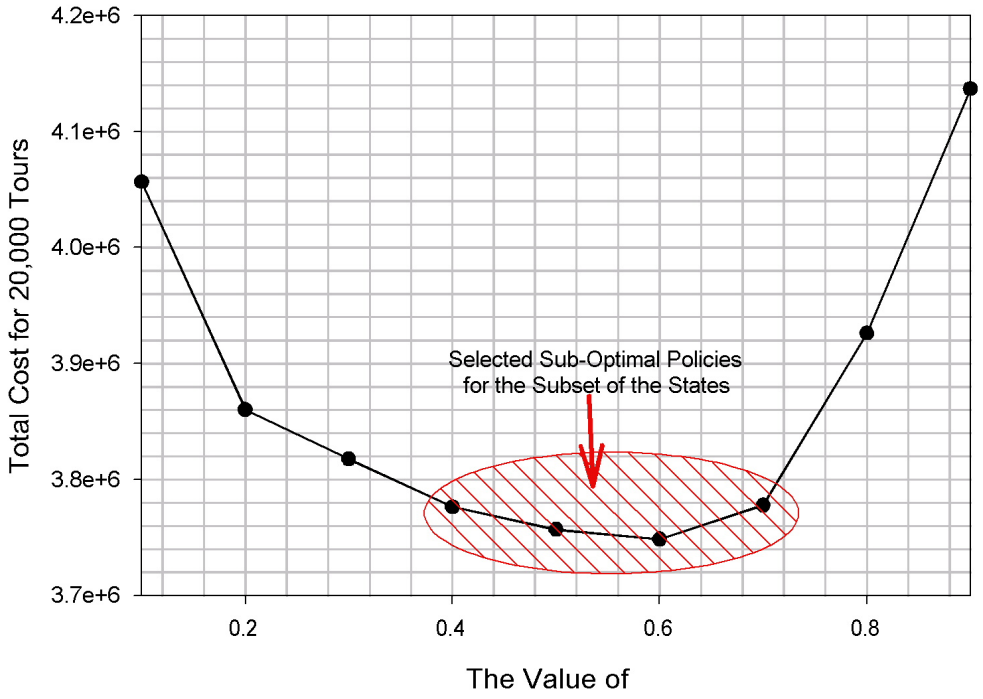

Figure 18: Example 2: Simulation Results of the suboptimal Policies

and modify the information state. To begin to develop solution approaches for this class of problems we introduced a new variant of the stochastic TSP. As a rigorous solution method for the problem, a conventional stochastic optimization method, stochastic DP was developed. However, due to the complexity of the problem, the conventional stochastic DP approach incurs high computational costs, especially in the Bellman iteration procedure for obtaining the optimal cost-to-go. The computational complexity of the conventional DP formulation was reduced, without significantly compromising the solution quality, by extending the heuristic synthesis used for the deterministic case by modification of the conventional stochastic DP formulation. We tested the computational and performance improvement via the method on 2 different examples with different problem sizes(small: 5 cost modes, 5 cities and larger: 20 cost modes, 5 cities). Finally, the basic idea of the proposed method, solving optimization problem through the rigorous search of a solution space that is composed of the states visited by suitable heuristics is quite general. The

Table 5: Example 2: Comparison of the Solutions by 3 Different Methods for Different Sets of Realizations(The values are the total costs for 20,000 tours)

Realization Set #	1 [+]	2
Full DP [*]	3732861	3717465
DP in the Subset [**]	3735435	3723314
The Best of Heuristics	3748535	3741832
% of Improvement [++]	83.58	76.00

[*] 1748 States : Computational Time for BI = 5.5 days
[**] 167 States : Computational Time for BI = 2.5 hours
for $\varepsilon < 0.01$, where $\|\hat{J}^{i+1}(\hat{x}(k)) - \hat{J}^{i}(\hat{x}(k))\| = \varepsilon$ on a Pentium III at 800 MHz: 512MB RAM
[+] Realization Used for Cost-to-Go Construction
[++] The amount of improvement from the best of the heuristic solutions.

introduced stochastic TSP is kept intentionally simple to facilitate the exposition of the main idea. Obviously, we could have complicated the problem further by, for example, introducing the possibility of cost transition after each segment of a tour, which will necessitate an information state update and a new decision at every segment. The proposed method can be generalized to this case without any difficulty. In fact, we expect it can be applied to many types of optimization problems, multi-stage, stochastic, or multi-objective, as long as some initial heuristics exist for their solutions.

Nomenclature for the Stochastic TSP

- Problem Description
  - C_i : cost matrix i for cost mode i, for $i = 1, 2, ..., M$
  - P : Markov Chain matrix for the cost mode transition
  - P^{∞} : the limiting probability of P
  - β : investigation cost
- States
  - $\hat{x}$: information state vector, which represents the conditional probability of each cost mode

- $\hat{z}$: information state after the investigation decision
- $\hat{x}^{data}$: the state visited by the simulation of the suboptimal policies
- e_i : possible realization of the information state after the investigation, for $i = 1, 2, ..., M$

- The suboptimal Policies
  - $\delta(k)$: investigation indicator, i.e. $\delta(k) = 1 \equiv$ investigation at time k, $\delta(k) = 0 \equiv$ no investigation at time k.
  - $\gamma(k)$: $\|\hat{x}(k)\|_\infty$, an uncertainty size indicator.
  - θ : investigation criteria threshold parameter
  - $\overline{C}$: mean cost matrix from X_∞
  - $mtour$: the optimal tour for the $\overline{C}$
  - $\hat{C}(k)$: expected cost matrix from $\hat{x}(k)$
  - $tour^*(k)$: the optimal tour for the $\hat{C}(k)$

- Current Cost and Cost-to-Go
  - $\phi(\hat{x})$: single tour cost
  - $\phi^{data}(\hat{x})$: a tour cost from the simulation results of the suboptimal policies
  - $\phi_i^{perfect}(\hat{x})$: the optimal current cost with cost mode i
  - $\hat{J}(\hat{x})$: approximate cost-to-go value for state $\hat{x}$
  - $\hat{J}^i(\hat{x})$: approximate cost-to-go at the ith Bellman iteration
  - $J^*(\hat{x})$: the optimal cost-to-go from the Bellman iteration
  - α : discount factor for the cost-to-go calculation

CHAPTER IV

HIGH DIMENSIONAL DISCRETE STATE SPACE: APPLICATION TO STOCHASTIC RESOURCE CONSTRAINED PROJECT SCHEDULING PROBLEMS

4.1 Introduction

A challenge in highly regulated industries, such as pharmaceuticals and agrochemicals, is the process of selecting, developing, and efficiently manufacturing new products that emerge from the discovery phase. Candidate products must undergo a set of tests related to safety, efficacy, and environmental impact, to obtain certification. The problem of scheduling these tasks and associated analysis can be considered as a generalization of the well-known job shop scheduling problem. The case in which all the problem data have known values belongs to the NP-hard class of combinatorial problems[14]. In general, task success or failure is uncertain and the time value of project reward varies, which adds more complexity to the scheduling problem. In a specialized R&D pipeline management problem, the time value of project reward decreases as the time to introduction of the product increases due to incoming competitive products and fixed patent periods. Hence a company has to manage its various resources, manpower, lab space, capital, pilot facilities, etc. to ensure its best return on its new product pipeline, with the added complication that the outcome of tasks is uncertain. Besides the uncertainty about the success of the task, there are several additional uncertain parameters in real problems such as uncertainties in task duration and resource(cost) requirement.

The project scheduling problem with unlimited resource [69] was introduced to the process systems engineering area using a mathematical programming(MILP) based solution approach. In the case of unlimited resource, the overall objective function(net present value) of the problem can be separated into the individual objective functions of each project since

one project does not influence the others. There has been significant progress in solution methods [46, 13, 53, 66] for the problem with resource constraints as well as uncertainty in the task outcome. However, previous solution methods for RCPSP have considered only a subset of the potential uncertainties and have been based on mathematical programming techniques. Even though the mathematical programming approach can account for uncertainties of the problem via scenario generation, the approach is limited to a fairly small number of scenarios due to the exponential increase in the computational load. Limitations in the mathematical programming approach lie not only in the computational tractability but also in the awkwardness in capturing richer representations of uncertainty. Notable exceptions are [77, 76, 78] where a broader set of uncertainties in the problem are addressed within a simulation and optimization (SIMOPT) framework. The SIMOPT framework developed in [77, 76, 78] achieved substantial improvement in combining stochastic simulation and optimization by taking a discrete-event dynamic system's view of the RCPSP. However, outer iteration process of the SIMOPT where constraints are added to the MILP to steer it away from decisions that gave poor outcomes in simulation cannot does not fully and rigorously account for the way information and outcomes can influence the decisions.

In this study, we address the uncertainties in the RCPSP using a discrete time Markov chain, which enables us to model correlations among the uncertain parameters. For example, the probability of success of a future task may not be independent of the outcomes of current or previous tasks. Furthermore, a novel solution method, *dynamic programming in a heuristically confined state space* developed and illustrated in [27, 23, 21], is tailored to the problem to obtain high quality solutions. The proposed approach is focused on solving the RCPSP as a multi-stage online decision making problem. Finally, the proposed approach is demonstrated by effectively solving a fairly complex stochastic RCPSP that can have up to 1.2 billion different outcomes depending on realization of the uncertainty.

4.2 Problem Description : Stochastic RCPSP

We consider a simplified version of RCPSP with M projects, each of which consists of m_i tasks, for $i = 1, ..., M$. There are N resources(Laboratories), a specific resource has to

be used to perform each task. In the example formulation studied in this work, the resources are represented as Laboratories(Lab.). Several problem parameters of a task, the result(success or failure), the duration, and the cost, are uncertain. A detailed description of the uncertainty model is given in section 4.2.1. A time-varying reward function is given for each project to represent the decreasing value of the project with time. The reward function(equation (58)) is characterized by three parameters: 'stiffness parameter', α, 'project deadline indicator', PD, and 'final value', β.

$$\begin{aligned} R(0) &= R_0 \quad \text{at} \quad k = 0 \\ R(k) &= R_0 - e^{\alpha k} \quad \text{for} \quad 0 < k \leq PD \\ R(k) &= \beta \quad \text{for} \quad k > PD \end{aligned} \tag{58}$$

Figure 19 shows the reward function with $R_0 = 5,000$, $\alpha = 0.235$ and $PD = 34$.

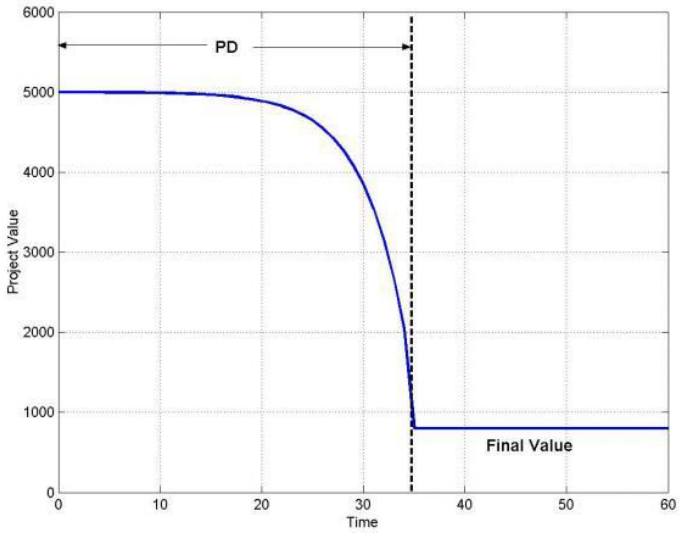

Figure 19: Decreasing Reward Function

4.2.1 Uncertain Parameter Modeling: Markov Chain & Conditional Probability

RCPSPs with diverse representations of uncertain parameters have been addressed in the literature [69, 46, 13, 53, 66, 77, 76, 78]. However, it appears that probabilistic correlation among the uncertain parameters in the RCPSP has not been addressed previously.

Our problem representation is based on the premise that the result, duration and cost of adjacent tasks in a project are correlated. For example, if a current task takes longer to complete, then the duration of the next task also tends to be longer. The assumption is particulary appropriate for the drug development pipeline management problem because a candidate(drug) has to pass similar types of tests with varying number of patients. In general, the correlation can exist between any 2 tasks in a project and can be modeled by introducing corresponding transition probability. However, in this work we assume the probabilistic correlation between 2 adjacent tasks only for simplification. The probabilistic correlation among uncertain parameters can be modeled with discrete time Markov chains. The nth task of a project i has r_{ni} realizations and each realization consists of the values of the result, duration, and cost of the task from a discrete set as shown in Figure 2. For example, 'F, D_{11i}, C_{11i}'(the first realization set of the task 1 in Figure 2) represents failure of the task with D_{11i} duration and C_{11i} cost. The possible discrete values for the parameters may represent the actual values or the mean values for the parameters. Furthermore, to represent the quality of the task result, multiple success levels can be introduced. For example, the result of task can be classified into 'failure(F)', 'moderate success(S_1)' and 'high success(S_2)' as shown in Figure 2. In the case of 'high success', the probability of success in the next task can be made higher by specifying the underlying Markov state transition probability accordingly. An explicit representation of the probabilistic correlation of the uncertain parameters in a project with 3 tasks is shown in Figure 20.

Here, there are 3, 2, and 3 possible realizations for tasks 1, 2, and 3 respectively($r_{1i} = 3$, $r_{2i} = 2$ and $r_{3i} = 3$). A Markov model for project i is defined with 3 probability matrices(vector), PI_i, PM_{1i} and PM_{2i}. The realization of the first task in project i is governed by 'initial probability vector'(PI_i), which consists of r_{1i} probabilities and for different potential realization of the task. The realization of the second and third tasks are conditioned by the realization result of the previous task and governed by r_{2i} by $r_{1i}(PM_{1i})$ and r_{3i} by $r_{2i}(PM_{2i})$ transition matrices respectively. The summation of each column of the matrix is equal to 1 and the ith column of the matrix represent a conditional probability vector when the previous task is completed with the ith realization. In the matrices PM_{1i} and PM_{2i},

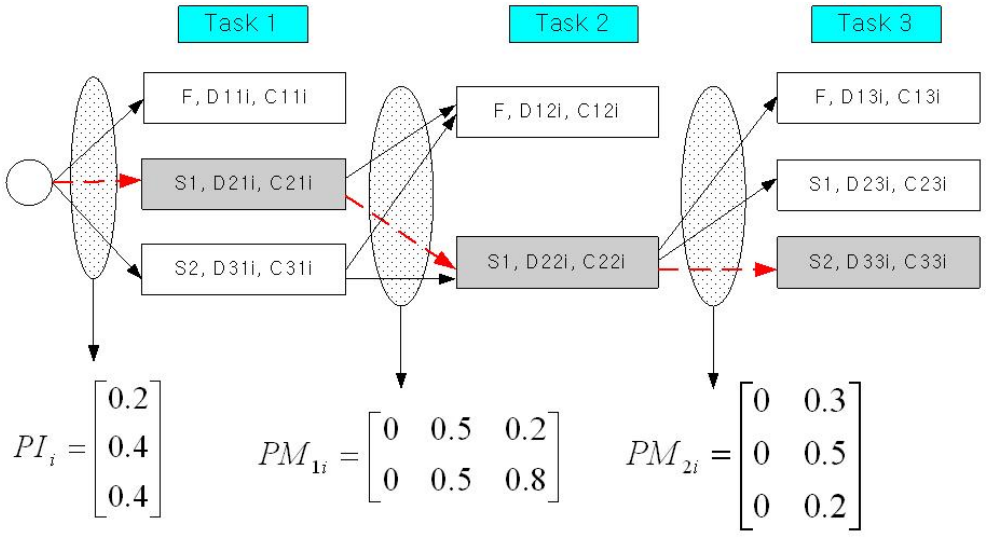

Figure 20: Uncertain Parameter Modeling for a Project with 3 Tasks

the columns with zeros represent the state transition probabilities, which are identically zero, which indicate that the task 2 is not performed if the task 1 fails. The shaded realization linked with dashed lines in Figure 20 represents the scenario of '2(moderate success, duration D_{21i}, cost C_{21i})-2(success, D_{22i}, C_{22i})-3(high success, D_{33i}, C_{33i})': each number represents realization index of the task. For the project in Figure 20, there can be 9 possible scenarios, 6 scenarios with a completion of all the three tasks, 2 scenarios ended with a task 2 failure and 1 scenario with a task 1 failure. With the propose uncertain parameter representation, the illustrative example with 1,214,693,756 scenarios is represented with one parameter table(Table 1 and 2). In summary, the proposed representation compactly represents quite complex interactions between task outcomes.

4.3 Dynamic Programming Formulation

The RCPSP is characterized by a sequence of combinatorial decisions made with respect to portfolio composition and resource allocation, both of which may depend on the state of the system at the time of the decision. The problem can be classified as a 'multi-stage stochastic optimization problem with recourse' or a 'stochastic optimal feedback control problem'. Stochastic dynamic programming(DP)[7] is widely considered to be an effective way to solve these types of problems. However, it suffers from what Bellman[5] referred to as "the

curse of dimensionality", meaning that its computational requirements grow exponentially with the number of state variables. If we can handle "the curse of dimensionality", DP will give us an optimal policy for the decision making based on the states(including the uncertain information) at the time of each decision. In this section, we develop a DP formulation, consisting of a definition of the state and action(decisions), the state transition rules, and the objective function(cost-to-go). The formulation can yield the optimal solution but is computationally infeasible. In the next section, an algorithmic framework that can overcome the computational intractability of the DP formulation and provide a suboptimal but good policy will be presented based on our previous work[27, 23].

4.3.1 State Space Definition

In defining the state of a system, it is important to adopt as parsimonious a state representation as possible because any redundancy will increase the computational complexity. Consider a RCPSP with M projects and N types of available resources(Laboratories). We propose the following definition of the state for the problem :

$$X = [s_1, s_2, \cdots, s_M, z_1, z_2, \cdots, z_M, L_1, L_2, \cdots, L_N, t]^T \tag{59}$$

In (59), s_i for $i = 1, 2, \cdots, M$ represents the current status of project i, containing the formulation of which tasks are finished and which task is on-going for project i. Because a finite number of tasks are involved in a project, s_i can be represented as an integer variable. For example, there can be 7 possible states(circled number) in a project with 3 tasks as illustrated in Figure 21.

z_i for $i = 1, 2, \cdots, M$ represents the information state of project i, which indicates the

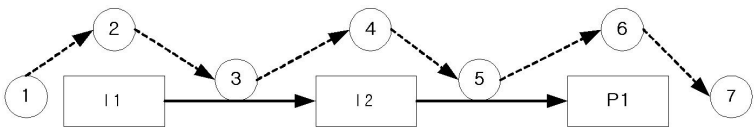

Figure 21: Possible Project Status of a Project with 3 Tasks

result for the most recent task in the project. As explained in the problem description, the parameters(i.e., the duration, cost, and result('success' or 'failure')) of each task are realized

according to the conditional probabilities in the corresponding Markov chain. Once the task in project i is completed, z_i is updated. z_i is an integer variable ranging from 1 to r_{ni}. r_{ni} is the number of possible realizations for the nth task(the most recently realized) in project i. The other state variables, L_j for $j = 1, 2, \cdots, N$ represents the time that the resource has been used for the currently on-going task. And $L_j = 0$ indicates that the resource is idle. Finally, time t is added as a state variable in order to consider the time-varying value of the reward of each project. The state definition is distinguished from the one introduced in [21] by the elimination of the state variables that represent the time spent so far for a task in each project. In the definition of state and its transition rules(4.3.3), only the states that influence the decisions are considered. This means, the state variable t, representing time does not have to increase uniformly between consecutive state transitions.

4.3.2 Decisions

With the state defined as in equation (59), the decision(action), U, can be defined as in the following equation (60).

$$U = [\delta_1, \delta_2, \cdots, \delta_M]^T \tag{60}$$

δ_i is a binary variable which represents whether to perform a task(1) or not to perform a task(0) in the project i, for $i = 1, 2, \cdots, M$. The decision can be made only when an appropriate resource is available, that is, $\exists\ L_j = 0$ for some $j = 1, 2, \cdots, N$. Otherwise, the decision remains a null vector, $U = [0, 0, \cdots, 0]^T$.

4.3.3 State Transition Rules

In a discrete time system, the state at time $k+1$, $X(k+1)$ can be derived from the state at k, $X(k)$, and the control action(decision) at k, $U(k)$. For this application, the state transition rules are given in an implicit rather than an explicit functional form.

1. **Initial State**
   According to the definition of the state(Section 4.3.1), there is only one initial state at time $t = 0$.

$$X(0) = \left[\underbrace{1, \cdots, 1}_{\text{Status of M Projects}}, \overbrace{0, \cdots, 0}^{\text{M Information States Variables}}, \underbrace{0, \cdots, 0}_{\text{N Types of Resource}}, \overbrace{0}^{\text{Time}} \right]^T \tag{61}$$

2. **Event-Based State Transition**

As introduced in a previous part of this work(Section 4.3.1), state transition occurs only after an 'event'. An event is defined as the completion or start of a task. In general, the start of a new task and the completion of a previous task happen at a same time because there are always tasks ready to be executed. The concept of an 'event' is a more efficient way of representing the state transition than the one in our previous work[21]. A common Gantt chart for a RCPSP with 5 projects(Figure 27) is shown in Figure 22 with indications(dotted lines) of the 'events' and corresponding states, $x(k)$. where k is the state index in terms of the event. According to the definition of an 'event', the Gantt chart can be represented with the 15 states instead of the 31 states with a uniform time discretization.

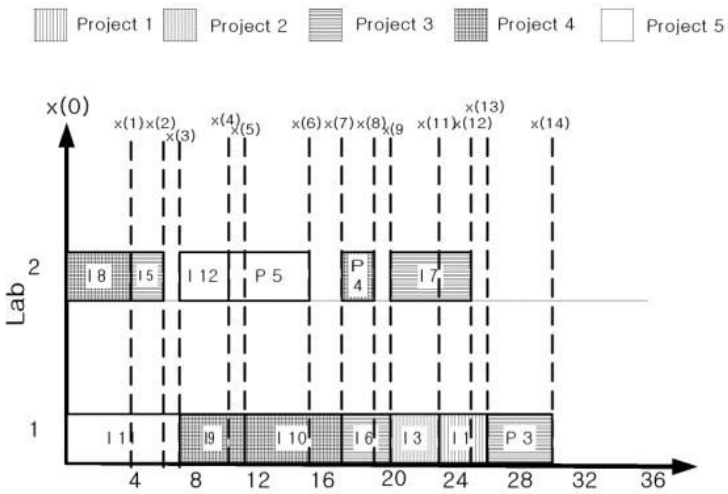

Figure 22: A Gantt Chart with Events and States

(a) **Starting of a Task with an Action(Decision)**

Suppose that an action, $U(k) = [\delta_1, \delta_2, \cdots, \delta_M]^T$ is given by a certain decision rule. According to the action, the current state, $X(k)$, transitions to a temporary state $X'(k+1)$. This temporary state transitions to the next state $X(k+1)$ after the completion of one or more of the on-going tasks. State transition from the current state, $X(k)$, to the temporary state, $X'(k+1)$, is defined by following

equation.

$$\begin{aligned} X'(k+1) &= f(X(k), U(k)) \\ &= [s'_1, s'_2, \cdots, s'_M, z_1, z_2, \cdots, z_M, L_1, L_2, \cdots, L_N, t]^T \end{aligned} \quad (62)$$

where, $X(k) = [s_1, s_2, \cdots, s_M, z_1, z_2, \cdots, z_M, L_1, L_2, \cdots, L_N, t]^T$,
$U(k) = [\delta_1, \delta_2, \cdots, \delta_M]^T$ and $s'_i = s_i + \delta_i$ for $i = 1, ..., M$. Besides s_i, the other state variables of $X'(k+1)$, z_i, L_j and t, are kept exactly the same as those of $X(k)$. The information state variable, z_i is updated after the completion of the corresponding task. The time spent in type j resource, L_j, also cannot be updated before the realization because we have no idea when the 'event' will occur. The 2nd step of state transition from $X'(k+1)$ to $X(k+1)$ is always accompanied by the completion of a task.

(b) **Completion of a Task with a Realization**

Given a decision with multiple actions, there can be more than one on-going project in the temporary state $X'(k+1)$. Suppose an on-going task(being performed in resource type n) in the ℓth project is completed earlier than the other on-going tasks and the mth values of the parameters were realized for the task. Then the state transition from $X'(k+1)$ to $X(k+1)$ will be :

$$X(k+1) = [s'_1, \cdots, s''_\ell, \cdots, s'_M, z_1, \cdots, z'_\ell, \cdots, z_M, L_1, \cdots, L'_n, \cdots, L_N, t']^T \quad (63)$$

where, $X'(k+1) = [s'_1, s'_2, \cdots, s'_M, z_1, z_2, \cdots, z_M, L_1, L_2, \cdots, L_N, t]^T$, $s''_\ell = s'_\ell + 1$, $z'_l = m$, $L'_n = 0$ and t' is the time at which the task is completed. In some cases, more than one on-going task can be coincidently completed at a same time. In the case of simultaneous completion of multiple tasks, the corresponding state variables for those tasks are updated in the same manner as it is described in equation (63).

3. **Terminal States**

One characteristic of the RCPSP is that the task network of the problem is not deterministic due to uncertain outcomes(success or failure) of the tasks in the problem.

Even though there is a unique initial state, the problem can end with one of numerous terminal states according to the realization. For example, in the Gantt chart(Figure 22), all the tasks in projects 3, 4 and 5 are completed. However, only one task in projects 1 and 2(task $I1$ and $I3$, respectively) is performed due to task failures. The number of possible terminal states depends on the stochastic complexity of the problem. We define the terminal state as the state where all the project are terminated. The termination condition for each project is defined either by the successful completion of the final task or the failure of an intermediate task.

4.3.4 Objective Function : Cost-to-Go

The objective of the RCPSP is the maximization of the final reward after finishing all projects. However, for the convenience of comparing solutions, the "Net Present Value" of the solution(schedule) has been generally used in previous problem formulations[77, 76, 46, 69, 13, 53]. In this study, we will set the objective function as the final reward of the problem for an exact evaluation of the solution. This objective can be translated into a 'cost-to-go' value, which represents the expected cost(-profit) to be spent from the current state to the terminal state. As described in section 4.2, the value of reward for each project decreases with time t. This reward decrease can be considered as an increase in the cost. Therefore, the expected 'cost-to-go', $J(X(k))$, at current state $X(k)$ is defined follows

$$\begin{aligned} J(X(k)) &= E\{\text{Future Cost to Complete All Remaining Projects} \\ &- \text{Rewards of Remaining Projects to be Retrieved in the Future}\} \end{aligned} \tag{64}$$

A large negative 'cost-to-go' means a high probability of retrieving a large amount of reward in the future. On the other hand, if the 'cost-to-go' is positive, one can expect more cost to be spent in order to complete the projects with less future rewards. To obtain initial guess values of the cost-to-go in the equation (64), simulations can be performed with the suboptimal heuristics introduced in section 4.5 and the cost-to-go values are evaluated for all the points of the state trajectories visited by the heuristics.

It should be noted that the DP formulation developed in this section is limited to the RCPSP, which has fairly simple problem structure, described in Section 4.2. However, the DP formulation is flexible to be extended for richer problem structure by modifying the

state definition and introducing new actions. Further extension of the DP formulation for more realistic RCPSP description will be discussed in Section 4.7.

4.4 Dynamic Programming in a Heuristically Confined State Space

All the necessary elements of the DP are defined in the previous section. Thus, the problem can be solved using the appropriate Bellman equation. However, the computational load of the DP for realistic size examples will be beyond current computational capabilities. Suppose that a problem is given with 3 projects, each of which consists of 3 tasks. Suppose each task has 3 possible realizations and 2 types of resources are available and the longest task duration is about 5 time units. If all the projects can be completed at $t = 20$, the approximate number of states defined by the state definition(4.3.1) is about 4,630,500($7 \times 7 \times 7 \times 3 \times 3 \times 3 \times 5 \times 5 \times 20 = 4,630,500$). The number of states tends to increase exponentially with the problem size, number of projects, number of possible realizations, and number of resources. In this work, a DP approach with a systematic approximation, DP in heuristically confined state space[27, 23, 21] is tailored for the given problem. The main idea of the algorithmic framework is to first find an important set of states via a large number of simulations with various heuristic policies and then solving the DP over the set of states visited by the heuristics to obtain an optimal solution within the confined state space as illustrated in Figure 23. The general steps for applying the algorithmic framework will be similar to those shown in [23].

1. Stochastic Simulations with Heuristic Policies.
2. Identification of the set of visited states and the first cost-to-go approximation.
3. Bellman iteration in a heuristically confined state space
4. Online evalutaion

A detailed description of the algorithm follows.

