The Pennsylvania State University

The Graduate School

Department of Industrial and Manufacturing Engineering

AUGUMENTED SIMULTANEOUS PERTURBATION STOCHASTIC APPROXIMATION (ASPSA) FOR DISCRETE SUPPLY CHAIN INVENTORY OPTIMIZATION PROBLEMS

A Thesis in

Industrial Engineering and Operations Research

by

Liya Wang

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

December 2006

The thesis of Liya Wang was reviewed and approved* by the following:

*Signatures are on file in the Graduate School

ABSTRACT

In recent years, simulation optimization has attracted a lot of attention because simulation can model the real systems in fidelity and capture the dynamics of the systems. Simultaneous Perturbation Stochastic Approximation (SPSA) is a simulation optimization algorithm that has attracted considerable attention because of its simplicity and efficiency. SPSA performs well for many problems but does not converge for some. This research proposes Augmented Spontaneous Perturbation Stochastic Approximation (ASPSA) algorithm in which SPSA is extended to include presearch, ordinal optimization, non-uniform gain, and line search. Extensive tests show that ASPSA achieves speedup and improves solution quality. ASPSA is also shown to converge. For unconstrained problems ASPSA uses random presearch whereas for constrained problems presearch search is used to find a feasible solution, thereby extending the gradient based approach. Performance of ASPSA is tested for supply chain inventory optimization problems including serial and fork-join supply chain without constraints and fork-join supply chain network with customer service level constraints. To evaluated performance of ASPSA, a naïve implementation of Genetic Algorithm is used to primarily test solution quality and indicate computation effort. Experiments show that ASPSA is comparable to Genetic Algorithms (GAs) in solution quality (worst case 16.67%) but is much more efficient computationally (12x faster).

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my gratitude and respect to my advisor, Dr. Vittal Prabhu. Without his support or supervision, this thesis would not be accomplished. His broad view, commitment and passion in research have been, and will be inspiring me forever.

I am deeply grateful to Dr. A. Ravi Ravindran, Dr. Hong Xu, and Dr. Ling Rothrock. I really appreciate their time spent on serving in my committee. The advices that they gave on my research and the knowledge they imparted in classes have improved the quality of the thesis.

I thank my lab mates at Discrete Lab for giving me a great time in past four years. I also thank all staff and faculty in Industrial Engineering Department for their top-quality teaching and good service. I really appreciate Penn State University for giving me such a good opportunity to study here by providing good environment and top facilities.

Finally, my special thanks are to my family (mom and dad) and friends (especially to my boyfriend, Rui Cheng) for their unselfish love and care. Every achievement of mine can not be got withouth their support. They are always of my inspiration of pursuing a meaningful life.

Chapter 1

Introduction

Global optimization refers to a mathematical program, which aims at a maximum or minimum objective function value over a set of feasible solutions (Spall, 2002). The adjective “global” indicates that the optimization problem very general in nature; the objective function maybe nonconvex, nondifferentiable, and possibly discontinuous over a continuous or discrete domain. The problem of designing algorithms that obtain global solutions is very difficult when there is no overriding structure that indicates whether a local solution is indeed the global solution.

Although global optimization problems are difficult to solve, applications of global optimization problems are very common in engineering and real world systems. Engineers are often required to provide some solution to their problem, even if it is a suboptimal one. Sometimes it is very difficult to get the global optimization such that a practitioner is satisfied with any feasible solution. Currently, simulation optimization has been being applied to complex systems optimization problems where the objective functions and constraints may only be evaluated using a simulation model because not only the problem lacks structure and fall into the category of “black box” functions, but also it has the additional complication of having randomness in the function evaluation.

The field of global optimization has attracted extensive attention from academia and has been developing at a rapid pace in the past decades. For example, stochastic methods, such as Simulated Annealing (SA) and Genetic Algorithms (GAs), are gaining in popularity among practitioners and engineers because they are relatively easy to program on a computer and can be applied to a broad class of global optimization problems.

However, there is a fundamental tradeoff between algorithm efficiency and algorithm robustness (reliability and stability in a broad range of problems). That is, algorithms that are designed to be very efficient on one type of problem tend to be "brittle" in the sense that they do not reliably transfer to problems of a different type (Spall, 2002). Hence, there can never be a universally best search algorithm in practice, which manifests the *no free lunch* (NFL) theorems in Wolpert and Macready (1997), which say that an algorithm is effective on one class of problems must be ineffective on another class. An informal mathematical representation of the NFL theorem is:

$$\boxed{\text{Size of domain of applicability} \times \text{Efficiency on domain of applicability} = \text{Constant}}$$

That is, an algorithm cannot have both wide applicability and uniformly high efficiency.

The ultimate goal of global optimization techniques is to develop a single method that (Spall, 2002):

- Works for a large class of problems,
- Finds the global optima with an absolute guarantee, and
- Uses very little computation.

However, we are far from achieving this goal.

The emergence of simulation optimization (Figure **1-1**) over the last decade has made optimization easier by providing the user the ability to obtain good performance of the system without knowledge and analysis of the objective function. In addition, Monte

Carlo simulation can also be used to analyze complex systems in the presence of uncertainties. Compared to analytical techniques, simulation provides the flexibility to accommodate arbitrary stochastic elements, and generally allows modeling of all of the complexities and dynamics of real world systems without undue simplifying assumptions.

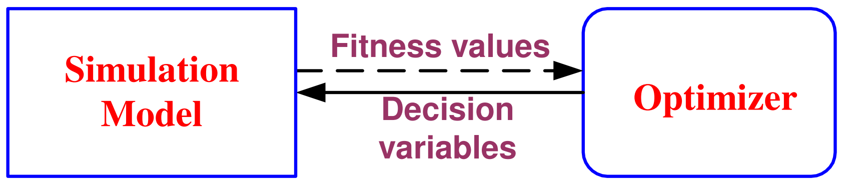


Figure 1-1: Simulation based optimization model

In addition, many of the leading simulation software companies also have added optimization packages in concordance with their simulation packages. For example, System Modeling Corporation, the makers of "Arena", have an optimization package called "OptQuest", which incorporates algorithms based on tabu search, scatter search, and integer programming techniques (Glover, 1996). It has a neural network acceleration feature which helps to eliminate sets of decision parameter values that would likely result in a poor solution. It also gives the user an option of performing Taguchi techniques within the experimental region in order to initialize the population of solutions. ProModel has introduced an optimization package called "SimRunner" (Carson and Maria, 1997). This simulation optimization computer uses strictly evolutionary strategy techniques. AutoMod has also introduced a simulation optimization software package called AutoStat, which is a statistical analysis package and employs a "Select the Best" ranking and selection procedure. This procedure finds the single best system, or a subset containing the best system from among a finite number of systems (Carson, 1996).

