UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

AN EXAMINATION OF MULTIPLE OPTIMIZATION APPROACHES TO THE SCHEDULING OF MULTI-PERIOD MIXED-BTU NATURAL GAS PRODUCTS

A DISSERTATION

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

degree of

Doctor of Philosophy

By

MICHAEL A. BOND
Norman, Oklahoma
2013

AN EXAMINATION OF MULTIPLE OPTIMIZATION APPROACHES
TO THE SCHEDULING OF MULTI-PERIOD
MIXED-BTU NATURAL GAS PRODUCTS

A DISSERTATION APPROVED FOR THE
SCHOOL OF INDUSTRIAL AND SYSTEMS ENGINEERING

BY

_____
Dr. Floyd H. Grant, Chair

_____
Dr. Faruk Civan

_____
Dr. Shivakumar Raman

_____
Dr. Binil Starly

_____
Dr. Theodore B. Trafalis

Acknowledgments

I would like to thank my committee members: Dr. Grant, Dr. Civan and Dr. Starly, Dr. Raman and Dr. Trafalis. They have all provided important insights and direction to my research. I cannot express how much I appreciate Dr. Grant's time. His guidance, friendship and most of all, patience has been invaluable throughout this time.

To my colleagues and the management of Science Applications International Corporation, Inc., I owe much. I would like to thank all of my friends who supported me through this. Don Essary deserves special thanks for supplying the computer hardware that was used on this project.

My parents have always supported my endeavors and to them I offer my gratitude and love. I would also like to thank my children, Erin, Cory, Daniel, Jackson, Olivia, Elizabeth, Joseph and Casey for their support and patience. Thank you for allowing me to read papers and do homework at soccer games, concerts, etc.

Finally, I cannot express my gratitude to my loving wife and best friend, Terri, whose tireless support, encouragement and patience made this possible.

Table of Contents

List of Tables

List of Figures

Abstract

As worldwide production and consumption of natural gas increase, so does the importance of maximizing profit when trading this commodity in a highly competitive market. Decisions regarding the buying, storing and selling of natural gas are difficult in the face of high volatility of prices and uncertain demand. With the introduction of alternative sources of fuels with lower levels of methane, the primary component of natural gas, these decisions become more complicated. This is an issue faced by investors as well as operational planners of industrial and commercial consumers of natural gas where incorrect planning decisions can be costly.

A great deal of research in the academic and commercial arenas has been accomplished regarding the problem of optimizing the scheduling of injection and withdrawal of this commodity. While various commercial products have been in use for years and research on new approaches continues, one aspect of the problem that has received less attention is that of combining gases of different heat contents. This study examines multiple approaches to maximizing profits by optimally scheduling the purchase and storage of two gas products of different energy densities and the sales of the same in combination with a product that is a blend of the two. The result provides an initial basis for planners to improve decision making and minimize the cost of natural gas consumed.

This multi-product multi-period finite (twelve-month) horizon product-mix problem is NP-Hard. The first approach developed is a Branch and Bound

(B&B) technique combined with a linear program (LP) solver. Heuristics are applied to limit the expansion the trinomial tree generated. In the second approach, a stochastic search algorithm-linear programming hybrid (SS-LP) is developed. The third approach implemented is a pure random search (PRS). To make each technique computationally tractable, constraints on the units of product moved in each transaction are implemented.

Then, using numerical data, the three approaches are tested, analyzed and compared statistically and graphically along with computer performance information. The best approach provides a tool for optimizing profits and offers planners an advantage over approaches that are solely history-based.

Chapter 1

Introduction

1.1 Overview

This dissertation describes the problem of planning natural gas scheduling and the experiment to examine multiple techniques to solve it. Chapter One contains the introductory material. It begins with this overview which is followed by a statement of the problem. The purpose of research and the questions researched are discussed, followed by a discussion of the significance of this study and the conceptual framework. The chapter's final section contains a summary of the methodology used and a discussion of the limitations that were observed in the research.

Chapter two presents a summary of the research found in the literature. Subjects relevant to this research were reviewed beginning with general concepts of natural gas and heat content. This chapter includes a review of various optimization techniques including the use of simulation and the combined simulation-optimization. Specific to this research is the section on natural gas scheduling optimization and the various approaches to this problem. The chapter concludes with a brief review of computer performance studies.

Chapter three details the research design and methodology including the approach taken, the procedures applied and the hardware upon which it was executed. The problem examined in this research is considered to be NP-Hard,

meaning that finding an exact answer would be very hard, if not impossible. Optimal or, rather, near-optimal solutions to NP-Hard problems are usually found through a random search technique of one type or another. This chapter describes the three approaches developed and tested. The Branch and Bound/Linear Programming hybrid algorithm was developed to examine all possible decisions during the twelve-month horizon. The discussion here includes the limitations and heuristics placed on this algorithm to make it computationally tractable. That approach was succeeded by a combination of a evolutionary search and Linear Programming. The details of the final approach, a Pure Random Search algorithm without Linear Programming, follow that. The development of each approach is a logical step from its predecessor, gaining in flexibility but becoming more expensive computationally.

Chapter Four reports and discusses the findings of this research. The data used is discussed. The results and analysis are presented.

There are still various avenues of research open on this topic. Chapter five includes the conclusions of the research and recommendations for further study.

1.2 Statement of the Problem

Investors in natural gas seek to maximize profit by taking advantage of the seasonal low and high prices. Decisions regarding buying, storing and selling natural gas are difficult in the face of high variability of prices and uncertain demand. Various management strategies exist. Buyers of natural gas

use various techniques for planning the buying, storage and selling of the product. These techniques are discussed in depth in Chapter Two.

This paper describes our approach of combining simulation and linear programming to optimize the selection process. While the focus is multi-cavern salt dome storage facilities, which have faster inventory turnover rates than the more common reservoir storage facilities, it is recognized that not all gas discussed in this paper is stored in such facilities (FERC 2004).

With economical stresses and increased emphasis on the protection of Earth's environment, the use of natural gas from alternate sources has increased. In many cases, such gas contains a lower energy content or Btu level, and while it may not be economically feasible to remove the impurities, it may still be desirable to use the gas rather than simply burning or 'flaring' it. Further complicating the problem is that the price curves of gas from different sources may not follow the same cost and price curves.

Consumers and investors seek a means of executing the planning process in the presence of gases of differing energy content levels.

1.3 Purpose

The primary purpose of this research is to acquire knowledge of techniques for optimizing the scheduling of buying, storage and selling of natural gas inventories of differing heat contents, specifically to maximize profits or minimize costs in these operations. Much work has been done in the area of scheduling standard pipeline-ready gas, but there exists a gap in the

literature regarding mixed content gas. This problem has nuances that differentiate it from existing research on mixed-product problems. The nature of natural gas and how it is stored separates it from other commodities. As the consumption of low-Btu gas increases, this will be a more important process.

Another purpose is to investigate the performance of different approaches to this problem. The combinatorial nature of this problem lends to solutions that tend to be computationally intensive. Technological advances continually increase the computational power available to the researcher; nevertheless, researchers and practitioners continue to seek more efficient and accurate ways to find better solutions to problems of this type. The balance between the number of variables examined and the amount of time taken to generate the solution and the accuracy of that solution are examined.

1.4 Research Questions or Hypotheses

This research investigates the combined use of branch and bound techniques and linear programming, specifically the Simplex Method, in finding a best or near-optimal solution to the mixed gas scheduling problem. Having developed and exercised the techniques, the results are then compared to random search algorithms developed both in conjunction with and without linear programming. These are all extensions and hybridizations of existing optimization techniques, all of which are discussed in chapter two of this dissertation. The Branch & Bound and simplex hybrid algorithm is compared to a stochastic optimization process.

The second area of investigation is the computational requirement of each method. This study examines the amount of time required to find a suitable solution and relates it to the quality of that solution.

1.5 Significance of the Study

The contribution this study makes in two areas lends to its significance. First, it adds to the research in this field by contribution to the study of mix-product natural gas scheduling. It provides initial information regarding the optimization of natural gas storage and scheduling, a logistical and financial problem that has been studied a long time and will continue to be investigated. Simply put, it investigates ways to maximize profits when buying and selling natural gas.

Secondly, it adds to the body of environmental studies work. Methane is the primary component of natural gas and is present in other bio-generated gases. It is considered to be a contributor to global warming and is seen as a pollutant when released into the atmosphere. As more and more low-Btu gas is captured to be used rather than released into the environment or 'flared', i.e. burned, it is useful to know how best to use it.

With the increasing importance of producing energy and with technology that makes it more affordable to do so, gas that would in the past have been economically infeasible to process may be produced. Also, due to increased emphasis on environmental concerns, low-BTU gas, such as landfill gas is

becoming available for consumption. Operators of facilities sometimes wish to combine gases of two different Btu levels to achieve an intermediate product.

1.6 Conceptual Framework

Figure 1 illustrates the flow of the product, natural gas, from source to consumption. The source may be the wellhead, another storage location or an alternative source of methane. Gas is purchaed and transferred into storage where it is held until sold or used.

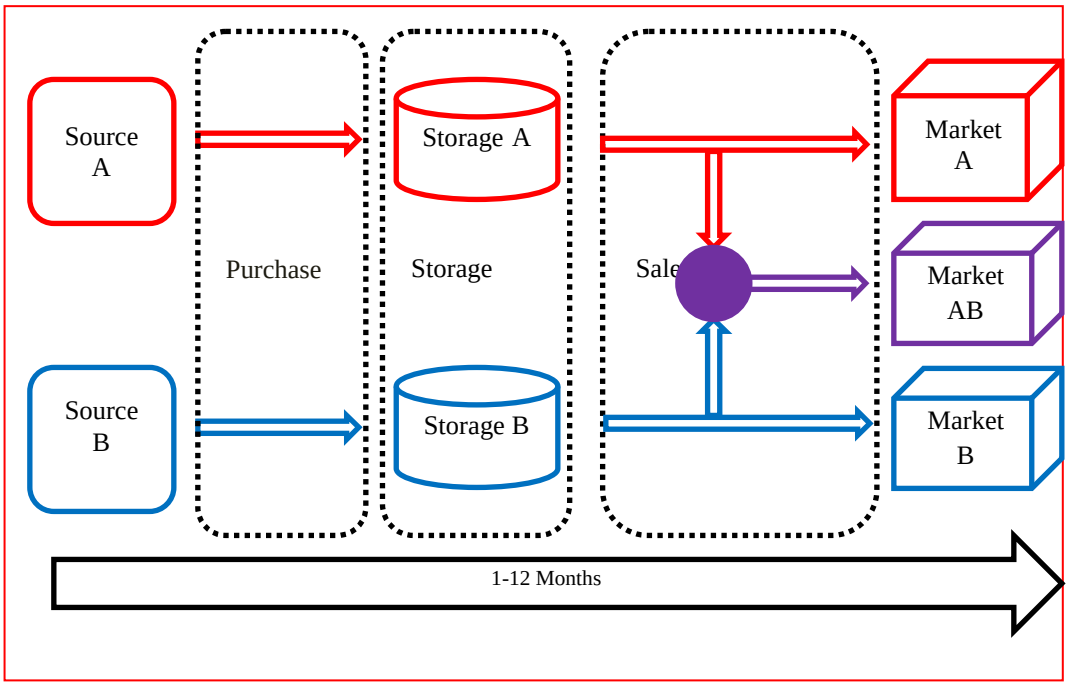


Figure 1 Conceptual Framework

In this model, consumption is considered equivalent to selling the gas for the current market price, the "spot price".

1.7 Summary of Methodology

The research examined the use of simulation optimization techniques in combination with linear programming to make optimal scheduling decisions

regarding holding times, product mix values, product injection and withdrawal schedules and transactional quantities. In addition to simplified test data designed to provide clear demonstrations of functionality and accuracy, we used price and cost data from past years as input for natural gas data and estimated landfill/low-Btu gas prices based on current trends and prices.

A twelve-month time horizon was used since many gas storage contracts are of that length. We used the simulation package Awesim and modules written in Microsoft C++ to generate scenarios and in some cases used a linear programming package to provide the economical product mix and then evaluated the results.

Three approaches were developed and compared. A Branch and Bound (B&B) algorithm combined with linear programming (LP), B&B-LP, was developed. This hybrid was implemented as a recursively created trinomial decision tree with options to buy, hold or sell gas at each node. Heuristics were applied to limit the number of node per branch and thereby make the algorithm more computationally tractable.

An evolutionary stochastic search algorithm (SS-LP) in combination with LP was developed. This direct stochastic search algorithm implemented the same heuristics as the B&B-LP hybrid.

Finally, in order to compare the computational efficiency of the LP solver versus pure random search of the solution space, a Pure Random Search (PRS)-based approach (SS) was developed. This approach did not operate under

the same bounds as the B&B-LP and SS-LP algorithms. It used more relaxed search criteria.

1.8 Limitations

Current modeling techniques contain many factors that affect the present and future value of gas in storage. These include but are not limited to parameters such as the cost of money (value of risk-free investment) and rate of inflation. The model used in this study did not contain all of these factors. That will be possible in future studies.

Current approaches to this problem may include any of many optimization techniques. These techniques are examined in chapter two. These may also be used in future research.

1.9 Definition of Terms

Natural Gas

Natural Gas is an odorless, colorless fossil fuel comprised primarily of methane, CH_4, but may also contain ethane, propane, butane, pentane, helium and hexane. It may be found in association with other fossil fuels (EIA 2012a). It is sold on residential, commercial, industrial and energy generation markets.

Heat content

The heat content or heat of combustion is the energy released when a substance undergoes combustion with oxygen under standard conditions, 60ºF

and 14.696 psia (Civan 2008). This may be known as heat of combustion, heating value or calorific value. Units are expressed as heating value per unit mass or volume. British thermal unit (Btu) per cubic foot is a common measurement of natural gas (NIST 2010). This value is typically expressed in units or energy per unit mass, (which may be expressed as volume for gasses at standard conditions). To compare the heat content of natural gas to that of other common fuels, refer to Table 1.

Fuel	Phase	Btu/ft^3
hydrogen	gas	324
landfill gas	gas	497
methane	gas	1,009
ethane	gas	1,768
propane	gas	2,516
butane	gas	3,263

Table 1 Higher Heat Value of Common Fuels (NIST 2010)

Low-BTU Gas

Natural gas that, as taken from underground, contains a significantly lower energy content than that of typical gas is known as low-Btu gas. The energy content may be as low as 500 Btu/ft^3 and present a challenge to engineers to recover, process and market economically (Newell et al., 2009).

Landfill Gas

Landfill gas is generated by the decomposition of organic material. While it is not natural gas, landfill gas is composed of approximately 50% methane, the primary ingredient of natural gas. Landfill gas also contains carbon dioxide, CO_2, and water vapor, H_2O. The gas may be collected and sold via small pipeline to local consumers (EPA 2012).

Chapter 2

2.1 Literature Review

This chapter summarizes the relevant literature for this dissertation and explains the background, concepts and basic methods used throughout this research. The literature review is presented in the following categories: general concepts, optimization, simulation optimization, and evaluation of performance.

2.2 General Concepts

This section presents the relevant subject matter background information to understand the problem and discusses the concepts needed to adequately understand the methods proposed in this dissertation.

2.1.1 Natural Gas

Natural gas is a non-renewable fossil fuel found underground and is commonly, though not always, associated with oil deposits. It is a major source of energy in the United States, supplying energy to residential, commercial, and industrial and power generation facilities. In 2010, US consumption reached 23 trillion cubic feet (EIA 2012). The US consumed all that it produced and imported another 4.6 tcf of gas via pipeline from Mexico and Canada or as liquefied natural gas-- gas chilled to -260 degrees Fahrenheit, the point of becoming liquid-- from various exporters (EIA 2006).

To collect gas from underground reservoirs, locations are evaluated and wells are drilled and prepared, and then gas, under geological pressure, flows up

through the wellhead and through a system of gathering lines to a field processing unit. From there, the gas stream may go to another processor for further treatment such as the removal of sulfur, other hydrocarbons or helium, for example. After it is market-ready, the gas is pumped through interstate pipelines to local distributors and back into smaller interstate pipelines to be distributed to users (EIA 2006).

The Federal Energy Regulatory Commission (FERC 2008) issued order no. 436 in 1985, no. 500 in 1987 and no. 636 in 1993 (Busby 1999). These orders uncoupled production and distribution and, as a result, storage facilities became opportunities for profit by gas owners and investors. Gas futures and options trading began on the New York Mercantile Exchange (NYMEX) on 3 April, 1990. These events have shaped the way gas in storage is assigned a financial value. Since then, owners of gas in storage have had the option to buy or sell some amount of the commodity each day.

A gas supplier may use a combination of tools to ensure that demands are met. A combination of long- and short-term contracts, vertical upstream integration, buying on the spot market, and the utilization of storage facilities are often employed. A supplier practices vertical upstream integration by acquiring or investing in oil and gas production companies. Gas may also be kept in storage for speculative reasons and as a precaution against short-term demand fluctuations (Baranes,et al. 2009).

Due to the nature of gas production and the time and capital investment required to bring new sources to market, current production cannot increase to

meet fluctuations in short-term demand. Therefore, gas must be kept in storage to meet seasonal increases in demand. Other than a relatively small amount of gas stored in aboveground containers by local distributors to buffer against peaks in daily demands, natural gas is stored underground. There are three major types of underground storage: depleted gas reservoirs, aquifers, and salt-caverns (Tek 1996).

Of the three types of underground natural gas storage facilities, the most common in North America is the depleted reservoir. After all recoverable gas has been extracted from a natural deposit, that reservoir may then be used as a storage location for the processed product. There are several advantages to using reservoirs. The wells, gathering systems and pipeline connections are already in place and the geology of the area is known. There are also disadvantages to this storage type. Since the formations have previously held hydrocarbons that have 'sealed' the formation, there is a requirement for more monitoring. The choices of storage field location and performance are limited by the inventory of depleted fields in any region (EIA 2006). The depleted reservoir is both the least expensive to develop and the fastest with a conversion time of 24-36 months.

Working gas or "top gas" is the volume of gas in the storage facility that is accessible for extraction. Gas that is present in a storage facility but is not accessible is called cushion gas or base gas. Base gas provides the pressure for the withdrawal of top gas. Top gas and cushion gas are the same mixture of

hydrocarbons in the storage facility? Top gas just indicates how much you can withdraw (FERC 2004).

Unlike storage facilities or warehouses for other commodities or materials, two salt caverns of the same volume may have different maximum capacities depending on their depth. The pressure at which gas is stored relates to the injection and withdrawal rates (Bagci and Ozturk, 2007). It can be approximated with a piece-wise linear function gas (Padberg and Haubrich, 2008).

2.1.2 Heat Content

One important attribute of all combustible fuels is the heat of combustion, the amount of heat released when that substance undergoes complete combustion with oxygen (Civan 2008). In its pure form, methane, CH_4, has an energy content of 23, 900 British Thermal Units (Btu's) per pound. Wood, by comparison, has roughly 6000 Btu's/lb (NIST 2010). One Btu is the amount of energy required to raise the temperature of one pound of water one degree Fahrenheit at standard temperature and pressure (UNCTAD 2011).

Natural gas, though mostly methane, contains other hydrocarbons such as ethane, propane, and butane or other impurities which may increase or, more likely, decrease the heat content. While the range of heat content may range from 500 to 1500 Btu/ft3, most gas has a heat content value in the range of 900 to 1100 Btu/ft^3. The average for gas produced in the US in 1995 was 1028 Btu (DOE 1995). Before being transported via the US interstate pipeline systems, gas must have a heat content of between 1030 and 1060 Btu/ft^3. Gas with a

higher value could pose a safety hazard by producing hotter flames. In other locations, Germany, for example, separate distribution systems exist for "High" or "Low" quality gas (Padberg and Haubrich, 2008).

At times, therefore, it is necessary to adjust the heat content of gas before shipping it. One method of lowering heat content is to dilute the gas with an inert substance such as air or nitrogen. This may also be accomplished by combining it with a gas of a different content resulting in a gas in the desired range. An example of this may be found in Lake Charles, Louisiana, in the southern United States where the Southern Union Company combines high quality imported gas with the locally produced lower quality product (DOE 2005).

2.3 Optimization

Optimization theory consists of a collection of techniques and methods that make it possible to find the best solution to a problem without actually examining each and every possible solution (Ravindran 2006). Table 2 shows the categories of techniques as they appear in the literature.

Optimization Problems		
Local Optimization		Global Optimization
Discrete Decision Space	Continuous Decision Space	Evolutionary Algorithms Tabu Search Simulated Annealing Bayesian/Sampling Algorithms Gradient Surface Method Model Reference Adaptive Search
Ranking and Selection Multiple Comparisons Ordinal Optimization Random Search Simplex/Complex Search Single Factor Method Hooke-Jeeves Pattern Search Complete Enumeration	Response Surface Methodology Finite Difference Estimates Perturbation Analysis Frequency Domain Analysis Likelihood Ratio Estimates Stochastic Approximation	

Table 2 Classification Scheme (Tekin and Sabuncouglu, 2004)

There are other categorization schemes for optimization techniques. Zlochin et al. (2004), for example, classifies continuous and combinatorial problems as either *instance-based* or *model-based*. The selection of solution candidates in instance-based approaches is based directly on the results of the previous solution searches. This group includes simulated annealing, genetic algorithms, Tabu search, and nested partitions. Model-based approaches, introduced more recently, tend to have two phases: (1) generate candidate solutions by randomly selecting from the solution space and (2) use the results to update the model so that it will be more likely to generate a new, higher quality candidate solution. The ant colony optimization (ACO) method, cross-

entropy (CE) and estimation of distribution algorithms (EDAs) methods are commonly used examples of model-based solutions (Hu, et al., 2007).

Model-based approaches, according to Hu, et al. (2011), are more robust, more easily parallelized and have been successfully applied to many difficult optimization problems.

2.4.1 Linear Programming

Linear programming is one of the most commonly used optimization techniques. First published in 1947 by Dantzig who developed the Simplex method and Neumann who developed the duality theorem, this mathematical approach to finding the best outcome of a linear objective function is used widely in operations research (Anderson et al. 2006). Avery, et al. (1992) used linear programming to model the purchase, storage and transmission contracts for natural gas utilities.

Many optimization problems are solved with a combination of simulation and linear programming. Arbib, et al. (2012) combined linear programming and a Tabu search algorithm to solve a cutting process problem. In this project, they generated possible solutions randomly, eliminated and marked inferior ones, and evaluated potential optimum solutions with linear programming.

2.4.2 Branch and Bound

Branch and Bound (B&B) is one of the most widely used approaches to optimizing NP-Hard combinatorial problems. For discrete problems with a

large number of variables, it is usually impractical to attempt to enumerate all possible combinations. A B&B algorithm searches the entire solution space but reduces processing by setting bounds on the combinations that will be examined (Papadimitriou and Steiglitz 1982). The solution space is typically represented as a multi-node tree, with each branch representing a decision point. If, at any point in the evaluation, a branch of the tree fails to meet a threshold value, it is pruned from the tree and as a result, its sub-branches are removed from further processing (Clausen 1999).

While B&B is conceptually simple, it is not without its limitations. Although it can produce an exact optimum, with larger problems the amount of computer time required to find that solution may be too great to be useful. Without careful pruning, the number of bud nodes on a tree increases exponentially. Mousavi et al. (2012) found this to be true, that the performance of B&B compared to a genetic or simulated annealing algorithm, which produced *near*-optimal solutions, was significantly poorer.

2.4.3 Nested Partitions

Developed by Shi and Olaffson in the 1990's, Nested partition (NP) is a randomized method for finding an optimal or near-optimal solution in a finite feasible region. This method seeks a solution by dividing the feasible solution space into subregions and selecting one of those as the potential location for the optimal solution. NP evaluates and ranks each subregion by randomly selecting points from it and evaluating them as the solution. The region with the best ranking is then examined in the same way (Shi, et al. 2000).

In practice, it is common to customize the subregion search function to the problem being solved. Yau, et al. (2009) used NP on a large-scale job shop problem and developed a weighted sampling function. Wei, et al. (2012) applied a variation of NP framework to the flexible resource flow shop problem. By modifying the random search portion of the algorithm, they achieved better performance than the generic NP provided.

2.4.4 Random Search

Random Search (RS) or Pure Random Search (PRS) can be used and works on an infinite parameter space when it is not possible to evaluate every possible solution. This is the general case of random solution searches, being performed without any heuristics or rules for reducing the set of solutions. The process ends after a predetermined number of searches have been completed, a limit of computer resources has been reached or an acceptable solution has been found. This process performs best when a neighborhood can be defined in the solution space (Olafsson and Kim, 2002). PRS has the advantage of avoiding local maxima. While it has been applied primarily to discrete problems, its closely related technique, sample path optimization, is practiced on continuous problems (April et al. 2003). While it can be shown that RS will converge to a near-optimal solution (Shi et al. 2000), one problem with this approach is the slow speed at which convergence is reached (Tekin and Sabuncouglu, 2004).

The existence of other, more guided approaches notwithstanding, this approach does find use in practice. Poland, et al. (2011) applied a PRS

algorithm to a smart home sensor placement problem and found that in 98.4% of test cases this approach produced superior results.

2.4.5 Nelder-Mead

Proposed in 1965, the Nelder-Mead (N-M) method, also known as the downhill simplex method, is a technique of minimizing an n+1 variable objective function in an n-dimensional parameter space without constraints. In each iteration, the worst point in the simplex is dropped and replaced by the reflective one across the centroid, the center of the remaining feasible solution space. The complex search method is a variation of the simplex method in which an effort is made to keep the centroid in the feasible area, i.e. constraints exist (Nelder and Mead, 1965). This heuristic can converge on non-stationary points

This technique has been modified or hybridized various times through the years. Liu, et al. (2012) and Baghmisheh, et al. (2012) both created particle swarm-Nelder-Mead hybrid optimization approaches and applied them to different problems. Kuriger and Grant (2010) presented a Lexicographic Nelder-Mead based method to solve multi-criteria optimization.

2.4.6 Other Direct Search methods

The Single-Factor Method (SFM) and Hooke-Jeeves Pattern Search Method (H-J) are both direct search techniques that may be applied over an infinite parameter space. SFM holds all parameters constant and moves one. H-J varies one of a set of theoretical parameters at a time and examines the

response. The magnitude of the parameter change decreases until the steps are deemed sufficiently small (Hooke and Jeeves, 1961). This method is often used in conjunction with other methods (Azadivar 1999).

2.4.7 Frequency Domain Analysis

Frequency Domain Analysis (FDA) screens factors in a simulation by oscillating the value of a parameter during simulation. The oscillation follows a sinusoidal function and gives an idea of the relative sensitivity of the parameter (Tekin, et al., 2004). This technique has drawbacks and is not frequently observed in recent literature.

2.4.8 Response Surface Methodologies

Response Surface Methodology (RSM), another continuous decision space approach, examines the relationship between multiple explanatory, i.e. independent, variables and subsequent response or dependent variables. In the context of simulation optimization, RSM is a representation of the response or value of an objective function as the input factors or variables are changed through simulation (April et al., 2003). RSM is used in two phases. In the first phase, a first-order model is fitted to the response surface and the steepest descent direction is estimated. This repeats until the slope nears zero, at which point the first order design is no longer a good fit. In the second phase a quadratic response surface is generated and the optimum is found from this. Performance of RSM compares favorably with many gradient-based methods (Azadivar, 1999).

2.4.9 Ranking and Selection

Ranking and Selection (RS) procedures are used when there is a fixed set of possible alternatives, i.e. the search for new candidates has ended, or a limit on computational resources limits the result set. They do have application within the simulation optimization arena and may be applied when there is a limit on computational resources. They may also be applied as screeners, eliminating unlikely solutions from a larger set based on some predetermined threshold. (Fu, et al. 2005). There may be cases where a search algorithm of a simulation-optimization system may not be the best selection procedure. Boesel, et al. (2003) use RS procedures at the end of a simulation-optimization run to identify the best of a set of candidate solutions. They also used a two-stage IZ ranking procedure to find the best system.

2.4.10 Multiple Comparison Procedures

The second approach to finding a satisfactory, though not guaranteed best, solution from a small, finite parameter space is Multiple Comparison Procedures (MCP). MCP's are statistical inference processes based on the confidence intervals of processes executed against multiple replications of a solution. There are three types commonly used: all pairwise multiple comparisons (MCA), multiple comparisons with the best (MCB), and multiple comparisons with a control (MCC) (Swisher, 2000). In general, the best performance is expected from the MCB approach since its goal is to find the best solution while reducing the number of comparisons (Fu, 1994).

2.4.11 Ordinal Optimization

Ordinal optimization (OO) is one of the approaches that may be used when the feasible region is discrete and finite but larger than computational resources can handle. It can be used when a 'good enough' solution is being sought, one of the tenets of OO being that "nothing but the best is very costly". The second tenet is that it is easier to assign an 'order' or arrangement to compared items than it is to assign them a value. (Ho, et al., 2007) OO attempts to quantify these tenets by (1) "softening" the goal and (2) searching a subset of the region. Figure 2 illustrates this.

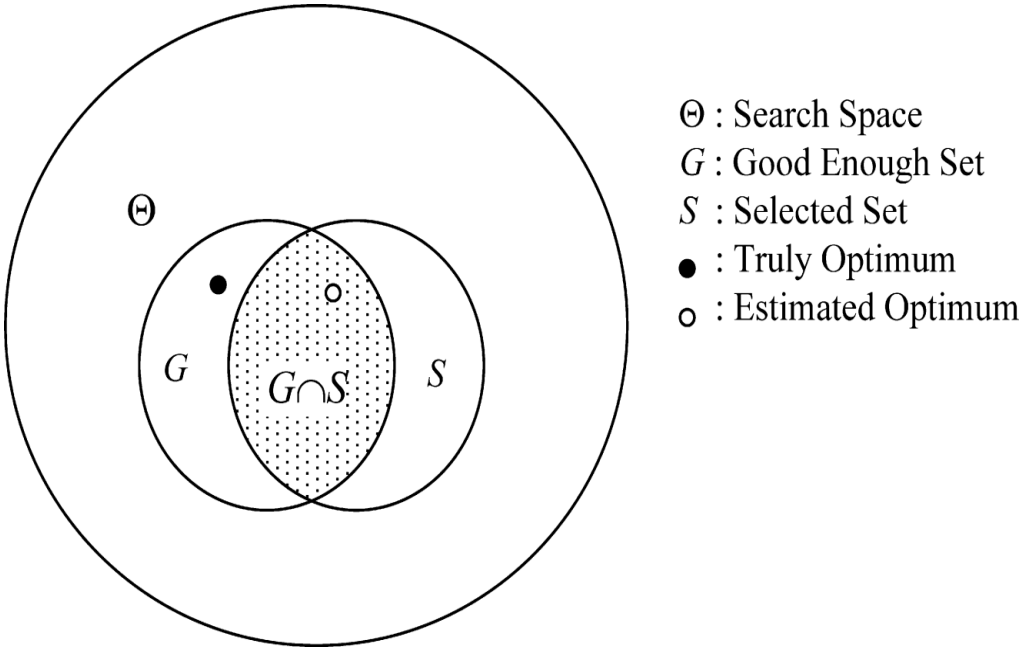


Figure 2 Illustration of Ordinal Optimization Concepts (Ho, et al. 2007)

2.4.12 Gradient Based Algorithms

Stochastic approximation (SA) methods encompass a family of algorithms in which an increasingly better solution is sought by iteratively

moving from one initial 'best guess' solution to another based on an estimate of the gradient (Olafsson 2002). These statistical inference tools are useful when there are noisy estimates of system performance, such as when parameters are generated by a Monte Carlo process and when gradients are not automatically available (Fu and Hill 1997).

The earliest algorithms of this type were by Robbins and Monro (1951). Based on that initial work, Kiefer and Wolfowitz (1952) developed the finite-difference approximation algorithm in which variables of the problem are varied one at a time. Spall (1992) developed an algorithm based on a simultaneous perturbation gradient approximation, Simultaneous Perturbation Stochastic Approximation (SPSA), in which all variables of the problem are varied at the same time. This approach reduces the computational requirement for large-dimensional problems and may be applied to any discrete event system that can be simulated (Fu and Hill, 1997) They found that the number of simulations required per gradient estimate, two, was not dependent on the number of parameters of interest. Suri and Zazanis (1998) found that the use of methods utilizing an infinitesimal perturbation analysis gradient estimator was the most efficient.

The assumption behind these processes is that a zero of the gradient for the original problem, $min_{\theta \in \Theta} J(\theta)$ can be found by solving$\nabla J(\theta) = 0$. The problem of local minima can be overcome through the use of heuristics (Fu 1994).

2.5 Metaheuristics

Metaheuristics are methods that may be employed when other procedures fail to move away from local optima. There are four primary metaheuristics: simulated annealing, genetic algorithms, Tabu Search, and scatter search (Fu et al., 2005).

2.5.1 Simulated Annealing

Simulated Annealing, SA, is a heuristic inspired by the physical process in which metals are combined to form an alloy and are slowly cooled to a lower-energy state (Ammeri, et al. 2010). In SA, a candidate solution is found and then possible replacement solutions are chosen randomly from a set of nearby solutions defined by a *candidate distribution* (Kirkpatrick, et al. 1983). SA is very similar to RS, with the exception that occasional downhill moves are allowed (Pritchitlamken and Nelson 2003). Vocaturo (2008) uses SA and simulation to optimize shipping container handling.