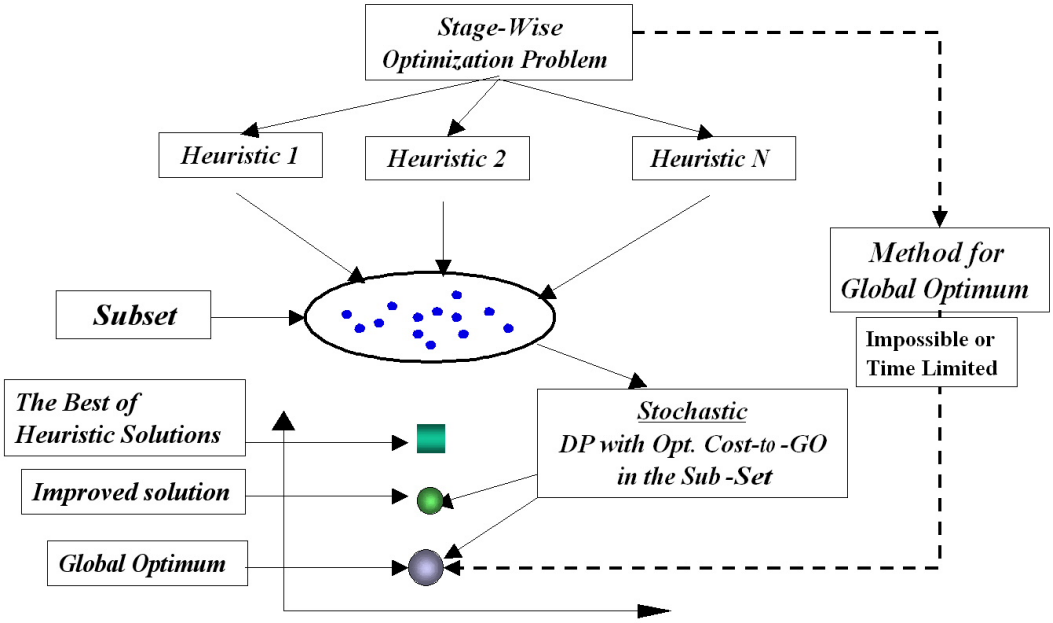

Figure 23: Stochastic DP in the Subset of the States

4.4.1 Simulation of Heuristic Policies

The purpose of the simulation is two-fold. First, the simulation is performed in order to obtain a meaningful set of the states within which the DP is to be performed. Obtaining a reasonably sized subset containing trajectories of good policies is critical for solving the problem because DP over the entire state space is computationally infeasible for the given problem. For the simulation, a large number of uncertain parameter realization sets are generated by the underlying Markov chains. Each realization set represents one scenario out of the enormous number of possible scenarios. Several different heuristics are applied for each realization and a set(trajectory) of visited states(as defined in 4.3.1) is obtained from each heuristic. Because each heuristic works in a different way, there can be several different state trajectories even for the same scenario. Those different state trajectories will be combined in the state space by the later step, Bellman Iteration, of the algorithmic framework. The heuristic policies applied for this problem will be described in Section 4.5.

Second, the simulation provides initial 'cost-to-go' values, which can be used in the Bellman iteration step, for the states. According to the definition of the 'cost-to-go'(4.3.4), a 'cost-to-go' value is calculated for each state in the state trajectories obtained by the

simulation of the heuristic policies. The same state in different state trajectories can have different estimates of its 'cost-to-go' values according to which heuristic is used. For example, every state trajectory starts with the unique initial state, shown equation (77), but different heuristic policies may give different average values of the reward and cost. In the Bellman iteration step, one(lowest or average among the heuristics tried.) can be assigned as the initial estimate of the cost-to-go for each state.

4.4.2 Cost-to-Go Calculation for the confined state space

The total number of state trajectories obtained by the simulation of the heuristic policies(30) is $\nu \times n$, where ν is the number of realizations and n is the number of heuristics tried in the simulation. The subset should consist of all non-redundant states in the $\nu \times n$ trajectories. This step requires substantial computation. If one state appears μ times in the set of trajectories, all the realized cost values obtained from the trajectories are added and divided by μ for the initial 'cost-to-go' value calculation. For example, the cost-to-go value for the initial state is chosen as the mean value of the total rewards minus the total costs over all the simulations. The initial guess for the 'cost-to-go' values obtained in the previous step are used as $\hat{J}^0$ to initialize the Bellman Iteration, where we iterate the following equation (65) for each state $X(k)$ in the subset, until $\hat{J}^i$ meets a certain convergence criteria, i.e. $||\frac{J^{i+1}-J^i}{J^i}||_\infty < 0.01$:

$$\hat{J}^{i+1}(X(k)) = \min_{u(k)} E\{\phi(X(k), u(k)) - R(X(k+1)|X(k), u(k)) + \hat{J}^i(X(k+1)|X(k), u(k))\} \quad (65)$$

In the above, $\phi(X(k), u(k))$ represents the cost incurred by the decision $u(k)$ for the state $X(k)$ and $R(X(k+1)|X(k), u(k))$ is the reward retrieved at the completion of a project as a result of decision making. The reward value will be zero unless a project is completed at state $k+1$. It should be noted that the Bellman Iteration equation(65) is consistent with the cost-to-go definition shown in the equation (64). Suppose that m_{th} state is the terminal state of a certain state trajectory. The total reward of the solution(state trajectory) is a consequence of all the costs spent and all the rewards retrieved along the state trajectory from the initial state to the terminal state. The total reward(TR) can be obtained by

following equation (66),

$$TR = \sum_{k=0}^{m-1} \left(R(X(k+1)|X(k), u(k)) - \phi(X(k), u(k)) \right) \tag{66}$$

Thus, if we convert the total reward into the cost-to-go, the cost-to-go value for the initial state, $J(X(0))$, is

$$\begin{aligned} J(X(0)) = -TR &= \sum_{k=0}^{m-1} \left(-(R(X(k+1)|X(k), u(k)) + \phi(X(k), u(k))) \right) \\ &= \phi(X(0), u(0)) - R(X(1)|X(0), u(0)) + J(X(1)) \\ &= \phi(X(0), u(0)) - R(X(1)|X(0), u(0)) + \underbrace{\phi(X(1), u(1)) - R(X(2)|X(1), u(1)) + J(X(2))}_{J(X(1))} \\ &\vdots \end{aligned} \tag{67}$$

The Bellman Iteration equation shown in the equation (65) is a generalization of the equation (67) with the expectation evaluation and cost-to-go minimization over the various stages.

For each state in the subset, we can identify all the possible decisions, $u(k)$, from the definition in 4.3.2. Once we know the possible decisions, the expected cost can be calculated for each of the possible decisions using the conditional probability. For each one of these decisions, the possible next states and their transition probabilities are obtained analytically according to the state transition rules and the given conditional probabilities. Each of those possible next states has the cost-to-go value calculated from the previous iteration and the information about the status of all the projects for the retrieved reward, $R(X(k+1))$, calculation. In the calculation of the reward, the last state variable, t, and the given reward functions(equation (58)) have to be used for obtaining the exact value of the reward at the time of its retrieval. After a sufficient number of iterations of equation (65), the converged cost-to-go $J^*(X(k))$ is obtained for every state in the subset.

- **Cost-to-Go Approximation for Partially Connected States**

  In the Bellman Iteration equation of (65), the calculation of $E\{\phi(X(k), u(k))\}$ and $E\{R(X(k+1)|X(k), u(k)))\}$ can be done exactly for every possible case. However, the exact calculation of $E\{\hat{J}^i(X(k+1)|X(k), u(k))\}$ is not possible because there is no guarantee that the subset is closed, i.e., for any state in the subset, all possible

next states are in the subset as well. The subset may be open for the following two reasons:

1. **A finite number of heuristic policies, which do not cover the entire decision space, are applied in simulation to form the subset.**

   The states in the subset are not arbitrarily chosen. A set of reasonable heuristics are implemented in simulation to collect all the visited states. Hence, in doing so, many state transitions, possible with certain decisions not covered by the heuristics, may never have occurred during the simulation. The states involved in the unrealized transitions would not have been included in the subset. Indeed, our intention was to reduce drastically the number of states we must examine.

2. **Only a finite number of realizations are simulated**

   If the number of possible scenarios are very large, it is unlikely that one can realize every possible scenario in simulation because some scenarios have a very low probability of occurring.

The states not included in the subset due to 1 and 2 have to be distinguished and dealt with differently in the Bellman iteration. In the case of 1, a group of possible next states associated with decisions not covered by the chosen heuristics, are not visited at all in the simulation.

In Figure 24, the decision $u_2(k)$ has never been made during the heuristic simulation. To confine the decision to those leading to a state in the subset, we propose to prevent the unseen decision by assigning a large cost-to-go values to those states. The large cost-to-go value will act as a barrier for the decision and one of the other decisions will be chosen in the minimization step of the Bellman Iteration. This implies that the large cost-to-go values will not be propagated to current or previous states. This cost-to-go approximation is based on the assumption that a reasonable number of good heuristics have been tried and all good decisions have been covered.

For reason 2, some of the next possible states associated with a simulated decision may be absent in the subset In Figure 24, a state linked from the decision $u_3(k)$ is

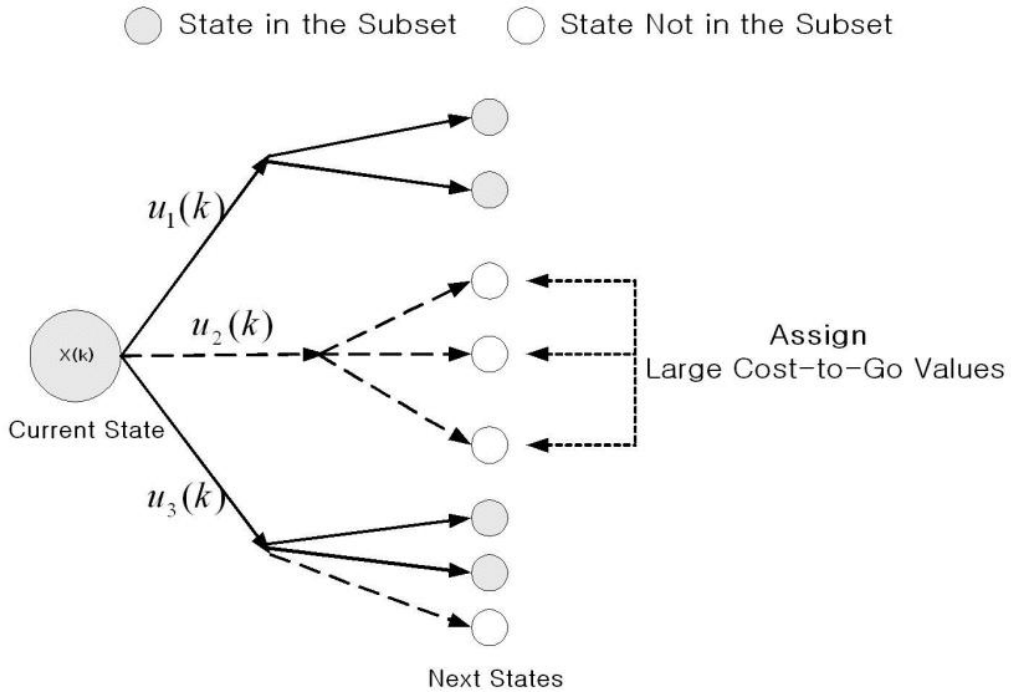

Figure 24: Cost-to-Go Approximation Type 1

not in the subset because the state transition is not only governed by the decision but by random factors as well. Theoretically, all the possible states under the tried heuristic policies can be included in the subset by performing a 'sufficient' number of realizations. However, for a problem with an enormous number of scenarios(i.e. Illustrative Example in Section 4.6), this may not be feasible. Thus, an approximation strategy is necessary to deal with this inevitable absence of some states in the subset. If a state is not in the subset due to the reason 2, it implies that the probability of transition to the state is comparatively small. Thus, we suggest that those states can be ignored and the state transition probabilities for the rest of the states are normalized accordingly as shown Figure 25.

With the proposed approximation methods, the Bellman Iteration gives 'converged' cost-to-go values rather than 'optimal' cost-to-go values. The issue of the open subset is re-examined in the next step of online decision making.

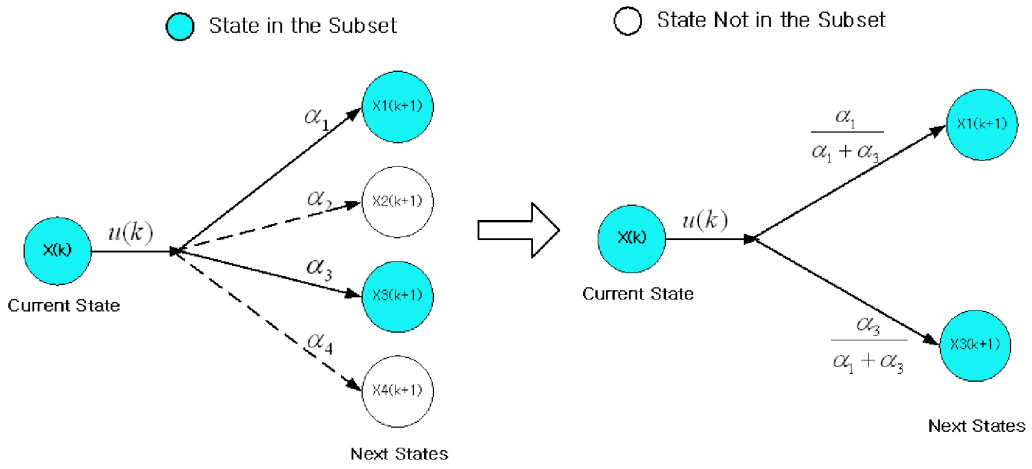

Figure 25: Cost-to-Go Approximation Type 2

4.4.3 Online decision making

The 'converged' cost-to-go values obtained in the previous step are used for online decision making as follows. If the 'converged' cost-to-go, $\hat{J}^*$, is the optimal cost-to-go, the following decision $u^*(k)$ also will also be optimal according to the 'Principle of Optimality' of DP.

$$u^*(k) = \arg\min_{u(k)} E\{\phi(X(k), u(k)) - R(X(k+1)|X(k), u(k)) + \hat{J}^*(X(k+1)|X(k), u(k))\} \quad (68)$$

However, the decision may be suboptimal because the converged cost-to-go, $\hat{J}^*$, is obtained by the approximation procedure described in section 4.4.2. Furthermore, the online decision making equation of (68) is not valid for every situation because the random factors may take the system outside the previously experienced subset. The online decision has to be robust for any possible realization, some of which may lead a state trajectory outside the subset, for which the cost-to-go is not available. In this work, we use the following two approaches to online decision making.

- Method 1 : Online decision making with a cost-to-go barrier

  A fixed high cost-to-go value is assigned to all states outside the subset, thereby making a decision leading to a state outside the subset highly unlikely. This approach is basically the same as the approximation method developed for the Bellman Iteration step.

- Method 2 : Online decision making with a guiding heuristic

  In this approach, we allow the state to step outside the subset. We use a heuristic policy whenever a state outside the subset is encountered. The best among the tried heuristic policies, in terms of the mean value of the reward can be used for this. Once the state comes back into the subset, the decision making is switched to the minimization of the cost-to-go, as shown in Figure 26.

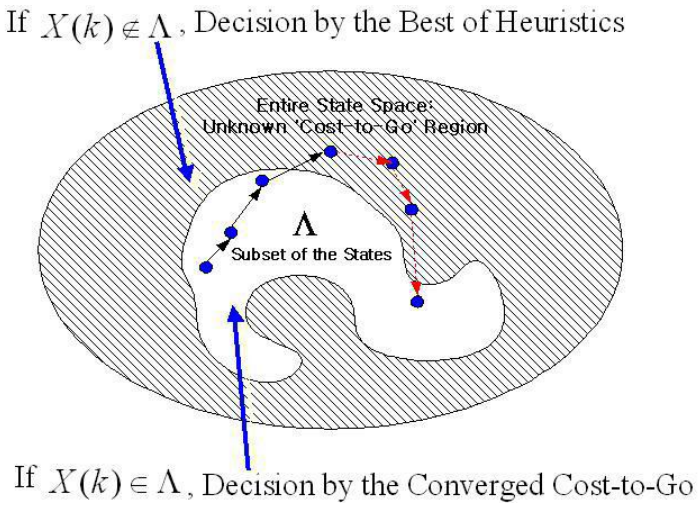

Figure 26: online Decision Making with A Guiding Heuristic

4.5 Suboptimal Policies : Heuristics

To apply the algorithmic framework developed in [23], developing a set of reasonable heuristics for the problem is very important so that a reasonable subset of the states can be formed. In this section, three heuristics, which utilize information from the state as defined in 4.3.1, are developed for the Resource-Constrained Project Scheduling Problem(RCPSP). These heuristics emphasize different information about the problem and hence combining them together could lead to a better overall performance.

4.5.1 Heuristic 1 : High Success Probability Task First

In RCPSP, the result(success or failure) of the task is a very important factor affecting the final reward, as well as the remaining part of the scheduling solution. Heuristic 1 is

developed based on maximizing the probability of the success of the next allocated task. For the decision, the expected success probability of each task is calculated based on the current information state, $z_i(k)$ for $i = 1, 2, ..., M$. Once we know $z_i(k)$ for each task, we can also obtain the corresponding conditional probability for each outcome. The expected success probability of the task can be calculated by summing the probability of each successful outcome. This heuristic can be modified for the case of multiple level of success by assigning appropriate weighting factors for the different levels of success.

4.5.2 Heuristic 2 : Short Duration Task First

Another way to increase the final reward of the projects in the RCPSP is to finish the projects as quickly as possible in order to minimize the loss of reward with time. Heuristic 2 considers the time value of the project in a greedy way by performing a task with shortest expected duration first in cases of resource conflicts. The expected duration can be calculated by utilizing current information state, $z_i(k)$,as in the calculation of expected success.

4.5.3 Heuristic 3 : High Reward Project First

Heuristic 3 gives priority to the impending task of the project which has the highest potential reward. This is a very greedy decision to get a highest reward in a short time with the smallest reward decrease. Heuristic 3 may work well if the project with the high reward is completed successfully. Its drawback is the other projects can be delayed too long. Therefore, if a project with the highest reward fails, the total reward can be decreased significantly. In the R&D pipeline management problem, priority of each project is decided in the order of initial reward value. The decision making procedure is straightforward, all the tasks in the project with the highest reward are performed first and so on. If there is an idle resource after assigning a pending task in the target project, the resource is used to perform a task in the next priority project.

Table 6: Example 1, Probabilities and Parameters

Project	Realized Result, Duration and Cost of the Task		
Project 1	I_1	I_2	P_1
	$\begin{bmatrix} F & 3 & 250 \\ F & 5 & 500 \\ S_1 & 4 & 300 \\ S_1 & 4 & 300 \\ S_2 & 5 & 400 \end{bmatrix}$	$\begin{bmatrix} F & 4 & 350 \\ S_1 & 5 & 300 \\ S_2 & 6 & 250 \end{bmatrix}$	$\begin{bmatrix} F & 4 & 300 \\ F & 7 & 650 \\ S_1 & 5 & 500 \\ S_1 & 4 & 450 \\ S_1 & 3 & 300 \\ S_2 & 3 & 400 \end{bmatrix}$
	PI_1	PM_{11}	PM_{21}
	$\begin{bmatrix} 0.1897 \\ 0.2441 \\ 0.2842 \\ 0.2276 \\ 0.0543 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0.1797 & 0.3013 & 0.0084 \\ 0 & 0 & 0.4143 & 0.6562 & 0.3001 \\ 0 & 0 & 0.4061 & 0.0425 & 0.6915 \end{bmatrix}$	$\begin{bmatrix} 0 & 0.2646 & 0.0057 \\ 0 & 0.1193 & 0.1109 \\ 0 & 0.0871 & 0.1964 \\ 0 & 0.0067 & 0.1991 \\ 0 & 0.3273 & 0.2430 \\ 0 & 0.1951 & 0.2449 \end{bmatrix}$
Project 2	I_3	I_4	P_2
	$\begin{bmatrix} F & 3 & 300 \\ F & 5 & 400 \\ S_1 & 4 & 350 \\ S_1 & 6 & 500 \\ S_1 & 5 & 600 \\ S_2 & 3 & 350 \end{bmatrix}$	$\begin{bmatrix} F & 5 & 700 \\ F & 6 & 650 \\ S_1 & 8 & 900 \\ S_1 & 7 & 600 \\ S_2 & 5 & 400 \end{bmatrix}$	$\begin{bmatrix} F & 4 & 550 \\ S_1 & 6 & 600 \\ S_2 & 5 & 450 \end{bmatrix}$
	PI_2	PM_{12}	PM_{22}
	$\begin{bmatrix} 0.1897 \\ 0.2441 \\ 0.2842 \\ 0.2276 \\ 0.0543 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0.1797 & 0.3013 & 0.0084 \\ 0 & 0 & 0.4143 & 0.6562 & 0.3001 \\ 0 & 0 & 0.4061 & 0.0425 & 0.6915 \end{bmatrix}$	$\begin{bmatrix} 0 & 0.2646 & 0.0057 \\ 0 & 0.1193 & 0.1109 \\ 0 & 0.0871 & 0.1964 \\ 0 & 0.0067 & 0.1991 \\ 0 & 0.3273 & 0.2430 \\ 0 & 0.1951 & 0.2449 \end{bmatrix}$

4.6 Illustrative Example

As an illustrative example of the RCPSP, we consider a generalized R&D pipeline problem that has 5 projects with 2 resources. AoN(Activity-on-Node) graph of the example is shown in Figure 27. The AoN displays the sequence of tasks involved in each project with the resources required to complete the tasks(e.g. task 'I_1' has to be performed in Laboratory 1 (Lab.1)). A parenthesized number over each task represents the possible number of outcomes(in terms of the duration, cost, and result, the multiple levels of success or failure of the task) of the task. A Markov chain is given for each project to represent correlations among the outcomes of adjacent tasks of a project. For example, for task I_1, which has 5 possible outcomes, a 5×1 probability vector is assigned each element of which represents a probability of the corresponding outcome. The conditional probabilities for the outcomes of the task I_2 is assigned based on the realized outcome of I_1. Since I_1 and I_2 have 5 and 3 possible realizations respectively, the size of the probability matrix for I_2 is 3×5. Each column represents the conditional probability vector for the possible outcomes of I_2. All the probabilities and parameters of the example are summarized in Table 13 and 7.

R_i, for $i = 1, ..., 5$, indicates the initial reward of project i at time $k = 0$. After time $k = 0$, the reward of each project decreases as shown in Figure 28. The reward profile in

Table 7: Example 1, Probabilities and Parameters (Continued)

Project	Realized Result, Duration and Cost of the Task			
Project 3	I_5	I_6	I_7	P_3
	$\begin{bmatrix} F & 4 & 400 \\ F & 6 & 600 \\ F & 7 & 650 \\ S_1 & 4 & 500 \\ S_1 & 5 & 450 \\ S_1 & 3 & 300 \\ S_2 & 2 & 250 \end{bmatrix}$	$\begin{bmatrix} F & 4 & 400 \\ S_1 & 6 & 550 \\ S_1 & 5 & 450 \\ S_2 & 3 & 150 \end{bmatrix}$	$\begin{bmatrix} F & 3 & 800 \\ F & 6 & 500 \\ S_1 & 5 & 450 \\ S_1 & 5 & 700 \\ S_2 & 7 & 300 \end{bmatrix}$	$\begin{bmatrix} F & 5 & 600 \\ S_1 & 7 & 300 \\ S_1 & 4 & 450 \end{bmatrix}$
	PI_3	PM_{13}	PM_{23}	PM_{33}
	$\begin{bmatrix} 0.0827 \\ 0.2480 \\ 0.1877 \\ 0.1064 \\ 0.1923 \\ 0.0692 \\ 0.1136 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0.2970 & 0.3376 & 0.3258 & 0.0396 \\ 0.3238 & 0.3149 & 0.2518 & 0.1095 \\ 0.1067 & 0.3696 & 0.4172 & 0.1065 \\ 0.1238 & 0.1261 & 0.1384 & 0.6117 \end{bmatrix}^T$	$\begin{bmatrix} 0 & 0.0703 & 0.1718 & 0.0033 \\ 0 & 0.1907 & 0.1705 & 0.0290 \\ 0 & 0.2856 & 0.1680 & 0.1754 \\ 0 & 0.1964 & 0.1537 & 0.3834 \\ 0 & 0.2571 & 0.3361 & 0.4089 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0.4666 & 0.0123 & 0.5212 \\ 0.3054 & 0.3217 & 0.3728 \\ 0.1060 & 0.3996 & 0.4944 \end{bmatrix}^T$
Project 4	I_8	I_9	I_{10}	P_4
	$\begin{bmatrix} F & 4 & 450 \\ F & 6 & 550 \\ S_1 & 3 & 600 \\ S_2 & 4 & 400 \end{bmatrix}$	$\begin{bmatrix} F & 7 & 600 \\ F & 3 & 450 \\ S_1 & 4 & 300 \\ S_1 & 7 & 800 \\ S_1 & 5 & 500 \\ S_2 & 2 & 300 \end{bmatrix}$	$\begin{bmatrix} F & 7 & 400 \\ S_1 & 5 & 500 \\ S_2 & 6 & 350 \end{bmatrix}$	$\begin{bmatrix} F & 3 & 300 \\ F & 8 & 700 \\ S_1 & 6 & 450 \\ S_1 & 4 & 600 \\ S_2 & 2 & 500 \end{bmatrix}$
	PI_4	PM_{14}	PM_{24}	PM_{34}
	$\begin{bmatrix} 0.0429 \\ 0.2211 \\ 0.3248 \\ 0.4112 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0.1285 & 0.0323 \\ 0 & 0 & 0.1664 & 0.0484 \\ 0 & 0 & 0.1825 & 0.1475 \\ 0 & 0 & 0.0335 & 0.2224 \\ 0 & 0 & 0.2658 & 0.2577 \\ 0 & 0 & 0.2233 & 0.2918 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0.3733 & 0.2010 & 0.4258 \\ 0.2203 & 0.3078 & 0.4720 \\ 0.2863 & 0.5542 & 0.1594 \\ 0.0422 & 0.3066 & 0.6511 \end{bmatrix}^T$	$\begin{bmatrix} 0 & 0.1564 & 0.0550 \\ 0 & 0.0912 & 0.0093 \\ 0 & 0.2111 & 0.1769 \\ 0 & 0.3007 & 0.4020 \\ 0 & 0.2406 & 0.3568 \end{bmatrix}$
Project 5	I_{11}	I_{12}	P_5	
	$\begin{bmatrix} F & 4 & 450 \\ F & 6 & 550 \\ S_1 & 3 & 600 \\ S_2 & 4 & 400 \end{bmatrix}$	$\begin{bmatrix} F & 7 & 600 \\ F & 3 & 450 \\ S_1 & 4 & 300 \\ S_1 & 7 & 800 \\ S_1 & 5 & 500 \\ S_2 & 2 & 300 \end{bmatrix}$	$\begin{bmatrix} F & 3 & 300 \\ F & 8 & 700 \\ S_1 & 6 & 450 \\ S_1 & 4 & 600 \\ S_2 & 2 & 500 \end{bmatrix}$	
	PI_5	PM_{15}	PM_{24}	
	$\begin{bmatrix} 0.2031 \\ 0.5663 \\ 0.2306 \end{bmatrix}$	$\begin{bmatrix} 0 & 0.0419 & 0.0004 \\ 0 & 0.1560 & 0.0025 \\ 0 & 0.0546 & 0.0389 \\ 0 & 0.2035 & 0.0717 \\ 0 & 0.0299 & 0.1655 \\ 0 & 0.1137 & 0.1986 \\ 0 & 0.1951 & 0.2391 \\ 0 & 0.2054 & 0.2833 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0.1528 & 0.2326 & 0.2605 & 0.3540 \\ 0.0954 & 0.2727 & 0.3136 & 0.3184 \\ 0.3116 & 0.3539 & 0.0762 & 0.2583 \\ 0.0013 & 0.2688 & 0.3490 & 0.3810 \\ 0.0564 & 0.2547 & 0.3190 & 0.3699 \end{bmatrix}^T$	

Figure 28 represents the "time value" of each project due to competitive market situation. If a project is delayed for too long(longer than PD_i, 'project deadline'), the reward from completing the project can be insignificant because similar drugs(products) developed by competitors may have taken a large market share.