The methods adopted in commercial software aims at solving a broad class of problems, and their inefficiency has become a big concern for large optimization problems. This has greatly limited their application in practice. Therefore, more research is needed to be explored for suitability to simulation optimization problems (Carson and Maria, 1997).

Among the well known simulation optimization methods, Simultaneous Perturbation Stochastic Approximation (SPSA) (Spall, 1992) has recently attracted considerable attention in areas such as statistical parameter estimation, feedback control, simulation-based optimization, signal and image processing (Spall, 1999) because of its simplicity to implement and fast convergence. In addition, SPSA is designed for large-scale practical optimization problems, which seems very promising for large supply chain optimization problems. Although SPSA is designed for continuous optimization problems, Gerencsėr and Hill (1999) also found out that SPSA algorithms can also be applied on discrete optimization problems.

SPSA proceeds in the general recursive form: $\hat{\theta}_{k+1} = \hat{\theta}_k - a_k \hat{g}_k(\hat{\theta}_k)$, where $\hat{g}_k$ is achieved by simultaneously perturb θ_k at two different directions. Although SPSA is easy to realize, it is noisy and biased (see wan et al., 2005). Therefore, to broaden its application, it is necessary to improve SPSA. Until now, SPSA algorithm only have been tested for toy problems, and its application in real complicated systems such as supply chain inventory network is still unknown. Therefore, it would be interesting to see how SPSA will perform for the complicated supply chain inventory optimization problems.

Efficient inventory management is an important function in supply chain operations since holding of inventories can cost anywhere between 20 to 40 percent of their value. Current markets have too many uncertainty elements such as demands for products due to seasonality or promotion, prices of raw materials and products, lead times of the suppliers, which all make supply chain management (SCM) a difficulty task. Usually, enterprises store the inventory when facing the uncertainties. Excessive inventory incurs unnecessary holding cost, while the inability to meet the customer needs results in both losses of profit and potentially, the long term loss of customers. The expected value of the ability to meet the product needs of a customer is traditionally called customer service level. Under competitive market conditions, customer service level is recognized as an import index that must be monitored and maintained at a high level (Jung et al., 2004). Although increasing inventory level can enhance customer service level, it increases inventory holding cost. Therefore, the optimal deployment of inventory is a vital business function for an enterprise. As Forrester Research pointed out in a recent report, the ability to increase inventory turns is a key differentiator between highly successful and more poorly performing companies (e.g., Wal-Mart vs. Kmart; Dell vs. Compaq).

Asset managers of large manufacturing enterprises, such as those found in the computer, electronics, and auto industries, oversee extremely large supply chains involving multiple products, each with a complex bill of materials (BOM). These asset managers have the responsibility of determining the appropriate inventory levels in the form of components and finished goods to hold at each level of the supply chain in order to guarantee specified end-customer service levels. Given the size and complexity of these supply chains, a common problem for these asset managers is not knowing how to quantify the trade-off between service levels and the investment in inventory required to support those service levels. The problem is made difficult by the fact that the supply chains are dynamic: products have short lifetimes, new products are introduced frequently, customer demands are erratic and non-stationary, and service-level requirements may change over time. The dynamic nature of complex supply chains means that the trade-off between service level and inventory investment changes over time, implying that asset managers must continually reevaluate this trade-off to make informed and timely decisions about inventory levels and service targets.

In the past, although a lot of study has been done for supply chain inventory optimization problems, most of research did not consider customer service levels as a constraint for the optimization. In addition, dominant analytical methods on supply chain inventory optimization problems can only tackle the problems with simple network structure such as serial supply chain and single node network, because fork-join network model is too complicated to be analyzed. Simulation optimization has begun to be used to tackle the complicated optimization problems because simulation can model the reality with sufficient fidelity. Compared with analytical techniques, simulation provides the flexibility to accommodate arbitrary stochastic elements, and generally allows modeling the complexities and dynamics of real world supply chains without undue simplifying assumptions. Until now, there is no efficient algorithm for discrete supply chain inventory optimization problems. Most of researches still adopt simulated annealing (SA) and genetic algorithms (GAs), which are well known for their inefficiency. Therefore, it is necessary to find faster algorithms for discrete supply chain inventory algorithms, which can take into account of customer service level constraints.

In this work, based on SPSA, Augmented Simultaneous Perturbation Stochastic Approximation (ASPSA) will be proposed, and its application on discrete supply inventory optimization problems (unconstrained and constrained) will be studied. ASPSA incorporate four functions to improve the performance of SPSA, and they are: 1) Presearch; 2) Some idea in Ordinal Optimization (OO) (ranking is much easier than valuing); 3) Line search, 4) Uncommon gain. In detail, through manipulating simulation length during optimization process, ASPSA achieves speedup; and through added functions, ASPSA improve solution quality. Chapter 4 will in detail discuss ASPSA algorithm, and its convergence is also proved there. In short, this study focuses on developing an efficient simulation optimization algorithm-ASPSA, and its performance will be tested by complicated discrete supply chain inventory optimization problems such as fork-join network with customer service level requirement, for which traditional analytical methods failed to solve and traditional heuristics are inefficient.

The thesis is made up of seven chapters. Chapter 2 generalizes the previous work on Work-In-Process (WIP) level settings problems in multi-product CONWIP systems. Then Chapter 3 presents a review of all the prominent simulation optimization techniques, the strengths and weakness behind each of the methods, and supply chain inventory study. In Chapter 4, the proposed ASPSA-I algorithm for unconstrained optimization problems is presented. And then the performance of ASPSA-I is tested by sets of experiments in Chapter 5. Chapter 6 presents ASPSA-II for constrained optimization problems, and it also shows numerical experiments and results for constrained fork-join supply chain problems with customer service requirement under stable demands and unstable demands respectively. Finally, we get our conclusion and discuss the future work in Chapter 7.

Chapter 2

A Parallel Algorithm for Setting WIP Levels for Multi-product CONWIP Systems

This chapter presents the published paper on IJPR for WIP level setting problems in multi-product CONWIP manufacturing systems.

2.1 Introduction

Global manufacturing enterprises continually strive to improve their respective manufacturing operations to gain a competitive advantage. One goal that the manufacturing enterprises are trying to achieve is to reduce mean WIP (Work In Process) subject to throughput requirements because reducing WIP can bring many benefits, such as reduced working capital requirements, lowering of storage requirement and associated costs**,** improving product quality, improving customer service, and maintaining flexibility, etc. The CONWIP (CONstant Work In Process) control policy (Sperman *et al.* 1990) is widely used to reduce WIP by controlling the number of parts allowed into the system. In CONWIP systems (Figure **2-1**), once the parts are released, they are attached with cards, also known as kanbans, and processed as quickly as possible until they wind up in the last buffer as finished goods (Hopp and Spearman, 1995, p.432). When an order arrives, a part is removed from the finished goods inventory; at the same time the card is released and returned to the beginning of the production line.