2.5.2 Genetic Algorithm

The genetic algorithm (GA) is a biologically-inspired approach to optimization. In it a set of values representing a candidate solution is encoded as a string of binary values, i.e. one and zero. Each such string is referred to as a 'chromosome'. An initial population (N) of chromosomes is randomly generated at the beginning of the process. They may represent values from across the solution space or a subset of it that is more likely to produce the optimal solution. Each individual is evaluated by a 'fitness function' and ranked

for survivability (optimality). Chromosomes are then selected randomly based on a weighted value of their fitness score. This is the Roulette Wheel selection (Man 1999).

Those selected are paired and combined to produce a new generation of individuals or chromosomes. The most common genetic operators used in the reproduction process are *crossover* and *mutation*, but others exist.

The new generation is then evaluated according to the fitness function, invalid solutions are discarded, i.e. the fittest survive and reproduction and selection takes place again. This process continues until a termination point is reached, the desired number of generations has been produced, a satisfactory solution has been found, computer or time resources have been depleted, etc. (Reeves 2003).

There are variations of this approach. Liu, et al. (2009) investigated a hybrid genetic algorithm and applied it to gas field pipeline networks. They replaced the Roulette selection process with a 'differential evolution algorithm' in which new individuals were produced through the linear combination of many parent individuals rather than through the crossover technique. Dhar and Datta (2008) used a numerical simulation routine and an elitist genetic algorithm to optimize operations of reservoirs for downstream water quality.

2.5.3 Tabu Search

In the Tabu Search (TS) heuristic, created by Fred Glover, a set of illegal moves is created, cataloged, and then avoided during the search for a solution. There are three categories of information cataloged. A (short-term) list of

recently considered (tabu) solutions is maintained and accessed to prevent the re-examination of a previously discarded solution. A list of rules intended to guide the search into more promising areas serves as an intermediate-term structure. And a long-term structure maintains rules that are applied if the search encounters a plateau or a local dead end (Glover 1989, 1990). This metaheuristic may also be combined with other approaches. Hansen (1996) developed an adaptation of the Tabu Search method to multi-objective problems. This method has grown in popularity, and is currently applied across various areas to solve combinatorial optimization problems. Beasley, et al. (2002) applied a Tabu search algorithm to an air traffic problem. Yang, et al. (2004) used a Tabu search to optimize a flow shop with multiple processors (FSMP) scheduling problem. Like many problems that are best served by simulation optimization, this one is NP-Hard and finding the exact optimal solution is computationally infeasible. Arbib, et al. (2012) combined a Tabu search algorithm and linear programming to solve a cutting process problem.

2.5.4 Scatter Search (SS)

This evolutionary (population-based) algorithm is closely related to the Tabu Search. It is designed to use combinations of reference points from potential solutions that have been marked as ‘good’. These combinations then generate a new potential solution. The first step in the process is to collect the information that is not contained in the original points. The next activity is to use existing heuristics, rules and techniques to generate and evaluate new points. The final step, rather than using randomization, is to apply a predetermined strategy to

complete component steps. The use of this technique has grown in recent years (Fu, et al., 2005).

The response surface methodology (RSM) performed on the entire solution set creates a metamodel that is then passed to a deterministic optimization process. The goal of RSM is to discover a relationship between that input data and the output objective function. When RSM is being used as part of an optimization process, then a form of sequential RSM is the one most often used (Fu 2005). It is noted that sequential RSM is a type of Stochastic Approximation in which the gradient is found from the regression model (Fu 1994).

A natural gas portfolio is a collection of long- and short-term contracts with different pricing structures, delivery times and rates that make use of the planned supply of gas, whether it is to be delivered to end users or consumed for industrial or energy generation purposed.

Like many models, Vautheeswaran and Balasubramanian (2010) use a similar approach to developing a lowest-cost model for the optimization of a natural gas portfolio for a power generation facility. Their model combined a Monte Carlo-based scenario generator based on short term gas prices and load demand with stochastic programming to find optimal combinations of gas contracts. To make the problem tractable, they applied a fast forward algorithm to reduce the number of scenarios from 10,000 to 200.

Seeking the lowest cost is not necessarily the safest plan. Hanjie and Baldick (2007) approach the problem of scheduling gas delivery for an electric

power producer with a utility-maximization framework, in which financial risk and user risk preferences are incorporated. Like many, their framework integrates Monte Carlo simulation and dynamic programming (Fu 2002) but includes the user's risk tolerance as a parameter.

2.6 Recent Developments

The Model Reference Adaptive Search Method (MRAS) was introduced in 2007 by Hu, et al. (2007). In this method, the solution space is 'modeled' and candidate solutions are taken from the model which is then updated after each iteration.

The Golden Region Search (GR), introduced in 2011 by Kabirian and Olafsson (2011) seeks an optimum by examining selected regions of the feasible space based on a score assigned which indicates the amount of the region that has been visited, a metamodel score which contains a metamodel-based predictive value of the objective function, and a quality score which represents the quality of the points evaluated within the region.

2.7 Simulation Optimization

Simulation is one of the most widely used tools in the field of operations research and industrial engineering (Pritsker and O'Reilly, 1999). It offers the practitioner the advantage of examining a proposed system without actually building it or evaluating changes to a system without modifying the existing system.

Simulation optimization problems are those in which simulation is integral to the evaluation of the objective function(s) or constraint(s) (Azadivar 1999).

The general form of the optimization objective function is

$$min_{\theta \in \Theta} J(\theta)$$

where $J(\theta)$ is the objective function, θ is a member of the constraint set, Θ. Since it is assumed that $J(\theta)$ is not available, the approach is to estimate $\hat{J}(\theta)$ through simulation and the expectation of J is represented by

$$J(\theta) = E[L(\theta, \omega)]$$

where L is a performance measure and w represents a simulation path (Fu, et al. 1994, 2005).

These techniques have been combined successfully across multiple industries for several years. Simulation optimization problems, then, are those in which simulation, primarily discrete event (Fu 2001), is integral to the evaluation of the objective function(s) or constraint(s) (Azadivar 1999). Examples of systems that may be well served by simulation optimization include manufacturing systems, supply chains, call centers and the optimization of financial portfolios (Fu 2001). Sinha and Ganesan (2011) use these techniques to optimize shipping container operations. Klaas and Fischer (2011) use simulation to generate scenarios, position of vehicles, positions of targets,

position of other vehicles, etc., to train and then test their approach for routing robotic material carriers.

This approach can be combined with other optimization techniques such as integer linear programming to find an optimal solution in cases of deterministic parameters or points of convergence in situations where stochastic parameters are present. This may be done by breaking the problem into multiple sub-problems and optimizing them separately. Abspoel et al. found this approach tenable while finding solutions using a D-optimal experimental design using intensive computer processing power (Abspoel et al. 2001). Padberg and Haubrich (2008) take a different path by identifying multiple objective functions throughout the storage-to-consumption system, combining them, and solving them as a Mixed Integer Quadratic Problem (MIQP).

Pitchitlamken and Nelson (2003) created a collection of algorithms in an attempt to adapt to variability and features of the response surface. For the optimization process, they used a combination of a Nested Partition (NP) approach, based on Branch and Bound, for sub-dividing the problem, a hill-climbing (HC) algorithm as a local-improvement scheme, and Sequential Selection with Memory (SSM) to select and retain the best of the solution candidates. Scenarios with integer-value decision variables were simulated and input values, some stochastically generated, were passed to the optimization process. They opined that the process would be very inefficient when applied to scenarios with discretized continuous decision variables. This is particularly relevant to natural gas futures.

The literature contains excellent survey papers by Azadivar (1992, 1999), Fu (1994), Andradottir (1998), Swisher et .at (2000), Olafsson, et al. (2002), Tekin, et al. (2004), and Ammeri, et al. (2010). It is common to classify simulation optimization by the nature of the feasible region. Tekin, et al (2004) classifies studies by local versus global optimization and further divides the local optimization studies according to discrete or continuous parameter spaces. Ammeri,et al. (2010) present a similar classification system. Discrete decision spaces can be further segregated into finite parameter space and infinite parameter space.

2.8 Optimization of Gas Storage Scheduling

Holland (2007) states that there are three common numerical techniques that are applied to the valuation of gas in storage: Monte Carlo simulation, binomial/trinomial trees, and numerical partial differential equation techniques.

2.8.1 Binomial/trinomial Techniques

This problem may be considered a Finite Horizon Markovian Dynamic Programming problem (FHDP) and as such is defined by a tuple $\{S, A, T, (r_t, f_t, f_a)^T_{t=1}\}$ where S is the State Space

A is the Action Space

T is the Horizon

So that r_t is the reward function of S,A

f_t is the transition function: S x A -> S

f_a is the feasible action correspondence: S->P(A)

As such, the problem can be represented as a bi- or trinomial tree with each decision point a state (s is element of S) and each branch an action (a is element of A) (Sundaram, 1996).

2.8.2 Differential Equation

In their seminal paper, Black and Scholes (1973) proposed that in a correctly priced market arbitrage, the ability to make sure profits through a compilation of a portfolio of long and short options and their stocks, should not be possible. Based on this idea, a formula was derived for determining the value of an option in terms of the price of the stock. Interestingly, the expected return of the stock is not incorporated in the formula. Also, the direction of change in value of the option is independent of the direction of change in value of the stock. Hodges (2004) incorporates an Ornstein-Uhlenbeck process rather than the Brownian motion process which is incorporated in the Black Scholes approach.

Ahn, et al (1991) separated the problem into two parts, a virtual storage problem consisting of traded instruments and the physical problem which identifies the actual gas in storage. He concluded that a strategy based on the seasonal spread of prices, the differences in the high demand for heating gas in the winter, the low demand in the spring and fall, and the lesser increase in demand for electrical generation in the summer, is not the optimal injection and withdrawal strategy. The market, according to the author, has incorporated the value of storage into the forward curve of natural gas, making it difficult to realize a profit when buying gas, holding it in physical storage, and then selling

at a later date. He describes the system state as a combination of cash and gas and establishes a model that implements a 'self-financing' framework. He derived a partial differential equation that showed an owner should inject gas at the maximum rate whenever the value of storage exceeds the spot price; in other words, whenever it appears that stored gas will be worth more if held in storage and sold later.

2.8.3 Monte Carlo

The nature of gas storage facilities forces the practitioner to use complex methodologies. Modeling the opportunity cost, for example, is complicated by the fact that as gas is released from storage, the ability to release more drops in the same way that the ability to accept more gas into storage decreases with the amount in storage. Of the three common numerical techniques that are applied to the valuation of gas in storage, Monte Carlo is the most efficient and most capable of injecting spikes into the projected prices. Holland (2007) modeled a one-year horizon with a price value for each day. One distinctive characteristic of gas storage is that the rate of injection or delivery is related to the amount in storage at the time (Holland 2008).

Holland developed a game theoretic model of a gas storage facility shared by multiple customers (Holland 2008). Emphasizing the worldwide importance of gas storage in times of price spikes, for example, this study sought the presence of a pure strategy Nash equilibrium. He considers the case where some owners may withdraw gas from storage before others, thereby

reducing the amount of gas those who wait can withdraw in a period of higher prices.

Consumption is seasonal with major fluctuations in residential consumption during cold months and lesser increases in summer consumption by power generation facilities (EIA 2006). This seasonality of gas consumption was investigated by Chaton (2005), who considered price shocks as well as various policies on prices.

Principal Component Analysis (PCA), introduced by Pearson in 1901, is one of the oldest multivariate analysis techniques and is still very popular. It is the simplest of the eigenvector-based multivariate analyses. In PCA, the dimensionality of a dataset containing many interrellated variables is reduced to a smaller, linear uncorrelated set of values by use of an orthagonal transformation. PCA techniques may also appear as 'factor analysis' or 'eigenvector analysis' (Jollife, 2002). Blanco (2002) and Bjerksund, et al. (2011) analyze the forward curve by applying PCA to segment it into its principal components

Typical natural gas contracts are normally written on a 12-month basis and gas is priced for delivery to the Henry Hub in Louisiana (Holland 2007). The price of gas obtained from other points on the interstate pipeline will be offset to reflect the transportation cost from the Henry Hub (NEB 2001).

Operators of gas storage facilities, then, must decide on a regular basis whether to inject, withdraw or hold the gas at its current level. Current prices, expected future prices (driven by expected demand) and contracted future

deliveries factor into that decision including estimation of forward curve process. The forward price curve differs from 'expected future prices' in that forward curve values are current and accurate rather than predictive. They are used for the purchase of gas for future delivery. It is common to use a Monte Carlo process to simulate the forward price curve, Blanco (2002), Bjerksund et al.(2011).

2.8.4 Real Option Theory

The practice of treating actual business opportunities as financial instruments is known as 'real options theory'. Frayer and Uludere (2001) identify five key components of real options: value of asset, exercise or strike price, time to expiration, volatility and risk-free rate. They modify the Black-Scholes model to real options and use it to evaluate a power production facility. Dixit and Pindyck (1994) made substantial contributions to the development of real option theory.

With this as a framework, then, for the purpose of assigning a value to the gas in storage, it is treated as a financial instrument known as an option. An option is a contract that gives the holder the right to buy or sell a certain amount of a commodity at a set price on a predetermined date. The holder of the option pays for this right and is not obligated to exercise the option. A second instrument is the futures contract in which the holder commits to pay a set price for a specified amount of gas to be delivered to a specified location on a date in the future (Hull, 2005).

Ignoring operating costs, Thompson treats the storage of gas as a series of call and put options (Thompson, 2002). In order to make the best possible decision, the owner considers the cost of the gas in inventory, the cost of capital, and expected demand, which translates to projected prices. Keppo and Lo (2003) develop a model for calculating the value of adding an electrical production facility to a corporation already in that market. Lai, et al. (2011) develop a more tractable heuristic model that combines real options and stochastic-dynamic-programming to valuate liquid natural gas storage by changing the high dimensional problem into one of lower dimensionality in regards to the forward price curves. Similarly, Chen, et al. (2006) develop a semi-Lagrangian approach that includes a timestepping scheme that effectively discretizes the system parameters.

While real options theory has become widely applied, it is not without problems. Smith and McCardle (1999) contend that the models described in the literature are oversimplified. They point out the many variations involved in developing and bringing an oil property to productivity.

Longstaff and Schwartz (2001) developed an approach to valuing American options through simulation using a least-squares approach. The framework of this approach was based on Black and Scholes' work. Boogert and Jong (2006) adapted this approach to include complexities of natural gas storage, such as injection and withdrawal rates and working volume, and used Monte Carlo to model prices. Hodge (2004) incorporates an Ornstein-Urhlenback process in the real options solution.

Prior to 1988, the focus was on continuous input parameter and included some form of path search and gradient estimation technique. Bettonvill, Fu and Ho, for example, investigated various gradient approaches. In the 1990's more attention was paid to discrete input parameters. Eglese and Fleischer, among others, investigated simulated annealing, and Liepins and Hilliard were some who worked on genetic algorithms (GA) (Swisher et al 2000). GA's have shown to be useful in non-parameterized problems (Azadivar 1999).

2.8.5 Computer Performance

Simulation optimization is, in general, very inefficient. The best convergence to be expected in "pure" stochastic optimization algorithms is $O(n^{-1/2})$ where n represents the computational effort (Fu 1994). Fu (2008) reviews common techniques and discusses the added complexity of managing limited computer resources

One approach to reducing computer resource consumption is to reduce the number of scenarios required in optimization simulation. In their natural gas power generation model, Vautheeswaran and Balasubramanian (2010) applied a fast forward algorithm to reduce the number of scenarios evaluated from 10,000 to 200.

Different optimization techniques demand different resources. Dhar and Datta (2008) found that conventional optimization techniques were prohibitively resource intensive due to a requirement to generate a Jacobian matrix during each iteration. They chose instead to use a Genetic Algorithm. Monte-Carlo is the most flexible approach and, when circumstances do not require an exact

optimal solution, provides a tractable method. It does not yield a provable best solution but, with added iterations, may produce improved results (Holland, 2007b).

The use of simultaneous perturbation rather than finite difference stochastic approximation led to a great reduction in computational requirements (Fu and Hill, 1997).

Felix and Weber (2008) compared recombining trees and dynamic programming and least squares methods both using Monte-Carlo simulation to generated scenarios. They found that the recombining tree algorithm performed better. These are trees that at some point converge rather than continue to branch. If, for example, two nodes at the same level have the same value and states, there would be no point in evaluating both of them and their subsequent branches.

The imposition of constraints is a common approach to reduce computer resource demand (April, et a. 2003). Boesel, et al. (2003) applied Ranking and Selection procedures to reduce the number of simulations by removing non-viable potential solutions from the solution set. To further reduce computational complexity, variables may be combined through principal component analysis techniques (Bjerksund 2011).

The problem is sometimes attacked by increasing the computer resources available. Parallelizing the simulation processes across multiple processors on the same computer allows more scenarios to be examined (Vocaturo, 2008). This is conceptually the same as executing the simulation on multiple, separate

computers. This concept has been extended, in fact, to the distributed computer model in which simulation optimization problems are separated into standalone units and generated or evaluated simultaneously on separate computers. The results are then pooled for evaluation (Garcia, et al. 2007, Fourer, et al. 2010)

2.8.6 Evaluation of Heuristic-Based Simulation Optimization

Convergence, in the context of simulation optimization, refers to the rate at which the optimal solution is found. Heuristic search algorithms cannot usually be proved to converge to the optimal solution. In a stochastic environment, a definite convergence can only be shown as the number of simulated solution sets approach infinity. It must be sufficient to find an *acceptable* solution. (Boesel, et al., 2003).

The best convergence rate to be expected with stochastic simulation optimization is on the order of $n^{-1/2}$ where n represents the computational effort (Fu 1994, Homem-de-mello, 2008). Fu goes on to state that simulation optimization itself is not efficient in general and should be used when other, more efficient approaches are not available.

The literature reports various comparisons between one optimization technique and another but, due to the variety of algorithms and variations in problems such as dimensionality, definite results are not likely. Hu investigated the efficiency of the Tabu search routine and found, in the standard test used, that it outperformed the random search and a composite GA (1992). Martin, et al. (1998) found that the Tabu outperformed the local search but was inferior to an SA approach. Yucesan and Jacobson (1996) compared the efficiency of

various SA algorithms to local search. Mousavi, et al. (2013) describe two metaheuristic algorithms for approaching the multi-item multi-period inventory problem. They use a genetic algorithm (GA), Branch and Bound and Simulated Annealing (SA) methodologies and compare the performance results. They found that the GA and SA approaches outperformed the B&B technique significantly.

Often, researchers will use a random search or complete enumeration algorithm as a baseline for comparisons to the one of choice (Tekin, et al. 2004, Dengiz, et al. 1997, Azadivar and Tomkins, 1999).

2.9 Modeling the problem

The optimization of complex systems through simulation involves the analysis of a series of possible solutions and selecting the best. There exists a set of alternative solutions, $A = (A_1, \ldots, A_n)$, where n is sufficiently large so that it is computationally difficult to examine all possibilities (Pepelaev, 2006). Due to their nature, it is impossible to analyze all possible alternatives, so a balance of optimality and time must be sought.

One approach is to reduce the size of the set A. In this experiment, initially gas volume and gas costs are continuous values. This is a mixed integer problem. To simplify, the gas volume values may be reduced to a discrete set of integer values.

The preferred approach of modeling gas storage is to include a high dimensional forward pricing, which overwhelms dynamic programming.

Therefore, the common approach is to apply a heuristic scheme (Lai, et al. 2010).

Chapter 3

3.1 Methodology

The problem being investigated is one of scheduling the buying and selling of natural gas as a mixed product. Gas is available with different energy contents or Btu levels. It may be combined for various uses. Natural gas has a cyclical demand pattern-- low in the fall, high in the winter as temperatures drop, low again in the spring, and then slightly higher in the hotter months as the demand for electricity for cooling increases. To hedge against the cyclical demand pattern, gas is placed into underground storage. Investors and operators of gas-consuming facilities seek ways to optimize the decision to buy, sell or hold natural gas.

This research investigates the scheduling of the purchase and sale or consumption of gases of mixed energy contents. To be specific, the research examines the purchase of two types of gas and the sale or consumption of three-- the original two plus a third, blended gas. Consumption may be thought of as an exchange of gas for heat or energy and can be viewed as a sale at the market spot buy price.

This multi-item, product-mix, multi-period inventory problem is non-deterministic polynomial time, NP-Hard, and finding the solution is computationally infeasible. It cannot be solved efficiently as is, but it can be approached by reducing it to a simpler problem through the application of heuristics and bounds. A result of this problem restatement is that an approach that provides a near-optimal solution must suffice.

This project investigated and compared three approaches to seeking an optimal schedule. The simulation-optimization approaches tested were (1) a B&B-LP hybrid; (2) a SS-LP hybrid; and (3) an SS approach with transaction volumes of variable sizes.

3.1.1 Simulation Optimization Methods

This type of scheduling problem, with decision points made across a finite horizon, lends itself nicely to a B&B solution, with each node representing a decision point. Decision points in the trinomial tree were created at each period of the 12-month horizon, three being generated from the previous node. B&B does find an optimum solution when the problem is sufficiently limited to not become computationally intractable due to the exponential growth of the tree. This was avoided by applying pruning and optimizing rules.

By selecting an action to be taken at a specific time, B&B identified a subregion of the solution set. That subregion was further searched by the LP routine to find the best combination of products to sell.

In the second approach, the use of a stochastic search (SS) routine to select sub-regions from the solution set replaced the B&B algorithm. Like the first approach, though, this one also used LP to optimize that selection.

The stochastic search routine alone was used as the third method. Full-horizon paths were generated and evaluated based on random selections from the solution set, with the best result being tracked. Rather than using LP, the volume moved in each transaction was the result of a random process.

3.1.2 General Framework

Using Awesim, each entity represented a potential path and contained all information required to define each period and to generate values for the next. This included all cost, price and inventory data as well as other parameters. The entity 'aged' through the time horizon, 12 months in most cases, changing value as different decisions were executed. At the end of the 12-period horizon, that value was compared to that of the best-valued decision path and replaced it if it was better.

Dependent Variables

The dependent variables in this experiment are the total profit, withdrawal and injection volumes, and computer processing time.

R_{ABi} - Calculated profit of Gas AB in period i

R_{Ai} - Calculated profit of Gas A in period i

R_{Bi} - Calculated profit of Gas B in period i

V_A - Volume of Gas_A Injected

V_B - Volume of Gas_B Injected

V_A - Volume of Gas_A Withdrawn

V_{AB} - Volume of Gas_{AB} Withdrawn

V_B - Volume of Gas_B Withdrawn

Independent Variables

C_A	Cost of Gas_A
C_{Ab}	Cost of Gas_{AB}
CAP_A	Max Facility Storage Capacity of Gas_A
CAP_B	Max Facility Storage Capacity of Gas_B
C_B	Cost of Gas_B
CS_A	Storage cost of GasA $/unit/month
CS_{AB}	Storage cost of Gas_B $/unit/month
h	Horizon – number of time periods
I_A	Max injection rate of Gas_A
I_B	Max injection rate of Gas_B
INV_A	Facility Current Inventory of Gas_A
INV_B	Facility Current Inventory of Gas_B
MDVa	max deliverable volume Gas_A
MDVb	max deliverable volume Gas_B
p_a	profit Gas_A
p_{ab}	profit Gas_{AB}
p_b	profit Gas_B
P_A	Sales Price of Gas_A
P_{Ab}	Sales Price of Gas_{AB}
P_B	Sales Price of Gas_B
r_A	Ratio of Gas_A in $Gas_{AB\ (1-\ r_B)}$
r_B	Ratio of Gas_B in Gas_{AB} $(1-r_A)$

$dvol_a$	change in volume of Gas_A one period
$dvol_{ab}$	change in volume of Gas_{AB} one period
$dvol_b$	change in volume of Gas_B one period
$VolMax_A$	Max leased storage capacity of Gas_A
$VolMax_{AB}$	Max leased storage capacity of Gas_B
W_A	Max withdrawal rate of Gas_A
W_{AB}	Max withdrawal rate of Gas_{AB}
W_B	Max withdrawal rate of Gas_B
Y_A	Number annual inventory turns for Gas_A
Y_B	Number annual inventory turns for Gas_B

Objective Function

At each decision point on the first and second approaches, if the choice to sell was selected, the LP_Solve functions were invoked through an Awesim USERF call and the entity was passed to the function. The objective function being solved was:

$$max\ p_a dvol_a + p_{ab} dvol_{ab} + p_b dvol_b \quad (1)$$

subject to:

$$dvol_a + percentAinAB * dvol_{ab} <= MDV_a$$

$$dvol_b + percentBinAB * dvol_{ab} <= MDV_b$$

$$dvol_a >= 0,\ dvol_b >= 0,\ dvol_{ab} >= 0$$

The general equation for the revenue of buys and sales of gas over time horizon h is often modeled as

$$revenue = \sum_{i=1}^{h} p_i Vol_i (dW_i - dI_i) \quad (2)$$

where W_i and I_i are decision variables, taking a mutually exclusive value of 1 or 0, representing the decision to withdraw (sell) or inject (buy) gas, respectively. In this project the objective was to maximize profit.

Expand equation (2) to include the combination of products. This models the rules that all products must be either bought or sold on any given day.

$$revenue = max \sum_{i=1}^{h} (dW_i - dI_i) \left(p_{a_i} dvol_{a_i} + p_{ab_i} dvol_{ab_i} + p_{b_i} dvol_{b_i} \right) \quad (3)$$

This is expanded to show the cost and price of the gases in equation (4)

$$value = max \sum_{i=1}^{h} ((dW_i - dI_i)(c_{a_i} dvol_{a_i} + c_{b_i} dvol_{b_i}) + (dW_i - dI_i)\left(p_{a_i} dvol_{a_i} + p_{ab_i} dvol_{ab_i} + p_{b_i} dvol_{b_i}\right)) \quad (4)$$

$$c_x = f(C_i, C_s) \quad (5)$$

C_x is a function of the initial cost of inventory and cost of storage. Throughout this model, a first-in-first-out (FIFO) pricing scheme is used for calculating the cost of gas sold. Note that other models include the present value of money (PVM), in the cost function. This would be discussed in Chapter 5 as future enhancement.

Equation (4) is expanded to allow a buy/sell decision to apply to individual products.

$$value = max \sum_{i=1}^{h} ((dW_{a_i} p_{a_i} dvol_{a_i} - c_{a_i} dvol_{a_i} dI_{a_i}) + \left(p_{ab_i} dvol_{ab_i} dW_{ab_i}\right) + (dW_{b_i} p_{b_i} dvol_{b_i} - c_{b_i} dvol_{b_i} dI_{b_i}))) \quad (6)$$

Constraints

To equation (6), multiple constraints are applied. The first is that gas must be in inventory the same month it is sold. Since gas can be sold and bought simultaneously, passing through, as it were, the inventory need not be in place at the beginning of the period but at the end.

There may be situations in which gas of one or both types is in a stream rather than in storage. In these cases, the total storage is equal to the maximum deliverable volume, effectively making the entire inventory pass through each period. Landfill gas would be such an example.

$$\forall i: i = 1..h, vol_{sold} \geq inv_i \geq 0 \tag{7}$$

The second constraint is that contracted capacity, CCAP, not be exceeded.

:

$$\forall i: i = 1..h, \ \sum_{n=1}^{i}((dW_n - dI_n)\left(dvol_a + \ _n + dvol_b\right))<= \text{CCAP} \tag{8}$$

One more constraint handles a feature common to many natural gas storage contracts, i.e., that gas still in storage at the end of the contracted period is forfeited, effectively creating a product with an increasingly short shelf life. Gas injected at the beginning of a contract has an effective life of twelve

months, while gas injected into storage two months prior to the end of contract has a shelf life of only two months. The penalty for having gas in storage at the expiration point of the storage contract is the loss of that gas at the current market price, the "spot" price.

Applying this constraint to the value equation (5) yields:

$$value = max \sum_{i=1}^{h} ((dW_{a_i} p_{a_i} dvol_{a_i} - c_{a_i} dvol_{a_i} dI_{a_i}) + (p_{ab_i} dvol_{ab_i} dW_{ab_i}) + (dW_{b_i} p_{b_i} dvol_{b_i} - c_{b_i} dvol_{b_i} dI_{b_i}))) - (Inv_a h * Spot_a + Inv_b h * Spot_b) \quad (9)$$

3.1.3 B&B-LP Approach

To create the trinomial tree, each surviving entity was copied thrice at the end of each generation, assigned a different value for the next decision point and reprocessed. This recursive process generated a trinomial tree with a maximum of

$$nodes_h = \sum_{i=1}^{h} 3^{i-1} \quad (10)$$

decision points to evaluate and potentially

$$paths_h = 3^h \quad (11)$$

total paths after generating paths for the full horizon. Appendix 8 contains the complete code from the Awesim model.

A set of heuristics and bounds were implemented to reduce the computational workload of the trinomial tree approach. The first two prune branches from the tree that would lead to an inferior final result while the third rule reduces search time without removing a branch.

- Do not purchase more product than can be sold by the end of the storage contract.
- Do not execute a 'hold' if that will result in having more gas than can be sold.
- If inventory is insufficient to complete a minimal transaction, do not invoke the LP solver.
- The decision made will apply to all types of gas. In the model, it was not legal to buy gas A while selling gas B.

Another variable that is not commonly found in problems of this type is the changing delivery rate of the product. While product delivery from a typical warehouse may be constrained by manpower or equipment, and the delivery rate affected by the level of concurrent orders to fill, natural gas delivery rates are primarily a function of the amount of product actually in storage. Deliverability refers to the rate at which gas can be withdrawn from storage. This rate is usually expressed in millions of cubic feet per day (MMcf/day) but may also

refer to the equivalent heat content of the gas and be expressed as dekatherms (100,000 BTU/day).

$$I = P * f(FAC) \tag{12}$$

I= rate of Injection

P=pressure, which is calculated with the Ideal Gas law

(Civan 2008)

$$P = \frac{nRT}{V} \tag{13}$$

n=moles of gas (converted from INV_t)

R= the Universal Gas Constant

$R = 0.08206 \frac{L \cdot atm}{mol \cdot K}$

T = temperature ^{o}Kelvin

V=Facility Volume

INV_t = Total facility inventory at time t

$f(FAC)$= a multiplier with a value from 0 to 1, to account for any flow constraint imposed by facility hardware, compressors, etc. If the aboveground hardware does not place a limit on the gas withdrawn, then *f(FAC)* = 1.

In practice, this may not be a significant parameter as a facility will contain gas owned by multiple operators and gas will be injected or withdrawn at rates unknown to planners. Also, there may be a deliverability rate guaranteed in the contract.

The inventory cycle time varies according to the type of gas storage facility. In the case of salt cavern storage, this period may be between 30 and 60 days, allowing a complete inventory turnover 6-12 times annually. Gas may be stored in one of several types of underground facilities. Of the three primary types, depleted reservoirs, aquifers and man-made salt-caverns, the later is considered due to its higher annual inventory turnover rates.

Type	Cushion to Working Gas Ratio	Injection Period(Days)	Withdrawal Period (Days)
Aquifer	Cushion 50% to 80%	200 to 250	100 to 150
Depleted Oil/Gas Reservoirs	Cushion 50%	200 to 250	100 to 150
Salt Cavern	Cushion 20% to 30%	20 to 40	10 to 20

Table 3 Natural Gas Storage Facility Characteristics (FERC 2004)

Algorithm 1: Branch and Bound Optimization with LP

for each entity loop

initialize independent variables & parameters

load prices, costs

apply variance process to price and cost data

(entity) enter B&B subroutine

```
repeat while at least one entity is in the B&B subroutine
   case action: ‘hold’
      if current inventory<periods remaining * transaction volume
         then apply action
      else prune branch
      end if
   case action: ‘buy’
      if current inventory<periods remaining * transaction volume
         and current inventory + purchase <= max storage capacity
         then apply action
      else prune branch
      end if
   case action: ‘sell’
      if current inventory or A and B
         then invoke LP_Solver
               apply results to value and inventory levels
      else process individual sale
      end if
   end case

   update status of entity
   if current entity is horizon (n)
      compare value to current best value
```

```
        if current value > best values
            swap values
        end if
      else spawn new entity for each action (buy,hold,sell)
          (entity) enter B&B subroutine
      end if
  end repeat
end loop
```

3.1.4 SS+LP Bounds

The SS-LP hybrid operated under the same constraints as the B&B-LP hybrid. The only change was the manner in which the test solution was selected from the solution space. The decision to buy, sell or hold was then selected from a uniform random distribution with each decision receiving equal weight, i.e., there was no bias toward either of the three decision actions.

Algorithm 2: Stochastic Selection with LP

```
for each entity (trial)
  initialize independent variables & parameters
  load prices, costs
  apply variance process to price and cost data
  (entity) enter SS-LP subroutine
```

```
repeat for each iteration
  repeat for s=1 to max solution samples
    repeat for n=1 to horizon
      generate action (stochastic process)
      case action: ‘hold’
        if current inventory<periods remaining * transaction volume
        then apply action
        else prune branch
      case action: ‘buy’
        if current inventory<periods remaining * transaction volume
        and current inventory + purchase <= max storage capacity
        then apply action
        else prune branch
      case action: ‘sell’
        if current inventory or A and B>
        then invoke LP_Solve
              apply results to value and inventory levels
        else process individual sale
      end case
      if value of plan > best plan
      then set best plan=current plan
      end repeat
   end repeat
```

end repeat

end loop

3.1.5 SS Bounds

Having two algorithms that applied LP, the third approach was developed so that the selection from the solution space in its entirety was the result of a random process. This provided a means of comparing the efficiency of the LP approach with the more flexible stochastic selection method. It was not expected that the approach would be more efficient when compared directly, but that its flexibility may increase the quality of solution.