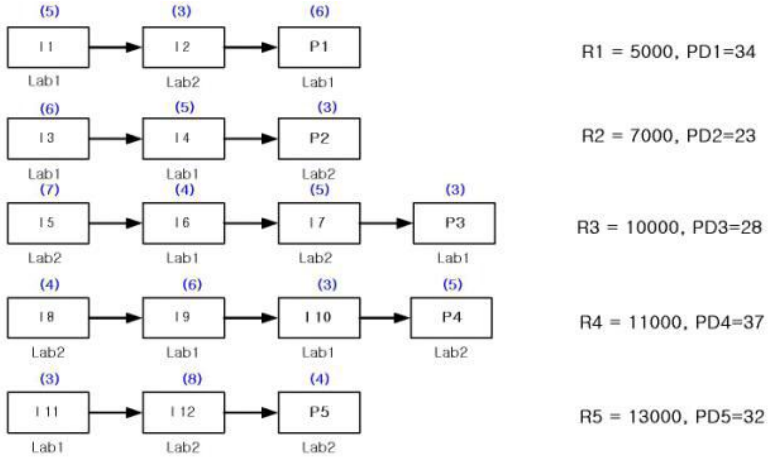

Figure 27: RCPSP Illustrative Example

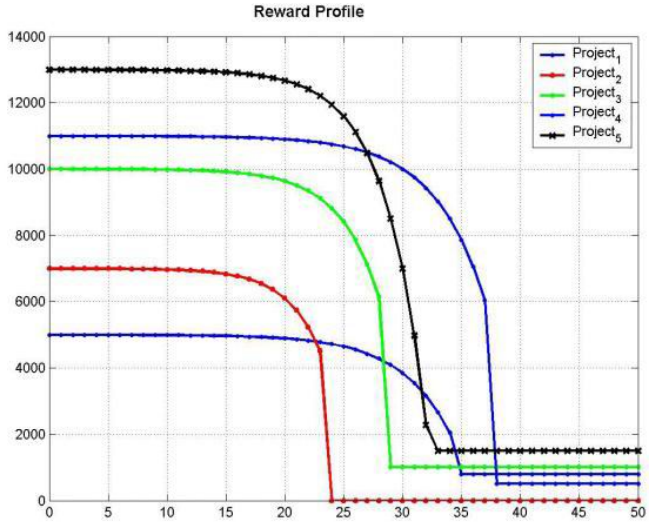

Figure 28: Reward Profile of the Projects in the Illustrative Example

- **Stochastic Complexity of the Example**

  The illustrative example is a small size RCPSP, which consists of only 5 projects. However, it is actually a large size problem due to its stochastic complexity. One measure

of the stochastic complexity of the problem is total number of possible scenarios under different parameters realizations. Figure 29 shows project 1 in the illustrative example and its realization data. According to the realization data, there are 36 scenarios, $6 \times 2 \times 3$(6 realizations in $P1$, 2 realizations in $I2$ linked to $P1$ and 3 realizations in $I1$ linked to $I2$), in case of project termination with $P1$ completion, i.e. all three tasks in the project are completed. In same way, in case of project termination with $I2$ and $I1$ completions, there are 3 and 2 scenarios respectively. Thus, the total number of scenarios of the project 1 is 41.

In summary, with given realization data in Table 13, total number of scenarios of

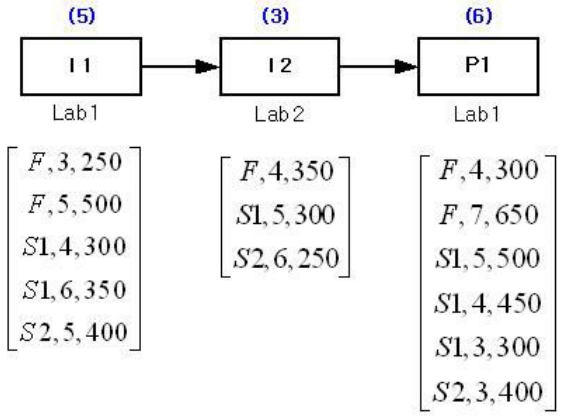

Figure 29: Project 1 in the Illustrative Example

the project 2,3,4 and 5 is 46, 139, 94, and 47 respectively. The scenarios of each project are independent of each other, thus, total number of scenarios of the problem is 1,214,693,756 found by multiplying the number of scenarios of all projects.

4.6.1 Simulation with the 3 Heuristic Policies

The three Heuristics introduced in section 4.5 were implemented on the illustrative example. For the simulation, 50,000 uncertain parameters realizations were performed according to the underlying Markov chains for each heuristic. The simulation results of Heuristic 1, 2 and 3 are shown in Figure 30. For all three heuristics, -8200 is the worst realized reward value corresponding to Project1 3-3-2(failure), Project2 5-4-2(success), Project3 4-2-3-1(failure), Project4, 4-6-2-1(failure) and Project5 3-5-1(failure). All the projects are successful before

the final tasks and all the final tasks fail except for project 2. Therefore, a large amount of money had been spent to perform all the tasks without any reward. The one project(P2), which has been completed successfully, retrieves only the minimum reward of the project(0) due to the long delay of the project under the heuristic rules. On the other hand, the maximum rewards of the heuristics and the corresponding realizations in which the maximum rewards are obtained are different for different heuristics, as summarized in Table 8. To achieve large rewards, projects 3,4, and 5 have to be completed successfully and quickly. Heuristic 1 works effectively to meet this requirement based on the expected success probability of each task. The first tasks($I1$ and $I3$) in projects 1 and 2 are performed at the end, as shown in Figure 32, and all the other projects are completed before their project deadline. Heuristic 3 also works in a similar way to Heuristic 1 though it completes projects 4 and 5(at time 19 and 15) earlier than Heuristic 1(at time 24 and 20) does. In the case of the Heuristic 2, tasks $I1$ and $I3$ in the failed projects are performed at earlier stages due to their short expected durations and the other projects are delayed. Most of all, the delay of the project 5 is critical because it is completed after its 'project deadline'. In summary, the simulation results(Table 8, 9 and Figure 30 to 32) indicate that none of the heuristics is uniformly superior. The relative performances of the different heuristics vary by realization. To obtain better results, the decision policy has to capture the overall stochastic complexity of the problem and utilize the information state appropriately. The simulation results will be used for obtaining a policy that performs better than any of the heuristics and this is done by performing DP within the visited region of the state space. The details of implementation procedures and results of the DP for the illustrative example will be described in the next section.

Table 8: Heuristic Solutions: The Maximum Rewards

	Realization #39804	Realization #7181	Realization #6452
Heuristic 1 Reward	27714*	20242	18502
Heuristic 2 Reward	16685	24902*	15531
Heuristic 3 Reward	25651	20429	25907*

* The chosen realizations correspond to those giving the maximum rewards for the three heuristics.

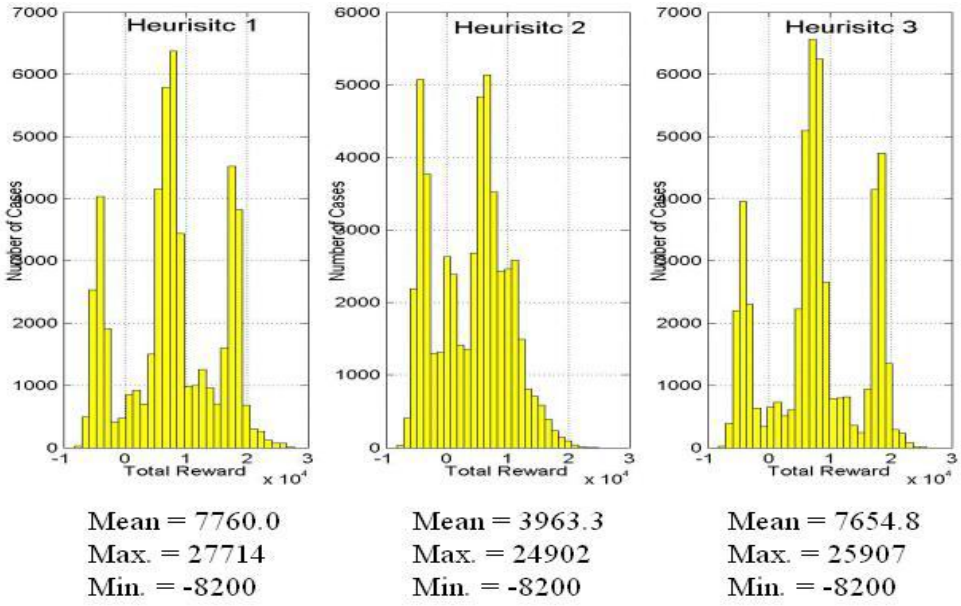

Figure 30: Heuristic Simulation Results for 50,000 Realizations

Table 9: Largest Positive Solution Difference Between 2 Heuristics through 50,000 Realizations

$\|\|H1_s^* - H2_s\|\|$	$\|\|H2_s - H1_s\|\|$	$\|\|H1_s - H3_s\|\|$	$\|\|H3_s - H1_s\|\|$	$\|\|H3_s - H2_s\|\|$	$\|\|H2_s - H3_s\|\|$
15571	10400	21943	12265	15822	26672

* Solution Obtained by the Heuristic 1

The results shown in Table 9 shows the possibility of improving the solutions given by the three heuristics by searching over the subset of the states visited by the heuristics because none of the heuristics is dominant for all cases.

4.6.2 Implementation of DP in a Heuristically Confined State Space

The state of the illustrative example consists of 13 variables according the state definition in 4.3.1 as shown in equation (69).

$$X = [s_1, s_2, s_3, s_4, s_5, z_1, z_2, z_3, z_4, z_5, L_1, L_2, t]^T \tag{69}$$

The calculation of the total state space size is complex due to the inability of certain combinations of completed tasks and event times to be realized. It is expected that the duration of the whole schedule will be about 40 time units. Using this and estimates of the longest task durations and an idea of the possible task parameter sets, approximately 230

billion states could be experienced.

4.6.2.1 Confining the State Space & Calculating the Initial Values of the Cost-to-Go

As a result of the heuristic simulation, 150,000 sets(trajectories) of states are obtained. The total number of states in the sets is 1,741,484 including redundant states. The initial state is visited most frequently(150,000 times) during the simulation because every heuristic simulation starts with a unique initial state. Each of the 150,000 initial states may have different values due to different realization and decision policy applied for the state. The first approximation of the 'cost-to-go' for the initial state is given as an average of those 150,000 values. The initial 'cost-to-go' values of the other states are obtained in same way. Although the idea is quite simple, this step requires significant computation. The identification step took about 49.3 hours implemented in MATLAB on a Pentium 4 at 2.4 GHz with 2GB RAM. The resulting subset is consist of 371,168 non-redundant states. The size of this subset is about $0.00016\%(= \frac{371{,}168}{227{,}820{,}600{,}000})$of the size of the the entire state space.

4.6.2.2 Bellman Iteration & Converged Cost-to-Go

For the 371,168 states in the subset, the Bellman iteration, equation (65), is performed with the initial cost-to-go values obtained in previous step. In the Bellman iteration, a 'cost-to-go' approximation procedure is necessary because the size of the subset is vanishingly small compared to that of the entire state space. Accordingly, the approximation methods developed in Section 4.4.2 are used for the iteration. At every iteration, 371,168 cost-to-go values are updated for corresponding states in the subset. The iteration scheme converged within an error tolerance $||\frac{J^{i+1}-J^i}{J^i}||_\infty < 0.01$ after 14th iteration and took 7.9 days implemented in MATLAB on a Pentium 4 at 2.4 GHz with 2GB RAM.

4.6.3 Improved Solution: Online Decision Making

The decision policy obtained by the proposed approach is represented by the converged cost-to-go values and the online decision making equation (68). Thus, once we have the converged cost-to-go values, we can make a valid decision for any realization generated by the underlying Markov chain model. In the online decision making, the future results of the

currently on-going tasks are not known at the point of decision. The transition from the current state is a consequence of both the decision made according to equation (68), and the parameter values chosen by the random process. State transitions outside the subset are handled by two different methods, a cost-to-go barrier and a guiding heuristic, as explained in Section 4.4.3.

4.6.3.1 Online Decision Making for the Realizations Used for Simulation of the Heuristic Policies

To verify the performance of the policy obtained by the proposed approach, it is compared to the heuristic solutions for the 50,000 realization used to synthesize the policy. The results shown in Table 10 indicates that the proposed approach improves the mean performance by about 8.5%. compared to the best heuristic policy, Heuristic 1. For the online decision policy with a guiding heuristic, the best heuristic, Heuristic 1, is used as a guiding heuristic and the policy is slightly more effective than the policy with a 'cost-to-go' barrier. This result can be explained by observing the overall behavior of the policy, which tends to work similarly to the best heuristic, the Heuristic 1, in many cases. Accordingly, for some realizations for which the other heuristics are preferable, the policy using Heuristic 1 as a guide does not really improve the solutions.

Table 10: Online Decision Making Results: 50,000 Realizations

	H1	H2	H3	Best*	Online 1+	Online 2-
Mean	7760.0	3963.3	7654.8	8409.1	8422.7	8450.4
Max.	27714	24902	25907	27714	28468.5	28468.5
Min.	-8200	-8200	-8200	-8200	-8200	-8200

* The best heuristic solution for each realization.

+ Online decision making with cost-to-go barrier.

- Online decision making with a guiding heuristic.

As shown in Figure 33, the policy outperforms the heuristics even when the decision maker presciently chooses the best heuristic for each given realization, an option impossible to implement in practice since realization is not known ahead. However, the comparison

Table 11: Online Decision Making Results: Set of 5,000 New Realizations

	H1	H2	H3	Best	Online 1	Online 2
Mean	7758.9	3960.8	7636.1	8396.1	8445.8	8460.7
Max.	28123.6	21928.4	26479.5	28123.6	28168.5	28168.5
Min.	-6950	-6950	-6950	-6950	-6950	-6950

demonstrates that there is a synergy among the heuristics and new policies that connect the best parts of the heuristic solutions are synthesized. The proposed approach is also computationally efficient. Average computational time of the online decision making for each realization is only 7.5 seconds.

4.6.3.2 Online Decision Making for A Set of New Realizations

As explained in the previous part of this section, the stochastic complexity of the problem is very high with 1,214,693,756 scenarios. The policy should be robust for any of these scenarios even though they were not seen during its creation. To demonstrate the robustness of the policy, it is tested for 5,000 realizations from the underlying Markov chain model that were not part of the the training set. The computational results summarized in Table 11 and Figure 34 shows the robustness of the proposed approach in this example.

4.7 Extensions and Generalizations

In this section, potential extensions of the basic DP formulation(Section 4.3) for the simplified version of the RCPSP described in Section 4.2 are discussed. Once the target problem(RCPSP in this work) is clearly defined as a stage-wise optimization problem, it can be solved by the DP approach with appropriate definitions of the state and state transition rules. Thus, the basic DP formulation developed in Section 4.3 is quite flexible and can be extended to handle richer problem representations. The key ideas (e.g., definitions of state and actions) for the extensions are briefly addressed in this section. We leave the detailed state transition rules and other elements necessary for the extensions as future work

because along with the definitions of state and transition rules appropriate algorithmic enhancements need to be made to handle the substantially larger state space resulting from the extensions.

4.7.1 Dynamic Task Sequencing

In the simplified version of the RCPSP, a fixed sequence of tasks is assumed. However, the sequence of tasks can be a decision variable in real problems. Dynamic task sequencing can be embedded in our framework by adding a new state variable, q_i for $i = 1, 2, ..., M$, which is used to represent different possible task sequences in a project i. With the additional state variable q_i, the original state definition in equation (59) is modified as following.

$$X = [s_1, s_2, \cdots, s_M, q_1, q_2, \cdots, q_M, z_1, z_2, \cdots, z_M, L_1, L_2, \cdots, L_N, t]^T \tag{70}$$

If tasks I6 and I7 of project 3 in the illustrative example in Section 4.6 and Figure 27 are exchangeable, the new state variable q_3 can be used to distinguish between the two different task sequences as shown in Figure 35.

The state variable s_i for project status indication remains the same as defined in Section 4.3.1 but it represents various project status combined with the given project sequencing variable q_i. The choice of task sequence can be a part of the decision with a slight modification of the sequence decision variable δ_3. In equation (60), δ_3 was defined as a binary variable; however, in the case where task sequence is to be chosen, after the task $I5$ is successfully completed($s_3 = 3$), δ_3 can be 0(not to perform either task), 1(to perform a task I6, with $q_3 = 1$), or 2(to perform a task I7, with $q_3 = 2$). Appropriate state transition rules need to be developed according to the new definitions. Of course, options for task sequences can take on a much more complex form, which can enlarge the state space and complicate the transition rule.

4.7.2 Complicated Task Sequences and Actions

In real RCPSPs, there can be branching or merging of the task sequence rather than a straight sequence assumed in the simplified version of the RCPSP [46, 78]. Such complicated task sequences can be handled by the DP formulation in Section 4.3 by modifying the state

variable s_i, which represents the project status, while keeping the state vector the same as in equation (59). Suppose that a project has a branching and merging task sequence as shown in Figure 36.

For the project shown in Figure 36, the state variable s_i can be defined as summarized in Table **??** to distinguish among the various states of the project.

As illustrated in Figure 36 and Table **??**, the extension is not limited to a specific type of structure as long as the state variable s_i can be appropriately defined. For example, we may have another branching to task $I3'$ as shown in the Figure 36. Task $I3'$ can be a task that must be completed prior to task $I4$, or it can be an outsourcing option, in which case it is treated as an alternative path to task $I3$. In addition to the modified state variables, additional action variables may be necessary to describe the decision regarding the options in the new structure. Figure 37 illustrates the extended action space at $s_i = 3$ for the task structure shown in Figure 36.

4.7.3 Uncertain Resource Requirements and Various Types of Resource Requirements

In more realistic RCPSPs, multiple types of resources may be required for a task. The multiple resource requirement can be considered in the current state definition in equation (59) by extending the value space of the state variables L_j for $j = 1, 2, ..., N$. Suppose that we have 5 type-1 Laboratories(equipment) and 3 type-2 Laboratories, then we need to define state variables, L_1 and L_2 as the number of remaining units for the respective resource type. For example, if two type-1 Laboratories and one type-2 Laboratories are required to perform a task when $[L_1\ L_2] = [5\ 3]$, the state variables become $[L_1\ L_2] = [3\ 2]$ after the task is started. Besides the equipment, amount of labor(number of technician) can be treated as a resource[46] and the requirement for this type of resource can be uncertain[76]. The labor(resource) requirement can be incorporated into the model by introducing additional state variables, p_i, for $i = 1, 2, ..., L$. where L is the number of different types of labor.

$$X = [s_1, s_2, \cdots, s_M, z_1, z_2, \cdots, z_M, L_1, L_2, \cdots, L_N, p_1, p_2, \cdots, p_L, t]^T \tag{71}$$

The state variable, p_i, is an integer variable ranging from 0 to the maximum number of units for the i^{th} type labor and indicates the number of units available at the time. For example, if we have two different types of labor for a RCPSP with five projects, p_1 and p_2 are added as state variables to the previously defined state. Suppose the maximum available number of type-1 labor and type-2 labor are 20 and 30 respectively and $p_1 = 10$ and $p_2 = 0$. Then we have 10 type-1 and 0 type-2 labor available at the time.

The uncertain labor requirement can be considered in the Markov chain, not in the definition of state and state transition rules, similar to how the cost of a task is treated as an uncertain parameter in Section 4.3.1.

4.7.4 New Project Arrival

In realistic RCPSP, new projects may arrive while the current project scheduling is ongoing. At the simplest level, one can reformulate the problem and develop a new policy at that point. However, for problems where new projects occur on a frequent basis, the decisions prior to their arrival may need to account for the possibilities of new projects. For example, one may want to reserve some resource in order to be able to accommodate very promising projects that may come later. Complicating this is that information about future projects including the number, arrival time, and characteristics of the projects is not known exactly. Hence, one can assume certain statics about these unknowns and accommodate them in the simulation based methodology.

In such a case M in the state definition would represent the maximum number of projects that can go on at any particular time. This way not all M projects may be active at any given time. Whenever a new project arrives, one of the "inactive" project slot is "activated." Also, once a project is completed or cancelled, the state variables relevant to that project are reset so that it can represent another project in the future.

It should be noted that all the extensions discussed in this section can be superimposed to represent a more complicated RCPSP structure. It doesn't require a fundamental change to the methodology. However, two practical limitations exist, which are the capability of the algorithm to handle a very large state space and the existence of reasonable heuristics.

Hence, for meaningful treatment of these extended problems, these issues will have to be resolved. This, however, is beyond the scope of the current work.

4.8 Conclusions

A stochastic resource-constrained project scheduling problem(sRCPSP) has been addressed by using Markov chains to model key uncertainties(the duration, cost, and result of a task). To solve the problem, a DP formulation has been developed with the appropriate definitions of state, including the information state variables, state transition rules, and actions. The conventional stochastic DP approach cannot be used as a solution method for the problem due to the enormous state space. A novel algorithmic framework, *DP in a heuristically confined state space*[23], was tailored for the problem. The algorithmic framework has been tested by solving an illustrative SRCPSP with significant stochastic complexity. By simulating the problem with three heuristic policies, we obtained a set of visited states, which corresponds to only about 0.00016% of the entire state space. We then performed DP over the states with reasonable computation time. The policy obtained by solving the DP showed superior performance to any of 3 heuristic policies. Indeed, the solution obtained by the policy on average outperformed the best heuristic solution chosen for each different realization. Furthermore, the robustness of the policy was confirmed by solving the problem with a different set of realizations, data of which were not used to create the policy.

The proposed algorithmic framework, *DP in a heuristically confined state space*, is a general solution approach that can handle a much wider class of sRCPSP. For example, including decisions to cancel an on-going project [66] is an important issue in problems in which new projects arrive during the scheduling period. This feature can be incorporated into our methodology by providing a reasonable way to represent project arrivals within the state space framework. Some extensions of the proposed approach needed to handle more realistic RCPSPs were discussed in Section 4.7. Beyond sRCPSPs, the algorithm approach presented may have applicability in supply chain planning and process design[18].

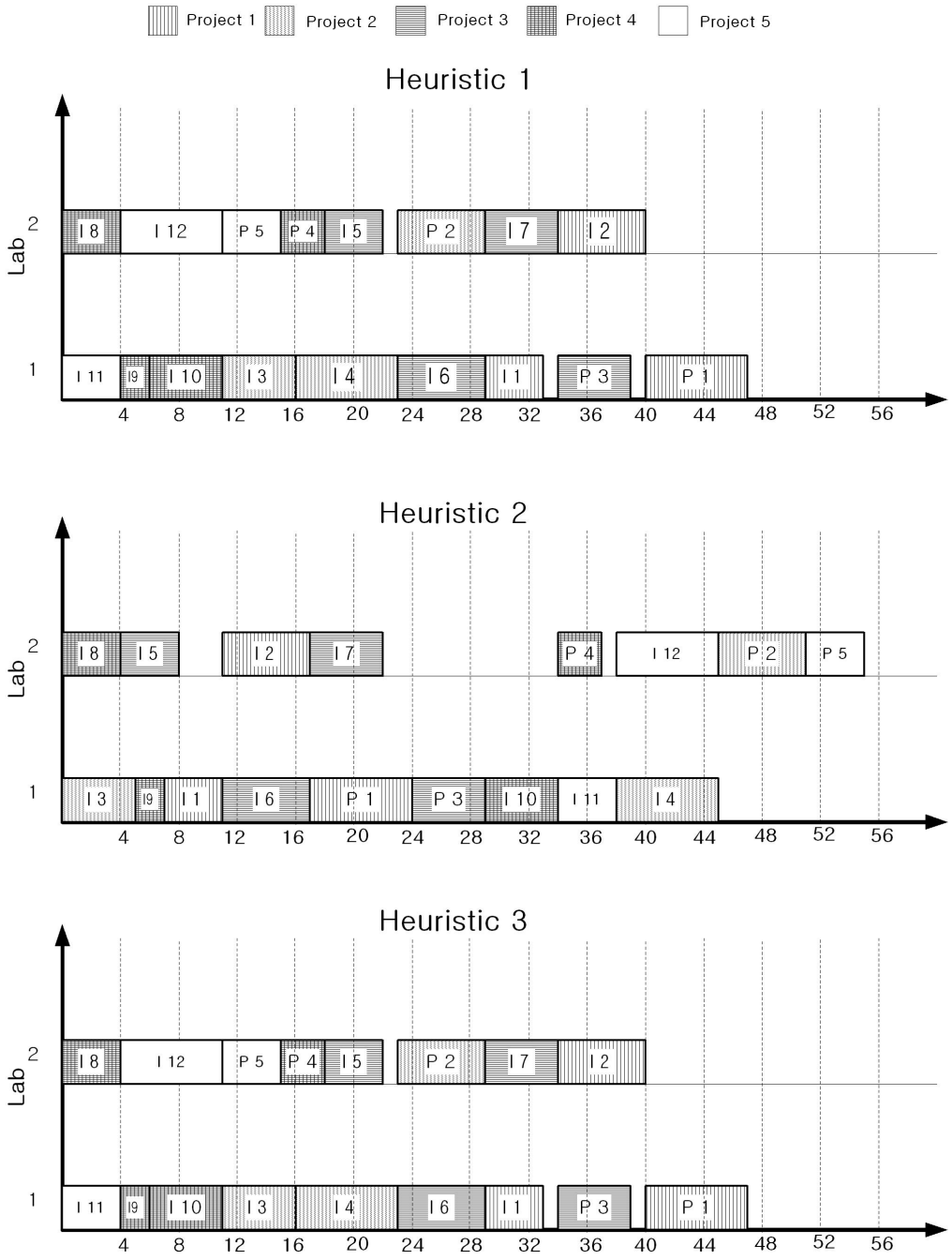

Figure 31: Gantt Charts of Heuristic Solutions for the Worst Case Realization # 3398

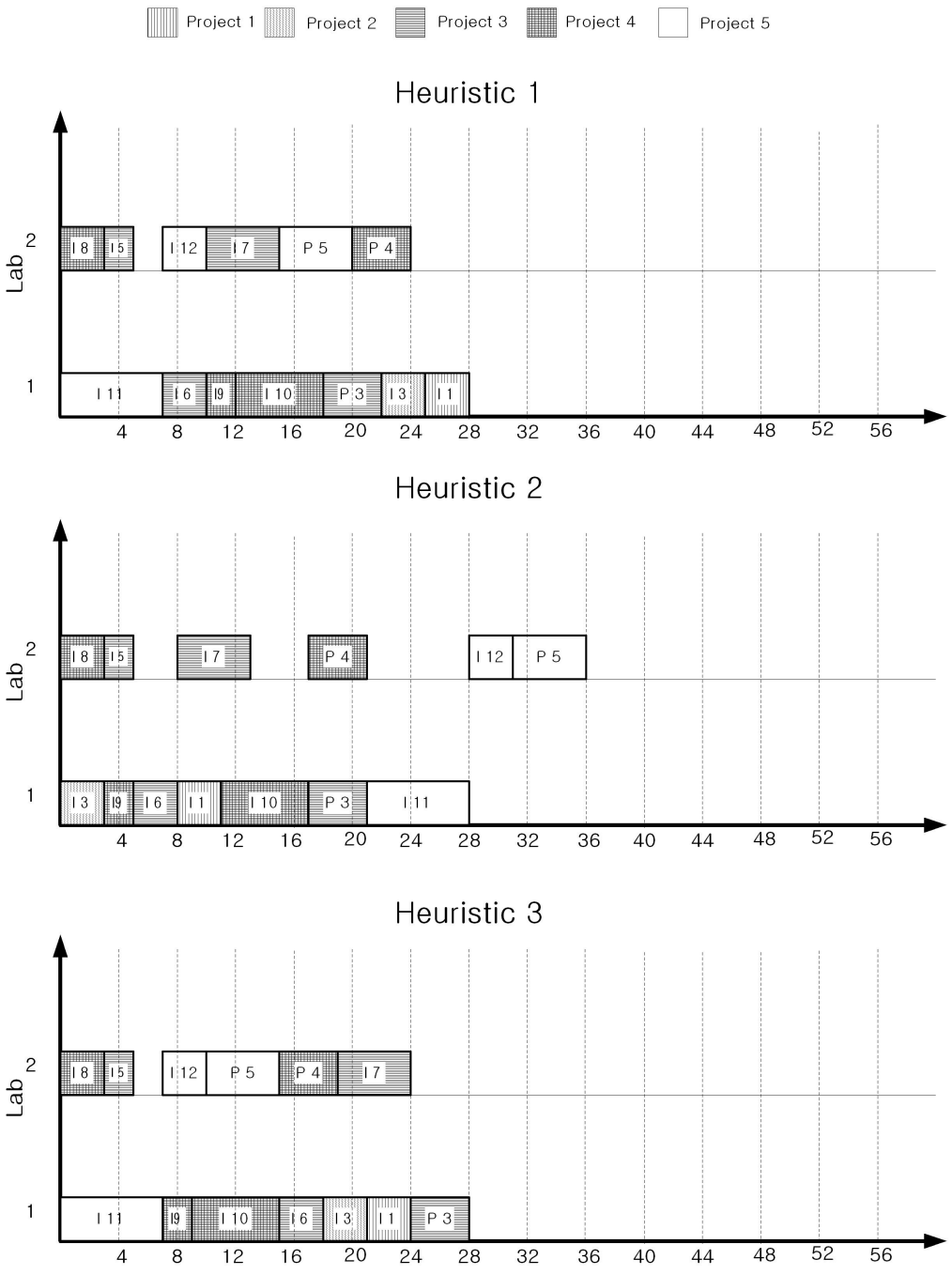

Figure 32: Gantt Charts of Heuristic Solutions for Realization # 39804

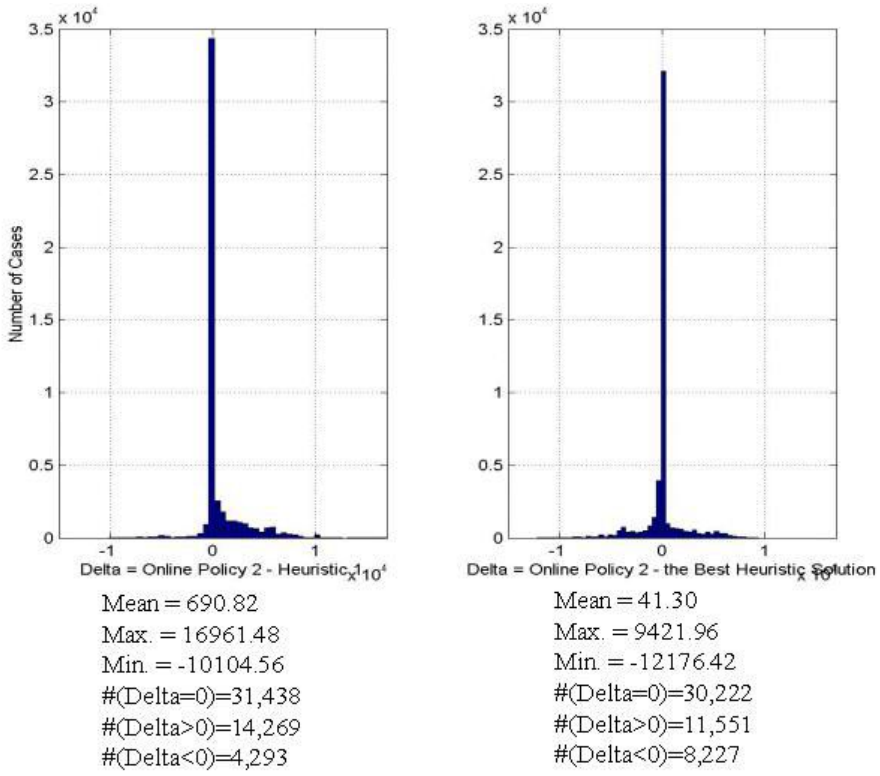

Figure 33: Evaluation of the Online Decision Making Performance for 50,000 Realizations