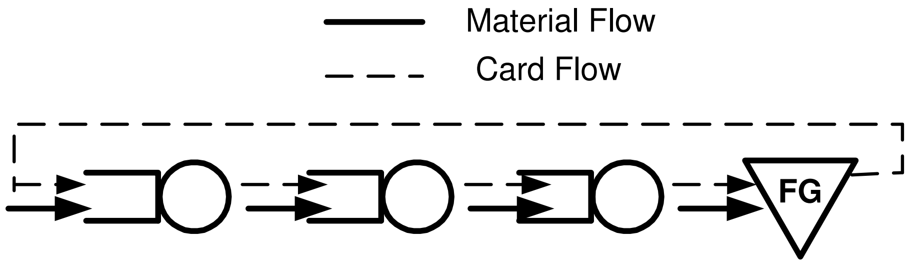


Figure 2-1: A CONWIP system

In an industrial environment usually several different product types are produced, which can be realized by using different processing plans. When CONWIP control policy is applied to such production systems, the systems will be multi-product CONWIP systems. In multi-product CONWIP systems, kanbans can be either dedicated to specific product types or shared among them (see Hopp and Roof, 1998; Duenyas, 1994; Framinan et al., 2000; for discussions). The study of Framinan et al. (2000) showed that the dedicated kanbans provided superior performance, and in his conclusion he pointed out it necessary to find ways to set the card counts for each part type.

The problem of setting the card counts for each part type in multi-product CONWIP systems can be seen as a discrete optimization problem (DOP), which belongs to the class of NP-hard problems. It is attractive to use parallel algorithms for solving NP-hard problems, as the massively parallel systems consisting of a number of powerful processors offer large computation power and memory storage capacity. Until now, intensive parallel algorithms have been developed for all kinds of applications, for example, see Ermakov and Krivulin (1995), Mitra et al. (1997), Phua and Fan (1998), Ralphs (2003) etc.

Previous research on setting WIP levels for multi-product CONWIP systems uses queueing theory and sequential algorithms. For example, MVA (Mean Value Analysis) algorithm (Reiser and Lavenberg, 1980) can be used when the system is modeled as a multi-chain multi-class closed queueing network. The disadvantage of this technique is that the solution is not guaranteed to be optimal. Hopp and Roof (1998) developed the STC (Statistical Throughput Control) method, which relies on two key assumptions: that the data is normally distributed and that the data are not correlated. However, simulation output data are typically auto-correlated (Robinson, 2002). Therefore, the application of this method is greatly limited. Ryan et al. (2000) suggested a heuristic allocation procedure in a CONWIP controlled job shop. Their card dealing heuristic is very time-consuming because the total card number is increased only by one at each iteration if the throughput requirements are not satisfied. Moreover, to decide which product type this card is dedicated to, the running time of this algorithm is proportion to the number of product types, which means it is not suitable to find card counts for a system which produce hundreds or thousands of different product types.

In this paper, a fast, easy to realize parallel algorithm is proposed to set the card counts for each part type in multi-product CONWIP systems subject to throughput requirements. The remainder of this paper is organized as follows. Section 2 is the problem definition. Section 3 describes the algorithms. In section 4, we test our algorithm with three different test cases. Finally we give our conclusion in Section 5.

2.2 Problem Formulation and Properties

Throughout this paper, the following assumptions are being made:

Assumptions:

(1) The sequence policy is FIFO (First In First Out).

(2) The cards are dedicated to a specific product type.

(3) A part will not change its part type during its whole manufacturing process in the system.

(4) The raw material is always available for each product type, so production never has to wait for supplies.

(5) Demand is infinite, so cards are removed from jobs as soon as they finish processing.

Under these assumptions, the CONWIP system can be modeled as a multi-chain multi-class closed network (Figure 2-2).

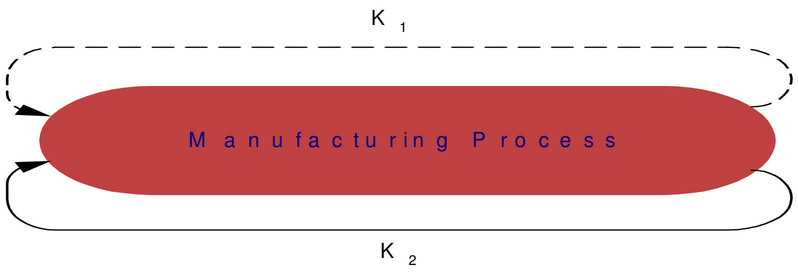


Figure 2-2: A two-product CONWIP system

Notations:

The following notations for the system parameters are defined:

$i = 1,......, N$: Indices for product types.

$j = 1,......, M$: Indices for workstations.

$k = 1,......, p$: Indices for computer processors.

K_i : Number of type i authorization cards, and it is also call WIP level for type i.

$K = (K_1, K_2,...,K_n)$: WIP vector.

$Z = \sum_{i=1}^{N} K_i$: Total number of authorization cards.

θ_i^* : The target throughput for product type i.

θ_i : The real throughput for product type i for a given WIP vector.

$D = \sum_{i=1}^{N} (\theta_i^* - \theta_i)^2$: The total bias between target throughput and real throughput.

Given a set of product types with known manufacturing requirements specified in the forma of a process plan, and a given throughput rate requirement for each product type, the problem of determining the optimal number of authorization cards can be expressed:

$$
\begin{aligned}
&Min \quad Z \\
&s.t. \quad K_1 + K_2 + ... + K_N = Z \\
&\qquad \theta_i(K_1,....., K_N) \ge \theta_i^*, \quad \forall i \\
&\qquad K_1,....., K_N \ge 1, \text{integer}
\end{aligned}
$$

This problem is a nonlinear integer programming problem. Although it is easy to state, it is hard to solve because the clear expression of θ as a function of the WIP vector is not known.

Buzacott and Shanthikumar (1993, p. 393) characterized the throughput properties of the multi-chain multi-class closed queuing network, which can be summarized as follows:

Property 1: *θ_i is increasing in K_i for each i.*

Property 2: *θ_i need not increase in K_j for each $j \neq i$.*

Property 3: *The total throughput $\theta = \sum_{i=1}^{N} \theta_i$ need not increase in K_i for any i.*

Based on the above properties, we propose the following lemma:

Lemma 1: *When θ_i^* is not sufficient at WIP vector $K = (K_1, K_2, ..., K_i, ..., K_N)$, increasing K_i is a way to increase θ_i.*

Proof: From properties 1 and 2, we can obtain lemma 1.