The third algorithm was implemented with more flexibility by using a stochastic process to select a candidate solution from the solution space, including the mix ratio of products bought or sold. With the stochastic solution sample processes, invalid decisions are allowed but may not be executed. For example, if 'Sell' if selected yet there is insufficient inventory to execute a Sell, then the net change to value and inventory will be \$0 and 0 cf^3 gas respectively.

Flexibility was extended in the SS algorithm by adding a random process to determine the percentage of the max transfer volume that would be bought or sold in each transaction. To modify the efficiency of this approach, the values returned by the random process are customizable. The initial tests were performed with the process returning values ranging from -100% to +100% in units of 25%.

Algorithm 3 Stochastic Selection (SS)

```
for each entity (trial)
   initialize independent variables & parameters
   load prices, costs
   apply variance process to price and cost data
   (entity) enter SS subroutine
    repeat for s=1 to max solution samples
        repeat for n=1 to horizon
           generate inventory delta gas A   -100 (sell) to 100 (buy) %
           generate inventory delta gas B   -100 (sell) to 100 (buy) %
           generate inventory delta gas AB   -100 (sell) to 0%
           case action: 'buy'
              update entity inventory, value
           case action: 'sell'
             update entity inventory, value
           end case
           if value of plan > best plan
           then set best plan=current plan
           end repeat
       end repeat
   end repeat
end loop
```

3.1.6 Required Minimum Delivery Option

To increase the rigor of the simulation, a second version of the model was created identical to the first, with one additional option, that a burden could be placed on the facility to provide a monthly minimum sale or use of any or all of the three products.

This option added parameters and logic that forced the facility to deliver a 'contracted' volume of gas to either a customer or for internal use. This delivered amount was due regardless of the strategic decision to buy, sell or hold. In cases where there was no gas in inventory, the model purchased it on the open market at the spot price and incurred a penalty, another parameter of the simulation

The purpose of this was to examine the flexibility of the three algorithms when facing change and to see how they performed with the necessarily increased processing.

3.1.7 Development System Information

The simulation tool used was AweSim and Visual SLAM, from Mapics. This software provides discrete event as well as continuous simulation. It offers the capability of creating and linking user-defined functionality with C++. For this, Microsoft C++ 6.0 was included. A mixed-integer linear programming (MILP) solver, LP-Solve 5.5, was linked to the system (LP_Solve 2010). Functions from this package were used to solve the mixed-product problems. Data analysis and charting were done with Microsoft Excel 2007.

The model was created and executed on an IBM PS2 with one (1) 3.4 GHz Pentium CPU and 1GB of RAM. It ran on Windows XP Professional, version 2002, service pack 2.

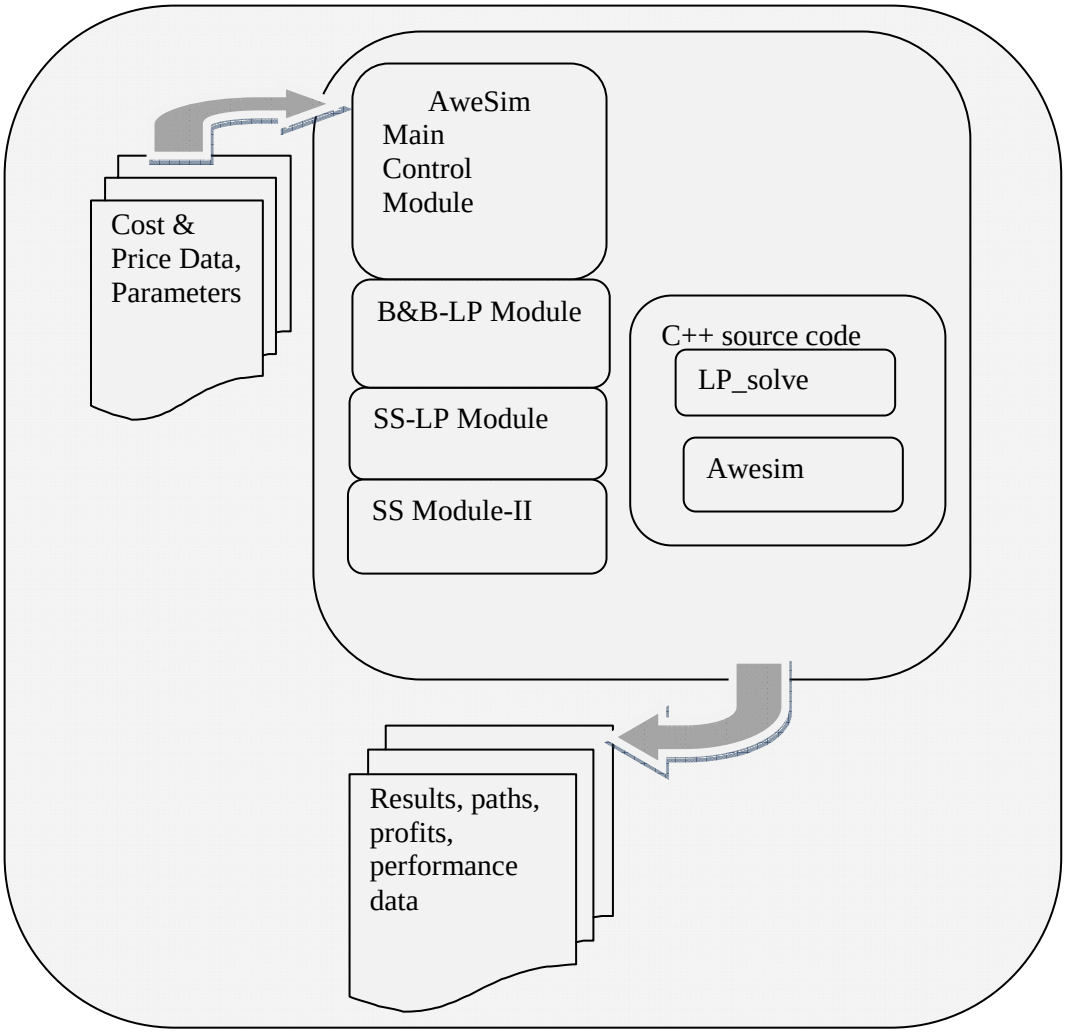


Figure 2 System Organization

3.2 Performance Measures

Optimization problems of this type do not always yield an optimum result but rather a near optimal one. There is an inherent trade-off in the amount of computer processing and the quality of the result. In this research both were measured and the relationship between the two was measured for each algorithm.

3.2.1 Speed of Computation

Computation speed can only be used as a relative expression as it is dependent on the system hardware, (CPU(s), memory, hard drives, etc.), operating system and software upon which the model executed. Computational speed is indirectly proportional to the effective cumulative processing power of the system. Equation 14 represents the factors that contribute to the computational speed.

Computational speed is also related to the structure of the problem itself. Some problems can only be approached sequentially, executing one step after another. In these cases, a multi-CPU machine would not demonstrate an advantage over a single-CPU machine. In such situations, the problem structure f_{ps} would have a detrimental effect on the processing time, S. With Monte Carlo optimization techniques, many problems may be parallelized, i.e., separated into smaller problems that can be solved on different threads, CPU's or other computers. After scenario generation and evaluation, the results are combined for analysis. Garcia, et al. (2007) used distributed processing to evaluate candidate solutions in a joint-strategy fictitious play simulation (Fourer, et al. 2010).

$$S = f_{pp} * f_{hw} * f_{sw} * f_{ps} \tag{14}$$

S = processing time

f_{pp} = function of problem parameters

f_{ps} = function of problem structure

f_{sw} = function of system software

f_{hw} = function of system hardware

The processing time may represent CPU time or total runtime. Problem parameters would include all variables and data used by the model as well as control parameters such as number of iterations, number of generations, thresholds, etc.

3.2.2 Quality of Solution

Each algorithm was executed multiple times, with different parameters and returned sets of results, candidate solutions. The accuracy and variance of values are analyzed for accuracy and variance. The results are compared and presented in Chapter 4.

3.2.3 Overall Performance

The quality of solutions and the computational time are tightly related in optimization problems using a stochastic method to select candidate solutions from the solution space. If the problem forces the investigator to accept a near-optimal solution, and it typically does, then the primary factor is how long to run the process.

3.2.4 Random Numbers

Many types of optimization require the generation of a set of alternative solutions. In doing so, a random number generator provides a number between 0 and 1 in a normal distribution. There are various ways to generate a ‘random’

number. One way is to import the numbers from a table of random numbers. Another method is to use a vacuum tube or some other generator of random noise. The third method, the one used in this experiment through AweSim, is to use a recursive equation $(i+1)^{st}$ random number from a previous set of random numbers. This deterministic approach does not produce true random numbers but rather pseudo-random numbers, which serve well for simulation problems. Awesim provides multiple streams as seeds for random numbers, allowing the user to execute reproducible simulations or vary the input to test multiple scenarios (Pritsker and O'Reilly, 1999)

Chapter 4

Computational Results and Analysis

4.1 Experimental Design

Simulation optimization was chosen as the tool to investigate this problem. Three algorithms were developed and tested—a branch and bound-linear programming B&B-LP, a stochastic search-linear programming SS-LP approach, and a fully stochastic search SS.

4.2 Trial Simulations

An initial set of tests was performed with the simplest data and parameters as a way to validate the functionality of the model. Appendix 1 contains the data and parameters used in these tests. Refer to appendix 2 for graphical representation of the results.

Subsequent trials were executed using historic price data from the Henry Hub for cost of gas and Oklahoma average city gate prices. Refer to appendices 6 and 7 for the Henry Hub and Oklahoma city gate data, respectively.

Historical data for the wellhead, city gate and consumer prices of natural gas are available from the Energy Information Agency (EIA 2012). This US government agency provides independent statistics and analysis of the production and consumption of petroleum products, coal, electricity and other

energy sources. Data is available by location and time. This research used a time period of one month and used similar available price data.

.

4.3 Computational Results

Branch & Bound-LP Hybrid

This algorithm provided, not surprisingly, the most accurate results. Within the constraints placed on it, the process enumerated and evaluated each possible path in the 12-period horizon. In 25 trials, the correct solution was found each time. The number of samples evaluated was based on the *maximum* number of candidate paths enumerated by the trinomial tree, 3^{12} = 531,441. With the bounds placed on the algorithm, and considering the samples per second evaluated by the other approaches, it is unlikely that the solution set was fully enumerated.

Horizon	12.0
Samples Evaluated	531,441
Value Generated	1200
Elapsed Seconds	300
Samples per Second	1771.5

Table 4 Branch & Bound-LP, Results

SS-LP hybrid

The SS-LP hybrid performed best when sampling 20000 solutions per iteration. It consistently found the optimum solutions with a STDDEV of 0.0.

Samples per Simulation	*250*	*2500*	*10000*	*20000*
Simulations	*20*	*20*	*20*	*20*
Total Seconds Elapsed	*60*	*240*	1800	3480
Samples per Second	83.3	119	111	114
Mean	1000.0	1126.3	1189.5	1200.0
Standard Error	15.3	22.7	10.5	0.0
Median	1000.0	1200.0	1200.0	1200.0
Mode	1000.0	1200.0	1200.0	1200.0
Standard Deviation	66.7	99.1	45.9	0.0
Sample Variance	4444.4	9824.6	2105.3	0.0
Range	400.0	200.0	200.0	0.0
Minimum	800.0	1000.0	1000.0	1200.0
Maximum	1200.0	1200.0	1200.0	1200.0
Sum	19000.0	21400.0	22600.0	22800.0
Count	19.0	19.0	19.0	19.0
Largest(1)	1200.0	1200.0	1200.0	1200.0
Smallest(1)	800.0	1000.0	1000.0	1200.0
Confidence Level(95.0%)	32.1	47.8	22.1	0.0

Table 5 SS-LP, Descriptive Statistics

SS

The SS algorithm was created with the option of generating specific volumes of gas to be bought or sold, with a range from -100% to 100% of the maximum transfer amount, and was initially generated in 25% increments. In practice, it turned out that this expanded the solution space to the point that the SS approach could not reliably find a near-optimal solution in a reasonable time.

The results shown here and in the follow-on model were generated with rates selected from the set {-100, 0,100}.

Samples per Simulation	*250*	*2500*	*10000*	*20000*
Simulations	*20*	*20*	*20*	*20*
Total Seconds Elapsed	*60*	*120*	360	840
Samples per Second	83.3	416.7	555.6	476.2
Mean	1010.0	1046.0	1090.0	1134.0
Standard Error	17.6	17.8	11.7	12.4
Median	1000.0	1000.0	1100.0	1120.0
Mode	1000.0	1000.0	1100.0	1100.0
Standard Deviation	78.8	79.5	52.1	55.5
Sample Variance	6210.5	6320.0	2715.8	3077.9
Range	400.0	280.0	200.0	200.0
Minimum	800.0	920.0	1000.0	1000.0
Maximum	1200.0	1200.0	1200.0	1200.0
Sum	20200.0	20920.0	21800.0	22680.0
Count	20.0	20.0	20.0	20.0
Largest(1)	1200.0	1200.0	1200.0	1200.0
Smallest(1)	800.0	920.0	1000.0	1000.0
Confidence Level(95.0%)	36.9	37.2	24.4	26.0

Table 6 SS Descriptive Statistics

The SS and SS-LP algorithms yielded very similar results due primarily to the simplicity of the scenarios. Figure 4 compares the number of solutions sampled with the mean value returned. Performance-wise, the SS model in these scenarios performed much more efficiently, evaluating 417% more solutions per second than SS-LP.

Branch & Bound-LP Hybrid w/ Minimum Required Deliveries

With the introduction of the minimal required deliveries constraint, the B&B-LP hybrid did find the optimal solution. It correctly returned a value of 1007 when a 0.1% out-of-stock penalty was applied and 1020 when there was no penalty. In both situations, the B&B-LP algorithm provided the best results and in the shortest time. Again, the number of samples evaluated was based on the maximum number of candidate paths enumerated by the trinomial tree, 3^{12} = 531,441.

Horizon	12.0	12.0
Samples Evaluated	531,441	531,441
Penalty	0.1	0.0
Value Generated	1007.0	1020.0
Elapsed Seconds	300	315
Samples per Second	1771.5	1687.1

Table 7 B&B-LP w/ Min Delivery, Results

SS-LP w/ Minimum Required Deliveries

With the addition of the minimum required delivery rule, the accuracy of the SS-LP and SS algorithms dropped. Variance and standard deviation were higher than the same algorithm without the additional constraint.

Samples per Simulation	*250*	*2500*	*10000*	*20000*
Simulations	*20*	*20*	*20*	*20*
Total Seconds Elapsed	*60*	*240*	1800	3480
Samples per Second	83.3	119.0	119.0	119.0
Mean	793.5	900.0	1020.0	1010.0
Standard Error	20.7	22.5	0.0	10.0
Median	820.0	820.0	1020.0	1020.0
Mode	820.0	820.0	1020.0	1020.0
Standard Deviation	92.6	100.5	0.0	44.7
Sample Variance	8571.3	10105.3	0.0	2000.0
Range	400.0	200.0	0.0	200.0
Minimum	620.0	820.0	1020.0	820.0
Maximum	1020.0	1020.0	1020.0	1020.0
Sum	15870.0	18000.0	20400.0	20200.0
Count	20.0	20.0	20.0	20.0
Largest(1)	1020.0	1020.0	1020.0	1020.0
Smallest(1)	620.0	820.0	1020.0	820.0
Confidence Level(95.0%)	43.3	47.0	0.0	20.9

Table 8 SS-LP w/ Min Delivery, Descriptive Statistics

SS w/ Minimum Required Deliveries

Again, the SS model outperformed the SS-LP speed-wise. The lack of computational overhead of the LP-Solve's simplex processing allowed the sample evaluations to run much faster. The accuracy of this approach was *significantly* enhanced by the constraint placed on the volume of quantity shipped.

Samples per Simulation	*250*	*2500*	*10000*	*20000*
Simulations	*20*	*20*	*20*	*20*
Total Seconds Elapsed	*60*	*120*	360	840
Samples per Second	83.3	416.7	555.6	476.2
Mean	784.4	901.2	970.2	999.2
Standard Error	18.0	13.9	11.2	9.5
Median	795.6	920.0	955.0	1013.3
Mode	720.0	920.0	1020.0	1020.0
Standard Deviation	80.5	62.1	50.2	42.5
Sample Variance	6476.6	3853.3	2520.9	1806.9
Range	300.0	253.2	153.2	200.0
Minimum	620.0	820.0	920.0	920.0
Maximum	920.0	1073.2	1073.2	1120.0
Sum	15688.2	18024.0	19404.4	19983.4
Count	20.0	20.0	20.0	20.0
Largest(1)	920.0	1073.2	1073.2	1120.0
Smallest(1)	620.0	820.0	920.0	920.0
Confidence Level(95.0%)	37.7	29.1	23.5	19.9

Table 9 SS w/ Min Delivery, Descriptive Statistics

4.4 Computational Speed

The only meaningful way to compare the speed of these algorithms is on the same hardware and software. With that consideration in mind, the Branch & Bound-LP hybrid was consistently superior on a result/unit time basis. Its ability to prune the decision tree and thereby avoid the evaluation of discarded nodes increased the efficiency. It is noted that the relatively low time horizon, twelve months, contributed to the B&B-LP success. Had the horizon been 24- or 52 periods, the problem would have become too computationally intensive for this approach.

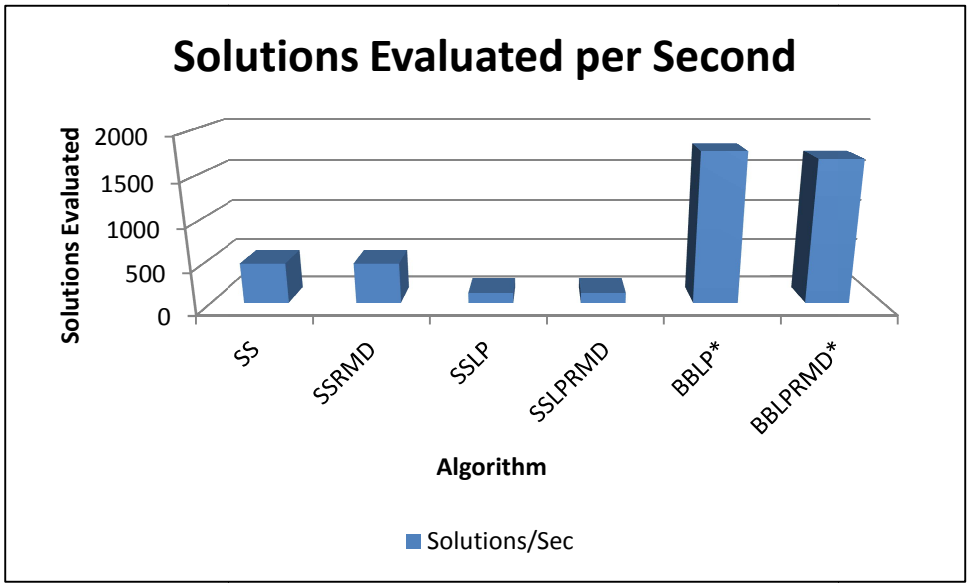


Figure 3 Solutions Evaluated/Second

The SS-LP hybrid was the least effective, computationally, due to the overhead of the LP-Solve software. The SS algorithm, the simplest

computationally, performed faster in this project due primarily to the constraints placed on its search of the solution space.

4.2.1 Quality of Solution

The following graphs compare the accuracy of each approach at each level of performance. The B&B-LP was superior in both variants of the model.

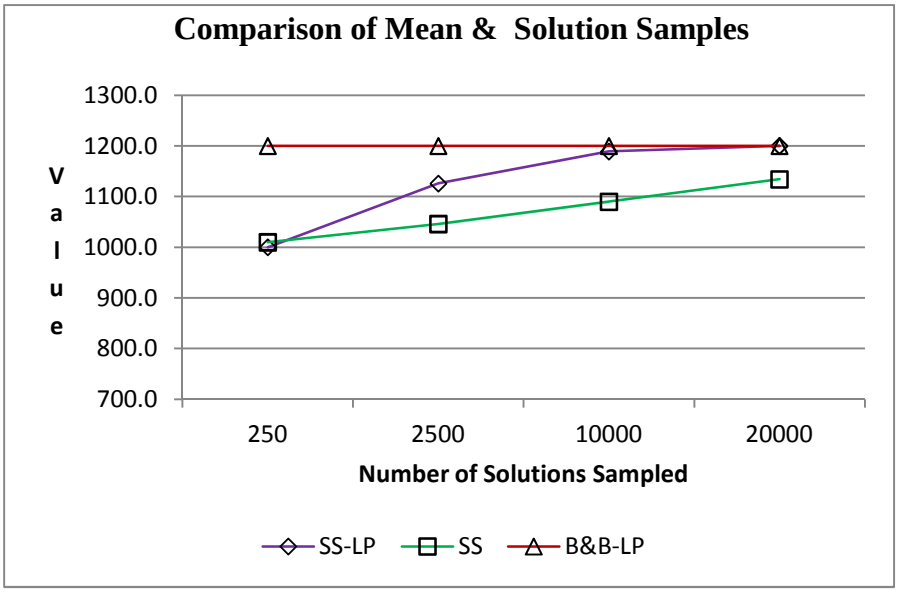


Figure 4 Comparison of SS-LP & SS Results

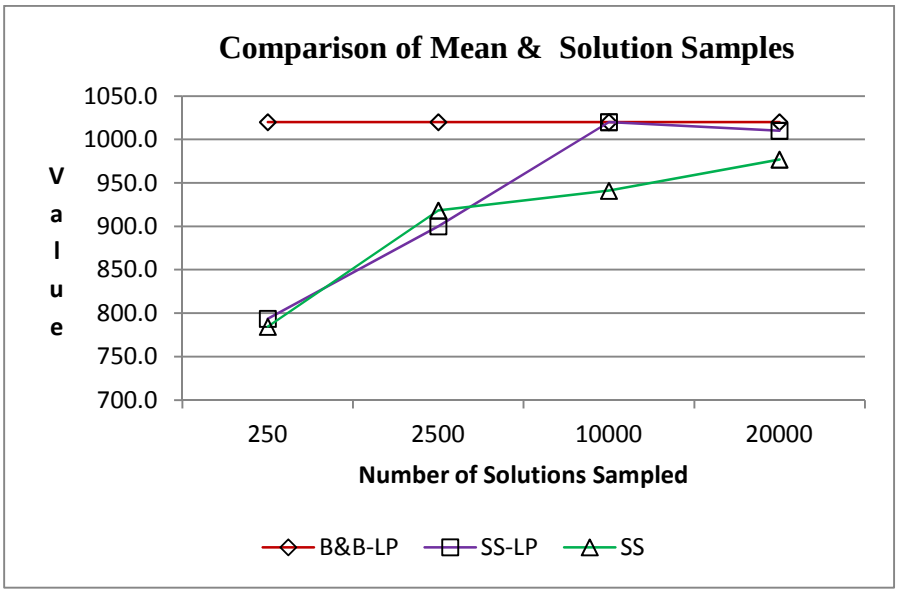


Figure 5 Comparison of SS-LP & SS w/ MRD

As expected, the stochastic approaches improved in accuracy with increased sample evaluations.

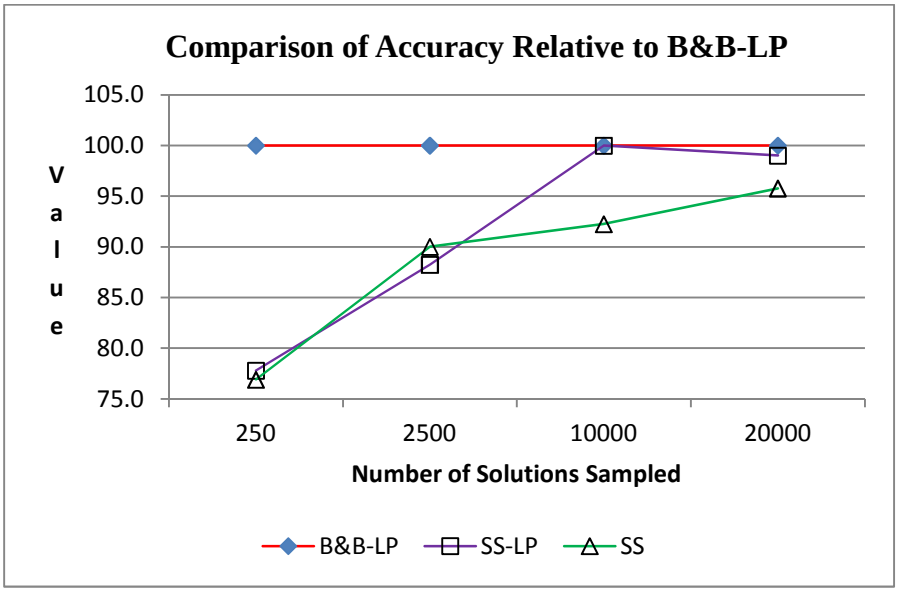


Figure 6 Relative Accuracy of Approaches

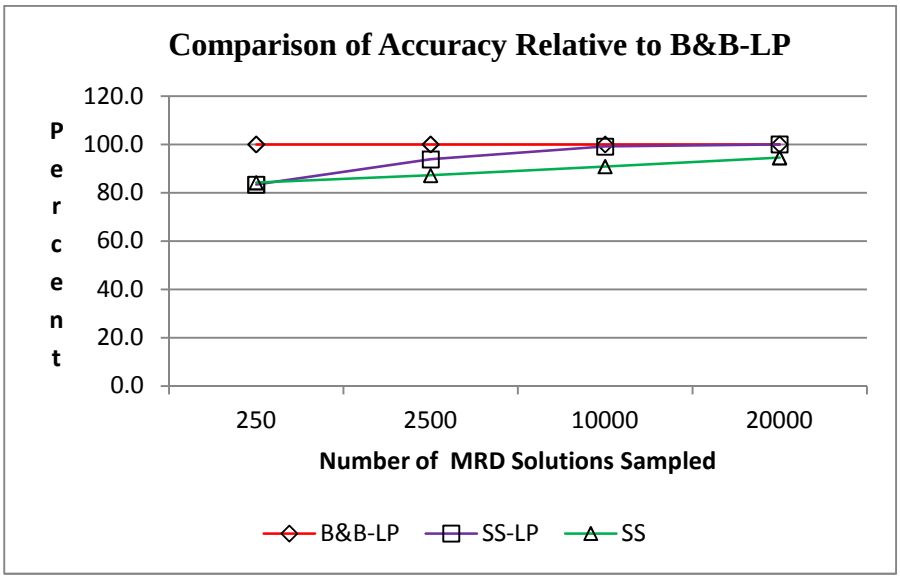


Figure 7 Relative Accuracy of Approaches w/ MRD

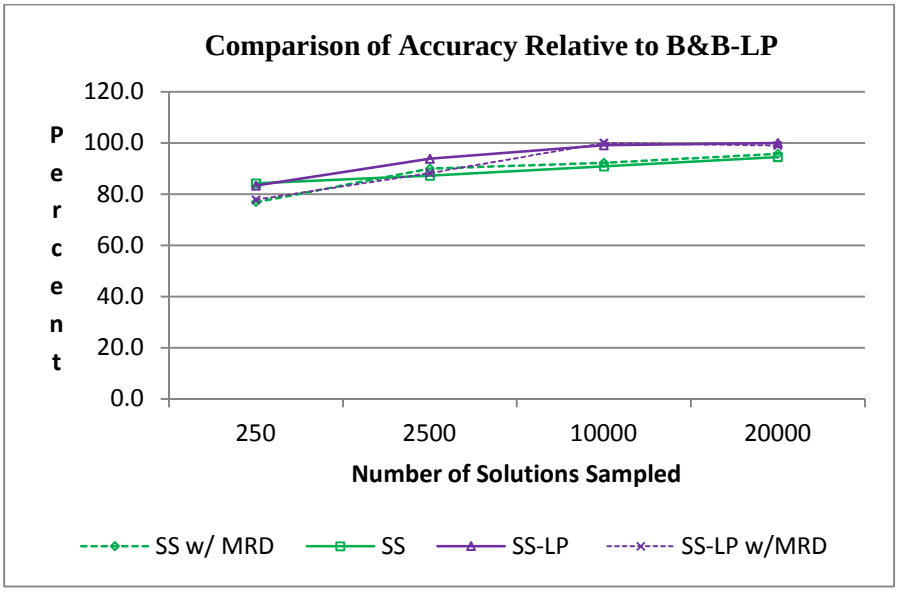


Figure 8 Relative Accuracy of SS approaches

4.2.2 Best Overall Performance

The Branch & Bound-LP hybrid was the best of the three approaches used in this project. It returned the optimal solution and, when compared to the SS-LP and SS algorithms that actually executed long enough to generate a reliable optimum or near-optimum solution, it was the least computationally expensive.

Chapter 5

Conclusions and Future Research

5.1 General Conclusions

With a worldwide market for natural gas, it is not surprising that a great amount of research has already been accomplished on a multitude of associated topics. So it is within the area of natural gas scheduling, an inventory scheduling problem with some attributes that are specific to natural gas.

This project has sought to extend that research by examining methods of optimizing the decisions that are made by gas investors and facility operators. The specific focus of the dissertation was the combination of gases of different energy contents, or Btu levels. This topic grows in importance as businesses seek to optimize resources and as environment pressures dictate the consumption of gas of lesser quality.

Simulation optimization is commonly employed to solve or find reasonable solutions to problems such as this. The literature review in chapter 2 discussed many variations of this powerful tool. This dissertation describes a research project that examined three algorithms for optimizing gas inventory decision making.

The B&B-LP hybrid was, within the constraints of the program, the most accurate, always returning an optimal solution and in the best time. This

accuracy was generated at the expense of flexibility. Heuristics were applied to reduce the number of decision points at each node, exponential growth being the nemesis of dynamic programming.

The advantage of the Stochastic Selection-Linear Programming (SS-LP) algorithms was its flexibility. It was not as efficient computationally as the B&B-LP approach, but it was more readily modified to new constraints.

While the simplest and most flexible approach, the generic Stochastic Selection (SS) algorithm proved to be too computationally expensive to use without some constraints. For example, percentage of shipping volumes were selected from a set of three options, -100%, 0, and +100%.

5.1.1 Computational Effort Conclusions

Being NP-Hard, this is a problem whose acceptable solution requires a high level of computational investment. Heuristics were applied that simplified the problem. Measurements of computational effort provide a comparison of the algorithms' resource requirements. These will vary from computer to computer and, as mentioned in chapter 3, from one implementation and software configuration to another. Even with knowledge of the specifications of the computer and software system used in this project, these results are not necessarily predictive of performance on another system.

The B&B-LP hybrid returned the best results in the least amount of time. The SS-LP and SS algorithms did generate optimal solutions when given sufficient time, but the time required was significantly greater than that of the B&B-LP.

5.1.2 Quality of Solution Conclusions

Each of the three algorithms produced optimal solutions in both test cases. The B&B-LP model found the optimal solution in the shortest time. Not surprisingly, the accuracy of the stochastic solution search routines was directly proportional to the number of sample solutions examined. The SS–LP model provided the optimal solution with a STDEV of 0.0 when 20,000 solutions were examined. The SS model exhibited the same performance.

There were two variants of the model, the basic one and a second that enforced a minimum delivery quantity. The quality of solution and relative consumption of computer resources were the same in each variant.

5.1.3 Overall Performance Conclusions

Performance of the three algorithms varied. The B&B-LP approach was the superior performer for this problem. Under other circumstances, that may not be the case when, for example, there are more decision factors to be evaluated.

5.2 Final Conclusions

This project has been very interesting. The energy industry and natural gas in particular is a global concern and, as it faces changes from economic, technological and environmental stimuli, there will be new and important areas of research. This project has examined and offered an useable approach, an approach superior to one based solely on historical performance, to a problem

that has become more prominent in the industry and will continue to receive attention.

5.3 Future Research

This is an exciting area of research and this project remains with many avenues to be investigated. There are many simulation optimization techniques that may be applied to this type of problem.

Exploration of performance improvement on distributed system would allow the researcher to examine multiple sets of simulated decision paths simultaneously.

The model designed for this project allowed for random variations of input price and cost data. To be more realistic, a model may include a procedure to apply a Brownian motion variance as well. Also, regarding price data, many models in practice currently include natural gas futures in the pricing scheme.

5.3.1 Parallel and Distributed Processing

With the availability of multi-core and multi-CPU architectures, even modern desktop computers offer significantly more processing power than was available for this research. Such hardware, when properly accessed, allows multiple independent processes to run simultaneously, as opposed to single-threaded processing which may appear to execute processes at the same time but actually switches between them rapidly. Many optimization techniques, particularly Monte-Carlo-based approaches, can be parallelized and executed simultaneously. Approaches that involve Markovian states also fit this scenario.

Software solutions and optimization programming techniques that take advantage of tread-level parallelism will provide much broader searches of the solution space (Happe et al. 2009, Hsu, et al. 2011, Lee 2010).

Perhaps the most promising technological development to increase processing power is distributed processing. Garcia, et al. (2007) used distributed processing to evaluate candidate solutions in a joint-strategy fictitious play simulation. Fourer, et al. (2010) developed a framework for distributed optimization, a system in which multiple computers not in a central location but connected via the Internet could be used as shared resources in solving optimization problems. Their work is conceptually similar to that of Luo, et al. (2000), but it is implemented with more recent and mature methods. They point out the need for the operations research (OR) community to be cognizant of the advances and innovations already in use in the information technology (IT) community. In service-oriented architecture (SOA) technology, service-level communication between servers is in widespread use commercially through standardized protocols that allow dissimilar applications to exchange data, invoke services on other servers, or execute services at the request of external machines. Using this or a similar framework, the practitioner could use modeling software of one server, data generation services of a different machine, the simulation services of another and the optimization service of yet another server.