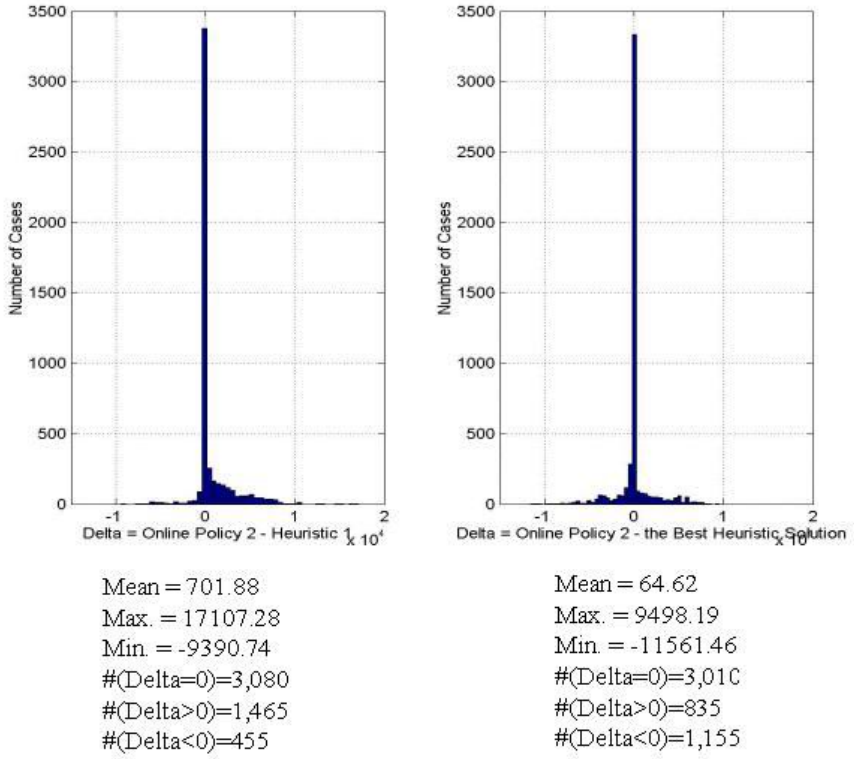

Figure 34: Evaluation of the Online Decision Making Performance for New 5,000 Realizations

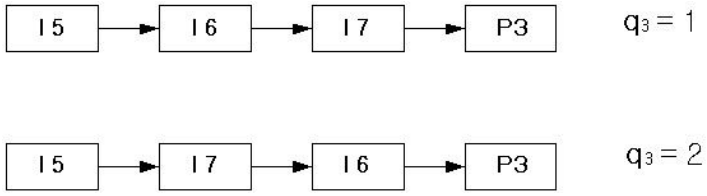

Figure 35: Two Different Task Sequences in Project 3 of the Illustrative Example

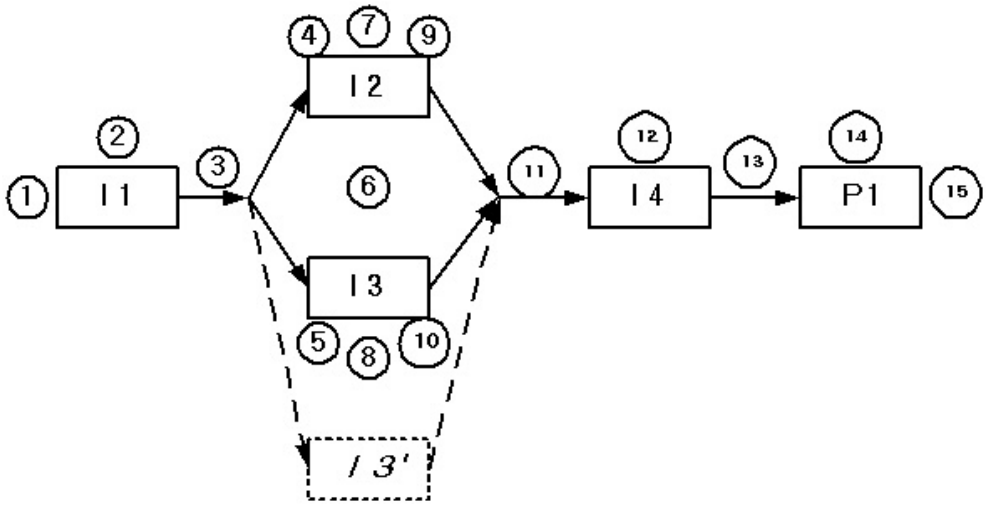

Figure 36: A Project with Branching and Merging Tasks, or Outsourcing Options

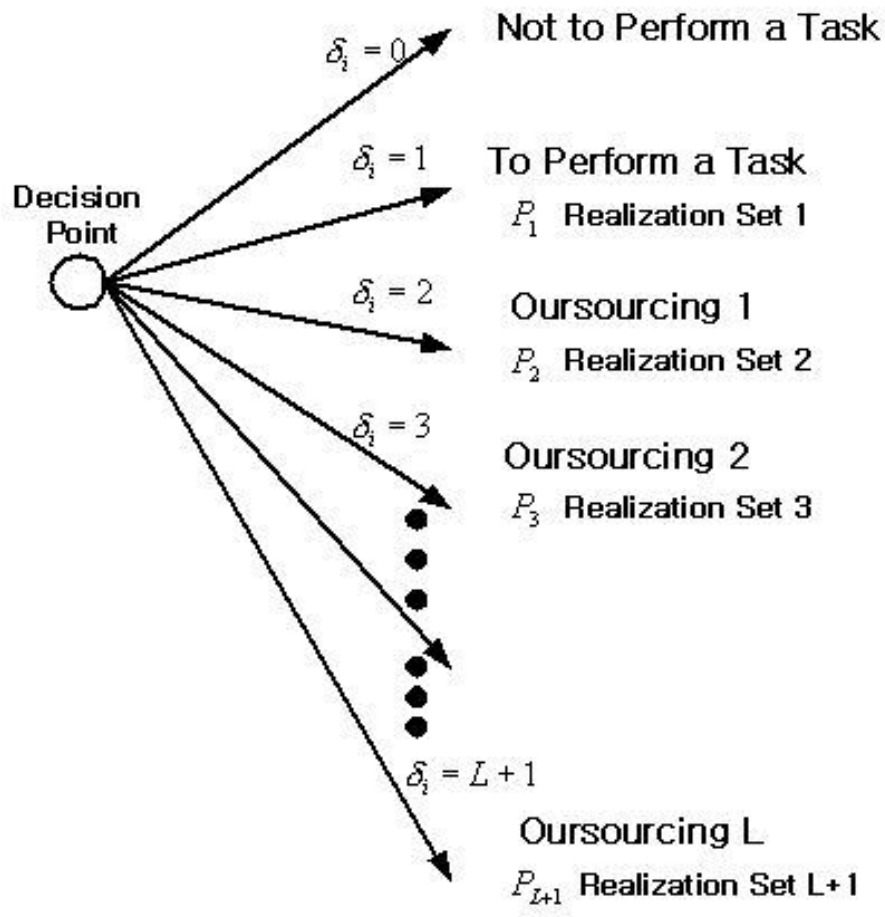

Figure 37: Outsourcing Actions

CHAPTER V

MODEL-FREE STATE TRANSITION RULES: APPLICATION TO STOCHASTIC RESOURCE CONSTRAINED PROJECT SCHEDULING PROBLEMS WITH NEW PROJECT ARRIVALS

5.1 Introduction

In many Resource-Constrained Project Scheduling Problems(RCPSP), the set of projects to be performed is dynamic. For example, while performing the projects according to a certain decision policy, a new project can emerge. To make an appropriate decision for the problem with dynamic project arrivals, project cancellation decisions [66] have to complement the conventional scheduling decisions.

In this study, a stochastic RCPSP(SRCPSP) with dynamic project arrivals is addressed with an appropriate project cancellation strategy. The proposed solution strategy is based on the simulation-based Dynamic Programming(DP) approach, which we have developed and applied to small SRCPSPs in our previous work [24, 22]. From an algorithmic standpoint, the approach is modified to handle an extended problem structure, dynamic project arrivals and expected profit changes. However our previous work has limitations in handling complicated SRCPSP. The analytic calculation of the all possible state transition probabilities is not practically feasible for a large size problem due to complex interactions among states, actions, and uncertain parameters. The bottleneck is overcome by developing an appropriate Q-Learning algorithm [86, 79, 8], which can be used when a model of the system is unavailable, for the problem. The Q-Learning algorithm can be viewed as a simultaneous identification of the probabilistic state transition rule and the Q-Values function. Hence, it removes the need to perform the analytical calculation of the transition rule, which can be painstakingly tedious. Stochastic simulation under certain suboptimal policies (heuristics)

is used to obtain the numerical state transition probabilities as well as the subset of states, which are defined as combinations of the conventional states and the corresponding actions, and initial cost-to-go values for the value (Q-Value) iteration.

In next section, we present a SRCPSP with new project arrivals and its complicated decision problem structure. Then, a generalized Q-Learning algorithm for the problem is discussed with appropriate definitions of state, action, state transition rules, and objective function. Finally, the proposed approach is verified by solving a SRCPSP with dynamic project arrivals and billions of scenarios.

5.2 Problem Description: Stochastic RCPSP with New Project Arrivals

We consider a RCPSP with M projects, each of which consists of m_i tasks, for $i = 1, ..., M$. There are N resources (laboratories), a specific resource has to be used to perform each task. On top of the basic structure of the RCPSP, there are L potential projects that can randomly emerge while performing tasks in the initially given M projects. A 'new project arrival' changes decision structure of the problem dramatically because of various types of decisions, such as cancelling on-going project or idling available resource for future usage, can be made to improve the overall profit upon the 'new project arrival'. Arrivals of the L potential projects are governed by arrival time distributions and their realization probabilities. Major problem parameters of a task, the result (success or failure), the duration, and the cost, are uncertain. The uncertainty is modelled by (underlying) discrete time Markov chain to represent correlation among uncertain parameters as we introduced earlier. A time-varying nonlinear reward function is given for each project to represent the decreasing value of the project with time (see Section4.2). For the new project candidates, the reward function starts at the time of corresponding project arrival.

5.3 Q-Learning for the Stochastic RCPSP

Our previous work on the RCPSP[24] was successful and gave a prototype stochastic dynamic programming formulation for the problem. Furthermore, we also developed an appropriate algorithmic framework to circumvent infamous 'curse of dimensionality' of conventional dynamic programming. However, application of the algorithmic framework for larger sizes of the RCPSP, with more complicated structure, is still limited due to heavy computational load of analytical state transition rules, in which all possible next states and their conditional probabilities of realizations are calculated. From a programming perspective, the analytical state transition rules are awkward to apply since the state, represented as integer to indicate a certain 'status' of a project, transition rules imply many logical constraints for the exact calculation of the every possible next state. Especially, with the new project arrival, the analytical state transition becomes much more complicated with the various types of decisions that can be made. We are motivated to develop more powerful solution method, Q-Learning approach, a model-free simulation-based optimization algorithm for the given problem because of the limitation of our previous work. The overall procedure of applying the Q-Learning algorithm for the given problem is similar to the algorithmic framework developed in our previous work[24] as shown in following Figure 38.

The objective of the heuristic simulation is to explore the system under a large number

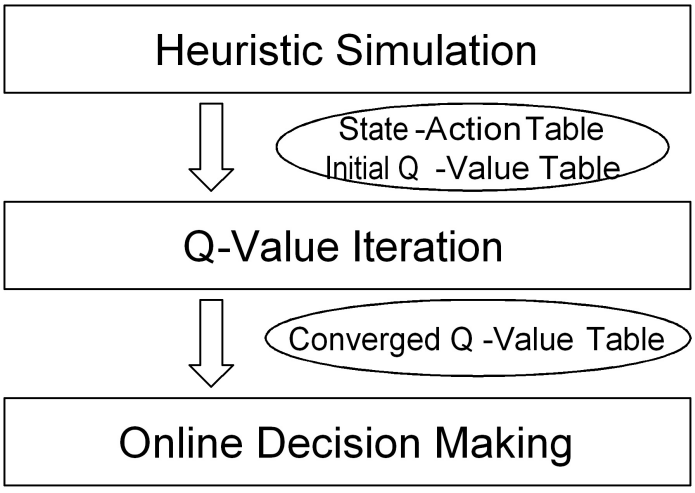

Figure 38: Q-Learning Approach

of realizations. As a result of the simulation, an initial Q-Value table is obtained as a function of state-action pair. At the beginning of the simulation, the state-action table is empty set and new state-action pairs are added as the simulation goes on. The heuristic simulation also can be viewed as an empirical model building process because it extends a coverage of the model for new state-action pairs as well as refines current Q-Value in the table for revisited state-action pairs. Since heuristic policies are applied in the simulation, the initial Q-Value table is not optimal. A generalized Q-Value iteration equation is shown in equation (72) and the equation is specialized for the given problem with fixed 'forgetting factor', $\gamma = 1$, as in equation (73). The 'forgetting factor' γ represents relative ratio of 'previous information', previous Q-Value, to 'new information', current Q-Value in calculating new Q-Value so that the Q-Value can be updated while exploring the system. However, in this work, we propose to set $\gamma = 1$ because simultaneous exploration and updating of the Q-Value is meaningless for the given problem due to significant stochastic complexity of the problem. In other words, any Q-Value is not reliable until the simulation covers a certain amount of states and it is why we propose to perform the Q-Value iteration after completion of the heuristic simulation as shown in the Figure 38. The initial Q-Value is iterated over the restricted state-action space built in the simulation stage and the Q-Value eventually converges.

$$Q(x(k), u(k)) = (1-\gamma)Q(x(k), u(k)) + \gamma\{g(x(k), x(k+1), u(k)) + \alpha \max_{u(k+1)\in U^{x(k+1)}} E[Q(x(k+1), u(k+1))]\} \tag{72}$$

$$Q(x(k), u(k)) = E\{g(x(k), u(k)) + \alpha \max_{u(k+1)\in U^{x(k+1)}} Q(x(k+1), u(k+1))\} \tag{73}$$

Then, the converged Q-Value table is utilized for the online decision making with equation (74).

$$u^*(k) = \arg \max_{u(k)\in U^{x(k)}} Q(x(k), u(k)) \tag{74}$$

The major difference between the stochastic DP based algorithmic framework and the Q-Learning approach is the 'state-action pairs' which are recorded during learning stage, which

is the stochastic simulation stage with suboptimal policies (heuristics). Any state visited in the simulation is recorded with the action taken at the state as well as resulting next state of the state and its state transition frequency. Since the simulation is performed over many realizations, the state transition frequency from a state to another state as a result of the action approximates the conditional probability of the corresponding state transition. Different state transitions from the same state with same action is due to the stochastic realization governed by the underlying Markov chains. Thus, in the Q-Learning approach, the objective of simulation is not only to obtain relevant states (or state-action pairs) but also to explore the system and identify empirical state transition rules. Another major difference of the algorithmic framework and the Q-Learning approach is the objective function. Instead of the cost-to-go value which is a function of the state, a Q-value, which is a function of the state and action is calculated in the Q-Learning approach. As a result, the Q-Learning approach requires more memory and computation in its iteration stage than the algorithmic framework because the iteration has to be done over every state-action pair instead of every state. However, this apparent computational drawback can be reconciled by its empirical state transition rule, which is computationally much more efficient than the analytic state transition rule, built in the simulation stage of the algorithm. Figure 39 illustrates the conceptual diagram of the state-action pair and the Q-value representation.

The state transition probabilities, $P1, P2, ..., Pn$, in Figure 39 are empirical conditional probabilities obtained vis simulation. Suppose that for a state, $x(k)$, an action, $u(k)$, was taken N times in the simulation and the state transition frequency from state $x(k)$ to state $x_i(k+1)$ as a result of action $u(k)$ is N_i for $i = 1, 2, ..., n$. From the definitions of N and N_i, it is obvious that $\sum_i N_i = N$ and $Pi = \frac{N_i}{N}$.

Due to similarity of the stochastic DP and the Q-Learning approach, all the necessary elements, state, action, and state transition rules, the mathematical formulation of the Q-Learning approach, are the same as those of the stochastic DP. Detailed definitions of the state, action, and state transition rules of the Q-Learning algorithm for the given problem are discussed in following sections. It should be noted that all following definitions are directly extension of the stochastic DP formulation developed in our previous work[24].

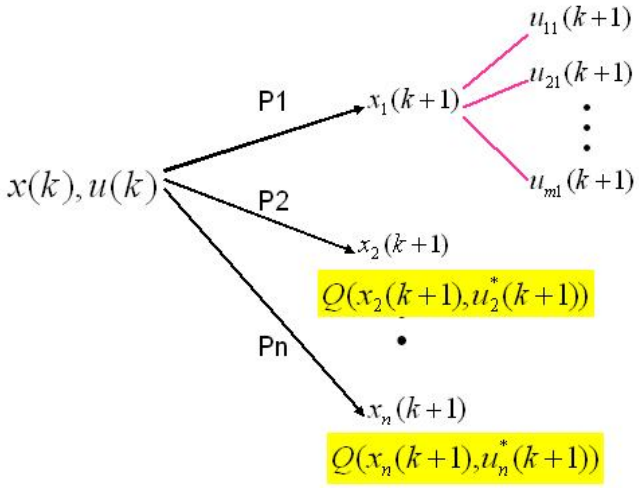

Figure 39: State-Action Pair and Q-Value

5.3.1 Definition of State

In defining the state of a system, it is important to adopt as parsimonious a state representation as possible because any redundancy will increase the computational complexity. For a RCPSP with M projects and R potential projects that may emerge in the future, the state is defined as following with L types of available resources (laboratories).

$$X = [s_1, s_2, ..., s_M, s_1, ..., s_R, z_1, z_2, ..., z_M, z_1, ..., z_R, L1, L2, ..., L_L, a_1, a_2, ..., a_R, t]^T \quad (75)$$

In (75), s_i for $i = 1, 2, ..., M$ and $r = 1, 2, ..., R$ represents the current status of project i, containing the information of which tasks are finished and which task is on-going for project i. Because each project consists of a finite number of tasks, s_i can be represented as an integer variable. For example, there can be 7 possible state (circled number) in a project with 3 tasks as illustrated in Figure 40.

z_i for $i = 1, 2, ..., M + R$ represents the information state of project i, which indicates

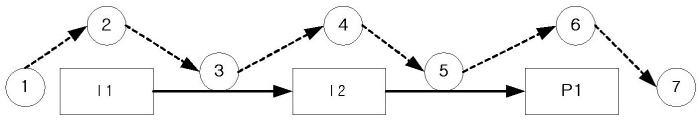

Figure 40: Possible project status of a project with three states

the result of the most recent task in the project. As explained in the problem description,

the parameters (i.e. the duration, cost, and result ('success' or 'failure')) of each task are realized according to the conditional probabilities in the corresponding Markov chain. Once a task in project i is completed, z_i is updated according to the realized result of the task. z_i is an integer variable ranging from 1 to r_{ni} where r_{ni} is the number of possible realizations for the nth task (the most recently completed and realized) in project i. The third set of state variables, L_j for $j = 1, 2, .., N$ represents the time that the resource has been used for the on-going task. And $L_j = 0$ indicates that the resource is idle. The next set of state variables, a_r for $r = 1, 2, ..., R$ represents the realized arrival time of project r. Finally, time t is added as a state variable in order to consider the time-varying value of the reward of each project.

5.3.2 Actions

With the state defined as in equation (75), the action, U, can be defined as in following equation (76).

$$U = [\delta_1, \delta_2, ..., \delta_M]^T \tag{76}$$

δ_i is an integer variable which represents whether to perform a task ($\delta = 1$) or not to perform a task ($\delta = 0$) or to cancel a task ($\delta = 2$) of the project i. The decision can be made only when an necessary resource for the task is available, that is, $\exists L_j = 0$ for some $j = 1, 2, ..., L$.

5.3.3 State Transition Rules

In the Q-Learning approach, the state transition rules are much simpler than the rules in our previous work[24] since complicated analytical calculation of state transition probabilities are not required. The Q-Value iteration and the online decision making stages do not require any state transition rules since the state transitions rules are already imposed in the Q-Value table obtained in the heuristic simulation stage. However, simple state transition rules have to be defined in the heuristic simulation stage to explore the system for many scenarios. According to the definition of the state (Section 5.3.1), there is only one initial

state at time $t = 0$.

$$X(0) = \left[\underbrace{1, \cdots, 1}_{\text{M Initial Projects}}, \overbrace{0, \cdots, 0}^{\text{R Potential Projects}}, \overbrace{\underbrace{0, \cdots, 0}}^{\text{M+R Information SV}}, \overbrace{0, \cdots, 0}^{\text{L Types of Resource}}, \underbrace{0, \cdots, 0}_{\text{R Arrival Time}}, \overbrace{0}^{\text{Time}} \right]^T \quad (77)$$

The initial state evolves with actions taken by heuristics and realizations of uncertain parameters until it reaches a terminal state. One characteristic of the RCPSP is that the task network of the problem is not deterministic due to uncertain outcomes (success or failure) of the tasks in the problem. Even though there is a unique initial state, the problem can end with one of numerous terminal states according to the realization of uncertainty. The number of possible terminal states depends on the stochastic complexity of the problem. We define the terminal state as the state where all the projects are terminated. The termination condition for each project is defined either by the successful completion of the final task or the failure of an intermediate task.

5.3.4 Objective Function: Q-Value

The objective of the RCPSP is the maximization of the final reward after finishing all the projects. The Q-Value iteration equation (73) also represents definition of the Q-Value as a recursive addition of one-stage profit function $g(x(k), u(k))$ so that it naturally reflects final reward. The one-stage profit $g(x(k), u(k))$ is a summation of cost incurred by an action, $u(k)$ and reward(profit) retrieved after successful completion of projects at the state, $x(k)$. Since the Q-Value table is expanded by the heuristic simulation, the Q-Value at the initial state with the optimal action, $Q(x(0), u^*(0))$, represents expected final reward of the online policy.

5.4 Suboptimal Policies

Defining or inventing suboptimal policies in Q-Learning is very important since it affects resulting model free state transition rules as well as quality of the final solution. In this section, three greedy heuristics, which utilize information from the state as defined in 5.3.1, are developed for the SRCPSP. These heuristics emphasize different information about the problem and hence combining them together could lead to a better overall performance.

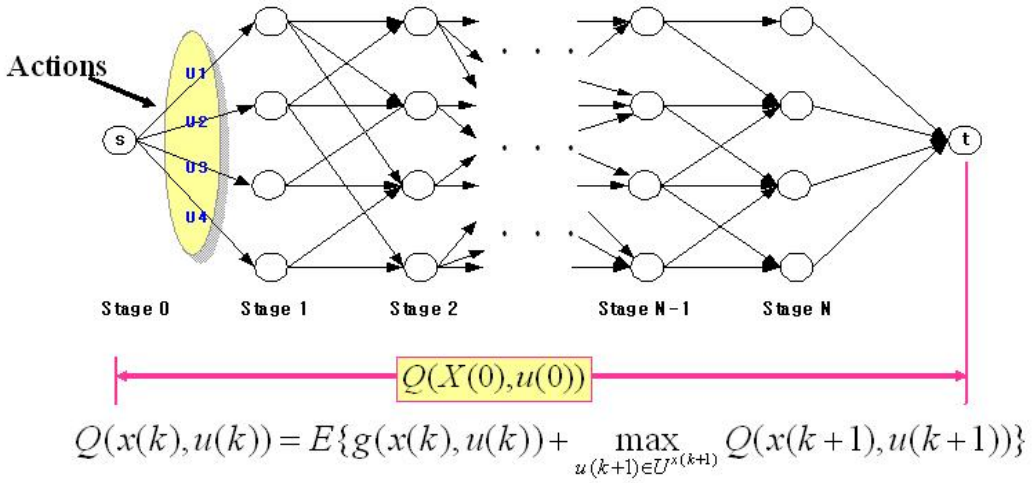

Figure 41: Definition of Q-Value

On top of the greedy heuristics, special types of actions (cancellation and idling), which cannot be taken by the heuristics, are randomly added.

5.4.1 Greedy Heuristics

5.4.1.1 Heuristic 1: high success probability task first

In the SRCPSP, the result (success or failure) of a task is a very important factor affecting the final reward, as well as the remaining part of the scheduling solution. Heuristic 1 is developed for maximizing the probability of the sucecess of the next allocated task. For the decision, the expected success probability of each task is calculated according to the current information state, $z(k)$ for $i = 1, 2, ..., M$. Once we know $z_i(k)$ for each task, we can also obtain the corresponding conditional probability for each outcome. The expected success probability of the task can be calculated by summing the probability of each successful outcome. This heuristic can be modified for the case of multiple level of success by assigning appropriate weighting factors for the different levels of success.

5.4.1.2 Heuristic 2: short duration task first

Another way to increase the final reward of the projects in the SRCPSP is to finish the projects as quickly as possible in order to minimize the loss of reward with time. Heuristic 2 considers the time value of the project in a greedy way by performing a task with

shortest expected duration first in cases of resource conflicts. The expected duration can be calculated by utilizing current information state, $z_i(k)$, as in the calculation of expected success.

5.4.1.3 Heuristic 3: high reward project first

Heuristic 3 gives priority to the impending task of the project which has the highest potential reward. This is a very greedy decision to get the highest reward in the shortest time with the smallest reward decrease. Heuristic 3 may work well if the project with the highest reward is completed successfully. Its drawback is the other projects can be delayed too long. Therefore, if a project with the highest reward fails, the total reward can be decreased significantly. In the R&D pipeline management problem, the priority of each project is decided in the order of initial reward value. The decision making procedure is straightforward, all the tasks in the project with the highest reward are performed first and so on. If there is an idle resource after assigning a pending task in the target project, the resource is used to perform a task in the next priority project.

5.4.2 Random Perturbation

The SRCPSP addressed in section 5.2 includes new project arrivals that can be realized while some initial projects are on-going. To maximize total reward for the new project arrivals, one may reserve resources (available laboratories) for the potential new project instead of utilizing them for currently on-going projects that may not be profitable. Furthermore, complete cancellation of currently on-going projects also has to be considered to allocate more resources for profitable new projects. Those 'idling' and 'cancellation' actions are not considered in the three heuristics because it cannot easily accommodate them due to their inherently greedy nature. To utilize benefits of the various actions, the cancellation and idling actions are added randomly in the heuristic simulation. The random actions are chosen with small probability to avoid significant perturbations that cause the overall reward to deteriorate significantly.