Hopp et al. (1996) suggested the principle of diminishing returns: The marginal improvement in a system from improvements of a single factor tends to decrease as that single factor is improved. In our case, this principle means that the increase in θ_i caused by increasing K_i will decrease as K_i becomes larger and larger. Based on this, we can induct the following lemma:

Lemma 2: *Given a WIP level K_i^{old} and its throughput rate θ_i^{old}, if $\theta_i^{old} < \theta_i^*$, then we have $\frac{\theta_i^{old}}{K_{old}} > \frac{\theta_i^*}{K_i^*}$* (Figure 2-3); *Otherwise, we have $\frac{\theta_i^{old}}{K_{old}} \leq \frac{\theta_i^*}{K_i^*}$.*

Based on Lemma 2, we can calculate the new WIP level for part type i under the following relationships:

If $\theta_{old} < \theta_i^*$, then $K_i^* > \frac{\theta_i^*}{\theta_i^{old}} K_{old}$. Therefore, we can use $K_i^{new} = \left\lfloor \frac{\theta_i^*}{\theta_i^{old}} K_{old} \right\rfloor + 1$ to adjust WIP level.

If $\theta_i^{old} \geq \theta_i^*$, then $K_i^* \leq \frac{\theta_i^*}{\theta_i^{old}} K_{old}$. To prevent waving of WIP level, we use $K_i^{new} = \left\lfloor \frac{\theta_i^*}{\theta_i^{old}} K_i^{old} \right\rfloor + 1$ to adjust the new WIP levels.

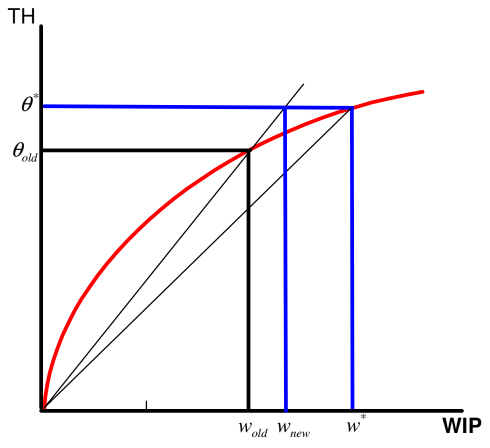


Figure 2-3: Throughput vs. WIP level

2.3 Algorithms

Based on the throughput properties and lemmas listed in Section 2, first we propose a sequential algorithm, and then a parallel algorithm is proposed.

2.3.1 Sequential Algorithm

We know that $K = (1, \quad 1, \quad .... \quad 1)$ is the lower bound for the WIP vector, so we can use it as the initial WIP vector. The Lemma 2 is adopted to adjust WIP levels until the target throughput of each part type is met. Therefore, the algorithms can be described as:

Step 1. ***Initialization***

Set iteration number $m = 0$, and initial WIP vector $K^0 = (1, \quad 1, \quad .... \quad 1)$.

Step 2. ***Run simulation***

Use WIP vector K^m to get the throughput of each product type.

Step 3. ***Check stop criteria***

If $\theta_i \geq \theta_i^* \quad \forall i$ **or the stopping criteria is met, stop.**

Else, go to Step 4.

Step 4. ***Calculate new WIP vector***

$$K_i^{m+1} = \left\lfloor \frac{\theta_i^*}{\theta_i} * K_i^m \right\rfloor + 1$$

$m \leftarrow m + 1$, go to Step 2 again.

Note: we use the iteration number to control the stop criteria.

The sequential algorithm begin to search optimal solution from the low bound of WIP level for each part type, thus it can guarantee the convergence of the algorithm.

2.3.2 Parallel algorithm

To find the best solution rapidly, a parallel search algorithm over distributed computer memory system, based on the sequential one, is proposed. In the parallel algorithm, p different WIP vectors are checked at each iteration. After simulation of these WIP vectors, the best WIP vector with the minimum D is found. Then the processor with best WIP vector will broadcast the best WIP vector and its simulation results to others. After that, each processor can calculate the new WIP vector like the sequential algorithm. To make different processor test different WIP vector, one more step, adjusting the new WIP vector according to some rules, is needed. The above procedures are repeated until one of computer processors find the feasible solution or the stop criteria is met. Therefore, we can summarize the parallel algorithm as the follows:

Step 1. ***Initialization***

Set iteration number $m = 0$, Processor k initializes its WIP vector $K^0 = (k, k, ..., k)$.

Step 2. ***In parallel, each processor runs simulation using their*** K^m

Step 3. ***Synchronization and communication***

The root processor gathers information from other processors to see if the feasible solution has been found or not. Then it broadcasts this information to others.

Step 4. ***In parallel, each processor checks the stop criteria***

If the feasible solution is found, go to Step 6.

Else, go to Step 5.

Step 5. ***Calculate the new WIP vector*** K^{m+1}

1) The root processor gathers D from others and compares them to find the best WIP vector with minimum D. Then it will broadcast the processor ID that has the best WIP vector to others.
2) The processor with best WIP vector will broadcast the best WIP vector and its simulation results to others.
3) In parallel, each processor calculates the new WIP vector using the following rules: $K_i^{m+1} = \left\lfloor \frac{\theta_i^*}{\theta_i} * K_i^m \right\rfloor + 1$
4) In parallel, each processor adjusts the new WIP vector according to the following rules:

```
If (Processor ID>1)
{
        J=Processor ID% Total number of infeasible part types;
        $n = 0$;
        For $(i = 0; i < N; i++)$
        {
            if ($\theta_i < \theta_i^*$)
            {
                    ++n;
                    if(J= =n)
                    $K_i^{m+1} += (\left\lfloor \frac{\text{Processor ID}}{\text{N}} \right\rfloor + 1)$
            }
        }
}
```

5) $m \leftarrow m+1$, go to Step 2 again.

Step 6. ***Find the best solution in all feasible solutions***

From all feasible WIP vectors, choose the best one that has minimum Z .

Step 7. ***Stop***

Usually we evaluate our parallel algorithm using the follows criteria:

Speedup: $S = \frac{T1}{T(p)}$

$T1$: The time it takes to run the same problem with 1 processor.

$T(p)$: The time it takes to run the same problem with p processors.

Referring to speedup, the most desirable one is linear, that is, if we use p processors, we want to get a speedup of p. However, linear speed-up cannot be achieved because the inevitable non-parallel part of a program execution would limit the speed-up. Other factors of restricting speedup are synchronization and communication time. In parallel search algorithms, speedup varies from one execution to another because the portions of the search space examined by each processor are determined dynamically.

2.4 Numerical Results

We tested the performance of our parallel algorithm using 3 test cases. The first test case represents a small system producing two product types on six workstations. The second test case is bigger, producing ten product types on six workstations. The last one is the biggest, which produce 20 different part types on 15 workstations.