In this particular example, such a system would provide the means of examining larger sets of potential solutions, whether they are stochastically-

generated combinations or regions of a solution space selected by a different approach, or as branches of a decision tree. The SETI@Home project is a well-known example of a massively-distributed computing program (Korpela, et al., 2001).

5.3.2 Geometric Brownian Motion

In applying a valuation to decisions to purchase or sell gas, it is a common practice to treat them as stock options, contracts for the future privilege to buy or sell a specified amount at a specified price. This practice is known as Real Options Theory and is widely practiced (Frayer, et al. 2001, Lai, et al. 2011).

Brownian motion was first described in the field of physics as observations of random movement of large particles when smaller ones struck them, but, interestingly, the prices of stocks tend to display Brownian motion as they fluctuate. Geometric Brownian motion varies from ordinary Brownian motion in that it holds that over time, the changes in price will fall into a normal distribution with a mean and standard deviation dependent only on the time elapsed. Brownian motion is a key component of the Black-Scholes equation. Formula 15 shows the basic Brownian motion value change (Chriss 1999).

$$dS = \mu S dt + \sigma S dz \qquad (15)$$

S = price

t=time

μ,σ = constants indicating drift

z =stochastic process, Wiener, $dz = \epsilon\sqrt{dt}$

Brewer, et. al. (2012)

It would be advantageous to apply Brownian motion to the changes in gas prices in future work. While other events and trends contribute to the volatility of natural gas prices, this would improve the model by making the volatility factor more realistic.

5.3.3 Heuristics, Metaheuristics, Multi-Criteria

This problem may be approached with various Metaheuristics, such as the tabu search. The problems continuous solution space makes it computationally intractable unless heuristics and constraints are applied to discretize the problem. Further examination of those constraints would prove interesting. The second part of this project sought to optimize profit while guaranteeing delivery of products. That or similar criteria for optimization may be expanded upon as well.

5.3.4 Natural Gas Futures

In this dissertation and the model it describes, natural gas 'spot prices' have been used. The spot price is the price for a transaction at the current price on the open market for immediate delivery of a specific quantity of gas at a specific location.

A futures price, however, is the price quoted for delivering a specified quantity of gas at a specified time and place in the future. Quotes or 'contracts' are usually written for delivery for a specific number of months in the future and

may be referred to as ‘contract n’ where n is 1,2,3 etc. and may go as high as 360 months, although that is unusual. Figure 12 shows the historic values of 1-month natural gas futures. Natural gas contracts expire three business days prior to the first calendar day of the delivery month. Thus, the delivery month for Contract 1 is the calendar month following the trade date.

While speculators buy and sell contracts as investments, futures are also purchased as a way of hedging against sharp price fluctuations. The addition of futures to the model would be a great enhancement.

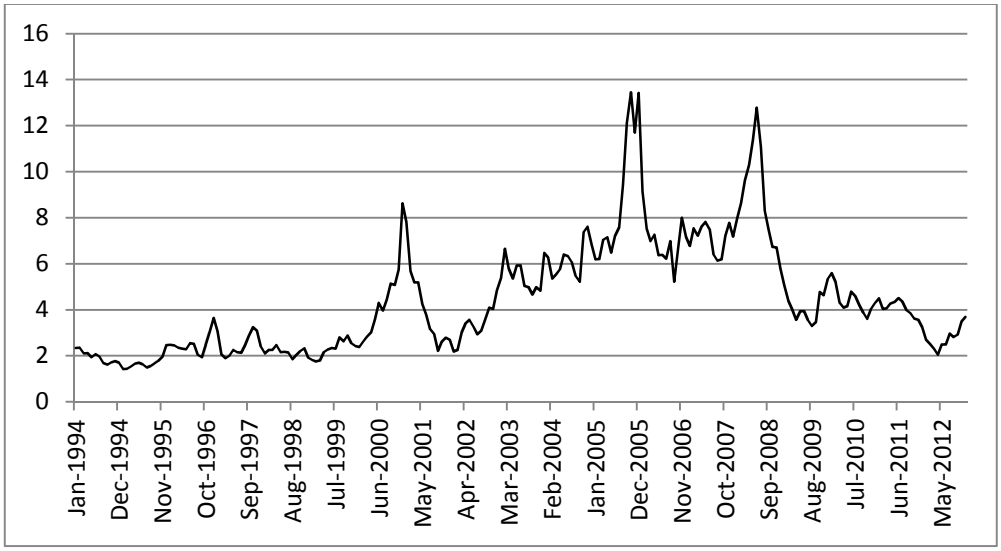


Figure 9 Natural Gas Futures Contract 1 ($US/MMBTU) (EIA2013)

There are many attributes and capabilities that may be added to this model that will increase its accuracy and perhaps efficiency as well. As new technologies emerge, economies change, and sources of energy fluctuate in priority, this field of research remains interesting and relevant.

Bibliography

1. Abspoel, S., Etman, L., Vervoort, J., Rooij, R., Schoofs, A., Rooda, J. (2001). Simulation Based Optimization of Stochastic Systems with Integer Design Variables by Sequential Multipoint Linear Approximation. *Structural and Multidisciplinary Optimization,* 22, 125-138.

2. Ahn, H., Danilova, A., Swindle, G. (2002, September). Storing ARB. *Wilmott*.

3. Al-Aomar, R. (2000). Product-Mix Analysis with Discrete Event Simulation. *Proceedings of the 2000 Winter Simulation Conference*, 1385-1392.

4. Amerault, P., Blount, J., Hopper, J., Gentges, R. (2005). Industry Leaders Discuss Future Gas Storage Trends. *Pipeline and Gas Journal* 232, 6.

5. Ammeri, A., Chabchoub, H., Hachicha, W., Masmoudi, F. (2010, May). A Comprehensive Literature Classification of Simulation-Optimization Methods. *Proceedings of Multi Objective Programming Goal Programming2010.*

6. Anderson, P., Evans, G., Biles, W. (2006, June). Simulation Optimization of logistics systems through the use of various reduction techniques and criterion models. *Engineering Optimization*, 38, 4, 441-460.

7. April, J., Glover, F., Kelly, J., Laguna, M. (2003). Practical Introduction to Simulation Optimization. *Proceedings of the 2003 Winter Simulation Conference*, 71-78.

8. Arbib, C., Marinellie, F., Pezzella, F. (2012). An LP-Based Tabo Search for Batch Scheduling in a Cutting Process with Finite Buffers. *International Journal of Production Economics*, 136, 287-296.

9. Avery, W., Brown, G., Rosenkranz, J., Wood, R. (1992). Optimization of Purchase, Storage and Transmission Contracts for Natural Gas Utilities. *Operations Research,* 40, 3, 446-462.

10. Azadivar, F. (1999). Simulation Optimization Methodologies. *Proceedings of the 1999 Winter Simulation Conference*, 93-100.

11. Bagci, A., Ozturk, E. (2007). Performance Prediction of Underground Gas Storage in Salt Caverns, *Energy Sources*, Part B, 2,155-165.

12. Baghmisheh, M., Peimani, M., Sadeghi, M.,Ettefagh, M.,Tabrizi, A. (2012). A Hybrid Particle Swarm-Nelder-Mead Optimization Method for Crack Detection in Cantilever Beams. *Applied Soft Computing Journal*, 12, 8, 2217-2226.

13. Baranes, E., Mirabel, F., Poudou, J. (2009). *The Economics of natural Gas Storage: A European Perspective*. Heidelberg: Springer-Verlag.

14. Bjerksund, P., Stensland, G., Vagstad, F. (2011). Gas Storage Valuation: Price Modeling vs Optimization Methods. *The Energy Journal*, 32, 203-226.

15. Black, F., Scholes, M. (1973). The Pricing of Options and Corporate Liabilities. *Journal of Political Economy*, 81, 637-659.

16. Beasley, J., Howells, H., Sonanser, J. (2002). Improving Short-Term Conflict via Tabu Search. *The Journal of the Operational Research Society*, 53, 6, 593-602.

17. Blanco, C., Stefiszyn, P. (2002). *Multi-factor Models for Forward Curves Analysis: An Introduction to Principal Component Analysis*. Financial Engineering Associates. Retrieved February 21, 2013, from http://www.docstoc.com/ docs /93749334/Multi-Factor-Models-for-Forward-Curve-Analysis

18. Blanco, C., Stefiszyn, P. (2007). *Valuing Natural Gas Storage Using Seasonal Principal Component Analysis*. Financial Engineering Associates. Retrieved February 21, 2013, from http://www.erasmusenergy.com/ articles/72/1/Valuing-Natural-Gas-Storage-Using-Seasonal-Principal-Component-Analysis/Page1.html

19. Boesel, J., Nelson, B., Kim,S. (2003). Using Ranking and Selection to 'Clean Up' After Simulation Optimization. *Operation Research*, 51, 814-825.

20. Boogert, A., De Jong, C. (2006). *Gas Storage Valuation Using a Monte-Carlo Method*. London: School of Economics, Mathematics and Statistics Birkbeck College University of London.

21. Bopp, A. (1996). An Optimization Model for Planning Natural Gas Purchases, Transportation, Storage and Deliverability, *Journal of Management Science*, 24, 5, 511-522.

22. Brewer, K., Feng, Y., Kwan, C. (2012). Geometric Brownian Motion, Option Pricing, and Simulation: Some Spreadsheet-Based Exercises in Financial Modeling. *Spreadsheets in Education (eJSiE)*, 5, 3, article 4

Retrieved February 13, 2013, from http://epublications.bond.edu.au/ejsie/vol5/iss3/4.

23. Busby, R. (1999). *Natural Gas in Non-Technical Language.* Institute of Gas Technology.Tulsa, Oklahoma: Pennwell.

24. Byers, J. (2006). Commodity Storage Valuation: A Linear Optimization Based on Traded Instruments. *Energy Economics,* 298, 275-287.

25. Chaton, C., Creti, A., Villeneuve, B. (2005, December). The Economics of Seasonal Gas Storage. *Energy Policy*, 36, 11, 4235-4246.

26. Chen, H., Baldick, R. (2007, February). Optimizing Short-Term Natural Gas Supply Portfolio for Electric Utility Companies. *IEEE Transactions on Power Systems*, 22, 1.

27. Chen, Z., Forsyth, P. (2007). A Semi-Lagrangian Approach for Natural Gas Storage Valuation and Optimal Operation. *SIAM Journal on Scientific Computing*, 30:339-368.

28. Chinneck, J. W. (2010). *Practical Optimization: A Gentle Introduction.* Carleton University, Ottawa, Canada. Retrieved September 6, 2010, from www.sce.carleton.ca/faculty/chinneck/po.html

29. Chriss, N. (1997). *Black-Scholes And Beyond.* Boston: McGraw Hill.

30. Chvatal, V. (1983). *Linear Programming*. W.H. Freeman & Company.

31. Civan, F. (2008). *Notes for "Oil and Gas Transportation and Storage"*. University of Oklahoma.

32. Clausen, J. (1999). *Branch and Bound Algorithms – Principles and Examples.* University Of Copenhagen.

33. Cormen, T., Leiserson, C., Rivest, R., Stein, C. (2001). *Introduction to Algorithms*. 2nd ed. Cambridge, Massachusetts: The MIT Press.

34. Dhar, A., Datta, B. (2008). Optimal Operation of Reservoirs for Downstream Water Quality Control Using Linked Simulation Optimization. *Hydrological Processes*, 22, 842-853.

35. Deng, S. (1998). *Stochastic Models of Energy Commodity Prices and Their Applications: Mean-Reversion with Jumps and Spikes.* University of California, Berkeley.

36. Dixit, A., Pindyck, R. (1994), *Investment Under Uncertainty*, Princeton University Press.

37. DOE (1993). *International Energy Annual 1993*, U.S. Department of Energy, Energy Information Administration Report DOE/EIA-0219(93).

38. DOE (1995). *Annual Energy Review 1995*, U.S. Department of Energy, Energy Information Administration Report DOE/EIA-0384(95).

39. DOE (2005). *Liquefied Natural Gas: Understanding the Basic Facts*. U.S. Department of Energy, Retrieved April 10, 2010, from http://fossil.energy.gov/ programs/oilgas/publications/lng/LNG_primerupd .pdf

40. EIA (2010). *Working Gas in Underground Storage*. US. Energy Information Administration, Office of Oil and Gas. Retrieved April 18, 2010, from http://ir.eia.gov/ngs/ngs.html

41. EIA (2012a). *U.S. Underground Natural Gas Storage Developments: 1998-2005* Energy Information Administration, Office of Oil and Gas. Retrieved February 18, 2012, from http://www.eia.gov

42. EIA (2012b). *Natural Gas Consumption by End Use.* Energy Information Administration, Office of Oil and Gas, Retrieved February 18, 2012, from http://www.eia.gov

43. EIA (2012c). *Natural Gas Prices*. Energy Information Administration, Office of Oil and Gas, Retrieved February 18, 2012, from http://www.eia.gov

44. EIA (2013). *Natural Gas Futures Contract 1*. Energy Information Administration, Office of Oil and Gas, Retrieved January 11, 2013, from http://www.eia.gov

45. EIA (2012d). Energy Information Administration, Retrieved January 11, 2013, from http://www.eia.gov/dnav/ng /ng_cons_heat_ a_EPG0_VGTH_btucf_a.htm

46. EPA (2012). *Landfill Methane Outreach Program*, Retrieved January 13, 2013, from http://www.epa.gov/lmop/

47. FERC (2004). *Current State of and Issues Concerning Underground Natural gas Storage*, Federal Energy Regulatory Commission Staff Report Sept 30, Retrieved January 13, 2013, from www.ferc.gov

48. FERC (2008). Federal Energy Regulatory Commission, Retrieved January 13, 2013, from www.ferc.gov

49. Felix, B., Weber, C. (2008). Gas Storage Valuation: Comparison of recombining Trees and Least Squares Monte-Carlo Simulation. *Engineering Management Conference. IEEE International*. 1-4.

50. Frayer, J., Uludere, N. (2001). What is it Worth? Application of Real Options Theory to the Valuation of Generation Assets. *The Electricity Journal*. 14, 8, 40-51.

51. Fourer, R., Ma, J., Martin, K. (2010). Optimization Services: A Framework for Distributed Optimization. *Operations Research*, 58, 6, 1624-1636.

52. Fu, M. (1994). Optimization via Simulation: A Review. *Annals of Operation Research*, 53, 199-248.

53. Fu, M., Hill, S. (1997). Optimization of Discrete Event Systems via Simultaneous Perturbation Stochastic Approximation, *IIE Transactions*, 29, 223-243.

54. Fu, M. (2001). Simulation Optimization. *Proceedings of the 2001 Winter Simulation Conference*, 53-61.

55. Fu, M. (2001a). *Encyclopedia of Operations Research and Management Science*, Kluwer Academic Publishers.

56. Fu, M., Laprise, S., Madan, D., Su, Y., Wu, R. (2001b, Spring). Pricing American Options: A Comparison of Monte Carlo Simulation Approaches. *Journal of Computational Finance*, 4, 3, 39-88.

57. Fu, M. (2002, Summer). Optimization for Simulation: Theory vs. Practice. *INFORMS Journal on Computing*, 14, 192-215.

58. Fu, M., Glover, F., April, J. (2005). Simulation Optimization: A Review, New Developments, and applications. *Proceedings of the 2005 Winter Simulation Conference*, 83-95.

59. Fu, M., Chen, C., Shi, L. (2008). Some Topics for Simulation Optimization. *Proceedings of the 2008 Winter Simulation Conference*, 27-38.

60. Gemen, H. (2006). Mean Reversion versus Random Walk in Oil and Natural Gas Prices. Retrieved January 13, 2013, from http://ieor.columbia.edu/files/seasdepts/industrial-engineering-operations-research/pdf-files/Geman.pdf 16 Jan 2013.

61. Garcia, A., Patek, S., Sinha, K. (2007, July-August). A Decentralized Approach to Discrete Optimization via Simulation: Application to Network Flow. *Operations Research*, 55, 4, 717-732.

62. Glover, F. (1989). Tabu Search – Part I. *ORSA Journal on Computing*, 1, 3, 190-206.

63. Glover, F. (1990). Tabu Search – Part II. *ORSA Journal on Computing*, 2, 1, 4-32.

64. Glover, F., Kelly, J., Laguna, M. (1999). New Advances for Wedding Optimization and Simulation. *Proceedings of the 1999 Winter Simulation Conference*, 255-260.

65. Hansen, M. (1996). Tabu Search for Multi-objective Optimization: MOTS, MCDM '97, Capetown, South Africa.

66. Happe, M., Lubbers, E., Platzner, M. (2010). A Multithreaded Framework for Sequential Monte Carlo Methods on CPU/FPGA Platforms. Lecture Notes in Computer Science. 5453, 380-385.

67. Ho, Y., Zhao, Q., Jia, Q. (2007). *Ordinal Optimization Soft Optimization for Hard Problems.* Springer.

68. Hooke, R., Jeeves, T. (1961). Direct Search Solution of Numerical and Statistical Problems. *Journal of the Association for Computing Machinery*, 8, 2, 212-229.

69. Hodges, S. (2004). The Value of a Storage Facility. *Techreport*, 04-142. Retrieved January 13, 2013, from http://www2.warwick.ac.uk/ fac/soc/wbs/research/wfri/rsrchcentres/forc/preprintseries/

70. Holland, A. (2007). Injection/Withdrawal Scheduling for Natural Gas Storage Facilities. *Proceedings of the ACM Symposium on Applied Computing.*

71. Holland, A. (2007b). Optimization of Injection/Withdrawal Scheduling for Natural Gas Storage Facilities. Retrieved June 26, 2012 from http://www.4c.ucc.ie/ ~aholland/publications/GasStorage.pdf

72. Holland, A. (2008). Welfare Losses in Commodity Storage Games (Extended Abstract). *Proceedings of the 8th Int. Conf on Autonomous Agents and Multi-Agent Systems (AAMAS2009)*, May 10-15. Budapest, Hungary, 1253-1254.

73. Homem-De-Mello, T. (2008). On Rates of Convergence for Stochastic Optimization Problems under Non–Independent and Identically Distributed Sampling, *SIAM Journal of Optimization*, 19, 2, 524–551.

74. Hsu, C., Pino, J., Bhattacharyya, S. (2011, June). Multithreaded Simulation for Synchronous Dataflow Graphs. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 16, 3.

75. Hu, J., Fu, M., Marcus, S. (2007). A Model Reference Adaptive Search method for Global Optimization. *Operations Research*, 55, 3, 549-568.

76. Hu, J., Chang, H., Fu, M., Marcus, S. (2011). Dynamic Sample Budget Allocation in Model-Based Optimization. *Journal of Global Optimization*, 50, 575-596.

77. Hull, J. (2005). *Options, Futures, and Other Derivative Securitie 6th ed*, Prentice Hall.

78. Jolliffe, I. (2002). *Principal Component Analysis, 2nd ed*. New York: Springer.

79. Kabirian, A., Olafsson, S., (2011). Continuous Optimization via Simulation using Golden Region Search. *European Journal of Operational Research*, 208, 19-27.

80. Keppo, J., Lu,H. (2003, Sept). Real Option and a Large Producer: the Case of Electricity Markets. *Energy Economics*, 25, 5, 459-472.

81. Kiefer, J, Wolfowitz, J, (1952). Stochastic Estimation of the Maximum a Regression Function. *Annals of Mathematical Statistics*, 23 462-466.

82. Kim, J., Olafsson, S. (2002). Two-Stage NP Method with Inheritance. *Proceedings of the 2002 Winter Simulation Conference*, 279-284.

83. Kirkpartick, S., Gelatt, C., Vecchi, M. (1983, May 13). Optimization by Simulated Annealing. *Science*, New Series, 220, 4598, 671-680.

84. Korpela, E., Werthimer, D., Anderson, D., Cobb, J., Lebofsky, M. (2001, January-February). SETI@Home—Massively Distributed Computing for SETI. *Computing in Science & Engineering*, 78-83.

85. Kuriger, G. (2006). Multi-Criteria-simulation Optimization with Stochastic Coefficients: Methods, Performance Measures and Test Bed Problems, Doctoral Dissertation, The University of Oklahoma.

86. Kuriger, G., Grant, F. (2010). A Lexicographic Nelder-Mead Simulation Optimization Method To Solve Multi-Criteria Problems. *Computers & Industrial Engineering*, 60, 4, 555-565.

87. Lai, G., Francois, M., Secomandi, N. (2011). An Approximate Dynamic Programming Approach to Benchmark Practice-Based Heuristics for Natural Gas Storage Valuation. *Operations Research*, 58, 3, 564-582.

88. Lai, G., Wang, M., Secomandi, N., Kekre, S., Scheller-Wolf, A. (2011). Valuation of Storage at a Liquefied Natural Gas Terminal. *Operations Research*, 59, 3, 602-616.

89. Lawler, E. (1976). *Combinatorial Optimization Networks and Matroids.* Mineola, New York: Dover Publications, Inc.

90. Lee, I. (2010). Analyzing Performance and Power of Multi-core Architecture Using Multithreaded Iterative Solver. *Journal of Computer Science*, 6, 4, 406-412.

91. Li, Y., (2007, December). Natural Gas Storage Valuation. Master's Thesis. Georgia Institute of Technology.

92. Liu, A., Yang, M. (2012). A New Hybrid Nelder-Mead Particle Swarm Optimization for Coordination Optimization of Directional Overcurrent Relays. *Mathematical Problems in Engineering*, 2012.

93. Liu, W., Li, M., Liu, Y., Xu, Y., Yang, X. (2009). Decision of Optimal Scheduling Scheme for Gas Field Pipeline Network Based on Hybrid Genetic Algorithm, *World Summit on Genetic and Evolutionary Computation (GEC '09).* Shanghai, China.

94. Longstaff, F., Schwartz, E. (2001). Valuing American Options by Simulation: A simple Least-Squares Approach. *The Review of Financial Studies*, 14, 1, 113-147.

95. Lp_solve (2010). Lp_solve Reference Guide. Retrieved December 10, 2010 from http://lpsolve.sourceforge.net/5.5/

96. Lou, Y., Chen, C., Yucesan, E., Lee, I. (2000). Distributed Web-Based Simulation Optimization. *Proceedings of the 2000 Winter Simulation Conference*, 1785-1793.

97. Man, K., Tang, K., Kwong, S. (1999). *Genetic Algorithms: concepts and designs.* London: New York: Springer.

98. Marks, F., (2011). *Landfill Gas to Power Wewoka Brick Company's Kilns*. Retrieved January 13, 2012 from http://newsok.com

99. Montgomery, Douglas C. (2001). *Design and Analysis of Experiments, 5th ed*. New York: Wiley.

100. Mousavi, S., Hajipour, V., Niaki, S. Alikar, N. (2013). Optimizing Multi-Item Multi-Period Inventory Control System with Discounted Cash Flow and Inflation: Two Calibrated Meta-Heuristic Algorithms. *Applied Mathematical Modeling,* 37, 4, 2241–2256.

101. NEB (2001). North American Natural Gas Liquids Pricing and Convergence. National Energy Board of Canada, Retrieved March 30, 2011 from www.neb-one.gc.ca/clf.../gslqdsprcngcvrgncnrthmrc2001-eng.pdf

102. Nelder, J., Mead, R. (1965). A Simple Method for Function Minimization. *The Computer Journal,* 7, 4, 308-313.

103. Newell, D., Bhattacharya, S., Sears, M. (2009, September 14). Low-Btu Gas in the US Mid-continent: A Challenge for Geologists and Engineers. *Oil& Gas Journal*.

104. NIST 2010. Retrieved April 20, 2010 from http://webbook.nist.gov/ chemistry

105. Olafsson, S., Kim, J. (2002). Simulation Optimization. *Proceedings of the 2002 Winter Simulation Conference*, 79-84.

106. Padberg, U., Haubrich, H. (2008). Stochastic Optimization of natural Gas Portfolios, *Electricity Market*, 1-6.

107. Pearson, K. (1901). On Lines and Planes of Closest Fit to Systems of Points in Space. *Philosophical Magazine,* 2, 6, 559–572.

108. Pepelyaev, V. (2006). Planning Simulation-Optimization Experiments. *Cybernetics and System Analysis*, 42, 6, 866-875.

109. Pichitlamken, J., Nelson, B. (2003, April). A Combined Procedure for Optimization via Simulation", *ACM Transactions on Modeling and Computer Simulation*, 13, 2, 155-179.

110. Papadimitriou, C., Steiglitz, K. (1982). *Combinatorial optimization Algotithms and Complexity*. New Jersey: Prentice Hall.

111. Poland, M., Nugent, C., Wang, H., Chen, L. (2011). Pure Random Search for Ambient Sensor Distribution Optimization In A Smart Home Environment. *Technology and Health Care*, 19, 137-160.

112. Powell, W. (2007). The Optimizing-Simulator: Merging Simulation and Optimization Using Approximate Dynamic Programming. *Proceedings of the 2007 Winter Simulation Conference*, 43-53.

113. Pritsker, A., O'Reilly, J. (1999). *Simulation with Visual SLAM and AweSim. 2nd ed*. Wiley.

114. Ravindran, A., Ragsdell, K., Reklaitis, G. (2006). *Engineering Optimization Methods and Applications, 2nd Edition.* Haryana, India: Wiley-India.

115. Reeves, C., Rowe, J. (2003). *Genetic Algorithms: Principles and Perspectives: A Guide to GA Theory*. Boston: Kluwer Academic Publishers.

116. Robbins, H., Monro, S. (1951). A Stochastic Approximation Method. *Annals of Mathematical Statistics*, 22, 400-407.

117. Rosen, S., Harmonosky, C., Traband, M. (2007). A Simulation Optimization Method That Considers Uncertainty and Multiple Performance Measures. *European Journal of Operational Research*, 181, 315–330.

118. Ross, S. (2003). *Introduction to Probability Models, 8th ed*. Academic Press.

119. Routledge, B., Seppi, D., Spatt, C. (2000, June). Equilibrium Forward Curves for Commodities. *The Journal of Finance*, 55, 3, 1297-1338.

120. Sieminski, A. (2007). Varying Views on the Future of the Natural Gas Market Secrets of Energy Price Forecasting. Retrieved April 20, 2010 from www.eia.gov/oiaf/aeo/conf/pdf/sieminski.pdf

121. Segars, M., Sanchez, R., Cannon, P., Binkowski, B., Gutierrez, C., Hailey, D. (2011). Blending Fuel Gas to Optimize use of Off-Spec Natural Gas, *ISA Power Industry Division 54th Annual I&C Symposium.*

122. Shi, L., Olafsson, S. (2000, May-June). Nested Partitions Method for Global Optimization. *Operations Research*, 48, 3, 390-407.

123. Smith, J., McCardle, K. (1999, January-February). Options in the Real World: Lessons learned in Evaluating Oil and Gas Investments. *Operations Research*, 47, 1.

124. Spall, J. (1992). Multivariate Stochastic Approximation Using a Simultaneous Perturbation Gradient Approximation. *IEEE Transactions on Automatic Control,* 37, 3, 332–341.

125. Sundaram, R. (1996). *A First Course in Optimization Theory*. Cambridge University Press.

126. Suri, R., Zazanis, M. (1988). Perturbation Analysis Gives Strongly Consistent Sensitivity Estimates for the M/G/1 Queue". *Management Science,* 34, 39-64.

127. Swisher, J., Hyden, P., Jacobson, S., and Schruben, L. (2000). A Survey of Simulation Optimization Techniques and Procedures. *Proceedings of the 2000 Winter Simulation Conference,* 119-228.

128. Tek, M. (1996). *Natural Gas Underground Storage: Inventory and Deliverability*, Pennwell.

129. Tekin, E., Sabuncuoglu, I. (2004). Simulation Optimization: A Comprehensive Review on Theory and Applications. *IIE Transactions,* 36, 1067-1081.

130. Thompson, M., Davidson, M, Rasmussen, H. (2002), Natural Gas Storage Valuation and Optimization: A Real Options Application, Department of Applied Mathematics, University of Ontario. Retrieved March 30, 2011 from www.apmaths.uwo.ca/~mdavison /_library/ preprints/Gasstorage.pdf

131. UNCTAD. Retrieved March 30, 2011 from http://www.unctad.org/ infocomm/anglais/gas/quality.htm

132. Vaitheeswaran, N. and Balasubramanian, R. (2010). Stochastic Model for Natural Gas Portfolio Optimization of a Power Producer. *Power Electronics, Drives and Energy Systems (PEDES) & 2010 Joint International Conference on Power India,* 1-5.

133. Vocaturo, F. (2008). Optimization via Simulation for Logistic Systems Planning and Control. *Journal of Operations Research,* 7, 97-100.

134. Wei, W., Wei, J., Guan, X., Shi, L. (2012). A Hybrid Nested Partitions Algorithm For Scheduling Flexible Resource In Flow Shop Problem. *International Journal of Production Research,* 50, 10, 2555-2569.

135. Yang, T., Kuo, Y., Chang, I. (2004, October 1). Tabu-Search Simulation Optimization Approach For Flow-Shop Scheduling With Multiple

Processors – A Case Study, *International Journal of Production Research* 42, 19, 4015-4030.

136. Yau, H., Shi, L., (2009). Nested Partitions for the Large-Scale Extended Job Shop Scheduling Problem. *Annals of Operations Research,* 168, 23-39.

137. Yucesan, E., Jacobson, S. (1996). Computational Issues for Accessibility in Discrete Event Simulation. *ACM Transactions on Modeling and Computer Simulation,* 6, 1, 53-75.

138. Zlochin, M; Birattari, M; Meuleau, N; Dorigo, M. (2004). Model-Based Search for Combinatorial Optimization: A Critical Survey. *Annals of Operations Research,* 131, 1,373-380.