5.5 Illustrative Example

As an illustrative example of the RCPSP, we consider a R&D pipeline problem that has 3 initially given projects and 2 new project candidates. The activty-on-node (AoN) graph of the example is shown in Figure 27. The AoN displays the sequence of tasks involved in each project with the resources required to complete the tasks (e.g. task 'I_1' has to be performed in laboratory 1 (Lab.1)). A parenthesized number over each task represents the possible number of outcomes (in terms of duration, cost, and result and the multiple levels of success or failure) of the task. A Markov chain is given for each project to represent correlations among the outcomes of adjacent tasks in a project. For example, for task I_2, which has three possible outcomes, a 3×1 probability vector is given to represent a probability of the three possible outcomes. The conditional probabilities for the outcomes of the task P_1 are assigned based on the realized outcomes of I_2. Since both of I_2 and P_1 have three possible realizations, the size of the probability matrix of P_1 is 3×3. Each column represents the conditional probability vector for the possible outcomes of P_1. All the probabilities and parameters of the example are summarized in Table 13.

R_i, for $i = 1, ..., 5$, indicates the initial reward of project i at time $k = 0$. After time $k = 0$,

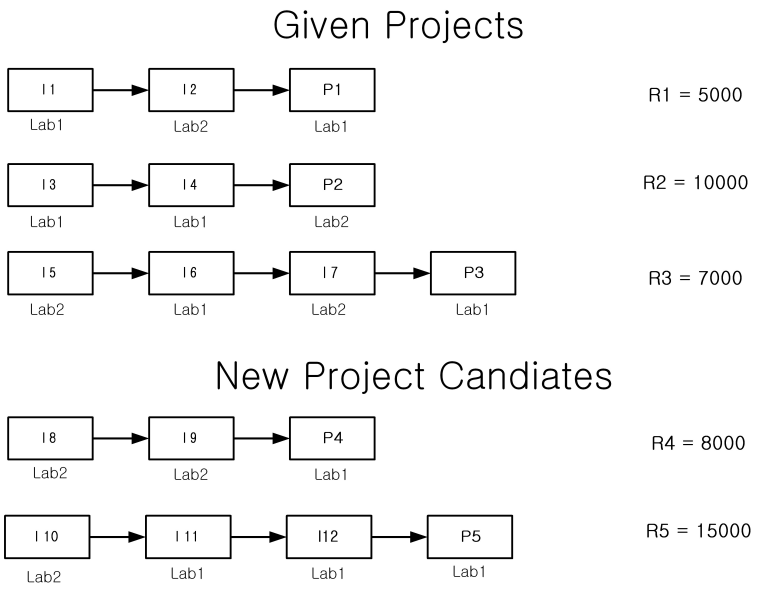

Figure 42: RCPSP example

the rewards of three initial projects decrease as shown in Figure 43. Those reward profiles represent the "time value" of each project due to competitive market situation. If a project is delayed for too long (longer than PD_i, 'project deadline'), the reward from completing the project can be insignificant because similar drugs (products) developed by competitors may have taken a large market share. Reward profiles of the potential project candidates are introduced at the time of their arrival. Figure 44 shows one of possible realizations of the reward profiles of this example in which both of potential project 4 and 5 arrive at time $t = 10$. Probabilities of the new project arrivals are summarized in Table 14 which implies 16 new project arrival scenarios including no project arrival case.

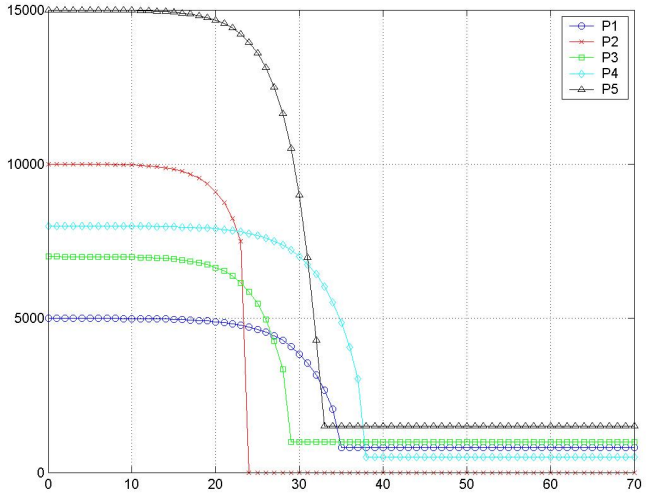

Figure 43: Reward profile of the projects in the illustrative example

- **Stochastic Complexity of the Example**

  The illustrative example is a small size RCPSP, which consists of only 5 projects including 3 initial projects and 2 potential new project candidates. However, it is actually a large size problem due to its stochastic complexity. One measure of the stochastic complexity of the problem is the total number of possible scenarios under different parameters realizations. Figure 45 shows project 1 in the illustrative example and its realization data. According to the realization data, there are 6 scenarios, $3 \times 2 \times 1$(3 realizations in $P1$, 2 realizations in $I2$ linked to $P1$ and 1 realizations in $I1$

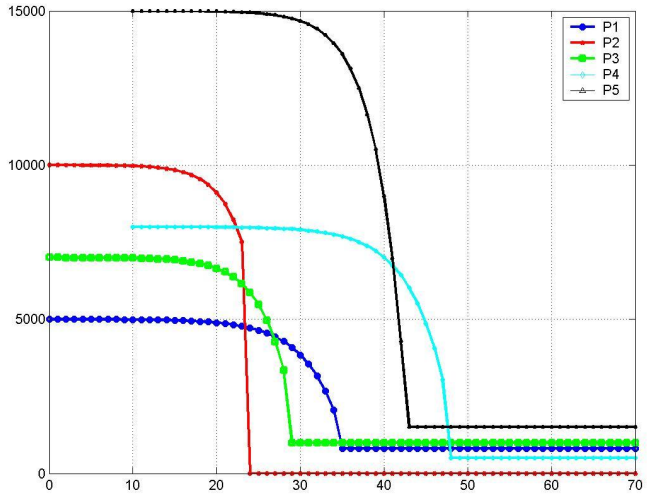

Figure 44: A realized reward profiles in the illustrative example

linked to $I2$), in case of project termination with $P1$ completion, i.e. all three tasks in the project are completed. In same way, in case of project termination with $I2$ and $I1$ completions, there is 1 scenario respectively. Thus, the total number of scenarios of the project 1 is 8.

In summary, with given realization data in Table 13, total number of scenarios of

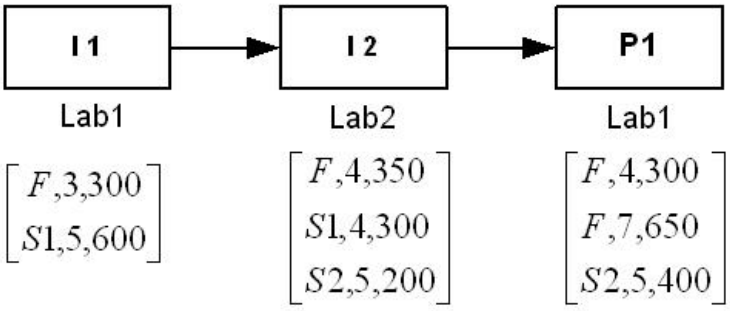

Figure 45: Project 1 in the Illustrative Example

the projects 2,3,4 and 5 is 7, 7, 7, and 13 respectively. Furthermore, total number of new project arrival scenarios is 16. The scenarios of each project and new project arrivals are independent of each other, thus, total number of scenarios of the problem is 570,752 obtained by multiplying the number of scenarios of all projects.

Table 13: Example 1, Probabilities and Parameters

Project	Realized Result, Duration and Cost of the Task			
Project 1	I_1	I_2	P_1	
	$\begin{bmatrix} F & 3 & 300 \\ S_1 & 5 & 600 \end{bmatrix}$	$\begin{bmatrix} F & 4 & 350 \\ S_1 & 4 & 300 \\ S_2 & 5 & 200 \end{bmatrix}$	$\begin{bmatrix} F & 4 & 300 \\ F & 7 & 650 \\ S_1 & 5 & 400 \end{bmatrix}$	
	PI_1	PM_{11}	PM_{21}	
	$\begin{bmatrix} 0.30 \\ 0.70 \end{bmatrix}$	$\begin{bmatrix} 0 & 0.25 \\ 0 & 0.50 \\ 0 & 0.25 \end{bmatrix}$	$\begin{bmatrix} 0 & 0.20 & 0.08 \\ 0 & 0.30 & 0.02 \\ 0 & 0.50 & 0.90 \end{bmatrix}$	
Project 2	I_3	I_4	P_2	
	$\begin{bmatrix} F & 3 & 300 \\ S_1 & 4 & 450 \\ S_2 & 5 & 600 \end{bmatrix}$	$\begin{bmatrix} F & 5 & 700 \\ S_1 & 7 & 500 \end{bmatrix}$	$\begin{bmatrix} F & 4 & 400 \\ S_1 & 6 & 600 \end{bmatrix}$	
	PI_2	PM_{12}	PM_{22}	
	$\begin{bmatrix} 0.25 \\ 0.40 \\ 0.35 \end{bmatrix}$	$\begin{bmatrix} 0 & 0.30 & 0.05 \\ 0 & 0.70 & 0.95 \end{bmatrix}$	$\begin{bmatrix} 0 & 0.25 \\ 0 & 0.75 \end{bmatrix}$	
Project 3	I_5	I_6	I_7	P_3
	$\begin{bmatrix} F & 3 & 400 \\ S_1 & 4 & 300 \end{bmatrix}$	$\begin{bmatrix} F & 5 & 700 \\ S_1 & 7 & 400 \end{bmatrix}$	$\begin{bmatrix} F & 4 & 500 \\ S_1 & 7 & 600 \\ S_2 & 5 & 300 \end{bmatrix}$	$\begin{bmatrix} F & 5 & 250 \\ S_1 & 3 & 300 \end{bmatrix}$
	PI_3	PM_{13}	PM_{23}	PM_{33}
	$\begin{bmatrix} 0.15 \\ 0.85 \end{bmatrix}$	$\begin{bmatrix} 0 & 0.30 \\ 0 & 0.70 \end{bmatrix}$	$\begin{bmatrix} 0 & 0.25 \\ 0 & 0.60 \\ 0 & 0.15 \end{bmatrix}$	$\begin{bmatrix} 0 & 0.25 & 0.10 \\ 0 & 0.75 & 0.90 \end{bmatrix}$
Project 4	I_8	I_9	P_4	
	$\begin{bmatrix} F & 5 & 500 \\ S_1 & 7 & 450 \\ S_2 & 5 & 600 \end{bmatrix}$	$\begin{bmatrix} F & 3 & 400 \\ S_1 & 6 & 300 \end{bmatrix}$	$\begin{bmatrix} F & 2 & 800 \\ S_1 & 6 & 450 \end{bmatrix}$	
	PI_4	PM_{14}	PM_{24}	
	$\begin{bmatrix} 0.35 \\ 0.45 \\ 0.20 \end{bmatrix}$	$\begin{bmatrix} 0 & 0.25 & 0.15 \\ 0 & 0.75 & 0.85 \end{bmatrix}$	$\begin{bmatrix} 0 & 0.20 \\ 0 & 0.80 \end{bmatrix}$	
Project 5	I_{10}	I_{11}	I_{12}	P_5
	$\begin{bmatrix} F & 5 & 600 \\ S_1 & 5 & 400 \\ S_2 & 6 & 900 \end{bmatrix}$	$\begin{bmatrix} F & 5 & 700 \\ S_1 & 7 & 400 \end{bmatrix}$	$\begin{bmatrix} F & 3 & 400 \\ S_1 & 5 & 800 \\ S_2 & 6 & 950 \end{bmatrix}$	$\begin{bmatrix} F & 4 & 1000 \\ S_1 & 5 & 700 \end{bmatrix}$
	PI_5	PM_{15}	PM_{25}	PM_{35}
	$\begin{bmatrix} 0.30 \\ 0.55 \\ 0.15 \end{bmatrix}$	$\begin{bmatrix} 0 & 0.25 & 0.02 \\ 0 & 0.75 & 0.98 \end{bmatrix}$	$\begin{bmatrix} 0 & 0.20 \\ 0 & 0.65 \\ 0 & 0.15 \end{bmatrix}$	$\begin{bmatrix} 0 & 0.20 & 0.05 \\ 0 & 0.80 & 0.95 \end{bmatrix}$

5.5.1 Simulation with the three heuristics and random perturbation

The three heuristics introduced in Section 5.4 are implemented on the illustrative example. For the simulation, 30,000 uncertain parameters realizations are performed, according to the underlying Markov chains, for each heuristic. The simulation results of Heuristic 1, 2, and 3 are shown in Figure 46. The simulation results shows all heuristics can generate a large loss in the worst case realizations in which all projects progress successfully until the last task which fails. Therefore, a large cost has been incurred to perform all the tasks without any reward. The simulation results (Table 15 and 16 and Figure 46) indicate that none

Table 14: Probability of Project Appearance Time

	10	20	30	never
P4	0.3	0.5	0.1	0.1
P5	0.4	0.3	0.2	0.1

Table 15: Heuristic Simulation Results for 30,000 Realizations

Total Profit	Heuristic 1	Heuristic 2	Heuristic 3
Mean	5914.00	3967.85	7276.98
Max.	30340.20	28726.64	32097.61
Min.	-9000	-9100	-8750

of the heuristics is uniformly superior. The relative performance of the different heuristics vary by realization.

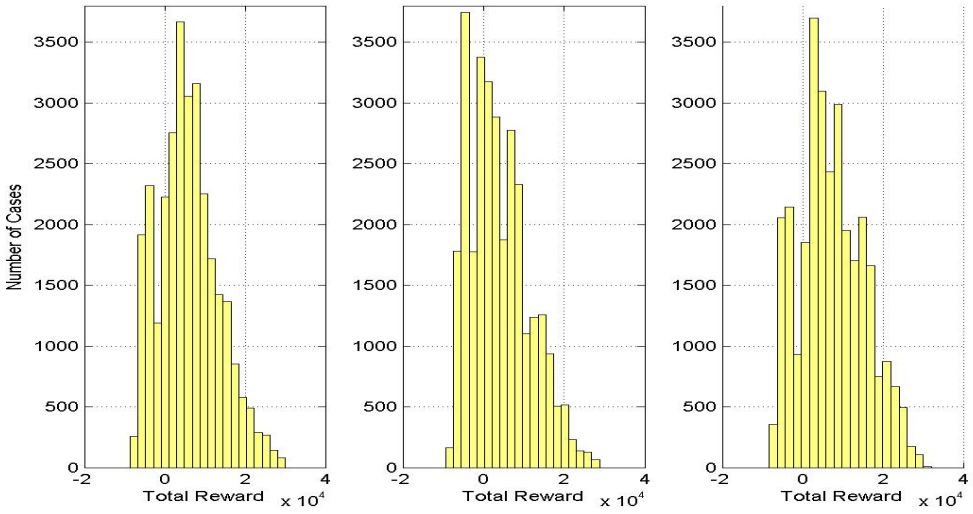

Figure 46: Simulation Results of the Three Heuristics for 30,000 Realizations

Since the heuristics are not able to take 'unusual' actions such as 'cancelling' and 'idling', those actions are randomly mixed with the actions chosen by the heuristics during the simulation. At each decision, the idling and the cancellation decisions replace the heuristic

Table 16: Performance of the Heuristics

	H1>H2	H1>H3	H2>H1	H2>H3	H3>H1	H3>H2
# of Cases	7188	13575	6839	4942	10881	16234
Mean	6190.36	2695.00	3662.54	2537.51	5441.92	6860.18

Table 17: Heuristic Simulation Results for 30,000 Realizations with 0.5% of idling action and 1% of cancellation action

Total Profit	Heuristic 1	Heuristic 2	Heuristic 3
Mean	5932.52	3927.43	7270.81
Max.	30340.20	28733.22	32050.05
Min.	-9000	-9100	-8750

decisions with 0.5% and 1.0% of probabilities respectively. The cancellation decision is applied to on-going projects and cancellation of a project is considered as failure of the project with zero action cost and zero reward. The idling decision for available resource is also considered as 'cost-free' action and the idling action is continued until next event(state). Table 17 shows performance of the heuristics with randomly introduced 'cancelling' and 'idling' actions for the same set of 30,000 realizations. Since the actions are introduced with small probabilities, overall performance of the heuristics is similar to the one (Table 15) without random actions.

The simulation is performed over 10 sets of 30,000 realizations, the three heuristics are applied with randomly mixed 'cancellation' and 'idling' actions.

5.5.2 Implementation of the DP in heuristically restricted state space

The state of the illustrative example consists of 15 state variables according to the state definition in 5.3.1 as shown in equation (78).

$$X = [s_1, s_2, s_3, s_4, s_5, z_1, z_2, z_3, z_4, z_5, L_1, L_2, t, p4t, p5t] \tag{78}$$

The calculation of the total state space size is complex due to the inability of certain combinations of completed tasks and event times to be realized. It is expected that the duration of the whole schedule will be about 40 time units, based on estimates of the longest task duration and an idea of possible task parameter sets, approximately 950 million states could be experienced.

5.5.2.1 State-Action pairs

As a result of the simulation, 263,053 non-redundant state-action pairs are obtained. Each of the 263,053 state has a Q-Value representing expected total reward from the current state to the terminal states. Among the 263,053 state-action pairs, 29,599 states have 'no action' to choose because they are identified as terminal states.

5.5.2.2 Q-Value Iteration

For the 263,053 state-action pairs, the Q-Value iteration, Equation (73), is performed with the initial Q-Values obtained in previous step. For a given state-action pair, the Q-Value iteration equation finds an optimal action for potential next state. The iteration scheme converged within an error tolerance $||(Q^{i+1} - Q^i)/Q^i||$ after the 21st iteration and took 3.1 hours for each iteration.

5.5.2.3 Improved Solution: Online Decision Making

The decision policy obtained by the proposed approach is represented by the converged Q-Values and the online decision making equation (74). Thus, after the converged Q-Values are obtained, we can make a valid decision for any realization generated by the underlying Markov chain model. In the online decision making, the future results of the currently on-going tasks are not know at the point of decision. The transition from current state is a consequence of both the decision made according to equation (74), and the parameter values chosen by the random process.

5.5.3 Computational Results

To verify the performance of the policy obtained by the proposed approach, it is compared to the heuristic solutions for the 30,000 realization used to synthesize the policy. The results shown in Table 18 indicates that the proposed approach improves the mean performance by about 39.13% compared to the best heuristic policy, the Heuristic #3. This significant improvement can be explained by the appropriate 'cancelling' and 'idling' decisions made by the policy. Although those actions are randomly mixed with the heuristics during the simulation, some of those actions are chosen appropriately to maximize total reward in the

Table 18: Heuristic Simulation Results for 30,000 Realizations with 0.5% of idling action and 1% of cancellation action Vs Online Decision Making with Q-Value

Total Profit	H1	H2	H3	Online
Mean	5914.00	3967.85	7276.98	10124.63
Max.	30340.20	28726.64	32097.61	30385.45
Min.	-9000	-9100	-8750	-8050

Table 19: Heuristic Simulation Results for 10,000 New Realizations with 0.5% of idling action and 1% of cancellation action Vs Online Decision Making with Q-Value

Total Profit	H1	H2	H3	Online
Mean	6054.31	4021.65	7339.34	10321.67
Max.	29874.99	28762.29	29974.99	30856.05
Min.	-8600	-8800	-8350	-7450

Q-Value iteration. The 'cancellation' and 'idling' actions are mainly chosen to prevent the 'worst' case in which a negative total cost is expected due to major project failure. The results in Table 85 shows that the minimum reward, the worst case, is increased to -8050. On the other hand, the maximum reward of the online policy is in same ranges as those of the heuristics. Hence, the significant improvement of the mean value is mainly due to reducing the worst case results (loss) with appropriate use of the cancellation or the idling actions.

The stochastic complexity of the problem is very high with 642,096 scenarios. The policy should be robust for any of these scenarios even though they were not seen during its creation. To demonstrate the robustness of the policy, it is tested for 10,000 realizations that were not part of the training set. The computational results summarized in Table 19 and Figure 47 shows the robustness of the proposed approach in this example. The Figure 47 shows an obvious shift of negative reward cases in the positive direction.

Figure 48 shows how the online policy can improve the total reward dramatically for a certain realization, realization #8863, among the 10,000 realizations used for the policy evaluation. In realization #8863, two of the three initial projects, project 1 and project 3,

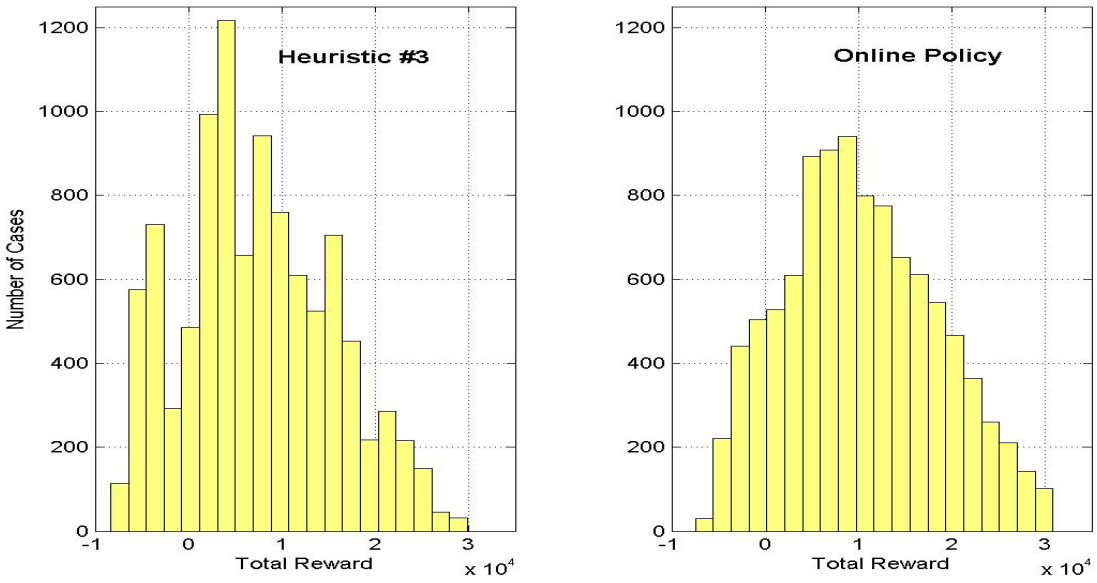

Figure 47: Evaluation of the online decision making performance for a new set of 10,000 realizations

turn out to fail in their second tasks. Meanwhile, both of the potential project candidates, project 4 and project 5, arrive at time 10 and both are successfully completed. The best heuristic, Heuristic #3, allocates resource to the project 1 and project 3 until they are completed with failure. However, the online policy cancels project 3 after its successful completion of the first task. Furthermore, after the new projects arrive at $t = 10$, it allocates resource to the new project and cancels the project 1. All these decisions are made by the online decision policy based on the Q-Value calculation, equation (74 and coupled with high level success of the first task of the project 2. (note that realization result '3' in the first task of project 3 indicates high level success of the task.). As a result of appropriate uses of the cancellation actions for less profitable projects, the online policy can boost the final reward up to 20588.36, which is more than a 60% improvement compared to the best heuristic result, 12752.99.

5.6 Conclusion

A stochastic resource-constrained project scheduling problem (sRCPSP) has been addressed by using Markov chains to model key uncertainties (the duration, cost, and results of a task). On top of the basic problem structure of the sRCPSP, a practical feature of the problem, new project arrival, is added to present realistic cases. To solve the problem, a Q-Learning approach has been developed with appropriate definitions of state, including the information state variables, and actions. The Q-Learning approach enables us to induced an empirical state transition rules from the simulation so that analytical calculations of highly complicated state transition can be avoided. The maximize advantages of using the empirical state transition rules, special types of actions, project cancellation and resource idling, that are difficult to include in randomly added in the simulation. Some of the random actions are filtered and confined during the Q-Value iteration and appropriately utilized in online decision making to maximize the total reward of the system. The proposed solution method has been tested by solving an illustrative sRCPSP with significant stochastic complexity with 642,096 scenarios. The solution obtained by the policy on average outperforms the best heuristic solution. Furthermore, by utilizing cancellation and idling actions properly, the resulting policy can reduce the worst case losses. The robustness of the policy is confirmed by solving the problem with a new set of realizations, the data of which were not used to create the policy.

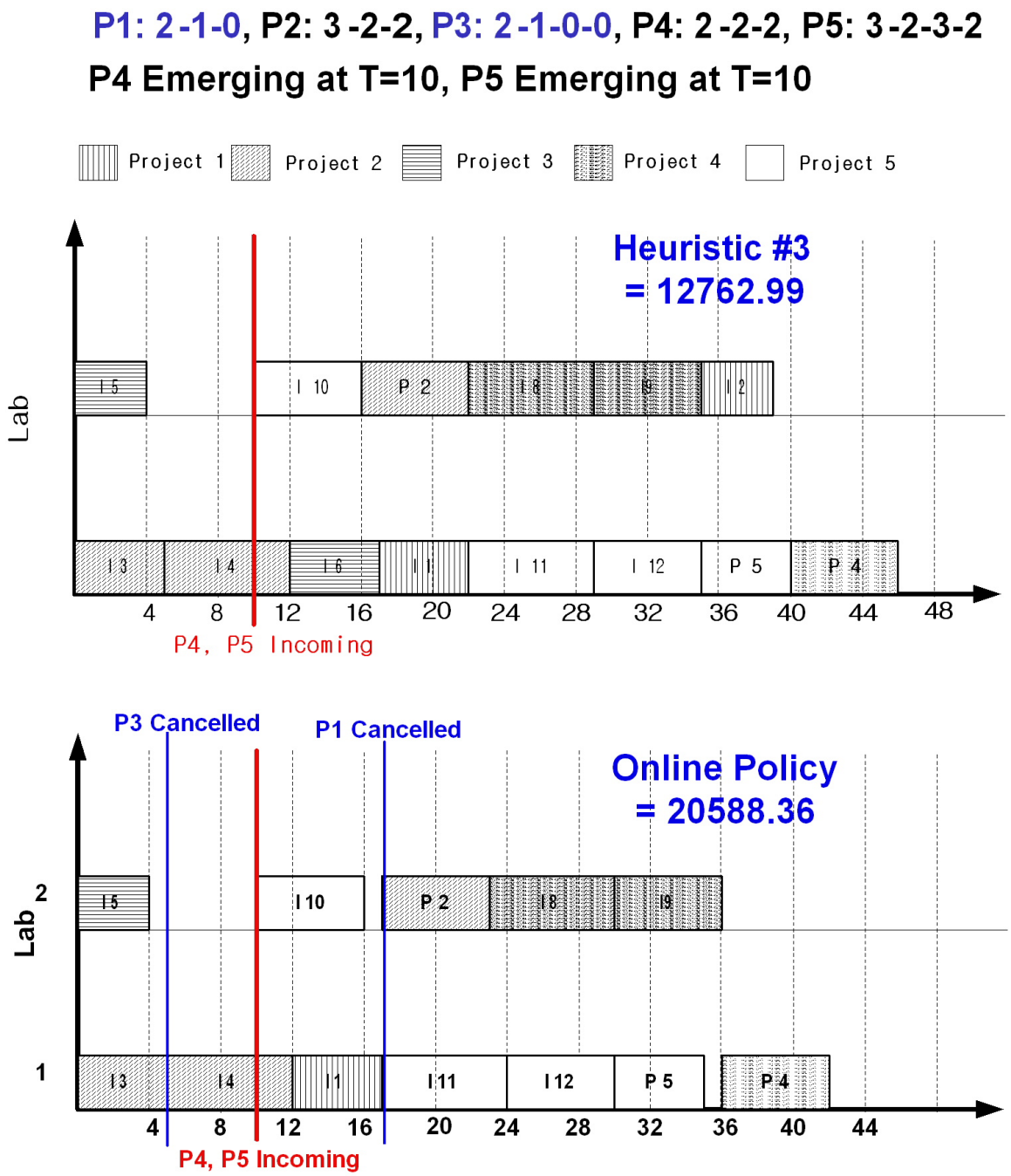

Figure 48: Gantt charts: the Heuristic #3 vs. the Online Policy for Realization #8863

CHAPTER VI

HANDLING LARGE ACTION SPACE: APPLICATION TO SUPPLY CHAIN MANAGEMENT PROBLEMS

6.1 Introduction

A significant problem for complex supply chain (SC) management is the effective handling of uncertainty in the system. The resulting SC operating policy has to be flexible enough to deal effectively with uncertain parameter variations, such as the volume and timing of market demands. Failure to account for significant product demand fluctuations by deterministic planning models may either lead to excessively high production costs (translating to high inventory charges) or unsatisfied customer demand and loss of market share. Recognition of this fact has motivated recent work aimed at studying process planning and scheduling under demand uncertainty. Most of the research on this problem has largely focused on mathematical programming approach [43, 71, 67, 80, 35, 6, 36]. The stochastic attributes of the problem are translated into an equivalent deterministic form with certain types of uncertain parameters, often normally or exponentially distributed.