The parallel simulation code was written in C language with MPI library, and run on a cluster connected by a high-speed network--Myrinet and made up of 128 dual 3.06 GHz Intel P4 Processors.

Test case 1: The first test case, shown in Figure 2-4, represents a system to product 2 types of products. The number in the box is the mean service time (Unit: min), and all service times follow exponential distribution. The bottleneck workstation throughput for product A is $r_A = 0.25$, and for type B is $r_B = 2$. The system was simulated for 60,000 minutes with a warm-up of 200 outputs using 10 different seeds for three different cases, which have different throughput requirements:

Case 1: $\theta_A^* = 0.2$ and $\theta_B^* = 1.3$.

Case 2: $\theta_A^* = 0.25$ and $\theta_B^* = 0.8$.

Case 2: $\theta_A^* = 0.15$ and $\theta_B^* = 1.6$.

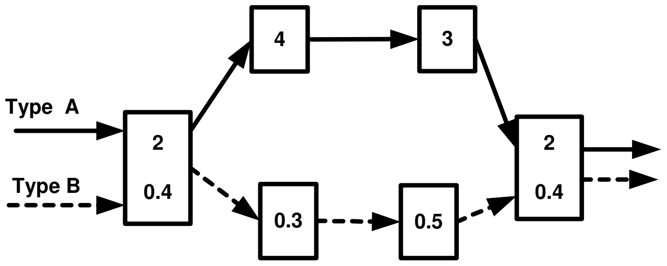


Figure 2-4: Product type flow schematic for test case 1

In Table 2-1, we list the card count for every product type and the real throughput, and we also show the speed vs. number of processors in different cases in Figure 2-5. From speedup figure, we see speedup waves at some point, but the whole trend of speedup still first increases and then decreases as usual parallel processing does. We can see that the largest speedups of these 3 instances are all over 4, which means we can shorten the simulation time more than 4 times, which can be considered as a significant improvement.

Table 2-1: Running results for test case 1

Case	Target TH θ^*		Real TH θ		Z	K_1	K_2
	A	B	A	B			
1	0.2	1.3	0.2	1.3	30	7	23
2	0.25	0.8	0.25	0.8	26	19	7
3	0.15	1.6	0.157	1.6	42	5	37

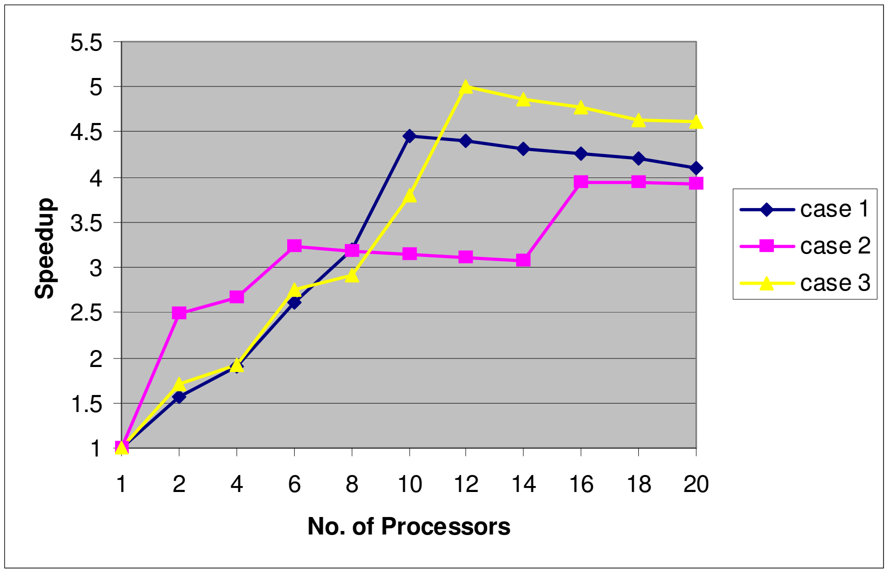


Figure 2-5: Speedup vs. number of processors for test case 1

Test case 2: Then we test the parallel algorithm in a system with 6 workstations, and producing 10 different product types. The service time of all workstations follow exponential distribution. In Table 2-2, the average mean service time of every part type on each workstation is listed (0 means that the part will not be processed on this workstation). The system was simulated using 10 different seeds for 60,000 simulation time units with a warm-up of 300 outputs. Table 2-3 shows the card count for every product type and the real throughput, and Table 2-4 lists the simulation time and speedup using different number of computer processors. And we also show the speedup in Figure 2-.

Table 2-2: Workstation service mean time for each part type

Part Type ID	1	2	3	4	5	6	7	8	9	10
Workstation 1	0.1	0.5	0.2	0.1	0.1	0	0	0	0	0
Workstation 2	0.2	0.1	0.3	0.2	0.5	0.1	0.2	0.1	0.3	0.3
Workstation 3	0.2	0.2	0	0	0	0.3	0.1	0.2	0.1	0.1
Workstation 4	0	0	0.3	0.5	0.1	0.3	0.2	1	0.2	0.1
Workstation 5	0	0	0.1	0.2	0.3	0	0	0	0	0
Workstation 6	0.1	0.1	0	0	0	0	0	0	0	0

Table 2-3: Test case 2 simulation results $Z = 81$

Part Type ID	1	2	3	4	5	6	7	8	9	10
Target TH	1.0	0.8	0.5	0.4	0.4	0.3	0.3	0.3	0.3	0.2
Real TH	1.01	0.81	0.51	0.41	0.41	0.32	0.32	0.32	0.32	0.21
Card count K_i	15	12	10	8	8	6	6	6	6	4

Table 2-4: Simulation Time and speedup for test case 2

Proc. #	1	2	4	6	8	10	12	14	16	18	20
Simu. Tim. (s)	530.4	380.4	322.2	225.6	175.8	175.3	176.7	177.4	178	178.9	179.3
Speedup	1.00	1.39	1.65	2.35	3.02	3.03	3.00	2.99	2.98	2.97	2.96

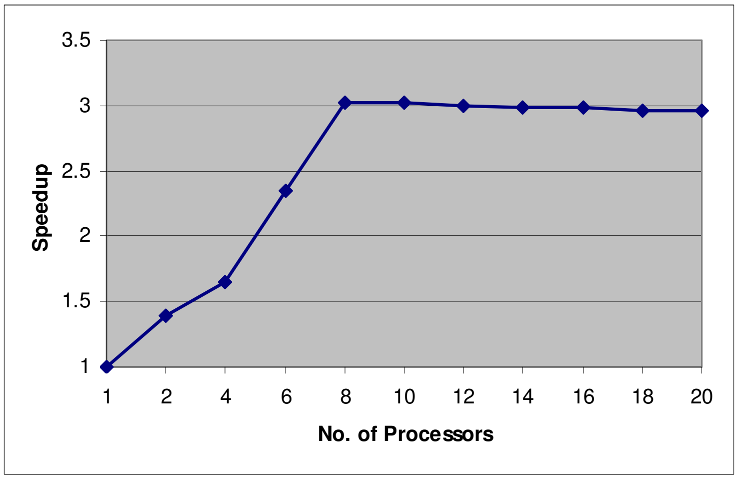


Figure 2-6:Test case 2 speedup vs. number of processors

From Table 2-4, we can see the simulation time dropped 355.14 seconds (about 6 minutes) from 530.40 seconds using 1 processor to 175.26 seconds using 10 processors, which proved the efficiency of the parallel algorithm in shortening the simulation time. Moreover, we see when more than 8 processors are in use, the speedup will decrease slowly, which can be due to the parallel programs run on fast network--Myrinet.