Appendix 1: Initial Test Data and Parameters

Test Data Set 1

Period	Cost A	Cost B	Price A	Price AB	Price B
1	1.0	1.0	2.0	1.0	2.0
2	1.0	1.0	2.0	1.0	2.0
3	1.0	1.0	2.0	1.0	2.0
4	1.0	1.0	2.0	1.0	2.0
5	1.0	1.0	2.0	1.0	2.0
6	1.0	1.0	2.0	1.0	2.0
7	1.0	1.0	2.0	1.0	2.0
8	1.0	1.0	2.0	1.0	2.0
9	1.0	1.0	2.0	1.0	2.0
10	1.0	1.0	2.0	1.0	2.0
11	1.0	1.0	2.0	1.0	2.0
12	1.0	1.0	2.0	1.0	2.0

Best results =([Buy 100%A, Buy100%B][Sell 100%A, Sell100%B])
*6
Max Expected Value:
1200.0

Max Trans A	100.0	aInAb	0.5
Max Trans B	100.0	bInAB	0.5
costStorage A	0.0	price variance	none
costStorage B	0.0	cost variance	none
max storage A	5000.0		
max storage B	5000.0		

Appendix 2: Results of Initial Test Runs

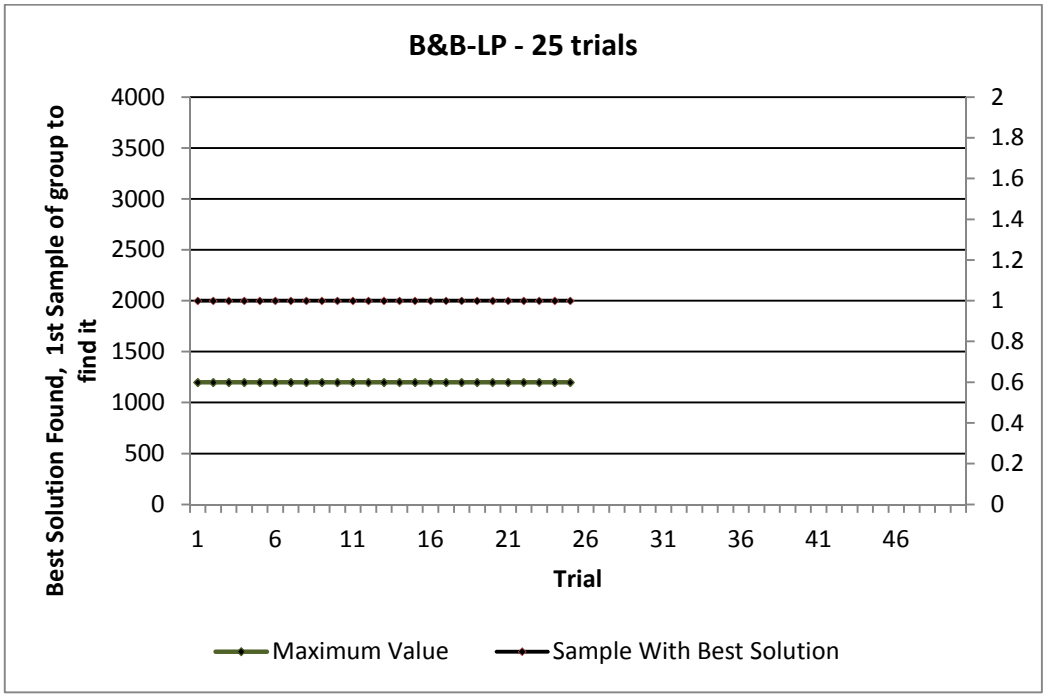


Figure 10 B&B-LP Trial Results

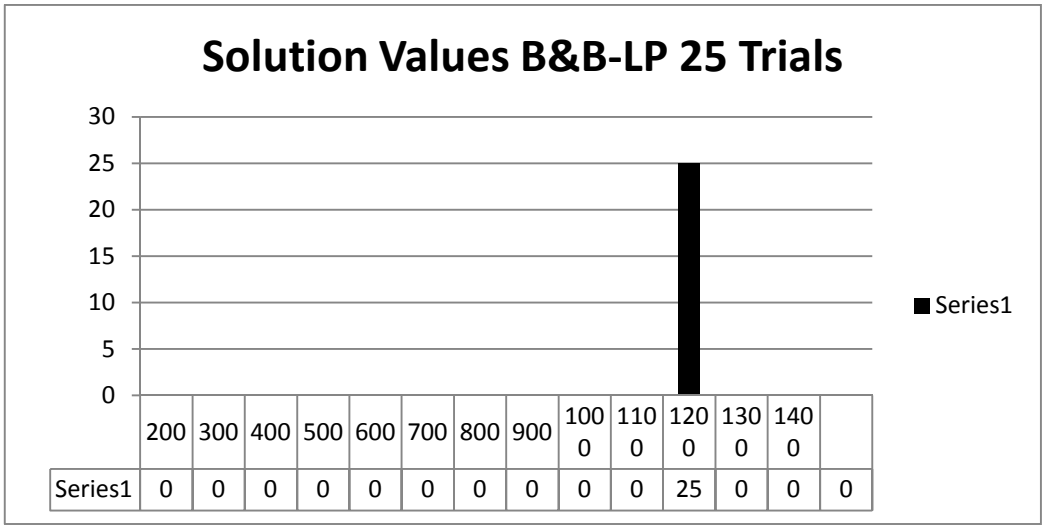


Figure 11 Solution Values B&B-LP 25 Trials

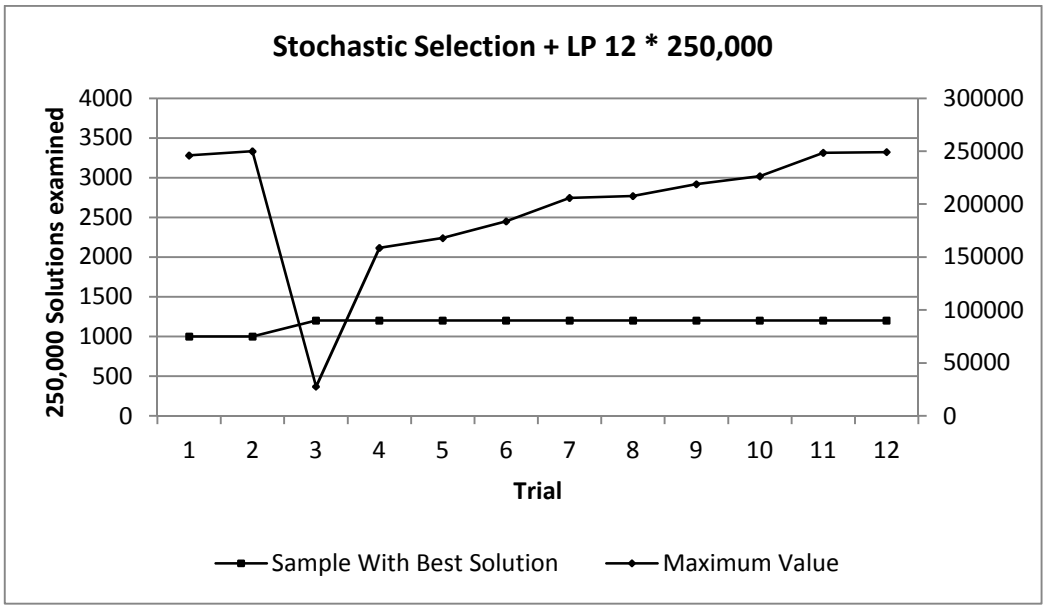


Figure 12 Results of SS-LP, 12x250,000

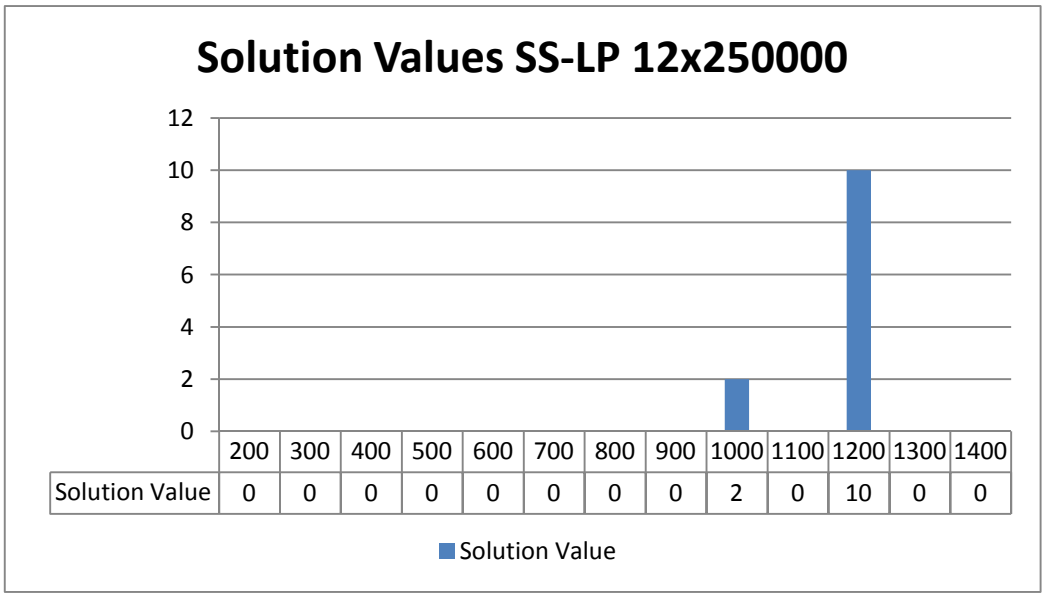


Figure 13 Solution Values SS-LP 12x250,000

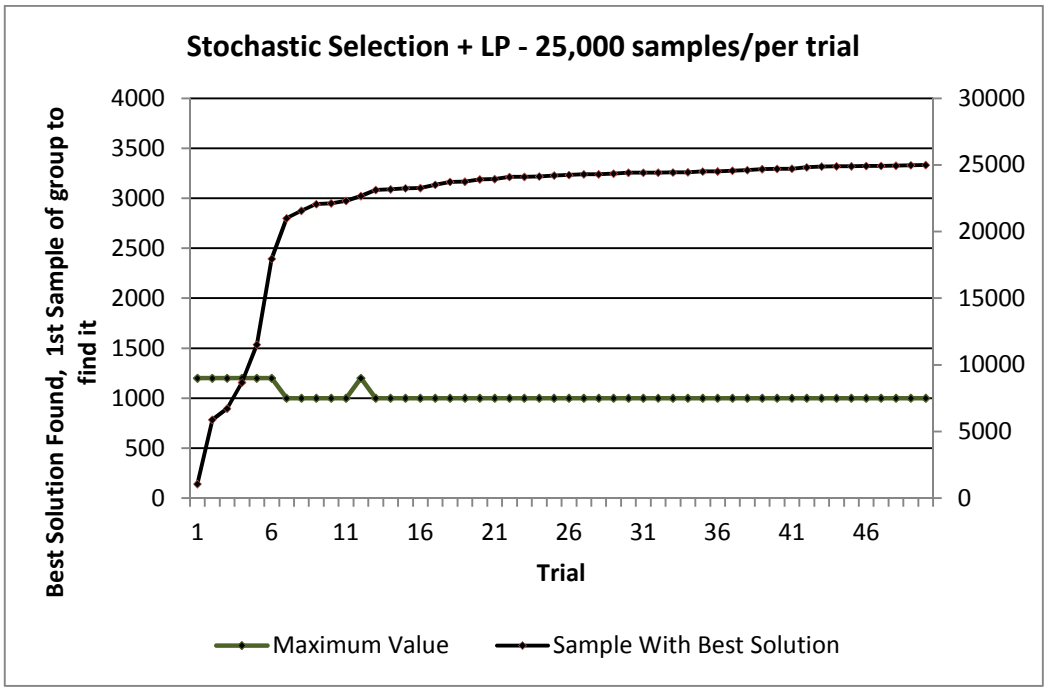
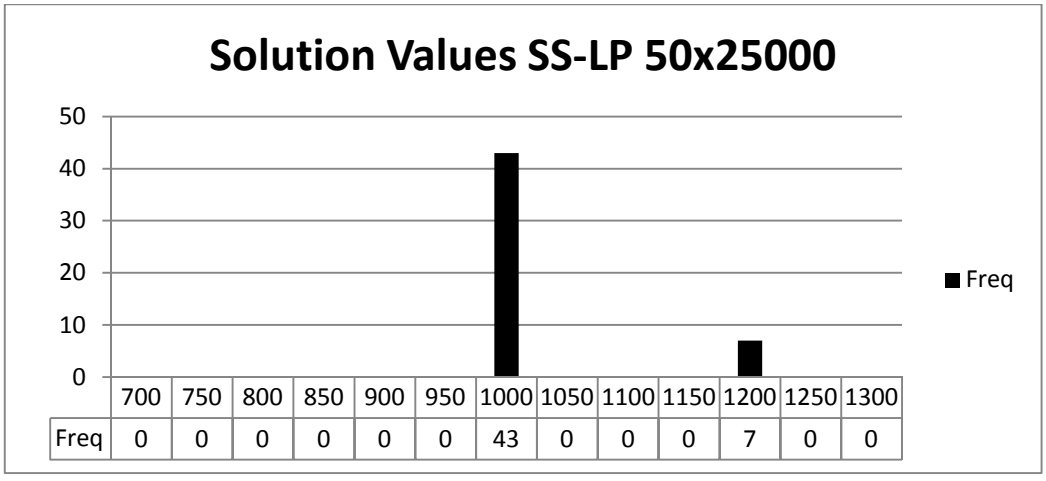


Figure 14 Results of SS-LP 50x25,000



	700	750	800	850	900	950	1000	1050	1100	1150	1200	1250	1300
Freq	0	0	0	0	0	0	43	0	0	0	7	0	0

Figure 15 Solution Values SS-LP 50x25,000

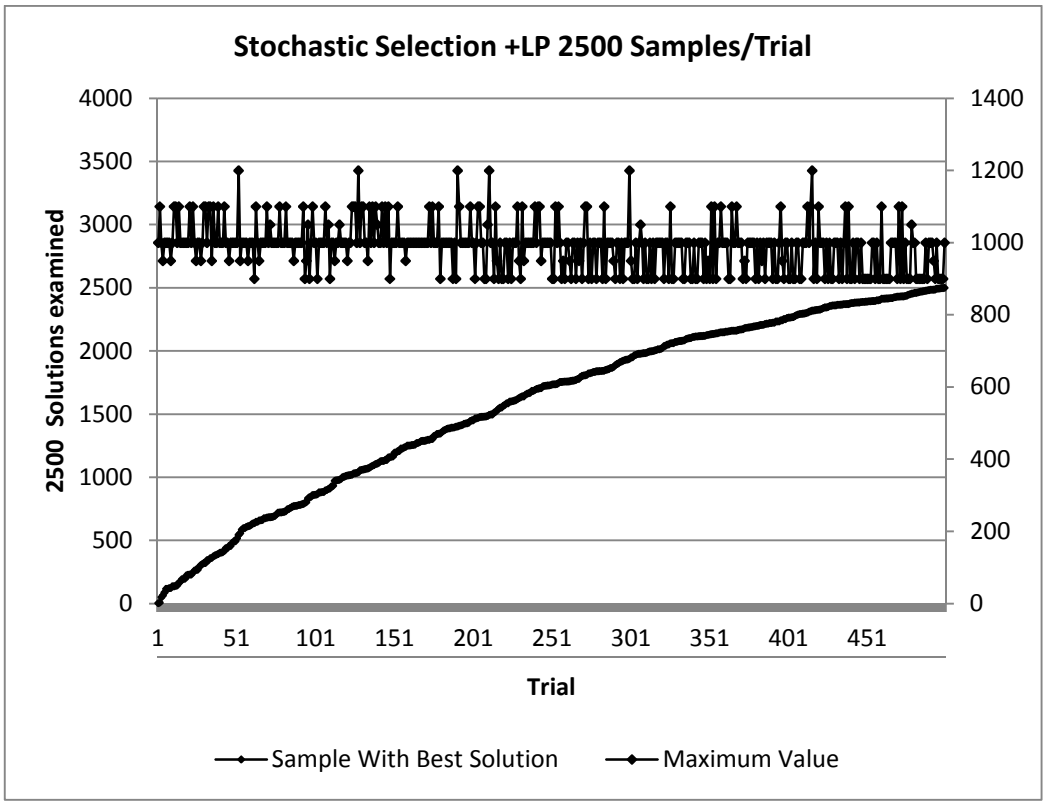
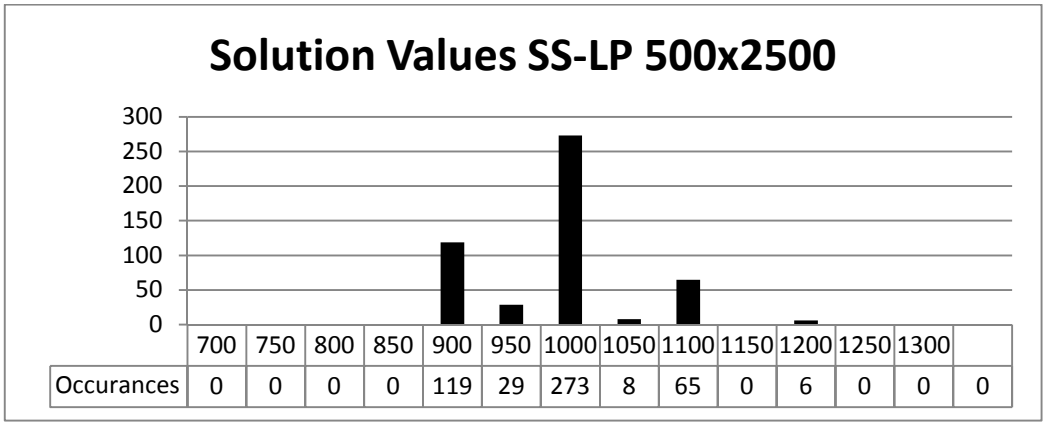


Figure 16 Results of SS+LP 50x2500



	700	750	800	850	900	950	1000	1050	1100	1150	1200	1250	1300	
Occurances	0	0	0	0	119	29	273	8	65	0	6	0	0	0

Figure 17 Solution Values SS-LP 500x2500

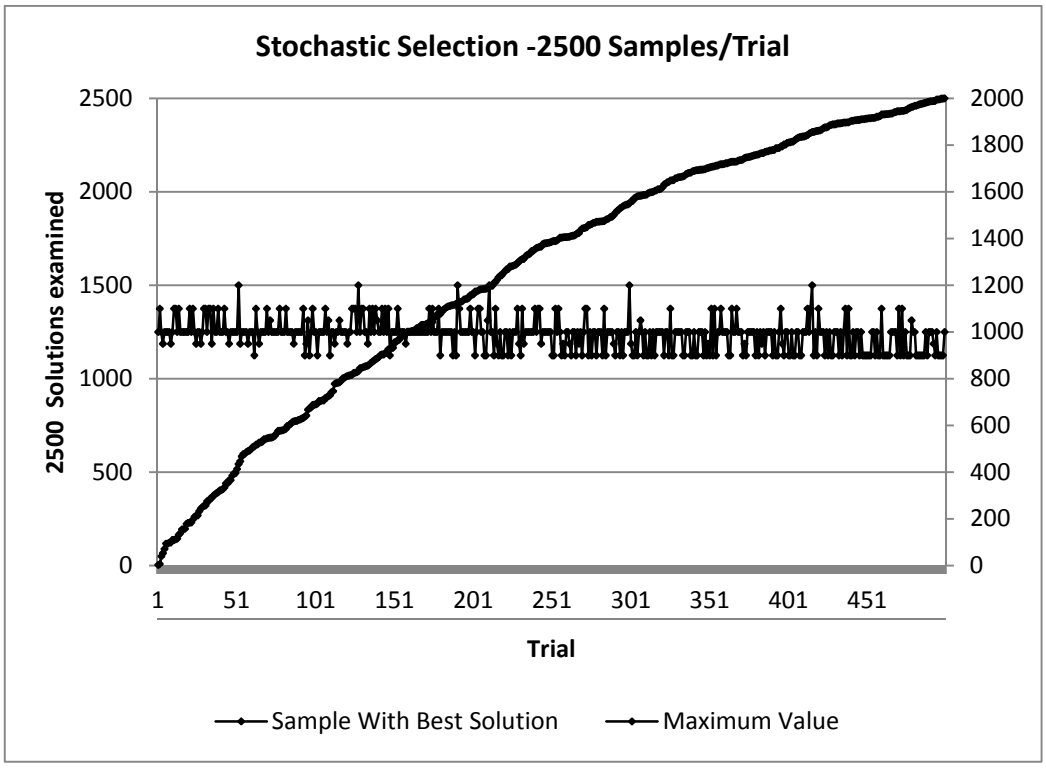


Figure 18 Stochastic Selection -2500 Samples/Trial

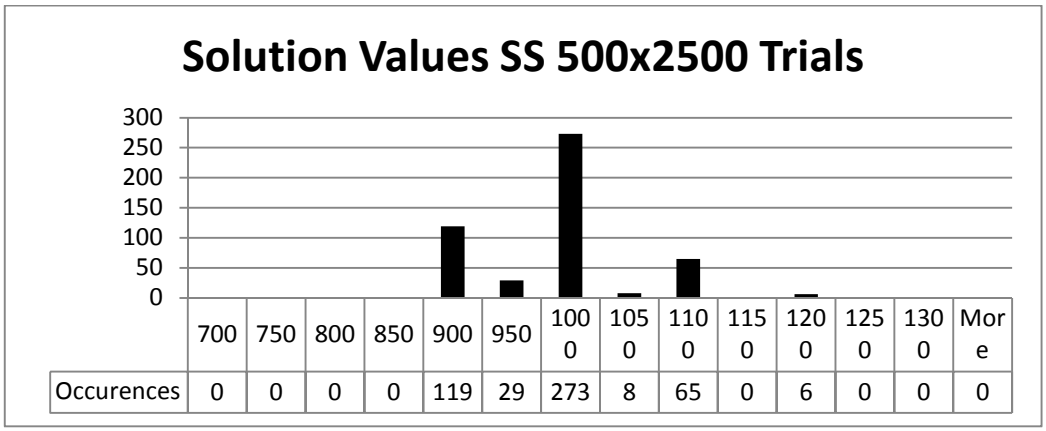


Figure 19 Stochastic Selection -500x2500 Samples/Trial

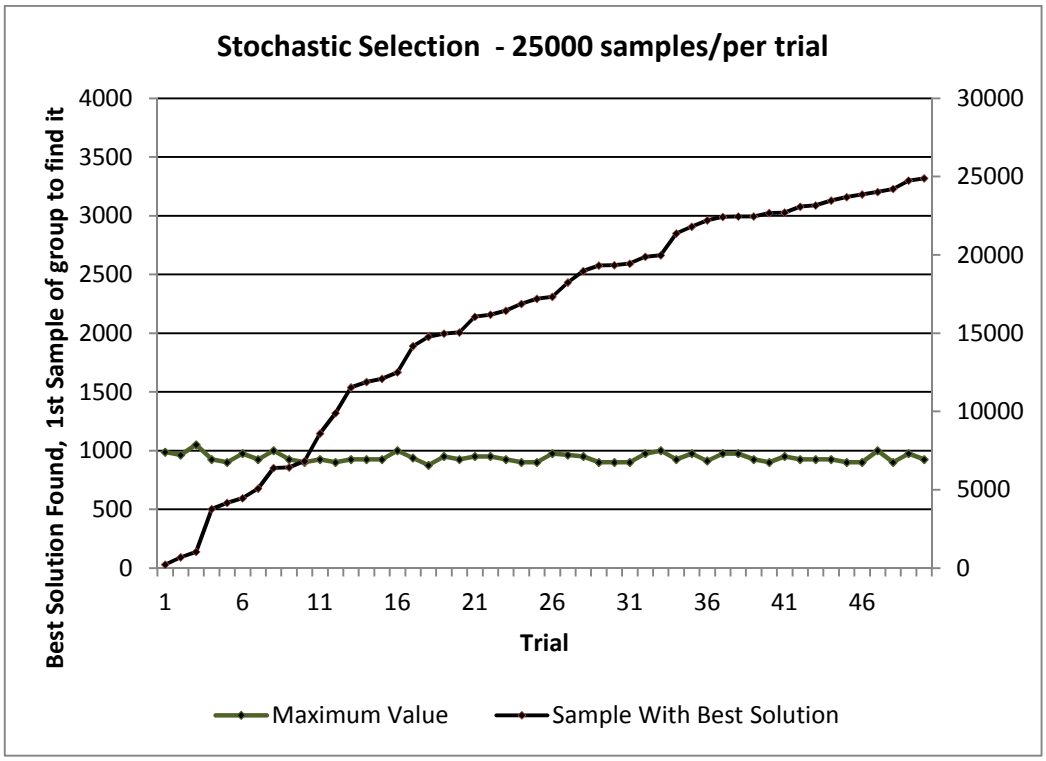


Figure 20 SS-LP 25000 Samples/Trial

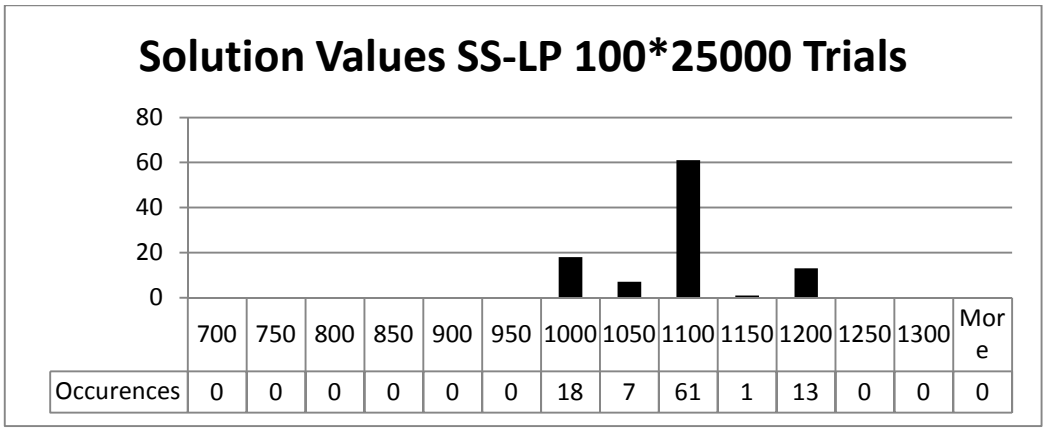


Figure 21 SS-LP 100x25000 Samples/Trial

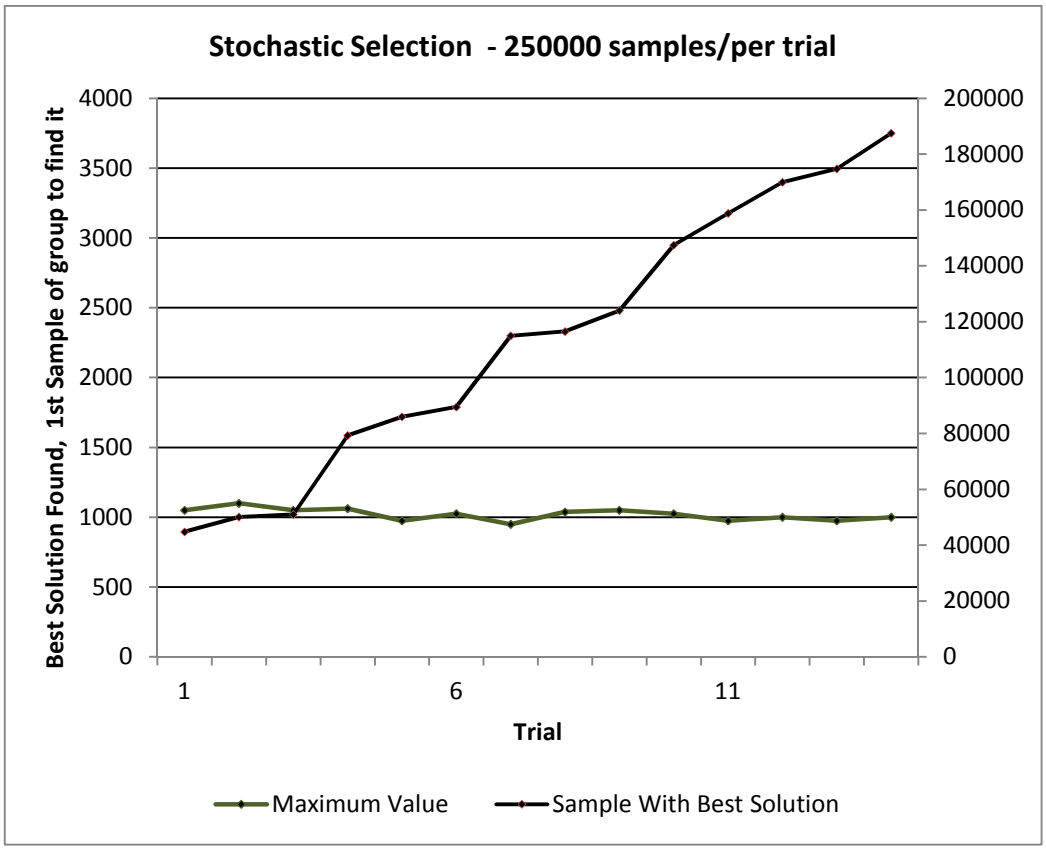


Figure 22 SS 100x250000 Samples/Trial

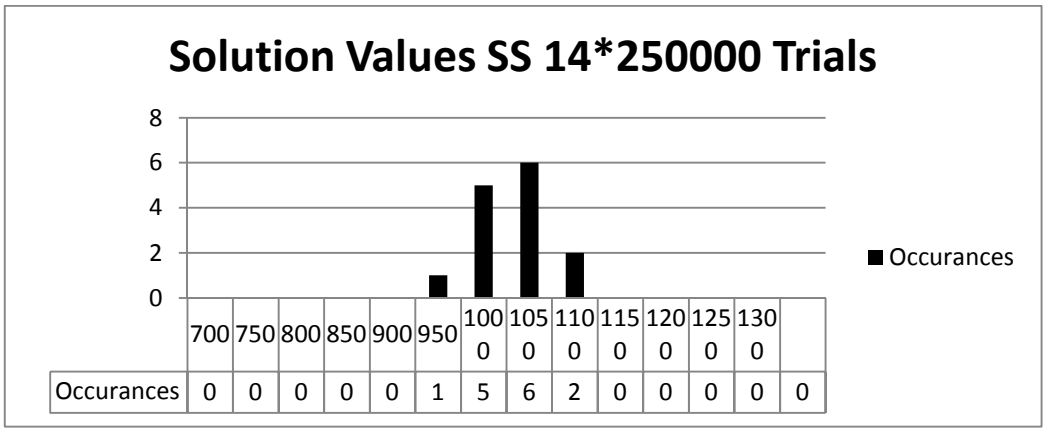


Figure 23 SS Values 100x250000 Samples/Trial

Appendix 3: Trial Price/Cost Data II

Month	GasA Cost	GasA Price	Markup	GasB Cost	GasB Price	GasAB Cost	Gas AB Price	GasAB Price Adjusted
Jan	5.83	5.52	-0.05	2.62	2.48	3.91	3.70	3.14
Feb	5.32	6.58	0.24	2.39	2.96	3.56	4.41	3.75
mar	4.29	6.38	0.49	1.93	2.87	2.87	4.27	3.63
Apr	4.03	6.02	0.49	1.81	2.71	2.70	4.03	3.43
May	4.14	6.64	0.60	1.86	2.99	2.77	4.45	3.78
Jun	4.80	6.41	0.34	2.16	2.88	3.22	4.29	3.65
Jul	4.63	6.63	0.43	2.08	2.98	3.10	4.44	3.78
Aug	4.32	7.29	0.69	1.94	3.28	2.89	4.88	4.15
Sep	3.89	7.23	0.86	1.75	3.25	2.61	4.84	4.12
Oct	3.43	6.05	0.76	1.54	2.72	2.30	4.05	3.45
Nov	3.71	7.07	0.91	1.67	3.18	2.49	4.74	4.03
Dec	4.25	5.22	0.23	1.91	2.35	2.85	3.50	2.97

Gas A Costs are based on Henry Hub spot prices – see appendix 5

Gas B Prices are based on Oklahoma City Gate prices – see appendix 6

Appendix 4: Natural Gas Futures Contract 1

Year	2000	2001	2002	2003	2004	2005	2006
Jan	2.385	7.825	2.190	5.381	6.272	6.186	9.136
Feb	2.614	5.675	2.263	6.657	5.363	6.203	7.520
Mar	2.828	5.189	3.015	5.786	5.542	7.045	6.979
Apr	3.028	5.189	3.410	5.358	5.765	7.150	7.264
May	3.596	4.244	3.563	5.926	6.398	6.486	6.372
Jun	4.303	3.782	3.259	5.925	6.334	7.206	6.385
Jul	3.972	3.167	2.942	5.034	6.064	7.579	6.222
Aug	4.460	2.935	3.092	4.978	5.471	9.427	6.989
Sep	5.130	2.213	3.569	4.667	5.219	12.111	5.218
Oct	5.079	2.618	4.088	4.986	7.371	13.454	6.633
Nov	5.740	2.786	4.040	4.834	7.608	11.695	7.995
Dec	8.618	2.686	4.838	6.469	6.828	13.425	7.161

Year	2007	2008	2009	2010	2011	2012
Jan	6.775	7.991	5.07	5.599	4.499	2.708
Feb	7.546	8.642	4.382	5.215	4.036	2.526
Mar	7.221	9.624	4.002	4.301	4.069	2.296
Apr	7.629	10.288	3.561	4.088	4.272	2.048
May	7.821	11.381	3.934	4.155	4.336	2.493
Jun	7.503	12.784	3.935	4.785	4.516	2.498
Jul	6.399	11.067	3.551	4.590	4.353	2.963
Aug	6.137	8.301	3.305	4.220	3.984	2.807
Sep	6.188	7.485	3.462	3.898	3.849	2.918
Oct	7.223	6.727	4.780	3.600	3.624	3.500
Nov	7.778	6.700	4.628	4.042	3.558	3.687
Dec	7.178	5.794	5.344	4.283	3.246	

(Dollars/Mil. BTUs) (EIA 2013)

Appendix 5: Henry Hub Natural Gas Spot Prices

Year	2000	2001	2002	2003	2004	2005	2006
Jan	2.42	8.17	2.32	5.43	6.14	6.15	8.69
Feb	2.66	5.61	2.32	7.71	5.37	6.14	7.54
Mar	2.79	5.23	3.03	5.93	5.39	6.96	6.89
Apr	3.04	5.19	3.43	5.26	5.71	7.16	7.16
May	3.59	4.19	3.50	5.81	6.33	6.47	6.25
Jun	4.29	3.72	3.26	5.82	6.27	7.18	6.21
Jul	3.99	3.11	2.99	5.03	5.93	7.63	6.17
Aug	4.43	2.97	3.09	4.99	5.41	9.53	7.14
Sep	5.06	2.19	3.55	4.62	5.15	11.75	4.90
Oct	5.02	2.46	4.13	4.63	6.35	13.42	5.85
Nov	5.52	2.34	4.04	4.47	6.17	10.30	7.41
Dec	8.90	2.30	4.74	6.13	6.58	13.05	6.73

Year	2007	2008	2009	2010	2011	2012
Jan	6.55	7.99	5.24	5.83	4.49	2.67
Feb	8.00	8.54	4.52	5.32	4.09	2.51
Mar	7.11	9.41	3.96	4.29	3.97	2.17
Apr	7.60	10.18	3.50	4.03	4.24	1.95
May	7.64	11.27	3.83	4.14	4.31	2.43
Jun	7.35	12.69	3.80	4.80	4.54	2.46
Jul	6.22	11.09	3.38	4.63	4.42	2.95
Aug	6.22	8.26	3.14	4.32	4.06	2.84
Sep	6.08	7.67	2.99	3.89	3.90	2.85
Oct	6.74	6.74	4.01	3.43	3.57	3.32
Nov	7.10	6.68	3.66	3.71	3.24	3.54
Dec	7.11	5.82	5.35	4.25	3.17	3.34

(Dollars/Mil. BTUs) (EIA 2013)

Appendix 6: Oklahoma Natural Gas Citygate Prices

Year	2000	2001	2002	2003	2004	2005	2006
Jan	6.61	9.63	4.07	4.94	6.21	7.12	10.01
Feb	2.66	6.85	3.78	5.41	6.48	6.70	10.59
Mar	3.01	6.39	4.07	7.71	6.31	6.95	9.52
Apr	2.88	6.76	4.48	5.13	6.82	7.11	8.17
May	3.36	4.50	4.13	6.04	6.11	7.62	8.11
Jun	3.19	4.25	3.77	5.90	6.48	7.23	7.89
Jul	4.14	4.10	3.63	5.34	6.42	7.89	8.66
Aug	4.48	5.30	3.57	5.53	6.32	8.85	8.58
Sep	3.57	5.18	4.20	5.36	6.18	10.59	8.48
Oct	4.94	4.95	4.37	7.14	5.68	10.74	6.40
Nov	5.60	5.10	4.93	6.36	6.94	11.14	8.28
Dec	5.58	4.49	4.72	6.17	8.00	11.39	8.78