On the other hand, a general supply chain can be viewed as serial and distributed inventory systems, referred to as 'multi-enchelon' inventory systems, if production facilities involved in the supply chain are simplified (i.e., assumed to be without large lead time in production). Previous research [28, 17, 83, 29] on the 'multi-enchelon' inventory systems have been focused on finding analytical optimal 'order-up-to-policies' for variants of the system under the 'balanced assumption', negative stock allocations to the retailers are possible. Although the analytical optimal policies for the multi-enchelon inventory systems are not directly applicable to the supply chain system addressed in this study, for which the balanced assumption is not valid, the 'order-up-to-policies', (s,S) policy, are adopted to generate heuristics for control (section 6.3 and 6.6.2).

This study develops a novel solution method that expands the representation of uncertainty to include a wider class of problems than addressed in the literature to date far. The solution of the SCM problem is a policy that can be interpreted as a series of decisions at each time unit. Thus, it is a "multi-stage decision making problem" with significant number of uncertain parameter realizations. Stochastic dynamic programming (DP)[7] can be used to solve this type of problem. However, stochastic DP is faced with the "curse of dimensionality", an exponential increase in the state space as the problem size increases. Hence, most of the research on the DP approach for SCM is limited to small sizes of the problem[9, 19, 38]. The size of the state space is coupled not only to the state of the supply chain but also factors in the "information states", which represent observed information regarding uncertain parameter variations. To overcome the computational intractability of the conventional DP approach, we employ an evolutionary algorithmic framework utilizing information obtained from stochastic simulation of the heuristics, which we call "DP in a heuristically restricted state space"[27, 23, 25]. This approach was applied to a stochastic resource constrained project scheduling problem (RCPSP) in [25], where the ability to address a large state space was verified by confining the original state space (with 230 billion states) and to obtain a reasonably sized confined state space(with 371,168 states). The development required to apply this to the SCM problem is taming the action space complexity. The decisions in the RCPSP and corresponding combination of the actions are much simpler than those of the SCM problem. In the SCM problem, the action space is continuous and, even though the actions can be aggregated and represented using a discrete action space, there are large number of actions for a supply chain involving multiple material flows between manufacturing sites. For example, if 10 material flows are involved in a supply chain and each material flow is discretized to 3 discrete values, the total number of possible actions at each time point is 59,049($= 3^{10}$). Thus, we cannot avoid large numbers of actions in the DP formulation of the problem. In conventional DP, a large number of actions makes the Bellman Iteration and online decision making procedures computationally intractable due to increased search space for the optimal action for each decision stage. In summary, the key contributions of this study are, first, developing an appropriate DP formulation for the

SCM problem, and second, enhancing the methodology of "DP in a heuristically confined state space" to handle the inevitable action space complexity.

The chapter is organized as follows. A SCM problem will be formulated with a Markov chain model to represent uncertain demand. An appropriate heuristic method for the given problem will be presented. Then, the conventional stochastic DP formulation will be developed as a basis for the "DP in a heuristically confined state space". A summary and further discussion will be given in the last part of the chapter.

6.2 Problem Description: SCM with Multiple Products Under Uncertain Product Demands and Prices

The prototypical process industry SCM problem addressed in this chapter has most of the essential components of a supply chain including production and inventory control decisions and intermediate product lines. It suppresses the details on logistics such as various transportation options and multiple customer(market) locations as these are often not as significant for the business-to-business component of supply chain. We consider a SCM problem with M products and the products are manufactured from p_i plants and stored in q_i inventories, for $i = 1, ..., M$. The plants that produce the product i have their own raw material inventory linked to r_i suppliers, for $i = 1, ..., M$. Thus, the supply chain is a partially connected network involved with $\sum_i p_i$ plants, $\sum_i q_i$ inventories, and $\sum_i r_i$ raw material suppliers. Connections between supply chains involved with different production/distribution lines arise due to product recipes that use raw materials to produce intermediates for other products. An illustrative example of such a supply chain is depicted in Figure 49. In this study, the focus is on the uncertainty in demand for products and that in raw material prices. This makes the allocation of intermediates to final production steps a very important decision. A detailed description of the uncertainty model is given in the next section. As stated earlier, logistic elements of the problem are kept simple by considering only one transportation option and unit transportation time for every material flow. Manufacturing time for the products are given as a multiple of time units and appear as manufacturing time delays in the plant. Other than the demands and prices of the

products, all the problem parameters (inventory costs, manufacturing costs, manufacturing time, and default setup cost for one batch of production) take known deterministic values.

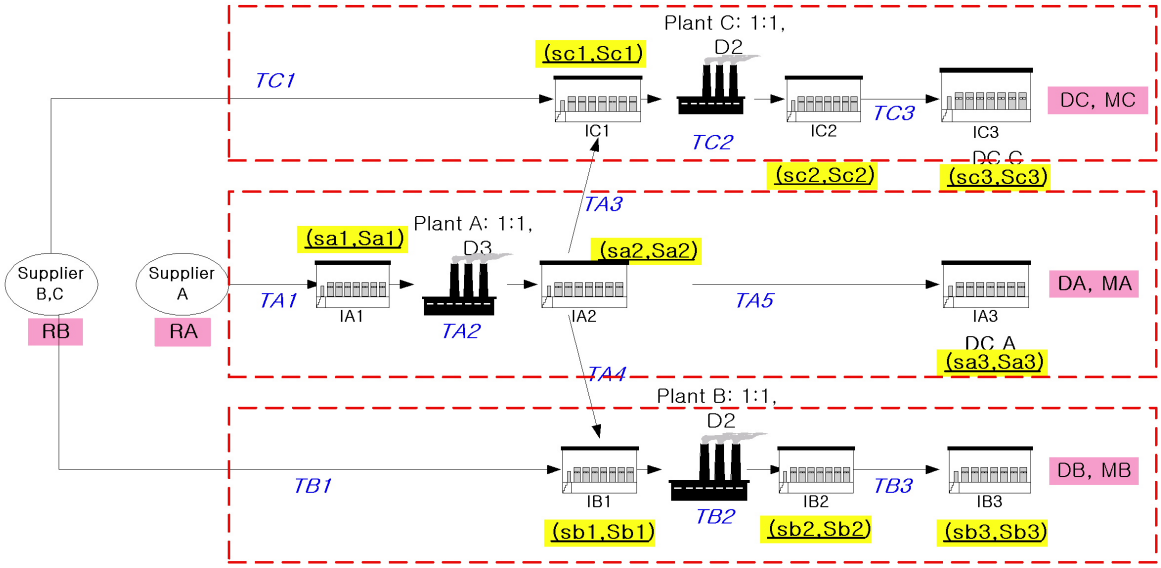

Figure 49: Illustrative Example, SCM with 3 Products Under Uncertainty

6.2.1 Markovian Model of the Uncertain Parameters

Demand and price variations encountered by retailers or manufacturers have many different sources. They are often correlated, both amongst themselves, due to an underlying cause such as oil prices, and in time such as in a seasonal variation. In previous supply chain literature, demand and price uncertainties in the SCM problem have been represented as Gaussian(normal) or exponential random variables [90, 37, 80, 36] or uniformly distributed random variables [87]. However, it appears that auto-and-cross correlation among the uncertain parameters in this context has not been addressed previously in the context of process supply chains with the exception of the contribution of [59], in which uncertain product demands are represented by a normal multivariate probability distribution. In our problem formulation, demand and price of each product are modeled with a Markov chain. This mirrors our previous work using Markov chains for applications in stochastic resource constrained project scheduling[25]. The use of Markov chain in modeling of market demand and price offers some advantages over the previous approaches. For example, if the

demand of a certain product is very high, the probability of a sharp demand decrease at next time unit (a week or a month) would be very small. Thus, the current demand can be an important indication of the future demand realization. Furthermore, demand and price may not be varied independently in the market due to their natural correlation. In our SC model, we assume the uncertain demand and price of a product are realized as a set and there are several different sets of demand and price evolving according to a given Markov chain. The possible discrete values for the uncertain demand and price may represent the actual values or the mean values of the parameters. In summary, random (but correlated) uncertain demand and price for a product in the market are represented with n sets of demand and price and n by n state(in the Markov chain) transition probability matrix as depicted in Figure 50. In the illustrative example (Figure 49), 5 Markov chains are given to represent uncertain demands and prices of 3 products and uncertain price of raw materials from 2 external suppliers.

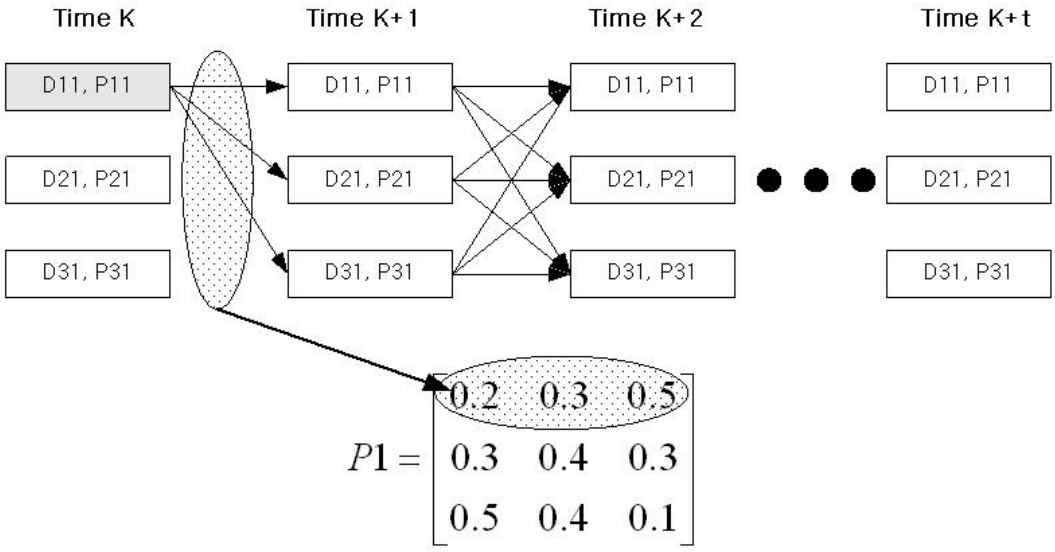

Figure 50: Representation of An Uncertain Demand and Price with a Markov Chain

6.3 Heuristics: Combination of Static Inventory Control Policies

A crucial step in applying our framework is to develop heuristics, which give a reasonable solution (policy) for the given problem and can be simulated without a significant computational burden. In this section, we propose to develop such heuristic policies by combining available static inventory control policies. The given SCM problem has many material flows that have to be decided at each time unit. Each of those material flows is linked to two inventories in the supply chain. Thus, the decision of each material flow can be made by a single inventory control policy. One of the simplest inventory control policies is a static inventory control policy[16], also known as an (s,S) policy, in which s is given as a reorder point to replenish the inventory level up to S. Even though the static inventory control policy is simple, with appropriate choice of s and S parameters, it is applicable to a wide range of operating conditions. Furthermore, under certain assumptions, the (s,S) policies can be shown to be optimal inventory control policies for classes of supply chain systems similar to the ones addressed in this study[28, 17, 83, 29]. However, for the given SCM problem, the static inventory control policy is not sufficient to get good overall performance policy due to the need to vary the replenishment levels under the uncertain demand. It would be better to employ a dynamic (s,S) policy that could respond to specific realizations of the uncertainty. As mentioned earlier, we as starting policies a set of static inventory control policies, which work reasonably and in a complementary manner. Each of the heuristics will be used for stochastic simulation in the next step of the solution framework as illustrated in section 6.5.1.

6.4 Conventional Stochastic DP Formulation

In this section, we develop a DP formulation, defining the state, action(decisions), the state transition rules, and the objective function(profit-to-go). Appropriate Bellman(value) iteration and online decision making equation will be presented in the last part of this section.

6.4.1 Definition and Aggregation of State

For the given SCM problem, three types of information are necessary to describe the status of the system.

1. **Inventory Level** All inventory levels in the system have to be included in the state. Thus, for the Q inventories in the system, the inventory level at time k, $I_{j1}(k)$, for $j1 = 1, ..., Q$, is defined as a state variable.

2. **On-Going Production** For P plants in the system, the amount of on-going production started at time $k - \ell$ at the plant $j2$ is devoted by $O_{j2}(k - \ell)$, for $j2 = 1, ..., P$ and $\ell = 1, ..., \tau_{j2} - 1$. where, τ_{j2} is the production time delay(time delay from the raw material inventory to the product inventory of the plant) for the plant $j2$.

3. **Information State** In addition to the physical state variables defined above, the information state, which represents the most current status of the uncertain parameters, also should be included in the state. The information state variable, which is represented as an integer, is a realized "state" of the Markov chain for the corresponding uncertain parameter or set of uncertain parameters. For L Markov chains in the system, the information state, $z_{j3}(k)$, for $j3 = 1, ..., L$, is including in the state vector. $z_{j3}(k)$ can be one of integers ranging from 1 to S_{j3} when the Markov chain $j3$ has S_{j3} possible states.

In summary, the state of the given problem is defined as follows:

$$\begin{aligned} X(k) = [&I_1(k), ..., I_Q(k), O_1(k-1), ..., O_1(k-\tau_1+1) \\ &, ..., O_P(k-1), ..., O_P(k-\tau_P+1), z_1(k), ..., z_L(k)] \end{aligned} \tag{79}$$

All the state variables in (79) are integers since the SCM problem is assumed to be a discrete system. The explosion of the state space is governed by the number and range of the inventory levels. To avoid a state space explosion, the state variables have to be aggregated and represented with discrete integer values. Besides the information state variables, z_{j3}, for $j3 = 1, ..., L$, which are naturally discrete, all other state variables(for

inventory and on-going production) are aggregated. The state aggregation is defined with three elements, 1) aggregation ranges, 2)representative index, 3) representative value. The aggregated range defines the range of the state values to be treated as belonging to a same state. Once a state variable is aggregated, a representative index is assigned to the state variable and represented in the state space with the index. The representative value of an aggregated state is required to disaggregate the state to an actual value (for example, an actual inventory level) and usually chosen as the median of the aggregation range.

6.4.2 Definition of Action

An action, u, of the problem is defined as a decision on all the material flows in the system.

$$u = [T_1, T_2, ..., T_R]^T \tag{80}$$

T_{j4} is an integer variable which represents a material flow $j4$, for $j4 = 1, 2, ..., R$. Some of the state variables, on-going production, defined in (79) are included in previous actions as production decisions. According to the definition of action in (80), the size of action space imposed of all possible actions is often too large to be investigated fully in the Bellman iteration and in the real-time decision making step of the DP. Thus, the action space also has to be aggregated as well as the state space. The simple aggregation rules suggested for the state in 6.3 is not appropriate for the action space. In general, a supply chain network includes many different material flows and the ranges of the material flows are diverse and large. Thus, the number of all possible actions will be astronomical even after aggregation. For example, if a supply chain network consists of 10 material flows to be decided at every unit time and each of the material flow is aggregated to just 3 representative values. The total number of actions in the action space will be $3^{10} = 59,049$. Therefore, the computational infeasibility of DP in a supply chain application is partly the result of the large number of possible actions. Note that, in a discrete system, the computational load of the Bellman iteration in equation (84) is proportional to multiplication of the number of states and number of actions to be investigated. The computational problem created by large action(decision) spaces is circumvented by introducing an 'implicit sub-action space' which will be explained in section 6.5.2 in detail.

6.4.3 State Transition Rules

In a discrete time system, the state at time $k+1$, $X(k+1)$ can be derived from the state at time k, $X(k)$, and the control action(decision), $u(k)$. For the given problem, the state transition rules are linear material balance equations of all inventories as generalized in equation (81)[1]

$$\text{Inventory Level at Time } k+1 \quad = \quad \text{Inventory Level at Time } k + \text{Input at Time } k - \text{Output at Time } k \quad (81)$$

The state variables can be classified into two types of variables in state transition. 1. **Controllable State Variables**: Those state variables that represent physical properties of the system such as the inventory levels and on-going productions are evolved with the current and previous actions. The inventory levels are partially controllable because of the uncertain demands also affects those state variables' transitions. The on-going productions are fully controllable state variables because they are actual actions taken in the past. If random yields were included in the problem, then this assumption would not hold, however. 2. **Uncontrollable State Variables**: Transitions of the information state variables are irrelevant to any action because the transitions are governed by the underlying Markov chains. The information state variables at time k only depends on the realized uncertain parameters at time k.

Due to the uncontrollable state variables, the state at time $k+1$ is not unique even though it evolves from same state at time k and same action by the state transition equation (81). According the underlying Markov chains, all possible next states, $X(k+1)$ are calculated with their realization probabilities for given $X(k)$ and $u(k)$. The realization probabilities of the possible next states are used in the calculation of the expected objective function value(profit-to-go) corresponding to the $X(k+1)$ states.

6.4.4 Objective Function: Profit-to-Go

The objective function to be maximized is the overall profit of the system. Since the operation of the supply chain is not limited to a specific finite time horizon, the problem is considered as an optimal control problem over an infinite time horizon. And the overall

[1]Detail state transition rules are illustrated with an example in Section 6.6.1.1 from equation (93) to (104).

profit is defined as the summation of one stage profit at each unit time over the infinite time horizon. To maximize the overall profit of the system in infinite horizon, an approximated 'profit-to-go' function, $J(X(k))$, is defined as following.

$$\begin{aligned} J(X(k)) &= E\{\text{Sum of all future profit}\} \\ &\simeq \hat{J}^0(X(k)) = E\{\sum_{i=0}^{H} \alpha^i \phi(X(k+i), u(k+i))\} \end{aligned} \tag{82}$$

One stage profit at time k, $\phi(X(k), u(k))$, is defined by the following equation (83).

$$\phi(X(k), u(k)) = \text{Revenue}(k) - \text{Inventory Cost}(k) - \text{Manufacturing Cost}(k) - \text{Raw Material Cost}(k) \tag{83}$$

The profit-to-go function over an infinite horizon can be approximated as shown in equation (82) with an approximation horizon H and a discounting factor α. $\phi(X, u)$ is a current profit function of the given state, X, and action, u. The initial value of the approximated profit-to-go, $\hat{J}^0(X)$, is calculated from appropriate sub-optimal simulation data and the $\hat{J}^0(X)$ is used as an initial profit-to-go in the Bellman iteration step of DP, which refines it to the optimal profit-to-go, $J^*(X)$.

6.4.5 Bellman Iteration and Real-Time Decision Making

In DP, one calculates numerically the optimal profit-to-go function J^* via the Bellman iteration step. This computation can be done offline, i.e., before the policy is applied to the actual system. For the SCM problem, the Bellman iteration equation is given as follows:

$$J^{i+1}(X(k)) = \max_{u(k) \in U} E\{\phi(X(k), u(k)) + \alpha J^i(X(k+1)|X(k), u(k))\} \tag{84}$$

In the above, U is a discrete action space of all actions defined in (80). In finding optimal $u(k)$, infeasible action, that make an inventory negative or violate the maximum production capacities of the plants in the system, have to be excluded. The Bellman iteration is continued until J^i meets a certain convergence criterion, e.g. $\| \frac{J^{i+1} - J^i}{J^i} \|_\infty < 0.01$. If the J^i meets the convergence criterion, it is considered as the optimal profit-to-go, J^* and used for online decision making as follows. According to the 'Principle of Optimality' of DP, the following decision $u^*(k)$ is the optimal action for the state, $X(k)$ given at any time k.

$$u^*(k) = \arg \max_{u(k) \in U} E\{\phi(X(k), u(k)) + \alpha J^*(X(k+1)|X(k), u(k))\} \tag{85}$$

6.5 The Algorithmic Framework: DP in A Heuristically Restricted State Space

All the necessary elements of the DP are defined in the previous section. Thus, the problem can be solved by using the appropriate Bellman equation shown in equation (84). However, the computational load of the full DP for a realistically sized example will be beyond the current computational capability. For example, the illustrative example shown in Figure 49 has a state composed of 5.832×10^{10} discrete states even with the state aggregation[2]. Furthermore, the number of discrete action defined in equation (80) will be very large even with the action space aggregation as discussed in the introduction part of this chapter. The number of states tends to increase exponentially with the problem size, number of inventories, number of products, and number of possible realizations. In out approximate DP framework, the part of the state space within which the cost-to-go is evaluated through the Bellman iteration is restricted to those visited during the simulation of the heuristic policies. The effectiveness of the algorithmic framework in handling large state space has been tested previously in [25]. However the algorithmic framework is not appropriate for the given problem with large number of actions. In this chapter, the algorithmic framework is modified to handle a large number of actions by introducing implicit sub-action space for each state in the restricted state space. The implicit sub-action space is defined with actions generated by a set of heuristics that have been applied during the heuristic simulation step of the framework. The general steps of applying the algorithmic framework will be similar to those shown in [25] except the Bellman iteration and online decision making is performed over the confined state space and the implicit sub-actions spaces of the states.

1. Stochastic simulation with the heuristic policies

2. Identification of the restricted state space which is composed of the states visited in the simulation and the first estimation of the profit-to-go values for the restricted state space using the simulation data.

[2]calculation of the number of states will be explained in later section 6.6 "illustrative example"

3. Bellman iteration in the heuristically restricted state space and corresponding implicit sub-action space

4. Evaluation of the policy performance when applied to real-time decision making

Detailed descriptions of the steps are given in the next sections.

6.5.1 Learning Stage: Simulation of the Heuristic Policies

The purpose of the simulation is three-fold. First, the simulation is performed in order to obtain meaningful, manageably sized, set of the states within which the DP is to be performed. Obtaining a reasonably sized subset containing trajectories of good policies is critical for solving the problem because DP over the entire state space is computationally infeasible for the given problem. For the simulation, a large number of uncertain parameter realization sets are generated by the underlying Markov chains. Each realization set represents one scenario along a certain time horizon out of the large number of possible scenarios. In the simulation, several different heuristics are applied for each realization and a set (trajectory) of visited states (as defined in the previous section) is obtained from each heuristic. Because each heuristic works in a different way as they are designed to do, there can be several different state trajectories even for the same scenario. Different state trajectories are obtained with different heuristics even for a same realization scenario. Those different state trajectories will be combined in the state space as profit-to-go in the later step, Bellman iteration, of the algorithmic framework. The heuristic policies applied for the given problem will be described in a later section.

Second, the simulation provides initial 'profit-to-go' values, which can be used to start up the Bellman iteration. There can be different 'profit-to-go' values for a same state evolving into different trajectories (corresponding to different heuristic or realization scenario). The initial 'profit-to-go' for the state is an averaged 'profit-to-go' value considering the number of times the state was visited.

Third, the initial 'profit-to-go' values can be directly used for the 'Rollout' approach to find quick solution(policy) of the problem without the Bellman iteration. The rollout approach is described in section 6.6.4 for the illustrative example.

6.5.2 Implicit Sub-Action Space for A State

As stated earlier, the problem has a large action space due to many material flows involved in the supply chain. To reduce the computational load of the Bellman iteration and the online decision making steps of the algorithmic framework, we propose to make a decision in an "implicit sub-action space", $U^{X(k)}$, a set of actions taken by the heuristics during the heuristic simulation, for each state in the confined state space. Hence, the number of the states in "sub-action space" is the same as the number of states in the restricted state space. Instead of a set of aggregated actions, the set of heuristics that visited the state in the simulation are recorded for the implicit sub-action space. In this way, we can avoid distorted state transitions introduced by action aggregation. The definition of action in equation (80) is kept the same other than that implicit heuristic rule h_i is applied for the state, $X(k)$, to reproduce a set of possible actions to be evaluated for the state.

$$u_i = h_i(X(k)) \quad \text{for } i = i_1, ..., i_N \tag{86}$$

$$U^{X(k)} = \{u_{i_1}, ..., u_{i_N}\} \tag{87}$$

where i is the index of the heuristics that visited the state $X(k)$ in the simulation. Restricting the decision in the sub-action space may prohibit choosing the optimal decision(action) for the given state. However, it ensures that a chosen action is feasible and increases the possibility of the next state being in the restricted state space.

6.5.3 Bellman Iteration over the Confined State Space

The Bellman iteration step of the proposed approach is same as in the conventional DP described in the previous section except for the following two details. First, it is done over the restricted state space instead of the entire state space for the obvious reason. Second in each iteration, the entire action space, U, is replaced with implicit sub-action space, $U^{X(k)}$, to provide possible actions to be evaluated.

$$J^{i+1}(X(k)) = \max_{u(k) \in U^{X(k)}} E\{\phi(X(k), u(k)) + \alpha J^i(X(k+1)|X(k), u(k))\} \tag{88}$$

In the calculation of equation (88), the current cost, $\phi(X(k), u(k))$, is deterministic for a given state, $X(k)$, and an action, $u(k)$. The profit-to-go of the next state, $J(X(k+$

1)|$X(k), u(k)$) is stochastic due to the uncertain demand and price parameters. The exact expected value of $J(X(k+1)|X(k), u(k))$ may not be calculated because all the possible next states may not be included are not in the restricted state space due to the limited number of realizations simulated. To obtain an approximate value of the $J(X(k+1)|X(k), u(k))$, the cost-to-go approximation method, in which normalized weighting factor of the profit-to-go(or cost-to-go) is applied for the state in the restricted state space as shown in Figure 51. Detailed description of the approximation method is given in our previous work, [25].

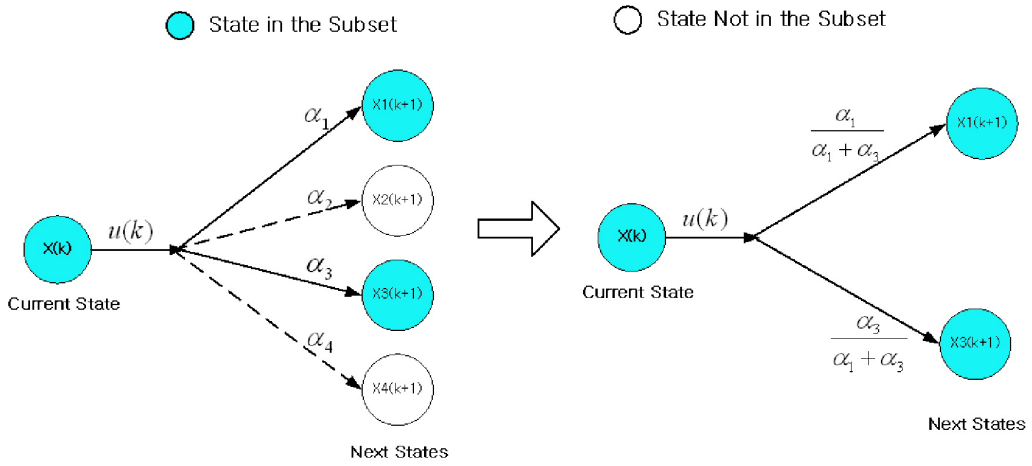

Figure 51: Profit-to-Go Approximation

6.5.4 Real-Time Decision Making

The 'converged' profit-to-go values obtained in the previous step are implemented for real-time decision making as follows.

$$u^*(k) = \arg \max_{u(k) \in U^{X(k)}} E\{\phi(X(k), u(k)) + \alpha J^*(X(k+1)|X(k), u(k))\} \tag{89}$$

The calculation in the online decision making is the same as in the Bellman iteration except that it is needed only for the specific encountered state at the time. The decision gets implemented and the actual system (ranther than the model) provides the next states. Because the decision belongs to the sub-action space, the next state is probabilistically assured to be in the restricted state space, if the enough simulations have been performed to allow for all probabilistically feasible outcomes. However, since those may not necessarily

hold, and could encounter an unvisited state not in the restricted state space. The state transitions can also be misled by the distortion induced by the state aggregation due to aggregation and disaggregation of the state in the state transition calculation. That is, even though the state is actually in the restricted state space, it could be represented as a state not in the restricted state space due to the distortion in the state aggregation and disaggregation processes . In this case, the current state is replaced with the most 'similar' state in the restricted state space according to the 'state similarity criteria'. In the given problem, the state is defined with three types of state variables as it is represented in equation (79).

$$X(k) = [\text{Inventory Level}, \text{On-Going Production}, \text{Information State}] \tag{90}$$

In finding the most similar state in the confined state space with the current state, priorities of the state variables are given in the order of information state variables, on-going production state variables, and then inventory level state variables. The information state variables are considered the most important have to be matched first. Because the information state variables are not aggregated or disaggregated in state transitions, existence of the states with exactly same information state variables in the restricted state space is assured under the assumption that the restricted state space is obtained with a sufficient number of realizations. The 'similar state' in the confined state space is searched in the following order.

1. States in the restricted state space with the same information state as the information state in the current state are searched

2. Among the states obtained in the previous step, states with the same on-going production of the current state are searched

    - If none of the states obtained in the previous step has the same on-going production state variables with the current state variable, find a state that has the minimum infinity norm of the difference with the current state.