Test case 3: In the last test case, we test a flow line system, in which every part type goes through 15 workstations in sequence, and 20 different product types are produced. The service time of all workstations follow exponential distribution and the mean service time for part type i on any workstation is $0.1*i$. The system was simulated using 10 different seeds for 200,000 simulation time units with a warm-up of 300 outputs. Table 2-5 shows the card count for every product type and the real throughput, and table 6 lists the simulation time and speedup. In Figure 2-6, the speedup vs. number of processors is shown.

Table 2-5: Test case 3 simulation results $Z = 158$

Type ID	Target TH	Real TH	Card Count	Type ID	Target TH	Real TH	Card Count
1	0.08	0.08	14	11	0.05	0.051	9
2	0.06	0.063	11	12	0.04	0.04	7
3	0.04	0.04	7	13	0.04	0.04	7
4	0.04	0.04	7	14	0.05	0.051	9
5	0.03	0.034	6	15	0.05	0.051	9
6	0.04	0.04	7	16	0.04	0.04	7
7	0.04	0.04	7	17	0.03	0.034	6
8	0.04	0.04	7	18	0.04	0.04	7
9	0.04	0.04	7	19	0.04	0.04	7
10	0.06	0.063	11	20	0.03	0.034	6

Table 2-6: Simulation time and speedup for test case 3

No. of Processors	Simulation Time (s)	Speedup	No. of Processors	Simulation Time (s)	Speedup
1	1470.28	1.00	12	552.61	2.66
2	1200.34	1.22	14	553.74	2.66
4	1165.50	1.26	16	555.09	2.65
6	896.43	1.64	18	557.64	2.64
8	547.66	2.68	20	558.08	2.63
10	551.24	2.67			

From Table 2-6, we see that the simulation time dropped from 1470 seconds using one processor to 547.66 seconds using 8 processors. The decrease is over 15 minutes, which implies the efficiency of the proposed parallel algorithm.

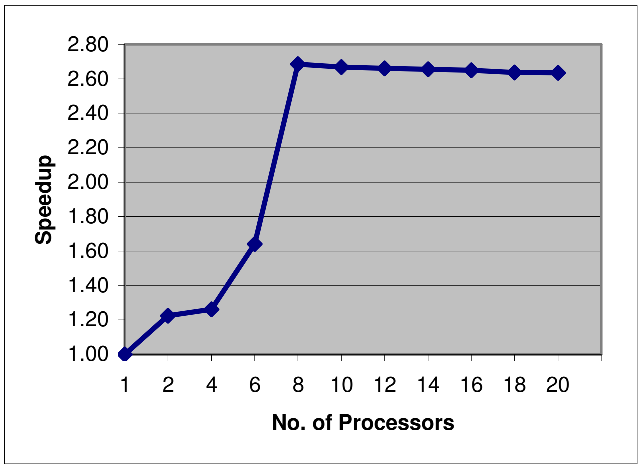


Figure 2-6: Test case 3 speedup vs. number of processors

2.5 Conclusion

Current cutting-edge technology promises parallel processing in solving DOPs. In this paper, a parallel algorithm is presented to solve a DOP, setting WIP level for each product type in multi-CONWIP systems to achieve desired throughput rates with minimum total WIP levels. The parallel algorithm can check p (the number of processors used) different WIP vectors at each iteration, then compare those WIP vectors to get the best one. The diminishing theory is used to calculate the new WIP vector, and some rules are adopted to adjust WIP vector in order that different processors will check different WIP vectors. Greatly decreasing the simulation time for very big systems proves its efficiency.

In all, the advantages of our parallel algorithm can be generalized as: 1) Reducing the simulation time. 2) Fully using the current advanced technology. 3) Convergence.

In this work, only WIP is considered. However, components and finished products are also important inventory in supply chain network, and can not be omitted. Therefore, in the next part of research, algorithms will be developed to consider all three types of inventory in the supply chain network.

Chapter 3

Literature Review

In this part, we reviewed the important methods in simulation optimization, their strength and weakness. After that, we also reviewed the research in supply chain inventory management.

3.1 Simulation Optimization

Simulation Optimization has attracted significant interest to practitioners interested in extracting useful information about an actual system (see, e.g., Hutchison and Hill 2001). Simulation optimization techniques can be divided into five categories (Figure **2-1**): gradient based search methods, response surface methodology, heuristic methods, asynchronous teams (A-Teams), and statistical methods.

Good review of the current research on simulation-based optimization developments can be found in Carson and d Maria (1997), Fu (1994), Swisher and Hyden (2000), Fu (2002), Fu and Hu (1997), April (2003), Swisher et al. (2004). Fu (2002) identifies that five methodologies are studied in the research area. They are: 1) Stochastic approximation (SA) (gradient-based approaches); 2) Response surface methodology; 3) Ranking and selection, multiple comparison procedures, and ordinal optimization; 4) Random search; 5) Sample path optimization. Next, before we introduce five major types of simulation optimization methods, we first give an introduction about problem formulation of simulation optimization.