Year	2007	2008	2009	2010	2011	2012
Jan	7.72	7.89	7.76	5.52	5.37	4.96
Feb	8.52	8.35	7.15	6.58	5.34	4.99
Mar	9.48	9.34	7.34	6.38	5.72	4.92
Apr	7.80	9.01	6.95	6.02	5.79	5.59
May	8.33	10.34	6.60	6.64	6.45	5.43
Jun	8.79	11.45	6.68	6.41	6.32	4.27
Jul	8.49	12.25	7.15	6.63	6.51	5.85
Aug	7.89	9.64	8.21	7.29	6.87	5.43
Sep	7.56	8.92	7.61	7.23	6.60	5.34
Oct	7.88	7.35	6.99	6.05	6.59	4.95
Nov	8.52	7.61	7.26	7.07	6.28	
Dec	7.80	7.78	5.84	5.22	5.18	

(Dollars per Thousand Cubic Feet) (EIA 2013)

Appendix 7: Natural Gas Consumption by End Use

Total Consumption	21,699	23,104	23,277	22,910	24,087	24,385
Lease and Plant Fuel	1,142	1,226	1,220	1,275	1,286	1,323
Lease Fuel	783	861	864	913	917	938
Plant Fuel	359	365	356	362	369	384
Pipeline & Distribution Use	584	621	648	670	674	684
Volumes Delivered to Consumers	19,973	21,256	21,409	20,965	22,127	22,378
Residential	4,368	4,722	4,892	4,779	4,782	4,714
Commercial	2,832	3,013	3,153	3,119	3,103	3,154
Industrial	6,527	6,655	6,670	6,167	6,826	6,905
Vehicle Fuel	24	25	26	27	29	32
Electric Power	6,222	6,841	6,668	6,873	7,387	7,574

(Billion Cubic Feet) (EIA 2013)

Appendix 8: C++ SOURCE CODE FOR LP-SOLVE INTERFACE

```
#include "c:\awesim\lib\vslam.h"
#include <math.h>
#include <stdlib.h>
#include "c:\awesim\lp_solve\lp_lib.h"
#include <windows.h>
#include "string.h"
#include <time.h>


double USERF(int IFN, ENTITY *peCur);
void getMixedProduct();
double minf(double vala, double valb);
double maxf(double vala, double valb);




add_constraint_func *_add_constraint;
add_constraintex_func *_add_constraintex;
delete_lp_func *_delete_lp;
get_col_name_func *_get_col_name;
get_objective_func *_get_objective;
get_variables_func *_get_variables;
make_lp_func *_make_lp;
print_lp_func *_print_lp;
print_solution_func *_print_solution;
read_LP_func *_read_LP;
set_add_rowmode_func *_set_add_rowmode;
set_col_name_func *_set_col_name;
set_maxim_func *_set_maxim;
set_obj_fn_func *_set_obj_fn;
set_obj_fnex_func *_set_obj_fnex;
```

```
set_verbose_func *_set_verbose;
solve_func *_solve;
write_LP_func *_write_LP;

FILE *runlogout;
FILE *runlogout2;
time_t now;


// Events
#define WRITELN 1
#define SIMPLEX 2
#define CALC_PROFITS 3
#define PRUNE_BUY 4
#define PRUNE_HOLD 5
#define PRUNE_SELL 6
#define CALC_BUY 7
#define ENDOFCYCLE 8
#define RESETINVENTORY 9
#define TIMER_GET 10
#define TIMER_SET 11
#define VARIABLE_BUY_SELL 12
#define RANDOM_A       13
#define RANDOM_AB      14
#define RANDOM_B       15
#define MANDATORY_SELL 16
#define DUMMY          99

#define PRUNE 0
#define NO_PRUNE 1



#define CostAidx          LL[1]
#define CostAidxBase      LL[2]
#define CostAidxMax       LL[3]
#define CostBidx          LL[5]
```

```
#define CostBidxBase       LL[6]
#define CostBidxMax        LL[7]
#define Horizon            LL[9]
#define PriceABidx         LL[11]
#define PriceABidxBase     LL[12]
#define PriceABidxMax      LL[13]
#define PriceAidx          LL[15]
#define PriceAidxBase      LL[16]
#define PriceAidxMax       LL[17]
#define PriceBidx          LL[19]
#define PriceBidxBase      LL[20]
#define PriceBidxMax       LL[21]
#define ReadResult         LL[23]
#define InventoryABase     LL[24]
#define InventoryBBase     LL[25]
#define CurrScenerio       LL[26]
#define AvgCostOfInvA      LL[30]
#define AvgCostOfInvB      LL[31]
#define ClockTime          LL[34]

#define CostOfStorageA     XX[3]
#define CostOfStorageB     XX[4]

#define Profit_A                peCur->ATRIB[11]
#define Profit_AB               peCur->ATRIB[12]
#define Profit_B                peCur->ATRIB[13]
#define aInAB                   peCur->ATRIB[14]
#define bInAB                   peCur->ATRIB[15]
#define CurrBuyVolA             peCur->ATRIB[17]
#define CurrBuyVolB             peCur->ATRIB[18]
#define changeAPerCent     peCur->ATRIB[19]
#define changeABPerCent    peCur->ATRIB[20]
#define changeBPerCent     peCur->ATRIB[21]
#define CurrDltVlA         peCur->ATRIB[22]
#define CurrDeltaVolB      peCur->ATRIB[23]
#define CurrDltVlAB        peCur->ATRIB[24]
#define aMinDelivery       peCur->ATRIB[25]
```

```
#define abMinDelivery     peCur->ATRIB[26]
#define bMinDelivery      peCur->ATRIB[27]
#define aLowVolPenalty    peCur->ATRIB[28]
#define bLowVolPenalty    peCur->ATRIB[29]
#define CashOnHand        peCur->ATRIB[1]
#define CurrSellVolA      peCur->ATRIB[2]
#define CurrSellVolAB     peCur->ATRIB[3]
#define CurrSellVolB      peCur->ATRIB[4]
#define InventoryA        peCur->ATRIB[5]
#define InventoryB        peCur->ATRIB[6]
#define MaxTransA         peCur->ATRIB[7]
#define MaxTransB         peCur->ATRIB[8]
#define MaxVolA           peCur->ATRIB[9]
#define MaxVolB           peCur->ATRIB[10]

#define CurrentAction     peCur->STRIB[1]
#define History           peCur->STRIB[2]

#define GENERATION        peCur->LTRIB[1]
#define rand02Action      peCur->LTRIB[5]



double USERF(int IFN, ENTITY *peCur)
{
double aVolSold = 0;
double abVolSold= 0;
double bVolSold = 0;
int c,i;
double volAToSell,volBToSell,volReq;
double costOfGasA = 0.0;
double volAAvailable  = 0.0;
double costOfGasB = 0.0;
double tgtSalesVol = 0.0;
double volBAvailable  = 0.0;
double volAInStorage = 0.0;
double volBInStorage = 0.0;
```

```
double CurrSellVolAinAB = 0.0;
double CurrSellVolBinAB = 0.0;
double aabMinReq= 0.0;
double babMinReq= 0.0;
double aMinReq= 0.0;
double bMinReq= 0.0;
double ARequiredVol = 0.0;
double BRequiredVol = 0.0;
double alreadyDeliveredPC= 0.0;
switch (IFN)
{
case WRITELN : SU_OUT(TRUE,TRUE,"Hello from procs.c USERF function. curr action=
%s\n",peCur->STRIB[1]);
runlogout =fopen("lp_run.txt","a");
runlogout2=fopen("lp_run2.txt","a");
break;



case DUMMY:
        fprintf(runlogout2, "DUMMY:  Percent: %8.2f InvA: %f  InvB: %8.2f tgtSalesVol
%8.2f   MaxTransA %8.2f  \n",
        changeAPerCent, InventoryA,  InventoryB ,tgtSalesVol,MaxTransA) ;


break;



case PRUNE_SELL :
if (((CurrentAction[0]=='*') || ( CurrentAction[0]=='S'))
&& (InventoryA > 0.0001 && InventoryB > 0.0001)
)
{
return(NO_PRUNE);
}
else
```

```
{
return(PRUNE);
}


break;


case TIMER_GET:
now = time (NULL);
ClockTime = now;
return 0.0;
break;



case PRUNE_HOLD :
if (((CurrentAction[0]=='*') || ( CurrentAction[0]=='H'))
 && (InventoryA < ((Horizon- GENERATION) * MaxTransA))
 && (InventoryB < ((Horizon- GENERATION) * MaxTransB)))
{
return(NO_PRUNE);
}
else
{
return(PRUNE);
}
break;



case PRUNE_BUY :
if (
   ((CurrentAction[0]=='*') || ( CurrentAction[0]=='B'))
&& ((InventoryA + MaxTransA) <= ((Horizon - GENERATION) * MaxTransA))
&& ((InventoryB + MaxTransB) <= ((Horizon - GENERATION) * MaxTransB))
&& ((InventoryA + MaxTransA) <= MaxVolA)
```

```
&& ((InventoryB + MaxTransB) <= MaxVolB)
)
{
return(NO_PRUNE); // 0=PRUNE   1=KEEP
}
else
{
return(PRUNE); // 0=PRUNE   1=KEEP
}
break;




case CALC_BUY:
// set total inventory

CurrBuyVolA=MaxTransA;
CurrBuyVolB=MaxTransB;


InventoryA = InventoryA + CurrBuyVolA;
InventoryB = InventoryB + CurrBuyVolB;


// set cash
CashOnHand =CashOnHand -
                              (CurrBuyVolA * peCur->ATRIB[CostAidxBase + GENERATION]
+
                               CurrBuyVolB * peCur->ATRIB[CostBidxBase + GENERATION] );

// set period inventory
peCur->ATRIB[InventoryABase + GENERATION] = CurrBuyVolA ;
peCur->ATRIB[InventoryBBase + GENERATION] = CurrBuyVolB ;

break;
```

```
case SIMPLEX:

        // first, calculate the profit of gas to be sold

        volAToSell= 0.0; // the amount KNOWN to be available for sale
        volReq = 0.0; // how much is needed to fill the 'order'
        costOfGasA = 0.0;

        // Set for gas A
        // loop through the inventory array from oldest to current (FIFO)
        //
        if (DEBUG1) fprintf(runlogout, "nSIMPLEX GENERATION %i  \n",
GENERATION);

        for(c=1;c<=GENERATION;c++)
                {
                if (volAToSell< MaxTransA)
                        // if we still need gas to fill the order
                        {
                        volReq=MaxTransA-volAToSell;    // how much do we still need for
this order?

                        // check the gas bought in this period
                        if ( peCur->ATRIB[InventoryABase+c]  >= volReq)
                                {
                                volAToSell=MaxTransA; // we have the max available

                                // increment the total cost by the cost of the gas bought in
this period
                                costOfGasA += volReq * peCur-
>ATRIB[CostAidxBase+c];

                                // add in the storage cost of the gas
                                costOfGasA += MaxTransA * CostOfStorageA *
((GENERATION + 1)- c);

                                // decrement the inventory
```

```
                                peCur->ATRIB[InventoryABase+c]  -= volReq;

                                // signal that the order is filled
                                volReq= 0.0;
                                }

                        else // we can fill part of the order from the gas purchased this period
                                {
                                volReq -= peCur->ATRIB[InventoryABase+c];
                                costOfGasA += peCur->ATRIB[InventoryABase+c] *
peCur->ATRIB[CostAidxBase+c];
                                // add in the storage cost of the gas
                                costOfGasA += peCur->ATRIB[InventoryABase+c] *
CostOfStorageA * ((GENERATION + 1)- c);

                                //
                                volAToSell+= peCur->ATRIB[InventoryABase+c];
                                peCur->ATRIB[InventoryABase+c] =0.0;
                                }
                        }
                }

        // at this point the amount of gasA available to sell, and it's total cost
        // is known



        volBToSell= 0.0; // the amount KNOWN to be available for sale
        volReq = 0.0; // how much is needed to fill the 'order'
        costOfGasB = 0.0;

        // Set for gas B
        // loop through the inventory array from oldest to current (FIFO)
        //
        for(c=1;c<=GENERATION;c++)
                {
                if (volBToSell< MaxTransB)
```

```
                        // if we still need gas to fill the order
                        {
                        volReq=MaxTransB-volBToSell;    // how much do we still need for
this order?

                        // check the gas bought in this period
                        if ( peCur->ATRIB[InventoryBBase+c]  >= volReq)
                                {
                                volBToSell=MaxTransB; // we have the max available

                                // increment the total cost by the cost of the gas bought in
this period
                                costOfGasB += volReq * peCur->ATRIB[CostBidxBase+c];

                                // add in the storage cost of the gas
                                costOfGasB += MaxTransB * CostOfStorageB *
((GENERATION + 1)- c);

                                // decrement the inventory
                                peCur->ATRIB[InventoryBBase+c]  -= volReq;

                                // signal that the order is filled
                                volReq= 0.0;
                                }

                        else // we can fill part of the order from the gas purchased this period
                                {
                                volReq -= peCur->ATRIB[InventoryBBase+c];
                                costOfGasB += peCur->ATRIB[InventoryBBase+c] *
peCur->ATRIB[CostBidxBase+c];
                                // add in the storage cost of the gas
                                costOfGasB += peCur->ATRIB[InventoryBBase+c] *
CostOfStorageB * ((GENERATION + 1)- c);
                                volBToSell+= peCur->ATRIB[InventoryBBase+c];
                                peCur->ATRIB[InventoryBBase+c] =0.0;
                                }
                        }
```

```
                }

   // set minimum delivery requirements
        aMinReq=aMinDelivery;
        bMinReq=bMinDelivery;
        aabMinReq = aInAB * abMinDelivery;
        babMinReq = bInAB * abMinDelivery;

        // if there is not enough in storage to meet requirements, buy some at SPOT price, with
penalty
        if ((aMinReq + aabMinReq) > volAToSell)
                {
                // how much is still needed?
                volReq = (aMinReq + aabMinReq)-volAToSell;
                volAToSell=volAToSell + volReq;
                // purchase gas on the market
                CashOnHand = CashOnHand - volReq * peCur-
>ATRIB[PriceAidxBase+GENERATION] * (1.0 + aLowVolPenalty);
                // add to inventory
                InventoryA = InventoryA + volReq;
                // increase cost of gas currently being solg
                costOfGasA = costOfGasA + volReq * peCur-
>ATRIB[PriceAidxBase+GENERATION] * (1.0 + aLowVolPenalty);
                }

        if ((bMinReq + babMinReq) > volBToSell)
                {
                volReq = (bMinReq + babMinReq)-volBToSell;
                volBToSell=volBToSell + volReq;
                // purchase gas on the market
                CashOnHand = CashOnHand - volReq * peCur-
>ATRIB[PriceBidxBase+GENERATION] * (1.0 + bLowVolPenalty);
                // add to inventory
                InventoryB = InventoryB + volReq;
                // increase cost of gas currently being solg
                costOfGasB = costOfGasB + volReq * peCur-
>ATRIB[PriceBidxBase+GENERATION] *(1.0 +  bLowVolPenalty);
```

```
            }
        // at this point the amount of gasB available to sell, and it's total cost
        // is known


        Profit_A  =  peCur->ATRIB[PriceAidxBase+GENERATION] -
costOfGasA/volAToSell;
        Profit_B  =  peCur->ATRIB[PriceBidxBase+GENERATION] -
costOfGasB/volBToSell;
        Profit_AB =  peCur->ATRIB[PriceABidxBase+GENERATION] -
                                        (aInAB*(costOfGasA/volAToSell) +
                                         bInAB*(costOfGasB/volBToSell));

        // then find the best mix to sell
        getMixedProduct(
                                volAToSell,
                                volBToSell,
                                Profit_A,
                                Profit_AB,
                                Profit_B,
                                aInAB,
                                bInAB,
                                aMinReq,
                                bMinReq,
                                aabMinReq,
                                babMinReq,
                                &aVolSold,
                                &abVolSold,
                                &bVolSold);


        CurrSellVolA  = aVolSold;
        CurrSellVolAB = abVolSold;
        CurrSellVolB  = bVolSold;
```

```
        CashOnHand = CashOnHand +       CurrSellVolA * peCur->ATRIB[PriceAidxBase +
GENERATION] +
                        CurrSellVolB * peCur->ATRIB[PriceBidxBase + GENERATION] +
                        CurrSellVolAB* peCur->ATRIB[PriceABidxBase +
GENERATION];

                        //* aInAB +
                        //CurrSellVolAB* peCur->ATRIB[PriceBidxBase + GENERATION]
* bInAB;

        InventoryA =InventoryA -( CurrSellVolA + CurrSellVolAB * aInAB);
        InventoryB =InventoryB -( CurrSellVolB + CurrSellVolAB * bInAB);

    break;

case ENDOFCYCLE:
// calculate the cost of the gas that is still in storage
//
//
        volAInStorage= 0.0; // the amount KNOWN to be available for sale
        costOfGasA = 0.0;
        volBInStorage= 0.0; // the amount KNOWN to be available for sale
        costOfGasB = 0.0;

        // Set for gas A
        // loop through the inventory array from oldest to current (FIFO)
        //

        for(c=1;c<=GENERATION;c++)
                {
                        //volReq=MaxTransA-volAToSell; // how much do we still need for
this order?

                        // check the gas bought in this period
                        costOfGasA += peCur->ATRIB[InventoryABase+c] * peCur-
>ATRIB[CostAidxBase+c];
                        volAInStorage+= peCur->ATRIB[InventoryABase+c];
```

```
                        costOfGasB += peCur->ATRIB[InventoryBBase+c] * peCur-
>ATRIB[CostBidxBase+c];
                        volBInStorage+= peCur->ATRIB[InventoryBBase+c];
                }

        CashOnHand = CashOnHand - costOfGasA;
        CashOnHand = CashOnHand - costOfGasB;

        break;


case RESETINVENTORY:
   for (c=0;c<=Horizon; c++)
        {
                peCur->ATRIB[InventoryABase+c] =0.0;
                peCur->ATRIB[InventoryBBase+c] =0.0;
                }
        InventoryA = 0.0;
        InventoryB = 0.0;


   break;



case RANDOM_A:
// set total inventory


if (changeAPerCent >=0)
{
        // indicate the purchase of Gas B
        rand02Action += 2;
        // set vol of Gas A to buy
        CurrDltVlA=MaxTransA * changeAPerCent;

        // confirm availability of Gas
```

```
        if ((InventoryA + CurrDltVlA) > MaxVolA)
                {
                CurrDltVlA=MaxVolA-InventoryA;
                }

        // execute buy
        InventoryA = InventoryA + CurrDltVlA;

        // set cash
        CashOnHand =CashOnHand -(CurrDltVlA * peCur->ATRIB[CostAidxBase +
GENERATION] );

        if (DEBUG11) fprintf(runlogout2, "RANDOM_A: peCur->ATRIB[InventoryABase +
GENERATION]=%8.2f  InventoryABase=%d   GEN%d  CurrDltVlA=%8.2f\n",
        peCur->ATRIB[InventoryABase +
GENERATION],InventoryABase,GENERATION,CurrDltVlA);

        // set period inventory
        peCur->ATRIB[InventoryABase + GENERATION] = CurrDltVlA ;

        if (DEBUG10) fprintf(runlogout2, "RANDOM_A:  BUY COMPLETE  CashOnHand
%8.2f  Percent: %8.2f InvA: %f  InvB: %8.2f MaxTransA %8.2f  \n",
        CashOnHand, changeAPerCent, InventoryA,  InventoryB ,MaxTransA );

        if (DEBUG11) fprintf(runlogout2, "RANDOM_Ab: CashOnHand %8.2f  peCur-
>ATRIB[InventoryABase + GENERATION]=%8.2f  InventoryABase=%d   GEN%d
CurrDltVlA=%8.2f\n",
        CashOnHand,peCur->ATRIB[InventoryABase +
GENERATION],InventoryABase,GENERATION,CurrDltVlA);

}
else

{

        // modify per cent change to exclude minimum already shipped
```

```
// changeAPerCent  MaxTransA   amindelivery
alreadyDeliveredPC = -1.0 * (aMinDelivery/MaxTransA); ///100.0;

// cannot ship more that 100% of capacity
changeAPerCent=minf((changeAPerCent-alreadyDeliveredPC),0.0);




// first, calculate the profit of gas to be sold
volAToSell = 0.0; // the amount KNOWN to be available for sale
volReq = 0.0;     // how much is needed to fill the 'order'
costOfGasA = 0.0;
tgtSalesVol = -1.0 * changeAPerCent * MaxTransA;
if (tgtSalesVol > InventoryA)
        {
        tgtSalesVol         = InventoryA;
        changeAPerCent = -1.0 * InventoryA/MaxTransA;
        }

// Set for gas A
// loop through the inventory array from oldest to current (FIFO)
//
for(c=1;c<=GENERATION;c++)
        {
        if (volAToSell< tgtSalesVol)
                // if we still need gas to fill the order
                {
                volReq=tgtSalesVol-volAToSell;    // how much do we still need for
this order?

                // check the gas bought in this period
                if ( peCur->ATRIB[InventoryABase+c]  >= volReq)
                        {
                        volAToSell=tgtSalesVol; // we have the max available
```

```
                                    // increment the total cost by the cost of the gas bought in
this period
                                    costOfGasA += volReq * peCur-
>ATRIB[CostAidxBase+c];

                                    // add in the storage cost of the gas
                                    costOfGasA += tgtSalesVol * CostOfStorageA *
((GENERATION + 1)- c);

                                    // decrement the inventory
                                    peCur->ATRIB[InventoryABase+c]  -= volReq;

                                    // signal that the order is filled
                                    volReq= 0.0;
                                    }

                            else // we can fill part of the order from the gas purchased this period
                                    {
                                    volReq -= peCur->ATRIB[InventoryABase+c];
                                    costOfGasA += peCur->ATRIB[InventoryABase+c] *
peCur->ATRIB[CostAidxBase+c];
                                    // add in the storage cost of the gas
                                    costOfGasA += peCur->ATRIB[InventoryABase+c] *
CostOfStorageA * ((GENERATION + 1)- c);

                                    //
                                    volAToSell+= peCur->ATRIB[InventoryABase+c];
                                    peCur->ATRIB[InventoryABase+c] =0.0;
                                    }
                            }
                    }

            // at this point the amount of gasA available to sell, and it's total cost
            // is known

            Profit_A  = peCur->ATRIB[PriceAidxBase+GENERATION] -
costOfGasA/volAToSell;
```

```
        CashOnHand = CashOnHand + volAToSell * peCur->ATRIB[PriceAidxBase +
GENERATION];

        InventoryA =InventoryA - volAToSell ;

        // set the sell volume
        CurrDltVlA  = -1.0 * volAToSell ;


// ================================================

}

break;
case RANDOM_B:


// set total inventory

// modify per cent change to exclude minimum already shipped


if (changeBPerCent >=0)
{
        // indicate the purchase of Gas B
        rand02Action += 1;
        // buy Gas B
        CurrDeltaVolB=MaxTransB * changeBPerCent;

        // confirm availability of storage for new Gas B
        if ((InventoryB + CurrDeltaVolB) > MaxVolB)
                {
                CurrDeltaVolB=MaxVolB-InventoryB;
                changeBPerCent=(MaxVolB-InventoryB)/MaxVolB;
                }
```

```
        // execute buy
        InventoryB = InventoryB + CurrDeltaVolB;

        // set cash
        CashOnHand =CashOnHand -(CurrDeltaVolB * peCur->ATRIB[CostBidxBase +
GENERATION] );

        // set period inventory
        peCur->ATRIB[InventoryBBase + GENERATION] = CurrDeltaVolB ;


}
else
{


        // changeBPerCent  MaxTransB   bmindelivery
        alreadyDeliveredPC = -1.0 * (bMinDelivery/MaxTransB) ; //* 100.0;

        // cannot ship more that 100% of capacity
        changeBPerCent=minf((changeBPerCent-alreadyDeliveredPC),0.0);


        // first, calculate the profit of gas to be sold
        volBToSell = 0.0; // the amount KNOWN to be available for sale
        volReq = 0.0;     // how much is needed to fill the 'order'
        costOfGasB = 0.0;
        tgtSalesVol = -1.0 * changeBPerCent * MaxTransB;



        if (tgtSalesVol > InventoryB)
                {
                tgtSalesVol         = InventoryB;
                changeBPerCent = -1.0 * InventoryB/MaxTransB;
                }
```

```
        // Set for gas B
        // loop through the inventory array from oldest to current (FIFO)
        //
        for(c=1;c<=GENERATION;c++)
                {
                if (volBToSell< tgtSalesVol)
                        // if we still need gas to fill the order
                        {
                        volReq=tgtSalesVol-volBToSell;    // how much do we still need for
this order?

                        // check the gas bought in this period
                        if ( peCur->ATRIB[InventoryBBase+c]  >= volReq)
                                {
                                volBToSell=tgtSalesVol; // we have the max available

                                // increment the total cost by the cost of the gas bought in
this period
                                costOfGasB += volReq * peCur->ATRIB[CostBidxBase+c];

                                // add in the storage cost of the gas
                                costOfGasB += tgtSalesVol * CostOfStorageB *
((GENERATION + 1)- c);

                                // decrement the inventory
                                peCur->ATRIB[InventoryBBase+c]  -= volReq;

                                // signal that the order is filled
                                volReq= 0.0;
                                }

                        else // we can fill part of the order from the gas purchased this period
                                {
                                volReq -= peCur->ATRIB[InventoryBBase+c];
                                costOfGasB += peCur->ATRIB[InventoryBBase+c] *
peCur->ATRIB[CostBidxBase+c];
```

```
                                    // add in the storage cost of the gas
                                    costOfGasB += peCur->ATRIB[InventoryBBase+c] *
CostOfStorageB * ((GENERATION + 1)- c);

                                    //
                                    volBToSell+= peCur->ATRIB[InventoryBBase+c];
                                    peCur->ATRIB[InventoryBBase+c] =0.0;
                                    }
                            }
                    }

            // at this point the amount of gasB available to sell, and it's total cost
            // is known

            Profit_B  = peCur->ATRIB[PriceBidxBase+GENERATION] -
costOfGasB/volBToSell;

            CashOnHand = CashOnHand + volBToSell * peCur->ATRIB[PriceBidxBase +
GENERATION];



            CurrDeltaVolB  = -1.0*volBToSell ;

            InventoryB =InventoryB + CurrDeltaVolB ;
            // set the sell volume


// ================================================
}


break;
case RANDOM_AB:


// modify per cent change to exclude minimum already shipped
```

```
// changeBPerCent  MaxTransB   bmindelivery
alreadyDeliveredPC = -1.0 * maxf((aInAB*abMinDelivery/MaxTransA) , //* 100.0,
                                                    (bInAB*abMinDelivery/MaxTransB) );
//* 100.0);

// cannot ship more that 100% of capacity
 changeABPerCent=minf((changeABPerCent-alreadyDeliveredPC),0.0);


 changeABPerCent*=-1.0;

// lower the percentage of AB to sell until
// it is at or below the amount amount possible to
// sell based on stock levels of A and B
// and the remaining shipping capacity of A and B

if (InventoryA < aInAB * changeABPerCent * MaxTransA)
        changeABPerCent = minf(changeABPerCent,(InventoryA/(aInAB* MaxTransA)));



if (InventoryB < bInAB * changeABPerCent * MaxTransB)
        changeABPerCent = minf(changeABPerCent,(InventoryB/(bInAB * MaxTransB)));


// check for max trans limit but allow passthrough
// i.e 100% gas in and 100% out is allowable
// if a sell occured CurrDltVlA is negative so MaxTransA + CurrDltVlA < MaxTransA
if (
        (CurrDltVlA<0.0)&& ((changeABPerCent * aInAB * MaxTransA)>(MaxTransA +
CurrDltVlA))
        )// this should only apply for gas A SELLS
        changeABPerCent=minf(changeABPerCent,((MaxTransA+CurrDltVlA)/aInAB));
```

```
// check for max trans limit but allow passthrough
// i.e 100% gas in and 100% out is allowable
// if a sell occured CurrDeltaVolB is negative so MaxTransA + CurrDeltaVolB < MaxTransB
if (
        (CurrDeltaVolB<0.0)&& ((changeABPerCent * bInAB *
MaxTransB)>(MaxTransB+CurrDeltaVolB))
        )
        changeABPerCent=minf(changeABPerCent,((MaxTransB+CurrDeltaVolB)/bInAB));

        // now execute the sales of A and B


        // first, calculate the profit of gas to be sold
        volAToSell = 0.0; // the amount KNOWN to be available for sale
        volReq = 0.0;     // how much is needed to fill the 'order'
        costOfGasA = 0.0;
        tgtSalesVol = changeABPerCent * aInAB * MaxTransA;

        // Set for gas A
        // loop through the inventory array from oldest to current (FIFO)
        //
        for(c=1;c<=GENERATION;c++)
                {
                if (volAToSell< tgtSalesVol)
                        // if we still need gas to fill the order
                        {
                        volReq=tgtSalesVol-volAToSell;    // how much do we still need for
this order?

                        // check the gas bought in this period
                        if ( peCur->ATRIB[InventoryABase+c]  >= volReq)
                                {
                                volAToSell=tgtSalesVol; // we have the max available

                                // increment the total cost by the cost of the gas bought in
this period
```

```
                                costOfGasA += volReq * peCur-
>ATRIB[CostAidxBase+c];

                                // add in the storage cost of the gas
                                costOfGasA += tgtSalesVol * CostOfStorageA *
((GENERATION + 1)- c);

                                // decrement the inventory
                                peCur->ATRIB[InventoryABase+c]  -= volReq;

                                // signal that the order is filled
                                volReq= 0.0;
                                }

                        else // we can fill part of the order from the gas purchased this period
                                {
                                volReq -= peCur->ATRIB[InventoryABase+c];
                                costOfGasA += peCur->ATRIB[InventoryABase+c] *
peCur->ATRIB[CostAidxBase+c];
                                // add in the storage cost of the gas
                                costOfGasA += peCur->ATRIB[InventoryABase+c] *
CostOfStorageA * ((GENERATION + 1)- c);

                                //
                                volAToSell+= peCur->ATRIB[InventoryABase+c];
                                peCur->ATRIB[InventoryABase+c] =0.0;
                                }
                        }
                }

        // at this point the amount of gasA available to sell, and it's total cost
        // is known

        Profit_A  = peCur->ATRIB[PriceAidxBase+GENERATION] -
costOfGasA/volAToSell;

        CurrSellVolAinAB  = -1.0 * volAToSell ;
```

```
        CashOnHand = CashOnHand +       volAToSell * peCur->ATRIB[PriceAidxBase +
GENERATION];

        InventoryA =InventoryA - volAToSell ;




        // process the sell of Gas B

        // first, calculate the profit of gas to be sold
        volBToSell = 0.0; // the amount KNOWN to be available for sale
        volReq = 0.0;     // how much is needed to fill the 'order'
        costOfGasB = 0.0;
        tgtSalesVol = changeABPerCent * bInAB * MaxTransB;


/////////

        // Set for gas B
        // loop through the inventory array from oldest to current (FIFO)
        //
        for(c=1;c<=GENERATION;c++)
                {
                if (volBToSell< tgtSalesVol)
                        // if we still need gas to fill the order
                        {
                        volReq=tgtSalesVol-volBToSell;    // how much do we still need for
this order?

                        // check the gas bought in this period
                        if ( peCur->ATRIB[InventoryBBase+c]  >= volReq)
                                {
                                volBToSell=tgtSalesVol; // we have the max available
```

```
                                        // increment the total cost by the cost of the gas bought in
this period
                                        costOfGasB += volReq * peCur->ATRIB[CostBidxBase+c];

                                        // add in the storage cost of the gas
                                        costOfGasB += tgtSalesVol * CostOfStorageB *
((GENERATION + 1)- c);

                                        // decrement the inventory
                                        peCur->ATRIB[InventoryBBase+c]  -= volReq;

                                        // signal that the order is filled
                                        volReq= 0.0;
                                        }

                                else // we can fill part of the order from the gas purchased this period
                                        {
                                        volReq -= peCur->ATRIB[InventoryBBase+c];
                                        costOfGasB += peCur->ATRIB[InventoryBBase+c] *
peCur->ATRIB[CostBidxBase+c];
                                        // add in the storage cost of the gas
                                        costOfGasB += peCur->ATRIB[InventoryBBase+c] *
CostOfStorageB * ((GENERATION + 1)- c);

                                        //
                                        volBToSell+= peCur->ATRIB[InventoryBBase+c];
                                        peCur->ATRIB[InventoryBBase+c] =0.0;
                                        }
                                }
                        }

        // at this point the amount of gasB available to sell, and it's total cost
        // is known

        Profit_B  = peCur->ATRIB[PriceBidxBase+GENERATION] -
costOfGasB/volBToSell;
```

```
        CurrSellVolBinAB  = -1.0 * volBToSell ;

        CashOnHand = CashOnHand + volBToSell * peCur->ATRIB[PriceBidxBase +
GENERATION];

        InventoryB =InventoryB - volBToSell ;


         CurrDltVlAB = -1.0 * (volAToSell + volBToSell);//
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

        changeABPerCent *=-1.0;


break;

case MANDATORY_SELL:

          // set minimum delivery requirements
                aMinReq=aMinDelivery;
                bMinReq=bMinDelivery;
                aabMinReq = aInAB * abMinDelivery;
                babMinReq = bInAB * abMinDelivery;

                ARequiredVol = aMinDelivery + aInAB * abMinDelivery;
                BRequiredVol = bMinDelivery + bInAB * abMinDelivery;


                // first, calculate the profit of gas to be sold

                volAToSell= 0.0; // the amount KNOWN to be available for sale
                volReq = 0.0; // how much is needed to fill the 'order'
                costOfGasA = 0.0;

                // Set for gas A
                // loop through the inventory array from oldest to current (FIFO)
                //
```

```
for(c=1;c<=GENERATION;c++)
        {
        if (volAToSell< ARequiredVol)
                // if we still need gas to fill the order
                {
                volReq=ARequiredVol-volAToSell;          // how much do
we still need for this order?

                // check the gas bought in this period
                if ( peCur->ATRIB[InventoryABase+c]  >= volReq)
                        {
                        volAToSell=ARequiredVol; // we have the max
available

                        // increment the total cost by the cost of the gas
bought in this period
                        costOfGasA += volReq * peCur-
>ATRIB[CostAidxBase+c];

                        // add in the storage cost of the gas
                        costOfGasA += ARequiredVol * CostOfStorageA *
((GENERATION + 1)- c);

                        // decrement the inventory
                        peCur->ATRIB[InventoryABase+c]  -= volReq;

                        // signal that the order is filled
                        volReq= 0.0;
                        }

                else // we can fill part of the order from the gas purchased
this period
                        {
                        volReq -= peCur->ATRIB[InventoryABase+c];
                        costOfGasA += peCur-
>ATRIB[InventoryABase+c] * peCur->ATRIB[CostAidxBase+c];
```

```
                                        // add in the storage cost of the gas
                                        costOfGasA += peCur-
>ATRIB[InventoryABase+c] * CostOfStorageA * ((GENERATION + 1)- c);

                                        //
                                        volAToSell+= peCur-
>ATRIB[InventoryABase+c];
                                        peCur->ATRIB[InventoryABase+c] =0.0;
                                        }
                                }
                        }

                // at this point the amount of gasA available to sell, and it's total cost
                // is known



                volBToSell= 0.0; // the amount KNOWN to be available for sale
                volReq = 0.0; // how much is needed to fill the 'order'
                costOfGasB = 0.0;

                // Set for gas B
                // loop through the inventory array from oldest to current (FIFO)
                //
                for(c=1;c<=GENERATION;c++)
                        {
                        if (volBToSell< BRequiredVol)
                                // if we still need gas to fill the order
                                {
                                volReq=BRequiredVol-volBToSell;         // how much do
we still need for this order?

                                // check the gas bought in this period
                                if ( peCur->ATRIB[InventoryBBase+c]  >= volReq)
                                        {
                                        volBToSell=BRequiredVol; // we have the max
available
```

```
                                        // increment the total cost by the cost of the gas
bought in this period
                                        costOfGasB += volReq * peCur-
>ATRIB[CostBidxBase+c];

                                        // add in the storage cost of the gas
                                        costOfGasB += BRequiredVol * CostOfStorageB *
((GENERATION + 1)- c);

                                        // decrement the inventory
                                        peCur->ATRIB[InventoryBBase+c]  -= volReq;

                                        // signal that the order is filled
                                        volReq= 0.0;
                                        }

                                else // we can fill part of the order from the gas purchased
this period
                                        {
                                        volReq -= peCur->ATRIB[InventoryBBase+c];
                                        costOfGasB += peCur-
>ATRIB[InventoryBBase+c] * peCur->ATRIB[CostBidxBase+c];
                                        // add in the storage cost of the gas
                                        costOfGasB += peCur-
>ATRIB[InventoryBBase+c] * CostOfStorageB * ((GENERATION + 1)- c);
                                        volBToSell+= peCur->ATRIB[InventoryBBase+c];
                                        peCur->ATRIB[InventoryBBase+c] =0.0;
                                        }
                                }
                        }


                // if there is not enough in storage to meet requirements, buy some at SPOT
price, with penalty
                if ((aMinReq + aabMinReq) > volAToSell)
                        {
```

```
                    // how much is still needed?
                    volReq = (aMinReq + aabMinReq)-volAToSell;
                    volAToSell=volAToSell + volReq;
                    // purchase gas on the market
                    CashOnHand = CashOnHand - volReq * peCur-
>ATRIB[PriceAidxBase+GENERATION] * (1.0 + aLowVolPenalty);
                    // add to inventory
                    InventoryA = InventoryA + volReq;
                    // increase cost of gas currently being solg
                    costOfGasA = costOfGasA + volReq * peCur-
>ATRIB[PriceAidxBase+GENERATION] * (1.0 + aLowVolPenalty);
                    }

          if ((bMinReq + babMinReq) > volBToSell)
                    {
                    volReq = (bMinReq + babMinReq)-volBToSell;
                    volBToSell=volBToSell + volReq;
                    // purchase gas on the market
                    CashOnHand = CashOnHand - volReq * peCur-
>ATRIB[PriceBidxBase+GENERATION] * (1.0 + bLowVolPenalty);
                    // add to inventory
                    InventoryB = InventoryB + volReq;
                    // increase cost of gas currently being solg
                    costOfGasB = costOfGasB + volReq * peCur-
>ATRIB[PriceBidxBase+GENERATION] *(1.0 +  bLowVolPenalty);

                    }
          // at this point the amount of gasB available to sell, and it's total cost
          // is known


          Profit_A  =  peCur->ATRIB[PriceAidxBase+GENERATION] -
costOfGasA/volAToSell;
          Profit_B  =  peCur->ATRIB[PriceBidxBase+GENERATION] -
costOfGasB/volBToSell;
          Profit_AB =  peCur->ATRIB[PriceABidxBase+GENERATION] -
                                        (aInAB*(costOfGasA/volAToSell) +
```

```
                                                    bInAB*(costOfGasB/volBToSell));


                    CurrSellVolA  = aMinDelivery;
                    CurrSellVolAB = abMinDelivery;
                    CurrSellVolB  = bMinDelivery;


                    CashOnHand = CashOnHand +       CurrSellVolA * peCur-
>ATRIB[PriceAidxBase + GENERATION] +
                    CurrSellVolB * peCur->ATRIB[PriceBidxBase + GENERATION] +
                                                            CurrSellVolAB* peCur-
>ATRIB[PriceABidxBase + GENERATION];


                    InventoryA =InventoryA -( CurrSellVolA + CurrSellVolAB * aInAB);
                    InventoryB =InventoryB -( CurrSellVolB + CurrSellVolAB * bInAB);


break;
} // end switch


// moved from here
} // end function


double minf(double vala, double valb)
{
if (vala>valb) return valb;
return vala;
}
double maxf(double vala, double valb)
{
if (vala>valb) return vala;
return valb;
```

```
}

//-------------------------------------------------------------

void getMixedProduct(
                                        double aAvail,
                                        double bAvail,
                                        double net_prof_a,
                                        double net_prof_ab,
                                        double net_prof_b,
                                        double aInABmix,
                                        double bInABmix,
                                        double aMinReq,
                                        double bMinReq,
                                        double aabMinReq,
                                        double babMinReq,
                                double *aVolSold,
                                double *abVolSold,
                                        double *bVolSold)

{
int *colno = NULL, Ncol,  ret = 0;
double row[99];
double var[99];


lprec *lp2;


HINSTANCE lpsolve;
lpsolve = LoadLibrary("lpsolve55.dll");
if (lpsolve == NULL)
            fprintf(runlogout,"Unable to load lpsolve shared library \n\n");
else
           {
fprintf(runlogout, "begin getMixedProd\n");
           }
```

```
//srand(time(NULL));

_make_lp    = (make_lp_func   *) GetProcAddress(lpsolve, "make_lp");
_add_constraint = (add_constraint_func *) GetProcAddress(lpsolve, "add_constraint");
_add_constraintex = (add_constraintex_func *) GetProcAddress(lpsolve, "add_constraintex");
_delete_lp  = (delete_lp_func *) GetProcAddress(lpsolve, "delete_lp");
_get_col_name = (get_col_name_func *) GetProcAddress(lpsolve, "get_col_name");
_get_objective = (get_objective_func *) GetProcAddress(lpsolve, "get_objective");
_get_variables = (get_variables_func *) GetProcAddress(lpsolve, "get_variables");
_print_lp = (print_lp_func *) GetProcAddress(lpsolve, "print_lp");
_print_solution = (print_solution_func *) GetProcAddress(lpsolve, "print_solution");
_read_LP = (read_LP_func *) GetProcAddress(lpsolve, "read_LP");
_set_add_rowmode = (set_add_rowmode_func *) GetProcAddress(lpsolve,
"set_add_rowmode");
_set_col_name = (set_col_name_func *) GetProcAddress(lpsolve, "set_col_name");
_set_maxim = (set_maxim_func *) GetProcAddress(lpsolve, "set_maxim");
_set_obj_fn = (set_obj_fn_func *) GetProcAddress(lpsolve, "set_obj_fn");
_set_obj_fnex = (set_obj_fnex_func *) GetProcAddress(lpsolve, "set_obj_fnex");
_set_verbose = (set_verbose_func *) GetProcAddress(lpsolve, "set_verbose");
_solve = (solve_func *) GetProcAddress(lpsolve, "solve");
_write_LP = (write_LP_func *) GetProcAddress(lpsolve, "write_LP");



/* Create LP  */
lp2 = _make_lp(0, 3);



row[1] = -1.0 * net_prof_a;
row[2] = -1.0 * net_prof_ab;
row[3] = -1.0 * net_prof_b;

_set_obj_fn(lp2, row);
```

```
// set the constraints
_set_add_rowmode(lp2, TRUE);
row[1] = 1.0;
row[2] = 0.0; /* also zero elements must be provided */
row[3] = 0.0;
_add_constraint(lp2, row, LE, aAvail); /* constructs the row: 1.0 * x1 <= available A */

row[1] = 1.0;
row[2] = 0.0; /* also zero elements must be provided */
row[3] = 0.0;
_add_constraint(lp2, row, GE, aMinReq); /* constructs the row: 1.0 * x1 >= minrequired A*/

row[1] = 0.0;
row[2] = 0.0; /* also zero elements must be provided */
row[3] = 1.0;
_add_constraint(lp2, row, LE, bAvail); /* constructs the row: 1.0 * x3 <= available B*/

row[1] = 0.0;
row[2] = 0.0; /* also zero elements must be provided */
row[3] = 1.0;
_add_constraint(lp2, row, GE, bMinReq); /* constructs the row: 1.0 * x3 >= minrequired B*/

row[1] = 1.0;
row[2] = aInABmix;
row[3] = 0.0;
_add_constraint(lp2, row, LE, aAvail); /*  1*x1 + aInABMix*x2 cannot contain more A than is
aAvail*/

row[1] = 0.0;
row[2] = aInABmix;
row[3] = 0.0;
_add_constraint(lp2, row, GE, aabMinReq); /* x2 aInABMix * x2  cannot be less the Avol in
mandatory minimum AB*/
```

```
row[1] = 0.0;
row[2] = bInABmix;
row[3] = 1.0;
_add_constraint(lp2, row, LE, bAvail);   /*x2 (1+bInABMix) * X2 cannot contain more B than is bAvail*/

row[1] = 0.0;
row[2] = bInABmix;
row[3] = 0.0;
_add_constraint(lp2, row, GE, babMinReq); /*x2  bInABMix * x2  cannot be less the Bvol in mandatory minimum AB*/


_set_add_rowmode(lp2, FALSE);


// ---------------------------------------------
Ncol=3;
/* I only want to see important messages on screen while solving */
//    _set_verbose(lp2, IMPORTANT);

/* Now let lpsolve calculate a solution */
ret = _solve(lp2);
if(ret == OPTIMAL)
     ret = 0;
else
    ret = 5;

if(ret == 0)
            {
            /* a solution is calculated, now lets get some results */

            /* objective value */
            //printf("Objective value: %f\n", _get_objective(lp2));

            /* variable values */
            //_get_variables(lp2, row);
```

```
            //for(j = 0; j < Ncol; j++)
      //      printf("%s: %f\n", _get_col_name(lp2, j + 1), row[j]);



            _get_variables(lp2, var);

            //for (j=0; j<3;j++)
            //     printf("var %i  = %f\n",j,var[j]);
            *aVolSold  =var[0];
            *abVolSold =var[1];
            *bVolSold  =var[2];
}

  if(colno != NULL)
    free(colno);

  if(lp2 != NULL) {
    /* clean up such that all used memory by lpsolve is freed */
    _delete_lp(lp2);
  }
return;
}
```