Table 20: Probabilities and Parameters of the Markov chains in the Illustrative Example

Markov Chain	Demand & Price	Probability Matrix
MC1 (Product A)	$\begin{bmatrix} 50 & 54 \\ 32 & 41 \\ 20 & 37 \end{bmatrix}$	$\begin{bmatrix} 0.60 & 0.10 & 0.10 \\ 0.30 & 0.50 & 0.20 \\ 0.10 & 0.40 & 0.70 \end{bmatrix}$
MC2 (Product B)	$\begin{bmatrix} 47 & 75 \\ 33 & 73 \\ 25 & 64 \end{bmatrix}$	$\begin{bmatrix} 0.50 & 0.30 & 0.15 \\ 0.35 & 0.50 & 0.25 \\ 0.15 & 0.20 & 0.60 \end{bmatrix}$
MC3 (Product C)	$\begin{bmatrix} 58 & 42 \\ 38 & 36 \end{bmatrix}$	$\begin{bmatrix} 0.8 & 0.3 \\ 0.2 & 0.7 \end{bmatrix}$
Markov Chain	RM Price	Probability Matrix
MC4 (RM for A)	$\begin{bmatrix} 18 \\ 25 \end{bmatrix}$	$\begin{bmatrix} 0.82 & 0.25 \\ 0.18 & 0.75 \end{bmatrix}$
MC5 (RM for B and C)	$\begin{bmatrix} 30 \\ 40 \end{bmatrix}$	$\begin{bmatrix} 0.85 & 0.40 \\ 0.15 & 0.60 \end{bmatrix}$

3. Among the state obtained in the previous step, find a state that has the minimum infinity norm of the difference with the current state.

6.6 Illustrative Example

As an illustrative example of the SCM problem with uncertainty, we consider a supply chain with three products and multiple inventories linked to external suppliers, plants, and markets (customers) as shown in Figure 1. As stated earlier, the objective is to maximize the overall profit by controlling all the relevant material flows in the system. Demand and price of each product are uncertain in the market and evolve according to underlying Markov chains. Besides the demand and price, the raw material price is also assumed to follow an independent Markov chain. Five Markov chains(three for the products and two for the raw materials) are introduced in the example to represent the uncertain parameter variations. All transition probability matrices and uncertain parameters for the Markov chains are summarized in Table 20.

Inventory cost arises from every inventory in the system and the cost is assumed to be piece-wise linear function shown in Figure 52. It is assumed that outsourcing is available for every inventory and hence the capacity of each inventory is infinite. However, as the amount

of product in an inventory increases, the total inventory cost increases more rapidly due to the higher inventory cost parameters(i.e. $IC_1 < IC_2 < IC_3$ in Figure 52). The inventory parameters of the problem are summarized in the Table 21.

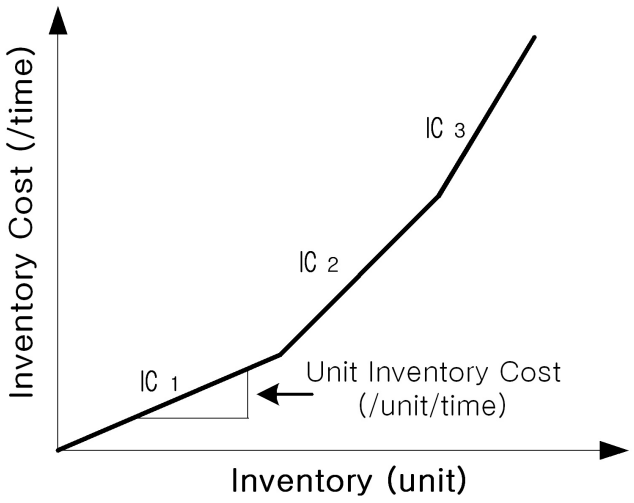

Figure 52: Piece-wise Linear Inventory Cost

Three batch plants are involved in the example and each plant has certain maximum production capacity(/unit time), production time, and minimum production cost for a single batch of production. Those parameters for the three plants are summarized in Table 22. Besides the parameters mentioned above, we set a fixed internal transaction cost(= 33) for the product A in the material flow $TA3$ and $TA4$. The internal transaction price will be used to evaluate the profit of the different product lines classified in Figure 49. Note that the overall profit of the entire system is same regardless of the internal transaction price for the product A.

6.6.1 Definition and Aggregation of the State and Action

The state for the illustrative example is defined as follows.

$$X(k) = [\underbrace{IA1(k), IA2(k), IA3(k), IB1(k), IB2(k), IB3(k), IC1(k), IC2(k), IC3(k)}_{\text{Inventory Levels}}$$
$$\underbrace{, TA2(k-2), TA2(k-1), TB2(k-1), TC2(k-1),}_{\text{On-going Production Amounts}}$$

Table 21: Inventory Cost Parameters for the Illustrative Example

Inventory	Parameters			
IA1	Range	$0 \leq i \leq 60$	$60 < i$	
	Inv. Cost(/unit)	1.5	2.4	
IA2	Range	$0 \leq i \leq 40$	$40 < i \leq 100$	$100 < i$
	Inv. Cost(/unit)	1.3	1.8	2.2
IA3	Range	$0 \leq i \leq 30$	$30 < i \leq 50$	$50 < i$
	Inv. Cost(/unit)	1.5	2.0	3.0
IB1	Range	$0 \leq i \leq 60$	$60 < i \leq 100$	$100 < i$
	Inv. Cost(/unit)	1.1	1.7	2.3
IB2	Range	$0 \leq i \leq 60$	$60 < i$	
	Inv. Cost(/unit)	1.4	2.0	
IB3	Range	$0 \leq i \leq 60$	$60 < i \leq 100$	$100 < i$
	Inv. Cost(/unit)	1.6	2.2	2.9
IC1	Range	$0 \leq i \leq 60$	$60 < i$	
	Inv. Cost(/unit)	1.5	2.5	
IC2	Range	$0 \leq i \leq 60$	$60 < i \leq 100$	$100 < i$
	Inv. Cost(/unit)	1.7	2.4	3.0
IC3	Range	$0 \leq i \leq 50$	$50 < i \leq 90$	$90 < i$
	Inv. Cost(/unit)	1.9	2.5	3.3

Table 22: Plant Parameters for the Illustrative Example

	Plant A	Plant B	Plant C
P. Cost(/unit)	5	9	4
Max. Capacity	120	70	70
P. Time	3	2	2
Fixed Cost(/batch)	120	100	95
Ratio(R:P)*	1:1	1:1	1:2

* Raw Material to Product Conversion Rate

$$\underbrace{Z1(k), Z2(k), Z3(k), Z4(k), Z5(k)]}_{\text{Information State}} \quad (91)$$

As discussed earlier, the state has to be aggregated to avoid an unmanageably large state space. Table 23 shows parameters used for the aggregation of the inventory levels and the on-going production state variables$(TA2(k-2), TA2(k-1), TB2(k-1), TC2(k-1))$.

Table 23: State Aggregation

Rep. Index*	1	2	3	4	5
Range of IA1**	0-25	25-50	50-75	75-100	100-∞
Rep. IA1***	13	38	63	88	113
Range of IA2	0-25	25-50	50-75	75 - 100	100 -∞
Rep. IA2	13	38	63	88	113
Range of IA3	0-25	25-50	50-75	75 - 100	100 -∞
Rep. IA3	13	38	63	88	113
Range of IB1	0-25	25-50	50-75	75-100	100-∞
Rep. IB1	13	38	63	88	113
Range of IB2	0-25	25-50	50- 75	75 - 100	100-∞
Rep. IB2	13	38	63	88	113
Range of IB3	0-25	25-50	50- 75	75 - 100	100-∞
Rep. IB3	13	38	63	88	113
Range of IC1	0-25	25-50	50- 75	75 - 100	100-∞
Rep. IC1	13	38	63	88	113
Range of IC2	0-25	25-50	50-75	75-100	100 - ∞
Rep. IC2	13	38	63	88	113
Range of IC3	0-25	25-50	50-75	75-100	100-∞
Rep. IC3	13	38	63	88	113
Rep. Index	0	1	2	3	4
Range of TA2	0	0 - 35	35 - 70	70 - 105	105 - ∞
Rep. TA2	0	18	43	78	120
Range of TB2	0	0 - 20	20 - 40	40 - 60	60 - ∞
Rep. TB2	0	10	30	50	70
Range of TC2	0	0 - 20	20 - 40	40 - 60	60 - ∞
Rep. TC2	0	10	30	50	70

* Representative Index in State Representation

** Aggregation Range of the Inventory Level

*** Representative Inventory Level in Reverse-Aggregation

Based on the state aggregation table and the state definition of the problem, the total number of states in the entire state space can be calculated to be $8.79 \cdot 10^{10} = 5^9 \cdot 5^4 \cdot (3^2 \cdot 2^3)$.

For the illustrative example, action consists of the 11 relevant material flows in the system.

$$U(k) = [\underbrace{TA1(k), TA2(k), TA3(k), TA4(k), TA5(k)}_{\text{Regarding the Product A}}, \underbrace{TB1(k), TB2(k), TB3(k)}_{\text{Regarding the Product B}}, \underbrace{TC1(k), TC2(k), TC3(k)}_{\text{Regarding the Product C}}] \quad (92)$$

6.6.1.1 State Transition Rules

Next state transition rules are defined. As discussed in section 6.4.3, there are two types of state variables, controllable and uncontrollable state variables. For the controllable state variables, the state transition rules are simply represented with the following material balance equations.

- State Transition Rules of the Controllable State Variables

$$IA1(k+1) = IA1(k) - TA2(k) + TA1(k) \quad (93)$$

$$IA2(k+1) = IA2(k) - TA5(k) - TA4(k) - TA3(k) + TA2(k-2) \quad (94)$$

$$IA3(k+1) = IA3(k) - DA(k) + TA5(k) \quad , \quad \text{if } IA3(k+1) > 0 \quad (95)$$

$$IA3(k+1) = 0 \quad , \quad \text{if } IA3(k+1) \leq 0 \quad (96)$$

$$IB1(k+1) = IB1(k) - TB2(k) + TB1(k) + TA4(k) \quad (97)$$

$$IB2(k+1) = IB2(k) - TB3(k) + TB2(k-1) \quad (98)$$

$$IB3(k+1) = IB3(k) - DB(k) + TB3(k) \quad , \quad \text{if } IB3(k+1) > 0 \quad (99)$$

$$IB3(k+1) = 0 \quad , \quad \text{if } IB3(k+1) \leq 0 \quad (100)$$

$$IC1(k+1) = IC1(k) - TC2(k) + TC1(k) + TA3(k) \quad (101)$$

$$IC2(k+1) = IC2(k) - TC3(k) + 2 * TC2(k-1) \quad (102)$$

$$IC3(k+1) = IC3(k) - DC(k) + TC3(k) \quad , \quad \text{if } IC3(k+1) > 0 \quad (103)$$

$$IC3(k+1) = 0 \quad , \quad \text{if } IC3(k+1) \leq 0 \quad (104)$$

Among the variables in controllable state transition equations, the retail inventory levels, $IA3(k), IB3(k), IC3(k)$, are only 'partially' controllable because demands of the products,

Table 24: Heuristic 1

s_a1	S_a1	s_a2	S_a2	s_a3	S_a3
80	120	100	200	80	150
s_b1	S_b1	s_b2	S_b2	s_b3	S_b3
70	150	90	150	70	150
s_c1	S_c1	s_c2	S_c2	s_c3	S_c3
90	130	80	95	70	120

Table 25: Heuristic 2

s_a1	S_a1	s_a2	S_a2	s_a3	S_a3
80	100	120	220	60	130
s_b1	S_b1	s_b2	S_b2	s_b3	S_b3
80	120	70	100	80	120
s_c1	S_c1	s_c2	S_c2	s_c3	S_c3
100	150	60	110	100	150

$DA(k), DB(k), DC(k)$, are uncertain. However, the state transition equations of those retained inventory levels are deterministic with particular realizations of the product demands. The state transitions of the uncontrollable(information) state variables depends solely on the realization of the five underlying Markov chains.

6.6.2 Simulation Results for the Heuristics

Heuristics can be created by combining static inventory control policies for all the inventories in the system to control the supply chain. Six heuristics are proposed where each heuristic consists of nine static inventory control policies. The stationary inventory control parameters, s and S, of each heuristic are summarized in Tables 24 through 29. In the heuristics, the two raw material inventories of the product B and C, $IC1$ and $IB1$, are replenished by the external supplier and the inventory $IA2$ with the ratio of 8 to 2(80% from the supplier, 20% from the inventory $IA2$). In distributing the internal material flows, TA3 and TA4, in all of the heuristics, TA4(to IB2) is considered first and TA3(to IC2) is considered later if IA2 still has surplus inventory after fulfilling the required TA3 according to the given heuristic. As an initial step of the algorithmic framework, the behavior of the supply chain under the 6 heuristics are simulated for a large number of realizations of Markov chains. Table 30 shows the performance of the heuristics for a certain set of realizations(30,000 time horizon).

Table 26: Heuristic 3

s_a1	S_a1	s_a2	S_a2	s_a3	S_a3
50	100	180	250	100	130
s_b1	S_b1	s_b2	S_b2	s_b3	S_b3
50	120	100	160	100	120
s_c1	S_c1	s_c2	S_c2	s_c3	S_c3
50	120	120	180	100	150

Table 27: Heuristic 4

s_a1	S_a1	s_a2	S_a2	s_a3	S_a3
80	100	120	220	70	130
s_b1	S_b1	s_b2	S_b2	s_b3	S_b3
80	120	100	160	100	150
s_c1	S_c1	s_c2	S_c2	s_c3	S_c3
70	120	100	250	80	150

Table 28: Heuristic 5

s_a1	S_a1	s_a2	S_a2	s_a3	S_a3
80	100	120	220	70	130
s_b1	S_b1	s_b2	S_b2	s_b3	S_b3
80	120	100	160	100	150
s_c1	S_c1	s_c2	S_c2	s_c3	S_c3
50	120	100	180	80	130

Table 29: Heuristic 6

s_a1	S_a1	s_a2	S_a2	s_a3	S_a3
80	140	110	230	80	110
s_b1	S_b1	s_b2	S_b2	s_b3	S_b3
90	140	90	160	100	130
s_c1	S_c1	s_c2	S_c2	s_c3	S_c3
40	80	100	170	120	160

From

Table 30: Results of Simulating the Heuristics for Realization(30,000 horizon) Set #1

	Profit A	Profit B	Profit C	Total Profit	CSLA*	CSLB	CSLC
H1	2.340e+6	2.107e+7	1.306e+7	3.647e+7	0.9441	0.9761	0.8882
H2	4.637e+6	1.585e+7	1.144e+7	3.193e+7	0.9546	0.7168	0.9199
H3	3.628e+6	2.078e+7	1.692e+7	4.133e+7	0.9796	0.8602	0.9525
H4	4.684e+6	2.004e+7	1.382e+7	3.854e+7	0.9608	0.9836	0.9249
H5	4.794e+6	2.002e+7	1.429e+7	3.910e+7	0.9618	0.9835	0.8009
H6	2.438e+6	2.011e+7	1.608e+7	3.863e+7	0.9164	0.9751	0.9287

* Customer Service Level of the Product A

the simulation result shown in the Table 30, Heuristic # 3 is better than the other heuristics in terms of the total profit. On the other hand, Heuristic # 4 is the best in terms of the average customer service level of the products. The customer service level of a product is defined as the amount of fulfilled demand over the total demand for the product. The results shown in Table 30 points to the possibility of improving the solutions obtained by the six heuristics by searching actions over the restricted state space visited by the heuristics in the simulation because none of the heuristics is universally best for all cases.

6.6.3 Restricted State Space and Sub-Action Space

In the previous section, the simulation is performed over 20 sets of realizations, which correspond to 600,000 realizations (20 sets with 30,000 realizations), which are used to obtain the restricted state space for the given problem. For each heuristic and each set of realization, the simulation is carried out with 3 different initial inventory conditions, low, medium, and high, to capture different transient state trajectories until the supply chain operation reaches a stable pattern. Thus, the total number of individual realizations in the series of simulation is 10,800,000. For each individual realization, its profit-to-go value for the visited state, $\hat{J}(X)_{sim}$ can be calculated from simulation data by the profit-to-go approximation equation (82) with $H = 100$ and $\alpha = 0.95$. If the state is already in the storage, the profit-to-go value of the state in the confined state space, $\hat{J}(X)^{old}_{cs}$, is updated by the following equation.

$$\hat{J}(X)^{new} = \frac{n\hat{J}(X))^{old} + \hat{J}(X))_{sim}}{n+1} \tag{105}$$

where, n is the total number of times of the state X was visited in the series of simulations. On the other hand, if the state is not among the stored states, it is added as a new state in the storage with visiting time of $n = 1$ along with the profit-to-go value $\hat{J}(X)_{sim}$. During the simulation, 1,433,694 states are visited and those states are recorded to define the restricted state space. The size(number of states) of the restricted state space increases as more and more simulation data are added. Figure 53 shows the increase of the states space size with the total number of realizations. As can be seen from the figure, the restricted state space is 'saturated' with 1,433,694 states implying probabilistically 'closed' at least for the 600,000 realizations used for the simulation. The sub-action space is also obtained

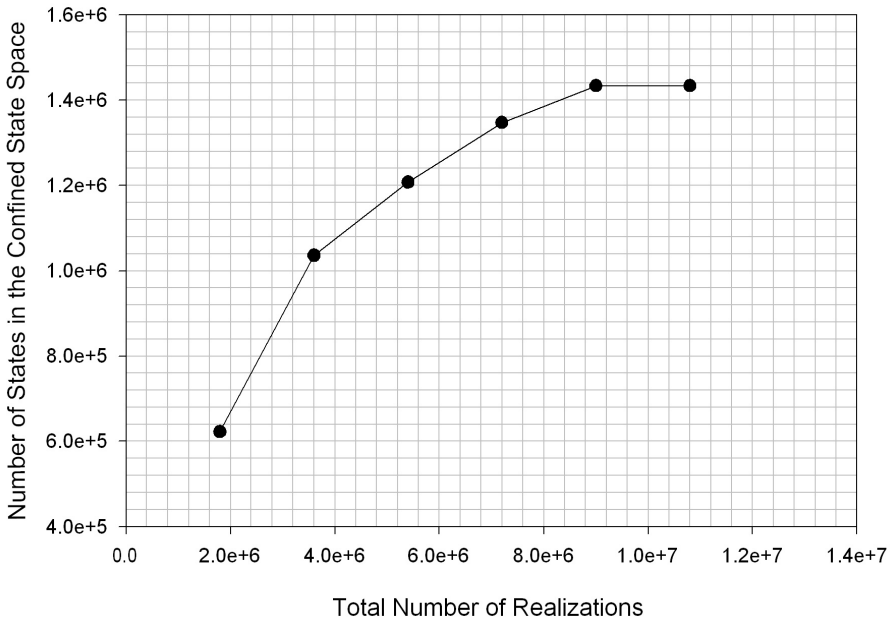

Figure 53: Increase in number of the States in the Restricted State Space with the Number of Realizations

along with each state in the restricted state space. By restricting the action space for an individual state, in the Bellman iteration and in the real-time decision making, we can reduce the computational load dramatically because the action space is represented with only 6 heuristics applied for the given problem rather than the enormous number represented by all possible combination of the individual actions. Out of 1,433,694 stored states, 1,335,784 states have a sub-action space defined by only one heuristic due to the limited overlap in the ranges of inventories in which the various heuristics operated. For such states, decision

Table 31: Distribution of the Number of Actions in Sub-Action Space

# of Heuristics	# of States
1	1,335,784
2	807,807
3	13,903
4	2,759
5	449
6	19

Table 32: Online Decision Making with Initial Profit-to-Go

	Profit A	Profit B	Profit C	Total Profit	CSLA	CSLB	CSLC
Rollout Approach	3.664e+6	2.109e+7	1.744e+7	4.219e+7	0.9809	0.8624	0.9578
Best Heuristic	3.628e+6	2.078e+7	1.692e+7	4.133e+7	0.9796	0.8602	0.9525

making is trivial because there is only one action to take. Table 31 shows tje distribution of the number of actions(taken by the heuristics) over the stored states.

6.6.4 Rollout Approach: Online Decision Making with Initial Profit-to-Go

The restricted state space and the sub-action space can be directly used for online decision making even without the Bellman Iteration. This way of approximating the 'Profit-to-Go' is called the 'Rollout Approach' and is particularly well-suited for deterministic combinatorial problems[7]. Based on the hypothesis that the heuristic simulation is done for enough number of realizations, the initial profit-to-go, $\hat{J}^0$, naturally contains complex stochastic variation of the system and is good enough to be used for the real-time decision making as shown in the following equation (106).

$$u(k) = \arg \max_{u(k) \in U^{X(k)}} E\{\phi(X(k), u(k)) + \alpha \hat{J}^0(X(k+1)|X(k), u(k))\} \tag{106}$$

The online policy obtained by the Rollout Approach is tested for the same set of realizations used for the results shown in Table 30. Computational results of the performance of the policy represented by the equation (106) are summarized in Table 32.

Comparing Table 32 with Table 30, the solution obtained by the 'Rollout Approach' is slightly better(about 2.1%) than the best heuristic policy, the Heuristic #3. The online decision making with the initial profit-to-go can be a quick alternative solution method when the Bellman iteration is not computationally feasible due to the large state space.

Average computational time of the decision making for each realization(unit time) was only 0.3 second when implemented in MATLAB on a machine with 2.66GHz CPUs and 2GB RAM.

6.6.5 Bellman Iteration Over the Restricted State Space

The Bellman iteration proposed in equation (88) is much faster than the conventional Bellman iteration in equation (84), for the given example because of the small sub-action space. As shown in Table 31, for more than 93%(1,335,784 out of 1,433,694) of the states in the state space, the decision making is trivial because only one exists action in the sub-action space. At every iteration, 1,433,694 profit-to-go values are updated for the states in the restricted set. The Bellman iteration is performed until a certain convergence criterion(i.e.$\| \frac{J^{i+1}-J^i}{J^i} \|_\infty < 0.01$) is met. According to the Bellman iteration equation in (88), a new set of profit-to-go values are obtained at each step of the iteration. With the intermediate profit-to-go values, online decision policy can be constructed by replacing the converged profit-to-go, J^*, with the intermediate profit-to-go, $\hat{J}^i$.

$$u(k) = \arg \max_{u(k) \in U^{X(k)}} E\{\phi(X(k), u(k)) + \alpha \hat{J}^i(X(k+1)|X(k), u(k))\} \tag{107}$$

Figure 54 shows the improvement in the total profit with the intermediate online policies when tested on the 30,000 realizations used for the simulation of the heuristics. As the profit-to-go values are updated by the Bellman iteration, the total profit of the intermediate solution are improved gradually. It should be noted that the profit improvement may be less new sets of realizations that are not experienced during the simulation. Comprehensive computational analysis of the online policy with the converged profit-to-go will be shown in the next section for various sets of realizations to verify the robustness of the proposed approach. The iteration was converged with the error tolerance of $\| \frac{J^{i+1}-J^i}{J^i} \|_\infty < 0.01$ after the 15th iteration and took 8.2 days when implemented in MATLAB on a machine with two 2.66GHz Xeon CPUs and 2GB RAM. Figure 55 shows the maximum relative error, $\| \frac{J^{i+1}-J^i}{J^i} \|_\infty < 0.01$, of the profit-to-go values in the Bellman iteration. The converged profit-to-go, J^*, is tested for online decision making next.

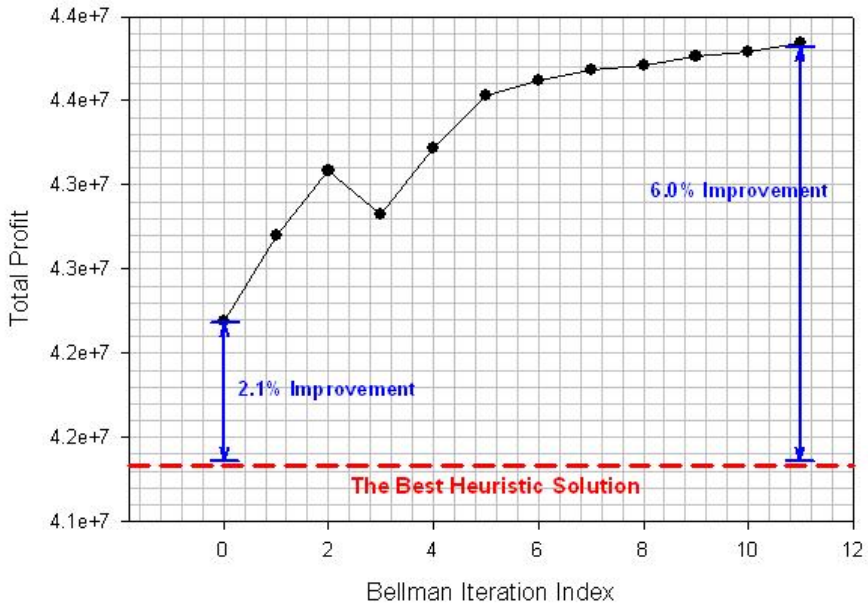

Figure 54: Total Profit Improvement with Intermediate Profit-to-Go Values

6.6.6 Online Decision Making with the Converged Profit-to-Go

The online decision making with the sub-action space and the converged profit-to-go defined in equation (89) is also computationally efficient compared to the conventional one in equation (85). If the state space is obtained with enough realizations under the simulated heuristics, the online decision making is guaranteed to be superior to any of the heuristics tested. To test the performance of the online supply operating policy based on the converged profit-to-go, 50 new sets of realizations were generated with the underlying Markov chain. Each set of realizations corresponds to 1,000 unit time horizon. Figure 56 shows the total profit improvement for the the supply chain system with the resulting operating policy compared to the best heuristic, Heuristic #3.

The average improvement is 4.53% with 1.32% and 7.84% being the minimum and maximum improvements respectively. The results in Figure 56 demonstrate that the online policy with the converged profit-to-go performs well even for new sets of realizations that are not used in the training stage. However, the performance improvements are irregular and depend on the realization because the profit-to-go is approximated over a relatively

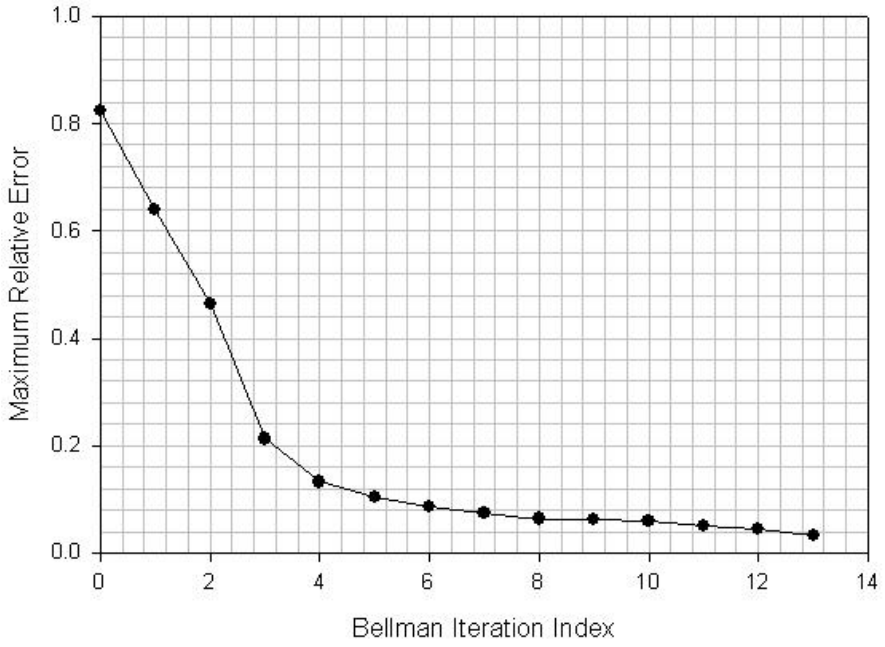

Figure 55: Maximum Relative Error, $\| \frac{J^{i+1}-J^i}{J^i} \|_\infty$, of the Bellman Iteration

long time horizon with $H = 100$ and $\alpha = 0.95$.

Figure 57 shows one stage total profits obtained by the best heuristic and the DP policy over a certain time horizon, 400 to 500, in one of the realization set (index #2) generated for the test shown in Figure 56. As we can see in the Figure, the new policy acts more 'conservatively' than the best heuristic. The one stage total profit of the DP policy neither results in as much profit nor incurs as much cost as the best heuristic. Indeed, for the given set of realization #2, standard deviations of the one stage total profits are 2396.44 and 2301.26 for the best heuristic and the DP policy respectively. The 'conservative' behavior of the DP policy is mainly due to its ability to blend future information into the decision. The profit-to-go value represents an approximate value of future profit and the online decision making equation considers all possible next states and their realization probabilities in the decision making action. Thus, the DP policy does not take an extreme action if a high loss is expected in the future. We hypothesized that the DP policy will be a combination of the heuristics used in the simulation. Figure 58 shows the 'shape' of the DP policy for the

Figure 56: Improvement in the total profit with the policy based on the converged Profit-to-Go: Distribution of Total Profit Improvement for 50 New Sets of 1,000 Realizations

same realization set shown in Figure 57. Since the action space is implicitly represented by the sub-action space introduced in section 6.5.2, every discrete action taken by the DP policy corresponding to a specific heuristic. Hence, the DP policy can be viewed as a piecewise combination of the heuristics. It chooses the best heuristic (heuristic #3) for the experienced state at each time, thus bringing an improvement over any single heuristic.