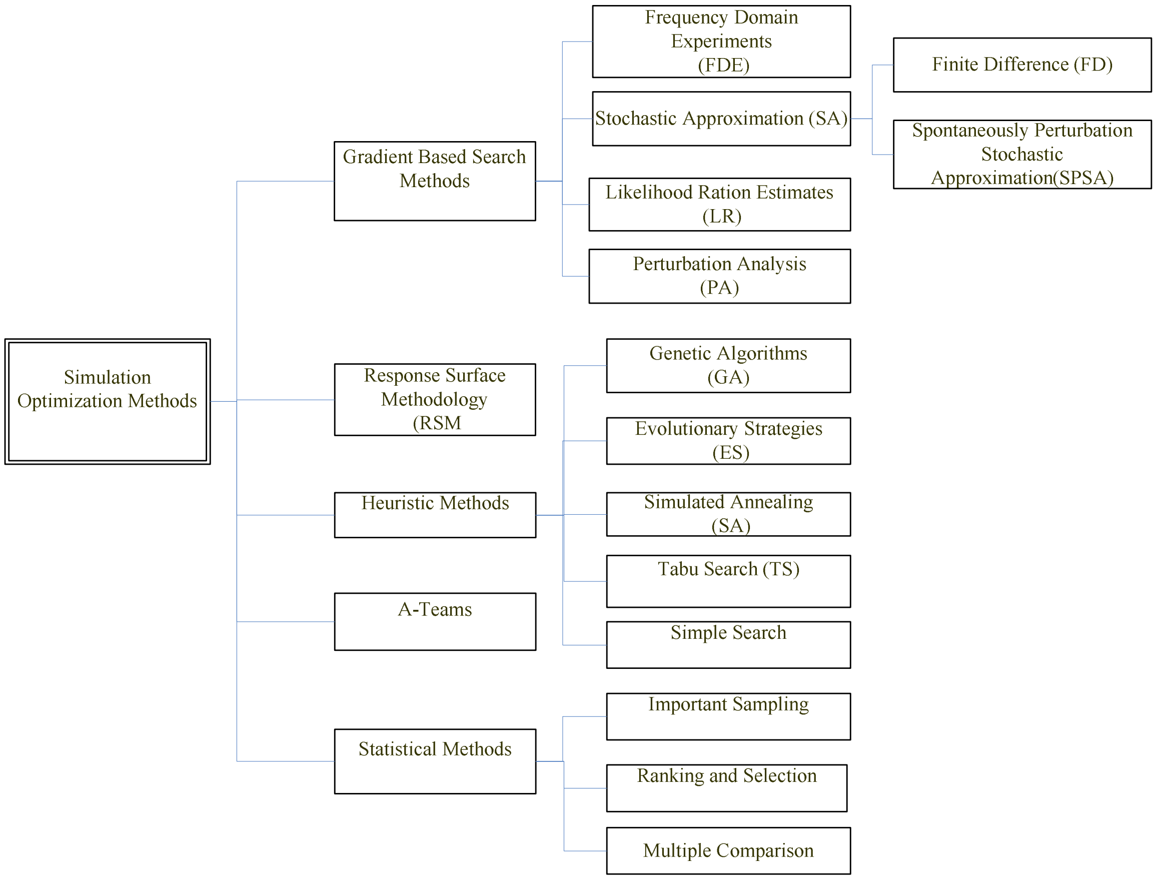

Simulation Optimization Methods
Gradient Based Search Methods
Response Surface Methodology (RSM
Heuristic Methods
A-Teams
Statistical Methods
Frequency Domain Experiments (FDE)
Stochastic Approximation (SA)
Likelihood Ration Estimates (LR)
Perturbation Analysis (PA)
Finite Difference (FD)
Spontaneously Perturbation Stochastic Approximation(SPSA)
Genetic Algorithms (GA)
Evolutionary Strategies (ES)
Simulated Annealing (SA)
Tabu Search (TS)
Simple Search
Important Sampling
Ranking and Selection
Multiple Comparison

Figure 3-1: Simulation Optimization Methods (adapted from Figure 3 in Carson and Maria, 1997)

Problem Formulation

Suppose that $\Theta \subset R^p$ and $\theta \in \Theta$ is a vector with components representing system parameters under control (Spall, 2002). Let $J(\theta)$ be the performance measure or loss function of interest. The exact values of loss are unavailable and are estimated by simulation. Our objective is to solve the following parametric optimization problem:

$$\min_{\theta\in\Theta} \quad J(\theta) = E[L(\theta,\omega)] \tag{3-1}$$

$$\text{s.t.} \quad l_i(\theta) = E[h_i(\theta,\omega)] \le 0; \quad i = 1, \quad 2, \quad \text{K} \quad m \tag{3-2}$$

where $L(\theta,\omega)$ is the sample performance, ω represents the stochastic effects of the system, $l_i(\theta)$ is constraint performance measurement, and $h_i(\theta,\omega_j)$ is sample constraint performance measurement. Using Monte Carlo procedures, we estimate the expectation of the system performance as:

$$J(\theta) = E[L(\theta,\omega)] = \lim_{n\to\infty} \frac{\sum_j L(\theta,\omega_j)}{n} \tag{3-3}$$

$$l_i(\theta) = E[h_i(\theta,\omega)] = \lim_{n\to\infty} \frac{\sum_j h_i(\theta,\omega_j)}{n}; \quad i = 1, \quad 2, \quad \text{K} \quad m \tag{3-4}$$

According to central limit theorem (CLT), we can see that $J(\theta) \sim N(\mu_L, \frac{\sigma_L^2}{n})$, and $h_i(\theta) \sim N(h_i, \frac{\sigma_{h_i}^2}{n})$. Therefore, the convergence rate is $O(\frac{1}{\sqrt{n}})$. Our aim here is to find the optimum $\theta^* = \arg\min_{\theta\in\Theta} J(\theta)$ subject to $h_i(\theta^*) \le 0$ for all i.

3.1.1 Gradient Based Methods

Methods in this category estimates the response function gradient (∇J) to assess the shape of the objective function and employ deterministic mathematical programming techniques. The best-known gradient estimation techniques include:

- Stochastic Approximation (SA), which include famous Finite Difference (FD) and Spontaneously Perturbation Stochastic Approximation (SPSA) methods
- Likelihood Ration/Score Function (LR/SF) method
- Perturbation analysis (PA)
- Frequency Domain Method (FDM).

3.1.1.1 Stochastic Approximation (SA)

SA is a class of algorithms used to minimize (maximize) a function when there is randomness in the optimization process (Andradite, 1998). First introduced by Robbins and Monro (1951) and Kiefer and Wolfowitz (1952), the model has been the subject of considerable research, expanding its applicability and power greatly (see, e.g., Fu, 1994; L' Ecuyer, Giroux, and Glynn, 1994; Shapiro, 1996, Spall, 1998a; Bhatnagar and Borkar, 2003).

Stochastic Approximation (SA) attempts to mimic the gradient search method used in deterministic optimization. The general form of the algorithm takes the following iterative form: $\theta_{k+1} = \theta_k - a_k g_k(\theta)$, where $g_k(\theta)$ is gradient at θ_k . Therefore, the procedures based on this methodology must estimate the gradient of the objective function in order to determine a search direction.

Stochastic approximation targets continuous variable problems because of its close relationship with steepest descent gradient search. In addition, this methodology has been applied to discrete problems (see e.g. Gerencsėr et al. 1999, 2004; Hill et al., 2003). Under appropriate conditions, one can guarantee convergence to the actual minimum with probability one, as the number of iterations goes to infinity (see, e.g., Kushner and Yin, 1997). Because of the estimation noise associated with stochastic optimization, the step size must eventually decrease to zero in order to obtain convergence. Again, because it is a gradient search method, SA generally finds local optimal point, so that enhancements are required for finding the global optimum.