Appendix 9: AWESIM Model Definition Files

```
[CONTROL FILE]
GEN,"SNTEST2","SNTEST2",,1,YES,YES,5;
LIMITS,800,60,5,800,10,10,999;
INITIALIZE,0.0,999999,YES,,YES;
EQUIVALENCE,{{CashOnHand,ATRIB[1]},
{CurrSellVolA,ATRIB[2]},
{CurrSellVolAB,ATRIB[3]},
{CurrSellVolB,ATRIB[4]},
{InventoryA,ATRIB[5]},
{InventoryB,ATRIB[6]},
{MaxTransA,ATRIB[7]},
{MaxTransB,ATRIB[8]},
{MaxVolA,ATRIB[9]},
{MaxVolB,ATRIB[10]},
{Profit_A,ATRIB[11]},
{Profit_AB,ATRIB[12]},
{Profit_B,ATRIB[13]},
{aInAB,ATRIB[14]},
{bInAB,ATRIB[15]},
{RVAL_PROFIT,ATRIB[16]},
{CurrBuyVolA,ATRIB[17]},
{CurrBuyVolB,ATRIB[18]},
{changeAPerCent,ATRIB[19]},
{changeABPerCent,ATRIB[20]},
{changeBPerCent,ATRIB[21]},
{CurrADeltaVol,ATRIB[22]},
{CurrBDeltaVol,ATRIB[23]},
{CurrABDeltaVol,ATRIB[24]},
{Generation,LTRIB[1]},
{CurrentAction,STRIB[1]},
{History,STRIB[2]},
{HistoryCh,STRIB[5]},
{HistoryNu,STRIB[6]},
{BestHistory,SZ[1]},
{BestHistoryCh,SZ[2]},
{BestHistoryNu,SZ[3]},
{BestProfit,XX[1]},
{CostAidx,LL[01]},
{CostAidxBase,LL[02]},
{CostAidxMax,LL[03]},
{CostAidxMaxG,LL[04]},
{CostBidx,LL[05]},
{CostBidxBase,LL[06]},
{CostBidxMax,LL[07]},
```