6.7 Conclusion

Supply chain management problems are growing in importance in the continuous process industries as the production of materials often involves a global enterprise. Solution methods for these problems need to address both the potentially complex models of the individual manufacturing facilities as well as the uncertainty in the information surrounding their operation, such as the market demands. We have developed an approach to apparently solve the stochastic dynamic programming problem that results from considering the evolution of the uncertain parameters as a Markov chain and allowing the decisions to be based on

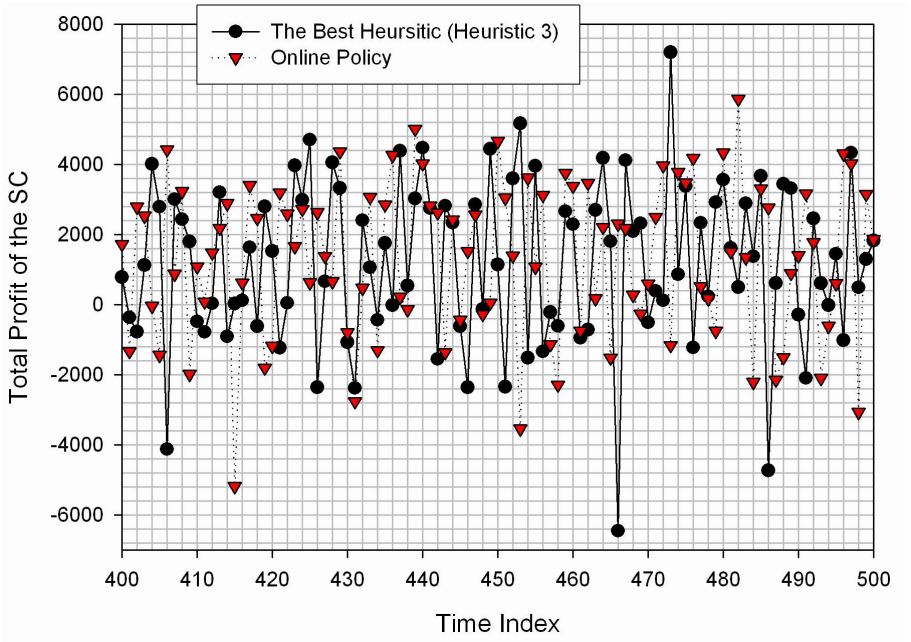

Figure 57: One Stage Total Profit (Cost) Comparison: Online Policy vs. Best Heuristic, Time Horizon 400 to 500 in the Test Realization Set #2

the information available at each time step. The approach is based on the idea of performing rigorous dynamic programming over a state space that is deliberately restricted. The restricted space is constructed by simulating a set of heuristics and storing the states that are visited during the simulation. The optimal trajectory constructed from these states can be significantly better than the individual heuristics that generated them. This results from the ability to choose the action based on a good estimate of the cost-to-go and context sensitive information. The approach can be applied to any problem formulated as a stochastic dynamic programm, provided that there are reasonable heuristics available for simulation. The major effort is in building the necessary simulation program and the Bellman iteration over the restricted space, which can still be very large for realistic problems.

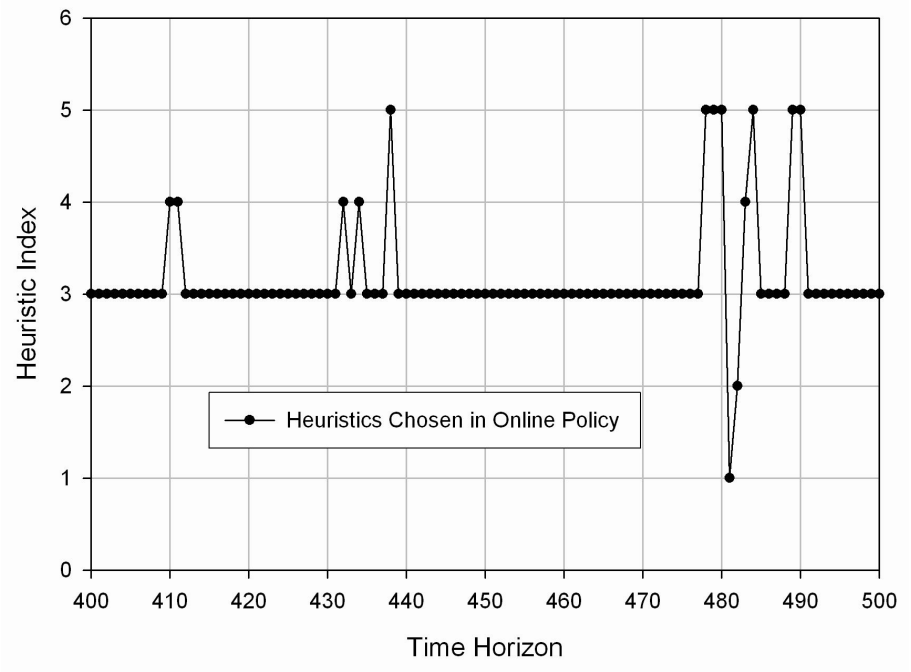

Figure 58: The Online Policy: Heuristics Taken for Time Horizon 400 to 500 in the Test Realization Set #2

CHAPTER VII

CONTRIBUTIONS AND FUTURE WORK

7.1 Contributions

The goal of this thesis is the development of a computationally tractable solution method for stochastic, stage-wise optimization problems. In order to achieve the goal, we have developed a novel algorithmic framework based on DP for improving heuristics. The proposed method represents a systematic way to take a family of solutions and patch them together as an improved solution. However, 'patching' is accomplished in state space, rather than in solution space.

In Chapter 3, a generalized version of the algorithmic framework, which can be applied to any deterministic/stochastic optimization problem formulated as a DP, is presented. Then, the efficiency of the proposed framework is verified by tailoring and applying the framework for deterministic/stochastic variants of traveling salesman problem.

In Chapter 4, the proposed framework is applied to a stochastic RCPSP, a real-world optimization problem with a *high dimensional state space* and significant uncertainty equivalent to billions of scenarios. The high dimensional state space is an inevitable consequence of DP formulation for the RCPSP, as we have pointed out in Section 1.2. The real-time decision policy obtained by the proposed algorithmic framework outperforms the best heuristic applied in simulation stage to form the policy. In Chapter 5, the proposed framework is applied to a RCPSP with new project arrivals which has *complicated state transition rules*, the second practical issue mentioned in Section 1.2. The complicated state transition increases the computational load for analytical state transitions, and eventually makes the Bellman iteration step of the framework computationally infeasible. To deal with the complicated state transition rules, we have adopted the idea of the Q-Learning approach, which enables us to build empirical state transition rules through simulation, into the proposed framework.

In Chapter 6, a stochastic supply chain management problem is addressed as an example of real-world problem with high dimensional state space and *high dimensional action space*, the last issue of practical importance mentioned in Section 1.2, it also has a high dimensional state space. The high dimensional action space, which describes astronomical numbers of discrete actions, increases the computational load of the Bellman iteration and real-time decision making enormously. We introduce the concept of an "implicit sub-action space", a systematic way of circumventing the high dimensional action space, and successfully added the idea to the proposed framework. The implicit sub-action space is defined with heuristic action operators mapping each state to corresponding discrete actions generated by the heuristics. Therefore, the implicit sub-action space provides significant computational benefits, in the Bellman iteration and in real-time decision making, by enumerating much smaller number of actions restricted by the heuristic operators, rather than all possible actions.

7.2 Future Work

There are a number of directions in which this thesis could be extended, including further computational improvements and a wider scope of applications.

- **Theoretical Proofs Improving Heuristics**

  Though the most important property of the proposed approach for, improving heuristics, is demonstrated via solving several practical applications in this thesis, the property has not been theoretically established. However, one should be able to show that the proposed approach can guarantee to improve heuristics in a given discrete state space under certain prerequisite conditions because of the underlying probability model for state action pairs. It should be noted that this guarantee will take the form of performance relative to the heuristics and not in an absolute sense with expect to the true optimal solution.

- **Efficient Bellman Iteration Scheme: Multi-Layered Bellman Iteration**

  Bellman iteration is a necessary step of the proposed approach, even if this takes place in a restricted state space. However, for large problems even the restricted state and action spaces can become large and the Bellman iteration slows down to an

unacceptable level. Therefore it would be beneficial to accelerate the Bellman iteration itself. The states in the restricted state space can be classified by the frequency they are visited which can be obtained during simulation. Since the Bellman iteration is performed over the restricted state space, 'Cost-to-Go' values of the frequently visited states are rapidly propagated with high conditional probabilities of realization. Thus, Bellman iteration over a subset of the states, consisting of the frequently visited states in the restricted state space, may converge faster than the Bellman iteration over the restricted state space and the converged cost-to-go values may be closer to the original cost-to-go values obtained by the Bellman iteration over the entire restricted state space. Once the 'cost-to-go' values are converged over the subset of the restricted state space, one may extend the subset and perform the Bellman iteration over the extended subset of the restricted state space. Since the cost-to-go values are already converged over the previous subset, the second Bellman iteration may converge faster over the extended subset. In same way, the Bellman iteration can be performed over the extended subsets of the restricted state space until it becomes same as the entire restricted state space. Extensive computational study and investigation of this tentative idea of 'multi-layered' Bellman iteration may lead to a systematic way of accelerating the Bellman iteration in the proposed framework.

- **Complicated Action Sampling: Adopting Deterministic Mathematical Programming Approach as A Heuristic**

  Complicated actions (decisions) are involved in real-world optimization problems under uncertainty. For example, the SCM problem introduced in Chapter 6 has a high-dimensional action space for simultaneous determination of the material flows in the supply chain system. As the decision structure gets complicated, in general, it is not easy to invent or choose reasonable heuristics for the problem. To obtain reasonable heuristics for stochastic optimization problems, one may utilize the deterministic solution method, mathematical programming, which can handle complicated decisions. With fixed problem parameters, the deterministic solutio method only provides a 'snapshot' solution that is only valid and feasible for a certain realization for to the

fixed problem parameters. Therefore, a single snapshot solution cannot be a robust action for stochastic optimization problem. However, a set of deterministic snapshot actions obtained by different realizations of the uncertain parameters can be a useful 'pool' of actions for stochastic optimization problems. The idea of utilizing mathematical programming was verified with a small size of SCM problem with grade transitions [20]. However, the idea has to be tested on more complicated, larger sized, stochastic optimization problems to be generalized.

REFERENCES

[1] Aarts, E. and Korst, J., *Simulated annealing and Boltzmann machines : a stochastic approach to combinatorial optimization and neural computing.* Wiley Press, 1st ed., 1989.

[2] Adjiman, C., Androulakis, I., and Floudas, C., "A global optimization method, abb, for general twice-differentiable constrined nlps - ii. implementation and computational results," *Computers and Chemical Engineering*, vol. 22, p. 1159, 1998.

[3] Adjiman, C., Dallwig, S., Neumaier, A., and Floudas, C., "A global optimization method, abb, for general twice-differentiable constrained nlps - i. theoretical advances," *Computers and Chemical Engineering*, vol. 22, p. 1137, 1998.

[4] Beaumont, N., "A generalization of binary variables," *Asia-Pacific Journal of Operational Research*, vol. 9, no. 2, pp. 177–181, 1992.

[5] Bellman, R., *Dynamic Programming.* Princeton University: New Jersey, 1st ed., 1957.

[6] Belvaux, G. and Wolsey, L., "Modelling practical lot-sizing problems as mixed-integer programs," *Management Science*, vol. 47, no. 7, pp. 993–1007, 2001.

[7] Bertsekas, D., *Dynamic Programming and Optimal Control*, vol. 1,2. Athena Scientific, Belmont, Massachusetts, 2nd ed., 1995.

[8] Bertsekas, D. and Tsitsiklis, J., *Neuro-Dynamic Programming*, vol. 1. Athena Scientific, 1st ed., 1996.

[9] Bhattacharjee, S. and Ramesh, R., "A multi-period profit maximizing model for retail supply chain management: An itegration of demand and supply-side mechanisms," *European Journal of Operational Research*, vol. 122, no. 3, pp. 584–601, 2000.

[10] Birge, J., "Decomposition and partitioning methods for multistage stochastic linear programs," *Operational Research*, vol. 33, pp. 989–1007, 1985.

[11] Birge, J. and Louveaux, F., "A multicit algorithm for two-stage stochastic linear programs," *European Journal of Operational Resesarch*, vol. 34, no. 3, pp. 384–392, 1988.

[12] Bisschop, J. and Roelfs, M., *AIMMS : The User's Guide.* Paragon Decision Technology, 3.5 ed., 2004.

[13] Blau, G., Mehta, B., Bose, S., Pekny, J., Sinclair, G., Kuenker, K., and Bunch, P., "Risk management in the development of new products in highly regulated industries," *Computers and Chemical Engineering*, vol. 24, no. 2-7, pp. 659–664, 2000.

[14] Blazewicz, J., Lenstra, J. K., and Kan, A. H. G. R., "Scheduling subject to resource contraints: Classification and complexity," *Discrete Appl. Maths*, vol. 5, pp. 11–24, 1983.

[15] Brooke, A., Kendrick, D., Meeraus, A., and Raman, R., *GAMS : A User's Guide*. GAMS Development Corporation, 1st ed., 1998.

[16] Buchan, J. and Koenigsberg, E., *Scientific Inventory Management*. Englewood Cliffs, NJ: Prentice-Hall, 1st ed., 1963.

[17] Chen, F., "Optimal policies for multi-echelon inventory problems with bactch ordering," *Operations Research*, vol. 48, no. 3, pp. 376–389, 2000.

[18] Cheng, L., E.Subrahmanian, and Westerberg, A., "Design and planning under uncertainty: issues on problem formulation and solution," *Computers and Chemical Engineering*, vol. 27, pp. 781–801, 2003.

[19] Cheng, T. and Kovalyov, M., "Single supplier scheduling for multiple deliveries," *Annals of Operations Research*, vol. 107, pp. 51–63, 2001.

[20] Choi, J., Lee, J., Lee, J., Realff, M., Lee, H., and Lee, J., "Optimization of a large scale, multi-product supply chain with grade transition operations under demand uncertainty," *AIChE 2004 Fall Meeting*, vol. Austin, TX, 2004.

[21] Choi, J., Lee, J., and Realff, M., "Simulation based approach for improving heuristics in stochastic resource-constrained project scheduling problem," *8th International Symposium on Process Systems Engineering, Kunming, China*, 2003.

[22] Choi, J., Lee, J., and Realff, M., "Simulation based approach for improving heuristics in stochastic resource-constrained project scheduling problem," *8th International Symposium on Process Systems Engineering*, pp. 439–444, 2003.

[23] Choi, J., Lee, J., and Realff, M., "An algorithmic framework for improving heuristic solutions part II : A new version of stochastic traveling salseman problem," *Computers and Chemical Engineering*, vol. 28, no. 8, pp. 1297–1307, 2004.

[24] Choi, J., Lee, J., and Realff, M., "Dynamic programming in a heuristically confined state space: A stochastic resorce-constrained project scheduling appplication," *Computers and Chemical Engineering*, vol. 28, no. 6-7, pp. 1039–1058, 2004.

[25] Choi, J., Lee, J., and Realff, M., "Dynamic programming in a heuristically confined state space: A stochastic resorce-constrained project scheduling appplication," *Computers and Chemical Engineering*, vol. 28, no. 6-7, pp. 1039–1058, 2004.

[26] Choi, J., Lee, J., Realff, M., Park, H., and Park, S., "Decision making under uncertainty," *AIChE Fall Annual Meeting, Reno, USA*, Nov 2001.

[27] Choi, J., Realff, M., and Lee, J., "An algorithmic framework for improving heuristic solutions part I : A deterministic discount coupon traveling salseman problem," *Computers and Chemical Engineering*, vol. 28, no. 8, pp. 1285–1296, 2004.

[28] Clark, A. and Scarf, H., "Optimal policies for a multi-echelon inventory problem," *Management Science*, vol. 6, pp. 475–490, 1960.

[29] Diks, D. and de Kok, A., "Optimal control of a divergent multi-echelon inventory system," *European Journal of Operational Research*, vol. 111, pp. 75–97, 1998.

[30] Floudas, C., *Nonlinear and Mixed-Integer Optimization : Fundamentals and Applications*. Oxford University Press, 1 ed., 1995.

[31] Garey, M. and Johnson, D., *Computers and interactability : a guide to the theroy of NP-completeness*. W.H. Freeman, 1st ed., 1979.

[32] Goldberg, D., *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Professional, 1st ed., 1989.

[33] Gooding, W., Pekny, J., and Mccroskey, P., "Eunymerative approaches to parallel flowshop scheduling via problem transformation," *Computers and Chemical Engineering*, vol. 18, no. 10, pp. 909–927, 1994.

[34] Gosavi, A., *Simulation-Based Optimization: Prametric Optimization Techniques and Reinforcement Learning*. Kluwer Academic Publishers, 1st ed., 2003.

[35] Grahovac, J. and Chakravarty, A., "Sharing and lateral transshipment of inventory in a supply chain with expensive low-demand items," *Management Science*, vol. 47, no. 4, pp. 579–594, 2001.

[36] Gupta, A. and Maranas, C., "Managing demand uncertainty in supply chain planning," *Computers and Chemical Engineering*, vol. 27, no. 8-9, pp. 1219–1227, 2003.

[37] Gupta, A., Maranas, C., and McDonald, C., "Mid-term supply chain planning under demand uncertainty: Customer demand satisfaction and inventory management," *Computers and Chemical Engineering*, vol. 24, pp. 2613–2621, 2000.

[38] Hall, N. and Potts, C., "Supply chain scheduling: Batching and delivery," *Operations Research*, vol. 51, no. 4, pp. 566–584, 2003.

[39] Harding, S. and Floudas, C., "Global optimization in multiproduct and multipurpose batch design under uncertainty," *Industrial and Engineering Chemistry Research*, vol. 36, no. 5, pp. 1644–1664, 1997.

[40] Higle, J. and Sen, S., "Stochaistic decomposition: An algorithm for two stage stochastic linear programs with recourse," *Mathematics of Operations Research*, vol. 16, no. 0, pp. 650–669, 1991.

[41] Ierapertritou, M. and Floudas, C., "Effective continuous-time formulation for short-term scheduling. 1. multipurpose batch processes," *Industrial and Engineering Chemistry Research*, vol. 37, no. 11, pp. 4341–4359, 1998.

[42] Ierapertritou, M. and Floudas, C., "Effective continuous-time formulation for short-term scheduling. 2. continuous and semicontinuous processes," *Industrial and Engineering Chemistry Research*, vol. 37, no. 11, pp. 4360–4374, 1998.

[43] Ierapetritou, M. G. and Pistikopoilos, E. N., "Batch plant design and operations under uncertainty," *Computers and Chemical Engineering*, vol. 24, pp. 2613–2621, 2000.

[44] Infanger, G., *Planning under uncertainty: Solving large scale stochstic linear programs*. Boyd and Fraser Publishing Co., 1st ed., 1994.

[45] Jackson, J. and Grossmann, I., "A disjunctive programming approach for the optimal design of reactive distillation columns," *Computers and Chemical Engineering*, vol. 25, no. 11-12, pp. 1661–1673, 2001.

[46] Jain, V. and Grossmann, I., "Resource-contrained scheduling of tests in new product development," *Industrial and Engineering Chemistry Research*, vol. 38, no. 8, pp. 3013–3026, 1999.

[47] Kalivas, J., *Adaption of Simulated Annealing to Chemical Optimization Problems (Data Handling in Science and Technology, Vol 15)*. Elsevier Publishing Company, 1st ed., 1995.

[48] Kall, P. and Wallace, S., *Stochastic Programming*. Willey, 1st ed., 1994.

[49] Kondili, E., Pantelides, C., and Sargent, R., "A general algorithm for short-term scheduling of batch operations-i. milp formulation," *Computers and Chemical Engineering*, vol. 17, no. 2, pp. 211–227, 1993.

[50] Kushner, H. and Yin, G., *Stochastic Approximation Algorithms and Applications*. New York: Springer, 2nd ed., 1997.

[51] Lawler, E. and Eugene, L., *The Traveling Salesman Problem : A guided Tour of Combinatorial Optimization*. Wiley Press, 2nd ed., 1985.

[52] Lee, S. and Grossmann, I., "Generalized convex disjunctive programming: Nonlinear convex hull relaxation," *Computational Optimization and Applications*, vol. 26, no. 1, pp. 83–100, 2003.

[53] Maravelias, C. and Grossmann, I., "Simultaneous planning for new product development and batch manufacturing facilities," *Industrial and Engineering Chemistry Research*, vol. 40, no. 26, pp. 6147–6164, 2001.

[54] Noon, C. and Bean, J., "An efficient transformation of the generalized traveling salesman problem," *INFOR*, vol. 31, no. 1, pp. 39–44, 1993.

[55] Norkin, V., Ermoliev, Y., and Ruszczynski, A., "On optimal allocation of indivisibles under uncertainty," *Operations Research*, vol. 46, pp. 381–395, 1998.

[56] Pekny, J., Miller, D., and Kudva, G., "An exact algorithm for resource constrained sequencing with application to production scheduling under an aggregate deadline," *Computers and Chemical Engineering*, vol. 17, no. 7, pp. 671–682, 1993.

[57] Penky, J. and Miller, D., "Exact solution of the no-wait flowshop scheduling problem with a comparison to heuristisc methods," *Computers and Chemical Engineering*, vol. 15, no. 11, pp. 741–748, 1991.

[58] Percus, A. and Martin, O., "The stochastic traveling salesman problem : Finite size scaling and the cavity," *Journal of Statistical Physics*, vol. 94, no. 5-6, pp. 739–758, 1999.

[59] Petkov, D. and Maranas, C., "Multiperiod planning and scheduling of multiproduct batch plants under uncertainty," *Industrial and Engineering Chemistry Research*, vol. 36, no. 11, pp. 4864–4881, 1997.

[60] Pinto, J. and Grossmann, I., "Assignment and sequencing models for the scheduling of process systems," *Annals of Operations Research*, vol. 81, pp. 433–466, 1998.

[61] Puterman, M., *Markov Decision Processes*. Wiley Interscience, 3rd ed., 1994.

[62] Raman, R. and Grossmann, I., "Modeling and computational techniques for logic-based integer programming," *Computers and Chemical Engineering*, vol. 18, no. 7, pp. 563–578, 1994.

[63] Recklaitis, G., "Overview of scheduling and planning batch process operations, technical report," *NATO Advanced Study Institute*, 1992.

[64] Rhee, W. and Talagrand, M., "A sharp deviation inequality for the stochastic taveling saleman problem," *Annals of Probability*, vol. 17, no. 1, pp. 1–8, 1989.

[65] Rockafellar, R. and Wets, R., "Scenarios and policy aggregation in optimization under uncertainty," *Mathematics of Operations Research*, vol. 16, no. 1, pp. 119–147, 1991.

[66] Rogers, M., Gupta, A., and Maranas, C., "Real options based analysis of optimal pharmaceutical research and developement portfolios," *Industrial and Engineering Chemistry Research*, vol. 41, pp. 6607–6620, 2002.

[67] Sabri, E. and Beamon, B., "A multi-objective apporach to simultaneous strategic and operational planning in supply chain design," *Omega-International Journal of Management Science*, vol. 28, no. 5, pp. 581–598, 2000.

[68] Sahinidis, N., "Baron: A general purpose global optimization software package," *Journal of Global Optimization*, vol. 8, no. 2, pp. 201–205, 1996.

[69] Schmidt, C. and Grossmann, I., "Optimization models for the scheduling of testing tasks in new product development," *Industrial and Engineering Chemistry Research*, vol. 35, no. 10, pp. 3498–3510, 1996.

[70] Schultz, R., Stougie, L., and van der Vlerk, M., "Solving stochastic programs with integer recourse by enumeration: A framework using grobner basis reductions," *Mathematical Programming*, vol. 83, pp. 229–252, 1998.

[71] Shah, N., "Single-and multisite planning and scheduling : Current status and future chellenges," *3rd International Conference on Foundations of Coputer-Aided Process Operations, AIChE Symposium Series*, vol. 94, no. 320, pp. 75–90, 1998.

[72] Shah, N., Pantelides, C., and Sargent, R., "A general algorithm for short-term scheduling of batch operations-ii. computational issues," *Computers and Chemical Engineering*, vol. 17, no. 2, pp. 229–244, 1993.

[73] Shapiro, A. and Wardi, Y., "Convergence analysis of gradient decent stochastic algorithm," *Journal of Operation Theory and Applications*, vol. 91, pp. 439–454, 1996.

[74] Smith, E. and Pantelides, C., "Global optimisation of nonconvex minlps," *Computers and Chemical Engineering*, vol. 21, no. S, pp. S791–S796, 1997.

[75] Subrahmanyam, S., Pekny, J., and Reklaitis, G., "Design of batch chemical plant under market uncertainty," *Industrial and Engineering Chemistry Research*, vol. 33, p. 2688, 1994.

[76] Subramanian, D., Pekny, J., and Recklaitis, G., "A simulation-optimization framework for research and development pipeline management," *AIChE Journal*, vol. 47, no. 10, pp. 2226–2241, 2001.

[77] Subramanian, D., Pekny, J., and Reklaitis, G., "A simulation-optimization framework for addressing combinatorial and stochastic aspects of an *r&d* pipeline management problem," *Computers and Chemical Engineering*, vol. 24, pp. 1005–1011, 2000.

[78] Subramanian, D., Pekny, J., and Reklaitis, G., "Simulation-optimziation framework for stochastic optimization of r&d pipeline management," *AIChE Journal*, vol. 49, no. 1, pp. 96–112, 2003.

[79] Sutton, R. and Barto, A., *Reinforcement Learning*. The MIT Press, 3rd ed., 2000.

[80] Tsiakis, P., Shah, N., and Pantelides, C., "Design of multi-echeon supply chain networks under demand uncertainty," *Industrial and Engineering Chemistry Research*, vol. 40, no. 16, pp. 3585–3604, 2001.

[81] Turkay, M. and Grossmann, I., "Logic-based minlp algorithms for the optimal synthesis of process networks," *Computers and Chemical Engineering*, vol. 20, no. 8, pp. 959–978, 1996.

[82] Vasantharnjan, S., Viswanathan, J., and Biegler, L., "Reduced successive quadratic-programming implementation for large-scale optimization problems with smaller degrees of freedom," *Computers and Chemical Engineering*, vol. 14, no. 8, pp. 907–915, 1990.

[83] Verrijdt, J. and de Kok, A., "Distribution planning for a divergent n-echelon network without intermediate stocks under service restrictions," *International Journal of Production Economics*, vol. 38, pp. 225–243, 1995.

[84] Vin, J. and Ierapetritou, M., "Robust short-term scheduling of multiproduct batch plans under demand uncertainty," *Industrial and Engineering Chemistry Research*, vol. 40, no. 21, pp. 4543–4554, 2001.

[85] Viswanathan, J. and Grossmann, I., "A combined penalty-function and outer-approximation method for minlp optimization," *Computers and Chemical Engineering*, vol. 14, no. 7, pp. 769–782, 1990.

[86] Watkins, C., "Learning from delayed rewards," *Ph.D. Thesis, Cambridge University*, 1989.

[87] Weng, Z. and McClurg, T., "Cordinated ordering decisions for short life cycle products with uncertainty in delivery tiem and demand," *European Journal of Operational Research*, vol. 151, pp. 12–24, 2003.

[88] Wets, R., "Stochastic programs with fixed recourse: The equivalent deterministic program," *SIAM Review*, vol. 16, pp. 309–339, 1974.

[89] Winston, W., *Operations Research : Applications and Algorithms*. Duxbury Press, 3rd ed., 1993.

[90] Yin, K., Liu, H., and Johnson, N., "Markovian inventroy policy with application to the paper industry," *Computers and Chemical Engineering*, vol. 26, pp. 1399–1413, 2002.

VITA

Jaein Choi was born in Seoul, Korea on November 23rd, 1976. He graduated from Hansung Science High School in 2 years, 1994. Then he had attended Korea Advanced Institute of Science and Technology (KAIST) at Teajon, Korea in 1994 until he obtained a B.S. degree as the 1st ranked student in Department of Chemical Engineering in February 1998. He then continued a graduate study in Process Systems Laboratory at KAIST until he graduated in August 2000 with a M.S. in Chemical Engineering. In August 2000, he attended Georgia Institute of Technology at Atlanta, Georgia. From May to August 2004, he worked for Owens Corning as an engineering intern at 'Modeling, Control, and Optimization' team in Manufacturing Technology Center, Granville, Ohio. His dissertation title was "Algorithmic Framework for Improving Heuristics in Stochastic, Stage-Wise Optimization Problems". He defended his thesis on November 19th, 2004 and obtained his Ph.D. in Chemical and Biomodecular Engineering with a Minor in Stochastic Optimization on December 11th, 2004.