{CostBidxMaxG,LL[08]},
{Horizon,LL[09]},
{PriceABidx,LL[11]},
{PriceABidxBase,LL[12]},
{PriceABidxMax,
LL[13]},
{PriceABidxMaxG,LL[14]},
{PriceAidx,LL[15]},
{PriceAidxBase,LL[16]},
{PriceAidxMax,LL[17]},
{PriceAidxMaxG,LL[18]},
{PriceBidx,LL[19]},
{PriceBidxBase,LL[20]},
{PriceBidxMax,LL[21]},
{PriceBidxMaxG,LL[22]},
{ReadResult,LL[23]},
{InventoryABase,LL[24]},
{InventoryBBase,LL[25]},
{CurrentScenerio,LL[26]},
{ScenerioCtr,LL[27]},
{AvgCostOfInvA,LL[30]},
{AvgCostOfInvB,LL[31]},
{variationMax,LL[32]},
{processType,LL[33]},
{clockTime,LL[34]},
{clockTime1,LL[35]},
{clockTime2,LL[36]},
{clockTime3,LL[37]},
{fReadResult,XX[2]},
{CostOfStorageA,XX[3]},
{CostOfStorageB,XX[4]},
{PRUNE,0},
{NO_PRUNE,1},
{SCENERIO,LTRIB[2]},
{nextAction,LTRIB[3]},
{variationNumber,LTRIB[4]},
{rand02Action,LTRIB[5]},
{RESULTFILE,STRIB[4]},
{STACK,LL[10]},
{RVAL_ACTION,STRIB[3]},
{cVaryPriceCostTest,-1},
{cVaryPriceCostNone,0},
{cVaryPriceCostNorm,1},
{cFileLoadTest,0},
{cFileLoadLocal,1},
{cFileLoadGlobal,2},
{cFileLoadG2L,3},
{cFileLoadL2G,4},

```
{cTrue,1},
{cFalse,0},
{cTypeRecursive,1},
{cTypeRandom,2},
{cTypeRand02,3},
{Prices_A_File,STRIB[10]},
{cTimerGet,
10},
{cTimerSet,11},
{cRand_A,13},
{cRand_AB,14},
{cRand_B,15}};
INTLC,{{ScenerioCtr,0},
{CurrentScenerio,1}};
NETWORK,READ;
FIN;



[NETWORK FILE]



[RECUR SUBNETWORK]
;DBF file created with Version 4
      VSN,SNRECUR1,,,,,,,,,{30,150};
NodeArrives: ENTERVSN,ProcessSVN,1,,,,,,,,{70,300};
      ACTIVITY,,,,,,,,,{2,4,,,};
Start: GOON,1,,,,,,,,,{140,300};
      ACTIVITY,,,,,,,,,{4,6,,,170,300};
SNRECUR1_WRITE_7: WRITE,"history_stack_init.txt",YES,"Stack = %d  \tenter:  gen:  %d
%s InvA: %f    InvB:
%f\n",{STACK,GENERATION,CURRENTACTION,INVENTORYA,INVENTORYB},1,,,,,,{
240,300};
      ACTIVITY,1,,,,,,,,{6,8,,,};
DirectAction: GOON,3,,,,,,,,,{430,300};
      ACTIVITY,,,((CurrentAction=="B")||(CurrentAction=="*")),,,,,,{8,12,,,};
      ACTIVITY,12,0,((CurrentAction=="H") ||
(CurrentAction=="*")),"SNRECUR1_ASSIGN_3",,,,,,{8,63,,,};
      ACTIVITY,13,0,((CurrentAction=="S") ||
(CurrentAction=="*")),"SNRECUR1_GOON_3",,,,,,{8,79,,,470,420,570,420};
SNRECUR1_WRITE_6: WRITE,"history_stack_init.txt",YES,"Stack = %d  \tpre-buy:  gen:
%d\n",{STACK,GENERATION},1,,,,,,{540,230};
      ACTIVITY,,,,,,,,,,{12,14,,,};
      GOON,2,,,,,,,,,,{740,230};
      ACTIVITY,,0,((CurrentAction=="B") || (CurrentAction=="*")) && ((InventoryA +
MaxTransA) <= ((Horizon - GENERATION) * MaxTransA)) && ((InventoryB + MaxTransB)
<= ((Horizon - GENERATION) * MaxTransB))&& ((InventoryA + MaxTransA) <= MaxVolA)
&& ((InventoryB + MaxTransB) <= MaxVolB),,,,,,,{14,17,,,810,170,790,130,850,130};
```

```
      ACTIVITY,,0,((InventoryA + MaxTransA) > ((Horizon - GENERATION) *
MaxTransA))||((InventoryB + MaxTransB) > ((Horizon - GENERATION) *
MaxTransB))||((InventoryA + MaxTransA) > MaxVolA) || ((InventoryB + MaxTransB) >
MaxVolB),"SNRECUR1_ASSIGN_2",,,,,,{14,57,,,1000,230};
calcBuyVols: ASSIGN,{{fReadResult,USERF(7)}},1,,,,,,,,{880,130};
      ACTIVITY,,,,,,,,,{17,19,,,};
SNRECUR1_ASSIGN_6: ASSIGN,{{STACK,STACK-1}},1,,,,,,,,{980,130};
      ACTIVITY,,,,,,,,,{19,21,,,};
SNRECUR1_WRITE_10: WRITE,"history_stack_init.txt",YES,"Stack = %d  \tpost-buyB:  gen:
%d\n",{STACK,GENERATION},1,,,,,,{1090,130};
      ACTIVITY,,,,,,,,,{21,23,,,};
PostBuy2:
ASSIGN,{{Generation,Generation+1},{History,strcat(History,"B")},{History,strcat(History,"[a]
")},{History,strcat(History,itoa(nint(CurrBuyVolA)))},{History,strcat(History,"[b]")},{History,s
trcat(History,itoa(nint(CurrBuyVolB)))},{HistoryCh,strcat(HistoryCh,"B")},{HistoryNu,strcat(
HistoryNu,itoa(nint(CurrBuyVolA)))},{HistoryNu,strcat(HistoryNu,",0,")},{HistoryNu,strcat(H
istoryNu,itoa(nint(CurrBuyVolB)))},{HistoryNu,strcat(HistoryNu,",")}},4,,,,,,,,{1250,130};
      ACTIVITY,,0,Generation <=Horizon,"BuyBuy",,,,,,{23,28,,,1670,190};
      ACTIVITY,,,Generation <= Horizon,"BuyHold",,,,,,{23,30,,,1690,250};
      ACTIVITY,,,Generation <= Horizon,"BuySell",,,,,,{23,32,,,1660,350};
      ACTIVITY,,,Generation > Horizon,"SNRECUR1_GOON_2",,,,,,{23,34,,,1430,350};
BuyBuy: ASSIGN,{{CURRENTACTION,"B"},{STACK,STACK+1}},1,,,,,,,,{1790,190};
      ACTIVITY,101,,,"Start",,,,,,{28,4,,,1870,70,140,70,110,70,120,230};
BuyHold: ASSIGN,{{CURRENTACTION,"H"},{STACK,STACK+1}},1,,,,,,,,{1810,250};
      ACTIVITY,102,,,"Start",,,,,,{30,4,,,1890,60,100,60};
BuySell: ASSIGN,{{CURRENTACTION,"S"},{STACK,STACK+1}},1,,,,,,,,{1820,350};
      ACTIVITY,103,,,"Start",,,,,,{32,4,,,1900,50,90,50};
SNRECUR1_GOON_2: GOON,2,,,,,,,,,{1450,490};
      ACTIVITY,,,,,,,,,{34,36,,,};
End_of_Cycle: ASSIGN,{{fReadResult,USERF(8)}},1,,,,,,,,{1460,460};
      ACTIVITY,,,,,,,,,{36,38,,,};
      GOON,2,,,,,,,,,{1540,520};
      ACTIVITY,,,CashOnHand>=BestProfit,,,,,,{38,41,,,1580,490};
      ACTIVITY,,,,"writehistory",,,,,,{38,54,,,1620,580};
PREPAREEXIT:
ASSIGN,{{BESTPROFIT,MAX(BESTPROFIT,CASHONHAND)},{RVAL_ACTION,HISTO
RY},{RVAL_PROFIT,MAX(BESTPROFIT,CASHONHAND)},{BestHistory,History},{BestH
istoryCh,HistoryCh},{BestHistoryNu,HistoryNu}},1,,,,,,,,{1590,490};
      ACTIVITY,1999,,,,,,,,,{41,43,,,};
BP: COLCT,1003,BESTPROFIT,"BestProfit_SVN",20,0,1000,1,,,,{1810,490};
      ACTIVITY,,,,,,,,,{43,45,,,};
SNRECUR1_WRITE_9: WRITE,"history_stack_init.txt",YES,"Stack = %d  \tenter:  gen:  %d
%s\n",{STACK,GENERATION,CURRENTACTION},1,,,,,{1940,490};
      ACTIVITY,,,stack<=-2,"SNRECUR1_ASSIGN_1",,,,,,{45,48,,,2100,490};
      ACTIVITY,,,stack != -2,"SNRECUR1_TERMINATE_1",,,,,,{45,53,,,};
SNRECUR1_ASSIGN_1:
ASSIGN,{{BESTPROFIT,MAX(BESTPROFIT,CASHONHAND)},{RVAL_ACTION,HISTO
```

```
RY},{RVAL_PROFIT,MAX(BESTPROFIT,CASHONHAND)},{readResult,USERF(cTimerG
et)},{ClockTime2,ClockTime-ClockTime1}},1,,,,,,,,{2270,490};
      ACTIVITY,,,,,,,,,{48,50,,,};
SNRECUR1_WRITE_8: WRITE,"results.txt",YES,"scen:,
%d,%d,%f,%s,%s%f\n",{CURRENTSCENERIO,ClockTime2,rval_profit,BestHistoryCh,BestH
istoryNu,AinAb},1,,,,,,{2490,490};
      ACTIVITY,,,,,,,,,{50,52,,,};
      RETURNVSN,BESTPROFIT,1,,,,,,,,{2840,490};
SNRECUR1_TERMINATE_1: TERMINATE,INF,,,,,,,,,{2090,520};
writehistory: WRITE,"history_stack_init.txt",YES,"stack = %d\t Cash=%f   \thistory: %s
\tInvA= %f InvB=
%f\n",{STACK,CASHONHAND,HISTORY,INVENTORYA,INVENTORYB},1,,,,,,{1710,580
};
      ACTIVITY,,,,,,,,,{54,56,,,};
      TERMINATE,INF,,,,,,,,,{1950,580};
SNRECUR1_ASSIGN_2: ASSIGN,{{STACK,STACK-1}},1,,,,,,,,{1030,250};
      ACTIVITY,,,,,,,,,{57,59,,,};
SNRECUR1_WRITE_1: WRITE,"history_stack_init.txt",YES,"Stack = %d  \tpost-buyA prune:
gen:  %d\n",{STACK,GENERATION},1,,,,,,{1130,250};
      ACTIVITY,,,stack != -2,,,,,,,{59,62,,,};
      ACTIVITY,,,stack == -2,"SNRECUR1_ASSIGN_1",,,,,,{59,48,,,1860,420,2000,420};
      TERMINATE,INF,,,,,,,,,{1320,250};
SNRECUR1_ASSIGN_3: ASSIGN,{{STACK,STACK-1}},1,,,,,,,,{610,300};
      ACTIVITY,,,,,,,,,{63,65,,,};
SNRECUR1_GOON_1: GOON,2,,,,,,,,,{740,300};
      ACTIVITY,,,((InventoryA <= ((Horizon- GENERATION) * MaxTransA))  &&
(InventoryB <= ((Horizon- GENERATION) * MaxTransB))),,,,,,,{65,68,,,};
      ACTIVITY,,,((InventoryA > ((Horizon- GENERATION) * MaxTransA))  ||  (InventoryB >
((Horizon- GENERATION) *
MaxTransB))),"SNRECUR1_WRITE_4",,,,,,{65,75,,,770,320,770,360};
SNRECUR1_WRITE_3: WRITE,"history_stack_init.txt",YES,"Stack= %d  \tpost-hold :  gen:
%d\n",{STACK,GENERATION},1,,,,,,{1110,300};
      ACTIVITY,,,,,,,,,{68,70,,,};
ExecuteHold:
ASSIGN,{{HISTORY,STRCAT(HISTORY,"H")},{GENERATION,GENERATION+1},{Histo
ryCh,strcat(HistoryCh,"H")},{HistoryNu,strcat(HistoryNu,"0,0,0,")}},4,,,,,,,,{1260,300};
      ACTIVITY,31,,Generation <=HORIZON,"BuyBuy",,,,,,{70,28,,,1670,190};
      ACTIVITY,32,,GENERATION<=HORIZON,"BuyHold",,,,,,{70,30,,,1690,250};
      ACTIVITY,33,,GENERATION<=HORIZON,"BuySell",,,,,,{70,32,,,1660,350};
      ACTIVITY,,,Generation > Horizon,"SNRECUR1_GOON_2",,,,,,{70,34,,,1410,360};
SNRECUR1_WRITE_4: WRITE,"history_stack_init.txt",YES,"Stack= %d  \tpost-hold (prune) :
gen:  %d\n",{STACK,GENERATION},1,,,,,,{880,360};
      ACTIVITY,,,stack != -2,,,,,,,{75,78,,,};
      ACTIVITY,,,stack == -2,"SNRECUR1_ASSIGN_1",,,,,,{75,48,,,1840,420,2000,420};
      TERMINATE,INF,,,,,,,,,{1230,360};
SNRECUR1_GOON_3: GOON,1,,,,,,,,,{600,420};
      ACTIVITY,,,(InventoryA> .0001 && InventoryB>.0001),,,,,,,{79,82,,,};
```

```
      ACTIVITY,,,(InventoryA< .0001 ||
InventoryB<.0001),"SNRECUR1_ASSIGN_4",,,,,,{79,97,,,700,470,870,800};
CalcSellQtys: ASSIGN,{{fReadResult,USERF(2)}},1,,,,,,,,{760,420};
      ACTIVITY,,,,,,,,,{82,84,,,840,420};
ExecuteSell:
ASSIGN,{{HISTORY,strcat(HISTORY,"S")},{HISTORY,strcat(HISTORY,"[a]")},{HISTORY
,strcat(HISTORY,itoa((nint(currSellVolA))))},{HISTORY,strcat(HISTORY,"[ab]")},{HISTOR
Y,strcat(HISTORY,itoa(nint(currSellVolAB)))},{HISTORY,strcat(HISTORY,"[b]")},{HISTOR
Y,strcat(HISTORY,itoa(nint(currSellVolB)))},{HistoryCh,strcat(HistoryCh,"S")},{HistoryNu,st
rcat(HistoryNu,itoa(nint(-
1.0*CurrSellVolA)))},{HistoryNu,strcat(HistoryNu,",")},{HistoryNu,strcat(HistoryNu,itoa(nint(
-
1.0*CurrSellVolAB)))},{HistoryNu,strcat(HistoryNu,",")},{HistoryNu,strcat(HistoryNu,itoa(nin
t(-1*CurrSellVolB)))},{HistoryNu,strcat(HistoryNu,",")}},2,,,,,,,,{880,440};
      ACTIVITY,,,,,,,,,{84,87,,,1140,420};
      ACTIVITY,,,,"SNRECUR1_WRITE_2",,,,,,{84,94,,,};
SNRECUR1_ASSIGN_7: ASSIGN,{{STACK,STACK-1}},1,,,,,,,,{1170,420};
      ACTIVITY,,,,,,,,,{87,89,,,};
PostSell: ASSIGN,{{GENERATION,GENERATION+1}},4,,,,,,,,{1270,420};
      ACTIVITY,41,,GENERATION<=HORIZON,"BuyBuy",,,,,,{89,28,,,1670,190};
      ACTIVITY,42,,GENERATION<=HORIZON,"BuyHold",,,,,,{89,30,,,1690,250};
      ACTIVITY,43,,GENERATION<=HORIZON,"BuySell",,,,,,{89,32,,,1660,350};
      ACTIVITY,,,Generation > Horizon,"SNRECUR1_GOON_2",,,,,,{89,34,,,1420,450};
SNRECUR1_WRITE_2: WRITE,"history_stack_init.txt",YES,"stack = %d  post-sell  gen:  %d
Current Action: %s   History= %s  Value=%f   InvA= %f  InvB:  %f   CurrSellVolA=%f
CurrSellVolAB=%f
CurrSellVolB=%f\n",{stack,GENERATION,CURRENTACTION,HISTORY,CASHONHAND,
INVENTORYA,INVENTORYB,CurrSellVolA,CurrSellVolAB,CurrSellVolB},1,,,,,,{1490,710
};
      ACTIVITY,,,,,,,,,{94,96,,,};
      TERMINATE,INF,,,,,,,,,{2020,710};
SNRECUR1_ASSIGN_4: ASSIGN,{{STACK,STACK-1}},1,,,,,,,,{930,800};
      ACTIVITY,,,,,,,,,{97,99,,,};
SNRECUR1_WRITE_5: WRITE,"history_stack_init.txt",YES,"Stack = %d  \tpost-sell:  prune
gen:  %d history: %s\n",{STACK,GENERATION,HISTORY},1,,,,,,{1100,800};
      ACTIVITY,,,stack== -
2,"SNRECUR1_ASSIGN_1",,,,,,{99,48,,,1290,790,2080,790,2080,560};
      ACTIVITY,,,stack != -2,,,,,,,{99,102,,,};
      TERMINATE,INF,,,,,,,,,{2400,800};

[RAND01 SUBNETWORK]

;DBF file created with Version 4
      VSN,SNRAND01,,,,,,,,,{40,30};
NodeArrives: ENTERVSN,ProcessSVN,1,,,,,,,,{50,300};
      ACTIVITY,,,,,,,,,{2,4,,,};
Start: GOON,1,,,,,,,,,{90,300};
      ACTIVITY,1,,,,,,,,{4,6,,,};
```

```
      ASSIGN,{{nextAction,nint(UNFRM(1,3,7))}},1,,,,,,,,{340,300};
      ACTIVITY,,,nextAction==1,,,,,,{6,11,,,450,230};
      ACTIVITY,,,nextAction==2,"BuyHold",,,,,{6,53,,,};
      ACTIVITY,,,nextAction==3,"BuySell",,,,,{6,60,,,450,420};
      ACTIVITY,,,,"SNRANDOM_WRITE_2",,,,,{6,71,,,440,460,490,480};
BuyBuy: ASSIGN,{{CURRENTACTION,"B"}},1,,,,,,,,{490,230};
      ACTIVITY,,,,"SNRECUR1_WRITE_6",,,,,{11,13,,,};
SNRECUR1_WRITE_6: WRITE,"history_stack_init_rnd.txt",YES,"variationNumber %d
\tbuy:  \tgen:  %d\t nextAction=%d\t InvA: %f \t InvB:
%f\n",{variationNumber,GENERATION,nextAction,InventoryA,InventoryB},1,,,,,,{620,230};
      ACTIVITY,,,,,,,,,{13,15,,,};
calcBuyVols: ASSIGN,{{fReadResult,USERF(7)}},1,,,,,,,,{900,230};
      ACTIVITY,,,,,,,,,{15,17,,,};
PostBuy2:
ASSIGN,{{Generation,Generation+1},{History,strcat(History,"B")},{HistoryCh,strcat(HistoryC
h,"B")},{HistoryNu,strcat(HistoryNu,itoa(nint(CurrBuyVolA)))},{HistoryNu,strcat(HistoryNu,"
,0,")},{HistoryNu,strcat(HistoryNu,itoa(nint(CurrBuyVolB)))},{HistoryNu,strcat(HistoryNu,",")
}},2,,,,,,,,{1200,230};
      ACTIVITY,,0,Generation <=Horizon,"SNRANDOM_GOON_1",,,,,{17,20,,,1500,240};
      ACTIVITY,,,Generation > Horizon,"SNRECUR1_GOON_2",,,,,{17,22,,,1460,420};
SNRANDOM_GOON_1: GOON,1,,,,,,,,{1570,260};
      ACTIVITY,,,,"Start",,,,,{20,4,,,1660,260,1660,50,60,50};
SNRECUR1_GOON_2: GOON,3,,,,,,,,{1470,490};
      ACTIVITY,,,,,,,,,{22,24,,,};
End_of_Cycle: ASSIGN,{{fReadResult,USERF(8)}},1,,,,,,,,{1520,490};
      ACTIVITY,,,,,,,,,{24,26,,,};
writehistory: WRITE,"history_stack_init_rnd.txt",YES,"variationNumber %d\t GEN
%d\tCash=%f   \thistory: %s  \tInvA= %f InvB=
%f\n",{variationNumber,generation,CASHONHAND,HISTORY,INVENTORYA,INVENTOR
YB},1,,,,,{1650,660};
      ACTIVITY,,,,,,,,,{26,28,,,1920,660,1920,600,1680,600,1610,490};
      GOON,2,,,,,,,,{1630,490};
      ACTIVITY,,,CashOnHand>BestProfit,,,,,,{28,31,,,};

ACTIVITY,,,CashOnHand<=BestProfit,"SNRANDOM_GOON_2",,,,,{28,41,,,1700,560,2730,
560};
PREPAREEXIT:
ASSIGN,{{BESTPROFIT,MAX(BESTPROFIT,CASHONHAND)},{RVAL_ACTION,HISTO
RY},{RVAL_PROFIT,MAX(BESTPROFIT,CASHONHAND)},{BestHistory,History},{BestH
istoryCh,HistoryCh},{BestHistoryNu,HistoryNu}},1,,,,,,,,{1710,490};
      ACTIVITY,1999,,,,,,,,{31,33,,,};
BP: COLCT,1003,BESTPROFIT,"BestProfit_SVN",20,0,1000,1,,,,{1940,490};
      ACTIVITY,,,,,,,,,{33,35,,,};
SNRECUR1_WRITE_9: WRITE,"history_stack_init_rnd.txt",YES,"variationNumber %d \tSet
BestProfit:  gen:  %d
%s\n",{variationNumber,GENERATION,CURRENTACTION},1,,,,,{2100,490};
      ACTIVITY,,,,"SNRECUR1_ASSIGN_1",,,,,{35,37,,,2270,490};
```

```
SNRECUR1_ASSIGN_1:
ASSIGN,{{BESTPROFIT,MAX(BESTPROFIT,CASHONHAND)},{RVAL_ACTION,HISTO
RY},{RVAL_PROFIT,MAX(BESTPROFIT,CASHONHAND)},{readResult,USERF(cTimerG
et)},{ClockTime2,ClockTime-ClockTime1}},1,,,,,,,,{2350,490};
    ACTIVITY,,,,,,,,,{37,39,,,};
SNRANDOM_WRITE_1: WRITE,"results_random.txt",YES,"scen:
%d,ct=%d,ct1=%d,ct2=%d,%d,%f,%s,%s\n",{CurrentScenerio,ClockTime,ClockTime1,ClockTi
me2,variationNumber,rval_profit,BestHistoryCh,BestHistoryNu},1,,,,,,{2610,490};
    ACTIVITY,,,,,,,,,{39,41,,,};
SNRANDOM_GOON_2: GOON,1,,,,,,,,,{2810,490};
    ACTIVITY,,,variationNumber<variationMax,,,,,,,{41,44,,,2790,400};
    ACTIVITY,,,variationNumber==variationMax,"SNRECUR1_WRITE_8",,,,,,{41,50,,,};
    GOON,1,,,,,,,,,{2830,400};
    ACTIVITY,,,,,,,,,{44,46,,,};
    ASSIGN,{{variationNumber,variationNumber
+1},{history,""},{CashOnHand,0},{historyCH,""},{historyNU,""},{Generation,1}},1,,,,,,,,{288
0,400};
    ACTIVITY,,,,,,,,,{46,48,,,};
ResetInventory: ASSIGN,{{fReadResult,USERF(9)}},1,,,,,,,,{3050,400};

ACTIVITY,,,,"SNRANDOM_GOON_1",,,,,,{48,20,,,3160,400,3160,300,2710,300,2710,380,25
45,380,2380,380,1860,380,1560,380};
SNRECUR1_WRITE_8: WRITE,"results.txt",YES,"scen:,
%d,%d,%f,%s,%s%f\n",{CURRENTSCENERIO,ClockTime2,rval_profit,BestHistoryCh,BestH
istoryNu,AinAb},1,,,,,,{2960,490};
    ACTIVITY,,,,,,,,,{50,52,,,};
    RETURNVSN,BESTPROFIT,1,,,,,,,,{3120,490};
BuyHold: ASSIGN,{{CURRENTACTION,"H"}},1,,,,,,,,{490,300};
    ACTIVITY,,,,,,,,,{53,55,,,};
SNRECUR1_WRITE_3: WRITE,"history_stack_init_rnd.txt",YES,"variationNumber %d\thold
: gen: %d\t nextAction=%d\n",{variationNumber,GENERATION,nextAction},1,,,,,,{620,300};
    ACTIVITY,,,,,,,,,{55,57,,,};
ExecuteHold:
ASSIGN,{{HISTORY,STRCAT(HISTORY,"H")},{GENERATION,GENERATION+1},{Histo
ryCh,strcat(HistoryCh,"H")},{HistoryNu,strcat(HistoryNu,"0,0,0,")}},2,,,,,,,,{1190,310};
    ACTIVITY,31,,Generation
<=HORIZON,"SNRANDOM_GOON_1",,,,,,{57,20,,,1500,260};
    ACTIVITY,,,Generation > Horizon,"SNRECUR1_GOON_2",,,,,,{57,22,,,1450,430};
BuySell: ASSIGN,{{CURRENTACTION,"S"}},1,,,,,,,,{490,420};
    ACTIVITY,13,,,"SNRECUR1_GOON_3",,,,,,{60,62,,,590,420};
SNRECUR1_GOON_3: GOON,1,,,,,,,,,{650,420};
    ACTIVITY,,,,,,,,,{62,64,,,};
CalcSellQtys: ASSIGN,{{fReadResult,USERF(2)}},1,,,,,,,,{790,420};
    ACTIVITY,,,,,,,,,{64,66,,,920,420,920,530,770,530,770,600};
SNRECUR1_WRITE_2: WRITE,"history_stack_init_rnd.txt",YES,"variationNumber
%d\tsell(post) nextAction=%d\tgen: %d Current Action: %s History= %s Value=%f InvA=
%f InvB: %f CurrSellVolA=%f CurrSellVolAB=%f
CurrSellVolB=%f\n",{variationNumber,nextAction,GENERATION,CURRENTACTION,HIST
```

```
ORY,CASHONHAND,INVENTORYA,INVENTORYB,CurrSellVolA,CurrSellVolAB,CurrSel
lVolB},1,,,,,,{830,600};
      ACTIVITY,,,,,,,,,,{66,68,,,1430,600,1430,530,1100,530,1100,420};
ExecuteSell:
ASSIGN,{{HISTORY,strcat(HISTORY,"S")},{HistoryCh,strcat(HistoryCh,"S")},{HistoryNu,st
rcat(HistoryNu,itoa(nint(-
1.0*CurrSellVolA)))},{HistoryNu,strcat(HistoryNu,",")},{HistoryNu,strcat(HistoryNu,itoa(nint(
-
1.0*CurrSellVolAB)))},{HistoryNu,strcat(HistoryNu,",")},{HistoryNu,strcat(HistoryNu,itoa(nin
t(-
1*CurrSellVolB)))},{HistoryNu,strcat(HistoryNu,",")},{Generation,Generation+1}},3,,,,,,,,{11
30,420};

ACTIVITY,41,,GENERATION<=HORIZON,"SNRANDOM_GOON_1",,,,,,{68,20,,,1500,280
};
      ACTIVITY,,,Generation > Horizon,"SNRECUR1_GOON_2",,,,,,{68,22,,,1440,450};
SNRANDOM_WRITE_2: WRITE,"history_stack_init_rnd.txt",YES,"variationNumber:
%d\tbad nextActionValue:  gen:  %d\t
nextAction=%d\n",{variationNumber,GENERATION,nextAction},1,,,,,,{530,480};
      ACTIVITY,,,,,,,,,,{71,73,,,};
      TERMINATE,INF,,,,,,,,,{780,480};

[RAND02 SUBNETWORK]
;DBF file created with Version 4
      VSN,SNRAND02,,,,,,,,,{40,30};
NodeArrives: ENTERVSN,ProcessR02,1,,,,,,,,{50,300};
      ACTIVITY,,,,,,,,,,{2,4,,,};
Start: GOON,1,,,,,,,,,{90,300};
      ACTIVITY,1,,,,,,,,{4,6,,,130,300,130,240};
      ASSIGN,{{rand02Action,0},{changeAPerCent,nint(UNFRM(-
4,4,7))*.25}},1,,,,,,,,{150,240};
      ACTIVITY,,,,,,,,,,{6,8,,,};
SNRAND02_WRITE_1: WRITE,"history_stack_init_rnd02.txt",YES,"VarNo: %d  gen %d
APct= %8.3f    ",{variationnumber,generation,changeAPercent},1,,,,,,{320,240};
      ACTIVITY,,,,,,,,,,{8,10,,,};
      ASSIGN,{{fReadResult,USERF(cRand_A)}},1,,,,,,,,{450,240};
      ACTIVITY,,,,,,,,,,{10,12,,,590,240,590,260,150,260,140,260,140,280};
      ASSIGN,{{changeBPerCent,nint(UNFRM(-4,4,7))*.25}},1,,,,,,,,{150,280};
      ACTIVITY,,,,,,,,,,{12,14,,,};
SNRAND02_WRITE_2: WRITE,"history_stack_init_rnd02.txt",YES,"BPct= %8.2f
",{changeBPercent},1,,,,,,{320,280};
      ACTIVITY,,,,,,,,,,{14,16,,,};
      ASSIGN,{{fReadResult,USERF(cRand_B)}},1,,,,,,,,{450,280};
      ACTIVITY,,,,,,,,,,{16,18,,,590,280,590,300,140,300,140,320};
      ASSIGN,{{changeABPerCent,nint(UNFRM(-4,0,7))*.25}},1,,,,,,,,{150,320};
      ACTIVITY,,,,,,,,,,{18,20,,,};
SNRAND02_WRITE_3: WRITE,"history_stack_init_rnd02.txt",YES,"ABPct= %8.2f
currDeltaVolAB   %8.2f  ",{changeABPercent,currABDeltaVol},1,,,,,,{320,320};
```

```
      ACTIVITY,,,,,,,,,,{20,22,,,};
      ASSIGN,{{fReadResult,USERF(cRand_AB)}},1,,,,,,,,{490,320};
      ACTIVITY,,,,,,,,,,{22,24,,,};
SNRAND02_WRITE_4: WRITE,"history_stack_init_rnd02.txt",YES,"  dltA= %8.3f  dltAB=
%8.3f  dltB= %8.3f   \n",{CurrADeltaVol,CurrABDeltaVol,CurrBDeltaVol},1,,,,,,{620,320};
      ACTIVITY,,,,,,,,,,{24,26,,,780,320,780,360,160,360,90,360,90,430};
SNRECUR1_WRITE_6: WRITE,"history_stack_init_rnd02.txt",YES,"varNo: %d  gen %d
Apct %8.2f  Bpct %8.2f  ABpct %8.2f  act %d  DltA %8.2f  DltAB %8.2f  DltB %8.2f invA
%8.2f
invB%8.2f\n",{variationNumber,generation,changeAPercent,changeBPercent,changeABPercent,
rand02Action,CurrADeltaVol,CurrABDeltaVol,CurrBDeltaVol,InventoryA,InventoryB},1,,,,,,{
170,430};
      ACTIVITY,,,,,,,,,,{26,28,,,580,430,550,490,530,510,550,510};
SNRECUR1_WRITE_2: WRITE,"history_stack_init_rnd_pre2.txt",YES,"var %d gen:  %d
Act=%d  $$=%8.2f  InvA= %8.2f  InvB:  %8.2f  hist:  %s
\n",{variationNumber,GENERATION,rand02Action,CASHONHAND,INVENTORYA,INVEN
TORYB,HISTORY},1,,,,,,{580,510};
      ACTIVITY,,,,,,,,,,{28,30,,,1190,510,1190,420,800,420,800,300};
SNRAND02_ASSIGN_1:
ASSIGN,{{Generation,Generation+1},{HistoryNu,strcat(HistoryNu,itoa(nint(CurrADeltaVol)))
},{HistoryNu,strcat(HistoryNu,",")},{HistoryNu,strcat(HistoryNu,itoa(nint(CurrABDeltaVol)))}
,{HistoryNu,strcat(HistoryNu,",")},{HistoryNu,strcat(HistoryNu,itoa(nint(CurrBDeltaVol)))},{
HistoryNu,strcat(HistoryNu,",")}},1,,,,,,,,{940,300};
      ACTIVITY,,,,,,,,,,{30,32,,,};
      GOON,1,,,,,,,,,{1180,300};
      ACTIVITY,,,generation==99999,,,,,,,{32,36,,,};

ACTIVITY,41,,GENERATION<=HORIZON,"SNRANDOM_GOON_1",,,,,,{32,39,,,1330,300
};
      ACTIVITY,,,Generation > Horizon,"SNRECUR1_GOON_2",,,,,,{32,41,,,1270,330};
ExecuteSell: ASSIGN,{{Generation,Generation+1},{HISTORY,strcat(HISTORY,"
")},{HISTORY,strcat(HISTORY,itoa(rand02Action))},{HISTORY,strcat(HISTORY,"[a]")},{H
ISTORY,strcat(HISTORY,itoa((nint(currADeltaVol))))},{HISTORY,strcat(HISTORY,"[ab]")},
{HISTORY,strcat(HISTORY,itoa(nint(currABDeltaVol)))},{HISTORY,strcat(HISTORY,"[b]")
},{HISTORY,strcat(HISTORY,itoa(nint(currBDeltaVol)))},{HistoryNu,strcat(HistoryNu,itoa(ni
nt(-
1.0*CurrADeltaVol)))},{HistoryNu,strcat(HistoryNu,",")},{HistoryNu,strcat(HistoryNu,itoa(nin
t(-
1.0*CurrABDeltaVol)))},{HistoryNu,strcat(HistoryNu,",")},{HistoryNu,strcat(HistoryNu,itoa(n
int(-1*CurrBDeltaVol)))},{HistoryNu,strcat(HistoryNu,",")}},2,,,,,,,,{1220,90};
      ACTIVITY,,,,,,,,,,{36,38,,,};
      TERMINATE,INF,,,,,,,,,{1480,70};
SNRANDOM_GOON_1: GOON,1,,,,,,,,,{1500,300};
      ACTIVITY,,,,"Start",,,,,{39,4,,,1530,300,1530,210,60,210,60,270};
SNRECUR1_GOON_2: GOON,1,,,,,,,,,{1470,490};
      ACTIVITY,,,,,,,,,,{41,43,,,};
End_of_Cycle: ASSIGN,{{fReadResult,USERF(8)}},1,,,,,,,,{1500,490};
      ACTIVITY,,,,,,,,,,{43,45,,,1600,490,1600,530,1460,530,1460,680};
```

```
writehistory: WRITE,"history_stack_init_rnd02.txt",YES,"End of Hor:   varNo= %d\t GEN
%d\tCash=%f   \thistory: %s  \tInvA= %f InvB=
%f\n",{variationNumber,generation,CASHONHAND,HISTORY,INVENTORYA,INVENTOR
YB},1,,,,,,{1500,680};
      ACTIVITY,,,,,,,,,,{45,47,,,1760,680,1760,640,1610,640,1610,490};
      GOON,2,,,,,,,,,{1630,490};
      ACTIVITY,,,CashOnHand>=BestProfit,,,,,,,{47,50,,,};

ACTIVITY,,,CashOnHand<BestProfit,"SNRANDOM_GOON_2",,,,,,{47,60,,,1700,560,2730,5
60};
PREPAREEXIT:
ASSIGN,{{BESTPROFIT,MAX(BESTPROFIT,CASHONHAND)},{RVAL_ACTION,HISTO
RY},{RVAL_PROFIT,MAX(BESTPROFIT,CASHONHAND)},{BestHistory,History},{BestH
istoryCh,HistoryCh},{BestHistoryNu,HistoryNu}},1,,,,,,,,{1710,490};
      ACTIVITY,1999,,,,,,,,,{50,52,,,};
BP: COLCT,1003,BESTPROFIT,"BestProfit_SVN",20,0,1000,1,,,,{1940,490};
      ACTIVITY,,,,,,,,,,{52,54,,,};
SNRECUR1_WRITE_9: WRITE,"history_stack_init_rnd.txt",YES,"variationNumber %d \tSet
BestProfit:  gen:  %d
%s\n",{variationNumber,GENERATION,CURRENTACTION},1,,,,,,{2100,490};
      ACTIVITY,,,,"SNRECUR1_ASSIGN_1",,,,,,{54,56,,,2270,490};
SNRECUR1_ASSIGN_1:
ASSIGN,{{BESTPROFIT,MAX(BESTPROFIT,CASHONHAND)},{RVAL_ACTION,HISTO
RY},{RVAL_PROFIT,MAX(BESTPROFIT,CASHONHAND)},{readResult,USERF(cTimerG
et)},{ClockTime2,ClockTime-ClockTime1}},1,,,,,,,,{2350,490};
      ACTIVITY,,,,,,,,,,{56,58,,,};
SNRANDOM_WRITE_1: WRITE,"results_random.txt",YES,"scen:
%d,%d,%f,%s,%s\n",{CurrentScenerio,variationNumber,rval_profit,BestHistoryCh,BestHistory
Nu},1,,,,,,{2610,490};
      ACTIVITY,,,,,,,,,,{58,60,,,};
SNRANDOM_GOON_2: GOON,1,,,,,,,,,{2810,490};
      ACTIVITY,,,variationNumber<variationMax,,,,,,,{60,63,,,2790,400,2804,400};
      ACTIVITY,,,variationNumber==variationMax,"SNRECUR1_WRITE_8",,,,,,{60,69,,,};
      GOON,1,,,,,,,,,{2830,400};
      ACTIVITY,,,,,,,,,,{63,65,,,};
      ASSIGN,{{variationNumber,variationNumber
+1},{history,""},{CashOnHand,0},{historyCH,""},{historyNU,""},{Generation,1}},1,,,,,,,,{288
0,400};
      ACTIVITY,,,,,,,,,,{65,67,,,};
ResetInventory: ASSIGN,{{fReadResult,USERF(9)}},1,,,,,,,,{3050,400};

ACTIVITY,,,,"SNRANDOM_GOON_1",,,,,,{67,39,,,3160,400,3160,300,2710,300,2710,380,25
45,380,2380,380,1860,380,1400,380,1400,310};
SNRECUR1_WRITE_8: WRITE,"results_02.txt",YES,"scen:,
%d,%d,%8.2f,%s,%s%3.2f\n",{CURRENTSCENERIO,ClockTime2,rval_profit,BestHistoryCh,
BestHistoryNu,AinAb},1,,,,,,{2960,490};
      ACTIVITY,,,,,,,,,,{69,71,,,};
      RETURNVSN,BESTPROFIT,1,,,,,,,,{3120,490};
```

```
[LOADFILE SUBNETWORK]
;DBF file created with Version 4
      VSN,LOADFILE,{{LoadType,LONGVAL, },{LocalIdx,LONGREF,
},{LocalIdxBase,LONGREF, },{LocalIdxMax,LONGREF, },{vFileName,STRINGVAL,
},{vEchoFileContents,LONGVAL,},{vFileNameEcho,STRINGVAL,}},,,,,,,,,{70,60};
Begin: ENTERVSN,LoadFile,1,,,,,,,,,{60,160};
      ACTIVITY,,,,,,,,,,{2,4,,,80,160};
      GOON,1,,,,,,,,,,{110,160};
      ACTIVITY,,,LoadType==cFileLoadTest,,,,,,,{4,10,,,};

ACTIVITY,,,LoadType==cFileLoadLocal,"ReadFileValue",,,,,,{4,11,,,140,210,140,260,140,300
};
      ACTIVITY,,,LoadType==cFileLoadGlobal,"ReadFileGlobal",,,,,,{4,28,,,130,400,140,490};
      ACTIVITY,,,LoadType==cFileLoadL2G,"CopyObjecToGlobal",,,,,,{4,43,,,120,790};

ACTIVITY,,,LoadType==cFileLoadG2L,"LOADFILE_WRITE_4",,,,,,{4,60,,,10,610,30,660};
Exit: RETURNVSN,0.0,1,,,,,,,,,{1050,190};
ReadFileValue: READ,vFileName,YES,ReadResult,"%f",{atrib[LocalIdxMax]},1,,,,,{220,300};
      ACTIVITY,,,ReadResult >0,,,,,,,{11,14,,,310,290,310,260};
      ACTIVITY,,,ReadResult <=0,"SetIdx",,,,,,{11,18,,,};
Inc_index: ASSIGN,{{LocalIdxMax,LocalIdxMax+1}},1,,,,,,,,,{320,260};
      ACTIVITY,,,,,,,,,,{14,16,,,};
wrteLocal: WRITE,"vFileNameEcho.txt",NO,"   value Read from %s =
%f\n",{vFileName,atrib[LocalIdxMax]-1},1,,,,,,{440,260};
      ACTIVITY,,,,"ReadFileValue",,,,,,{16,11,,,530,260,530,240,190,240,190,280};
SetIdx: ASSIGN,{{LocalIdx,LocalIdxBase+1}},1,,,,,,,,,{350,300};
      ACTIVITY,,,vEchoFileContents==cTrue,,,,,,,{18,21,,,};

ACTIVITY,,,vEchoFileContents==cFalse,"GMIX1_GOON_1",,,,,,{18,26,,,440,340,830,340,87
0,310};
WriteData: WRITE,vFileNameEcho,YES,"%f    \n ",{atrib[LocalIdx]},1,,,,,,{580,300};
      ACTIVITY,,,LocalIdx <LocalIdxMax,,,,,,,{21,24,,,660,280,680,260};
      ACTIVITY,,,LocalIdx >=LocalIdxMax,"GMIX1_GOON_1",,,,,,{21,26,,,};
Inc_idx_2: ASSIGN,{{LocalIdx,LocalIdx+1}},1,,,,,,,,,{710,260};
      ACTIVITY,,,,"WriteData",,,,,,{24,21,,,820,260,820,240,560,240,560,270};
GMIX1_GOON_1: GOON,1,,,,,,,,,{900,300};
      ACTIVITY,,,,"Exit",,,,,,{26,10,,,};
ReadFileGlobal: READ,vFileName,YES,ReadResult,"%f",{XX[LocalIdxMax]},1,,,,,{220,490};
      ACTIVITY,,,ReadResult >0,,,,,,,{28,31,,,310,480,310,450};
      ACTIVITY,,,ReadResult <=0,"SetIdx2",,,,,,{28,33,,,};
Inc_index_2: ASSIGN,{{LocalIdxMax,LocalIdxMax+1}},1,,,,,,,,,{320,450};
      ACTIVITY,,,,"ReadFileGlobal",,,,,,{31,28,,,470,450,470,420,200,420,200,470};
SetIdx2: ASSIGN,{{LocalIdx,LocalIdxBase+1}},1,,,,,,,,,{350,490};
      ACTIVITY,,,vEchoFileContents==cTrue,,,,,,,{33,36,,,};
```

```
ACTIVITY,,,vEchoFileContents==cFalse,"LOADFILE_GOON_1",,,,,,{33,41,,,440,530,830,53
0,870,500};
WriteData2: WRITE,SZ[3],YES,"%f    \n ",{XX[LocalIdx]},1,,,,,,{580,490};
      ACTIVITY,,,LocalIdx <=LocalIdxMax,,,,,,,{36,39,,,660,470,680,450};
      ACTIVITY,,,LocalIdx >LocalIdxMax,"LOADFILE_GOON_1",,,,,,{36,41,,,};
Inc_idx4: ASSIGN,{{LocalIdx,LocalIdx+1}},1,,,,,,,,{710,450};
      ACTIVITY,,,,"WriteData2",,,,,,{39,36,,,820,450,820,430,560,430,560,460};
LOADFILE_GOON_1: GOON,1,,,,,,,,,{900,490};
      ACTIVITY,,,,"Exit",,,,,,{41,10,,,};
CopyObjecToGlobal: ASSIGN,{{XX[LocalIdx],atrib[LocalIdx]}},1,,,,,,,,{180,840};
      ACTIVITY,,,LocalIdx<LocalIdxmax,,,,,,,{43,46,,,};
      ACTIVITY,,,LocalIdx >=LocalIdxMax,"LOADFILE_ASSIGN_3",,,,,,{43,48,,,310,840};
      ASSIGN,{{LocalIdx,LocalIdx+1}},1,,,,,,,,{310,790};

ACTIVITY,,,,"CopyObjecToGlobal",,,,,,{46,43,,,430,790,430,760,240,760,200,760,160,760};
LOADFILE_ASSIGN_3: ASSIGN,{{LocalIdx,LocalIdxBase+1}},1,,,,,,,,{340,840};
      ACTIVITY,,,,,,,,,,{48,50,,,};
      GOON,1,,,,,,,,,,{460,840};
      ACTIVITY,,,vEchoFileContents==cTrue,,,,,,{50,53,,,};

ACTIVITY,,,vEchoFileContents==cFalse,"LOADFILE_GOON_3",,,,,,{50,58,,,510,880,830,88
0,870,850};
LOADFILE_WRITE_2: WRITE,"vFileNameEcho",YES,"%f    \n
",{XX[LocalIdx]},1,,,,,,{580,840};
      ACTIVITY,,,LocalIdx <LocalIdxMax,,,,,,,{53,56,,,660,820,680,800};
      ACTIVITY,,,LocalIdx >=LocalIdxMax,"LOADFILE_GOON_3",,,,,,{53,58,,,};
LOADFILE_ASSIGN_4: ASSIGN,{{LocalIdx,LocalIdx+1}},1,,,,,,,,{710,800};
      ACTIVITY,,,,"LOADFILE_WRITE_2",,,,,,{56,53,,,820,800,820,780,560,780,560,810};
LOADFILE_GOON_3: GOON,1,,,,,,,,,{900,840};
      ACTIVITY,,,,"Exit",,,,,,{58,10,,,};
LOADFILE_WRITE_4: WRITE,SZ[3],YES,"G2L (pre)  localIDX %d   \n
",{LocalIdx},1,,,,,,{70,600};
      ACTIVITY,,,,,,,,,,{60,62,,,170,650,160,690};
CopyGlobalToObject: ASSIGN,{{atrib[LocalIdx],XX[LocalIdx]}},1,,,,,,,,{190,680};
      ACTIVITY,,,LocalIdx<=LocalIdxmax,,,,,,,{62,65,,,};
      ACTIVITY,,,LocalIdx >LocalIdxMax,"SetIdx3",,,,,,{62,69,,,320,680};
LOADFILE_WRITE_3: WRITE,SZ[3],YES,"%d   obj:  %f   xx:  %f\n
",{LocalIdx,atrib[LocalIdx],xx[LocalIdx]},1,,,,,,{320,620};
      ACTIVITY,,,,"LOADFILE_ASSIGN_2",,,,,,{65,67,,,410,620,430,620};
LOADFILE_ASSIGN_2: ASSIGN,{{LocalIdx,LocalIdx+1}},1,,,,,,,,{450,620};
      ACTIVITY,,,,"CopyGlobalToObject",,,,,,{67,62,,,540,600,540,550,240,550,200,600};
SetIdx3: ASSIGN,{{LocalIdx,LocalIdxBase+1}},1,,,,,,,,{350,680};
      ACTIVITY,,,,,,,,,,{69,71,,,};
      GOON,1,,,,,,,,,,{470,680};
      ACTIVITY,,,vEchoFileContents==cTrue,,,,,,{71,74,,,};
```

```
ACTIVITY,,,vEchoFileContents==cFalse,"LOADFILE_GOON_2",,,,,,{71,79,,,520,720,840,72
0,880,690};
LOADFILE_WRITE_1: WRITE,SZ[3],YES,"G2L  output localIDX %d %f    \n
",{LocalIdx,atrib[LocalIdx]},1,,,,,,{590,680};
      ACTIVITY,,,LocalIdx <LocalIdxMax,,,,,,,{74,77,,,710,670,710,640};
      ACTIVITY,,,LocalIdx >=LocalIdxMax,"LOADFILE_GOON_2",,,,,,{74,79,,,};
LOADFILE_ASSIGN_1: ASSIGN,{{LocalIdx,LocalIdx+1}},1,,,,,,,,{740,640};
      ACTIVITY,,,,"LOADFILE_WRITE_1",,,,,,{77,74,,,830,640,830,620,570,620,570,650};
LOADFILE_GOON_2: GOON,1,,,,,,,,,{910,680};
      ACTIVITY,,,,"Exit",,,,,,{79,10,,,};


[PRCECOST SUBNETWORK]

;DBF file created with Version 4
      VSN,PRCECOST,{{ValueBase,LONGVAL,0 element of price
array},{ValueMax,DOUBLEVAL,Max element of value
array},{ActionCode,LONGVAL,}},,,,,,,,,{30,70};
      LIMITSVSN,2,2,2,2,2,2,,,,,{0,0};
      EQUIVALENCE,{{IndexCtr,LLINST[1]}},,,,,,,,,,{0,0};
VaryPriceCost: ENTERVSN,VaryPriceCost,1,,,,,,,,,{60,190};
      ACTIVITY,,,ActionCode==cVaryPriceCostNone,,,,,,,{4,8,,,100,110,1470,110};
      ACTIVITY,,,ActionCode==cVaryPriceCostNorm,"SetPriceBidx",,,,,,{4,9,,,};

ACTIVITY,,,ActionCode==cVaryPriceCostTest,"ExitPriceCost",,,,,,{4,8,,,110,230,720,230,149
0,230};
ExitPriceCost: RETURNVSN,0.0,1,,,,,,,,,{1530,190};
SetPriceBidx: ASSIGN,{{IndexCtr,ValueBase+1}},1,,,,,,,,,{200,190};
      ACTIVITY,,,,,,,,,,{9,11,,,};
WritePriceB: WRITE,"PRCECOSTEchoValueArray.txt",YES,"%f    \n
",{atrib[IndexCtr]},1,,,,,,{310,190};
      ACTIVITY,,,IndexCtr<ValueMax,,,,,,,{11,14,,,420,170,420,150};
      ACTIVITY,,,IndexCtr>=ValueMax,"PRCECOST_ASSIGN_3",,,,,,{11,16,,,};
inc_index_03: ASSIGN,{{IndexCtr,IndexCtr+1}},1,,,,,,,,,{440,150};
      ACTIVITY,,,,"WritePriceB",,,,,,{14,11,,,540,150,540,120,270,120,270,160};
PRCECOST_ASSIGN_3: ASSIGN,{{IndexCtr,ValueBase+1}},1,,,,,,,,,{530,190};
      ACTIVITY,,,,,,,,,,{16,18,,,};
ModifyPrice: ASSIGN,{{atrib[IndexCtr],RNORM(atrib[IndexCtr],1)}},1,,,,,,,,,{640,190};
      ACTIVITY,,,,,,,,,,{18,20,,,810,190};
PRCECOST_ASSIGN_4: ASSIGN,{{IndexCtr,IndexCtr+1}},1,,,,,,,,,{860,190};

ACTIVITY,,,IndexCtr<ValueMax,"ModifyPrice",,,,,,{20,18,,,950,170,950,150,800,150,620,150
,620,170};
      ACTIVITY,,,IndexCtr>=ValueMax,,,,,,,{20,23,,,};
PRCECOST_ASSIGN_1: ASSIGN,{{IndexCtr,ValueBase+1}},1,,,,,,,,,{1040,190};
      ACTIVITY,,,,,,,,,,{23,25,,,};
```

```
PRCECOST_WRITE_1: WRITE,"PRCECOSTEchoValueArray.txt",YES,"%f    \n
",{atrib[IndexCtr]},1,,,,,,{1210,190};
      ACTIVITY,,,IndexCtr<ValueMax,,,,,,,{25,28,,,1320,180,1320,160};
      ACTIVITY,,,IndexCtr>=ValueMax,"ValuesPrinted",,,,,,{25,30,,,};
PRCECOST_ASSIGN_2: ASSIGN,{{IndexCtr,IndexCtr+1}},1,,,,,,,,{1340,160};

ACTIVITY,,,,"PRCECOST_WRITE_1",,,,,,{28,25,,,1440,160,1440,130,1170,130,1170,170};
ValuesPrinted: GOON,1,,,,,,,,,{1470,190};
      ACTIVITY,,,,"ExitPriceCost",,,,,,{30,8,,,};
```