3.1.1.1.1 Finite Differences

FD method (Dennis and Schnabel, 1989) is the oldest method for gradient approximation. Because many problems do not allow for the computation of $\frac{\partial L}{\partial \theta}$, as need in the stochastic gradient form of the root finding (Robbins-Monro) algorithm. This section introduces an alternative to the stochastic gradient algorithm for the case where only the measurements $y(\theta) = J(\theta) + \varepsilon(\theta)$ are available at various values of θ. In fact, the algorithm has the even weaker requirement of only requiring measurements of the difference of two values of the loss function, as opposed to measuring the loss functions themselves.

In FD method, the one-sided gradient approximations involve measurement $y(\theta)$ and $y(\theta + perturbation)$, while two-sided approximations involve measurements of the form $y(\theta \pm perturbation)$. The two-sided FD approximation for use with is

$$g_k(\theta) = \begin{bmatrix} \dfrac{y(\theta_k + c_k\xi_1) - y(\theta_k - c_k\xi_1)}{2c_k} \\ \vdots \\ \dfrac{y(\theta_k + c_k\xi_p) - y(\theta_k - c_k\xi_p)}{2c_k} \end{bmatrix} \qquad \text{3-5}$$

Where ξ_i denotes a vector with a 1 in the i^{th} place and 0's elsewhere and $c_k > 0$ defines the difference magnitude. The pair $\{a_k, c_k\}$ are the gains (or gain sequences) for the FD algorithm. An obvious analogue to Eq. 2-5 holds for the one-sided FD approximation: The i^{th} component of the gradient approximation is $\dfrac{y(\theta_k + c_k\xi_i) - y(\theta_k)}{c_k}$. Because the simplicity of FD method, it has been employed widely in practice such as simulation optimization (Fu and Hu, 1997; p.5).

From the above description of FD method, we can see FD is computationally inefficient. One-sided finite difference, for example, requires $p+1$ function evaluations, while two-sided FD needs $2p$ function evaluations. If p is large and the function evaluations difficult or time-consuming, the computational effort could be substantial. To address this difficulty, Spall (1987, 1988, 1992) developed the simultaneous perturbation stochastic approximation (SPSA) algorithm, which requires only two function evaluations to estimate the gradient at each iteration, regardless of the dimension of θ.

3.1.1.1.2 Simultaneous Perturbation Stochastic Approximation (SPSA)

The theoretical basis for SPSA was developed by Spall (1987, 1992, 1998a, 1998b) and expanded in subsequent work (see Fu and Hill, 1997; Sadegh and Spall, 1998a; Spall, 2000; Bhatnagar and Borker, 2003). The method relies on a computationally efficient estimate of the gradient and calculates gradient by perturbing in all directions simultaneously, as shows in Eq. 3-6.

$$g_k(\theta) = \begin{bmatrix} \dfrac{y(\theta_k + c_k\Delta_k) - y(\theta_k - c_k\Delta_k)}{2c_k\Delta_{k1}} \\ \vdots \\ \dfrac{y(\theta_k + c_k\Delta_k) - y(\theta_k - c_k\Delta_k)}{2c_k\Delta_{kp}} \end{bmatrix} \qquad \textbf{3-6}$$

where $\Delta_k = [\Delta_{k1}, \quad \Delta_{k2}, \quad \ldots \quad \Delta_{kp}]^T$ represents a perturbation vector.

SPSA has been considered as an efficient tool for the solution of difficult optimization problems. It is essentially a randomized Kiefer-Wolfowitz method where the gradient is estimated using only the two measurements per iteration. The method is particularly suited to problems where the cost function can be computed only by expensive simulations. The almost sure convergence, the limit distribution and the rate of convergence of higher order moments to the estimator process have been established or reported in a series of papers (e.g., Spall, 1992; Spall, 1999).

It is clear that the SPSA above will not act the same as the FD approximation as an estimate of the gradient per iteration. The FD approximation will generally be superior in that sense. However, the interest is not in the gradient per iteration. Rather, the interest is in how the approximations operate when considered in optimization over multiple iterations with changing point of evaluation θ. Spall (2002, Section 7.4) discusses the fundamental efficiency issue further, establishing the fundamental result:

Under reasonable general conditions, the SPSA and FD recursions achieve the same level of statistical accuracy for a given number of iterations even though SPSA uses only $1/p$ times the number of function evaluations of FD.

Because SPSA algorithm has been found to be particularly effective in cases where the parameter dimension is high, it has been employed in many areas such as queueing systems (Fu and Hill, 1997), controlling of a heavy ion beam (Hopkins, 1997), industrial quality improvement (Rezayat, 1995), pattern recognition (Maedal et al., 1995), air traffic management (Kleinman et al., 1997), neural network training (Cauwenberghs, 1994), neural network training for adaptive control of dynamic systems (Spall and Cristion, 1994; Maeda and De Figueiredo, 1997), classification of ECG signals for heart monitoring (Gerencsėr, 1998), statistical model parameter estimation/fault detection (Alessandri and Parisini, 1997), human-machine interaction (Nechyba and Xu, 1997), sensor placement and configuration (Sadegh and Spall, 1998b), signal inversion for a complex physical model (Chin, 1999), real-time optimization of model predictive control (Baltcheva and Cristea, 2000), hidden Markov models (Bhatnagar et al., 2001), airplane delays with constraint (Hutchison and Hill, 2001), collision avoidance (Burnett, 2004).

In addition, SPSA has also been modified to discrete set (e.g., see, Gerncsėr et al., 1999, 2001, 2004; Hill et al., 2003). Two common methods have been used to modify SPSA algorithm to discrete optimization problems. One is using truncated method for the product of gain and gradient, and the other is using generalized sign function on the product of gain and gradient.

Besides applying SPSA on discrete optimization problems, it has also been adapted to constrained stochastic optimization problems (e.g., see Wang and Spall, 1999; Sadegh, 1997). The research on this aspect showed SPSA converges almost surely for explicit constraints using projection. More difficult constraints can be handled by penalty function, transforming the problem into one that is unconstrained or mildly constrained. Pflug (1981) showed that stochastic approximation using penalty functions converges almost surely, and Wang and Spall (1999) have extended this result to SPSA. Hutchison and Hill (2001) applied SPSA on constrained airplane delay optimization problems.

Recently, several researches have focused on improving the performance of SPSA algorithm. For example, Spall (1997) proposed second-order SPSA algorithm to speed up convergence, which requires five loss function evaluation at each iteration. Hutchison (2002) studied efficient distribution of perturbation for simulation optimization using SPSA. Kocsis et al. (2003) proposed RSPSA (resilient SPSA) to enhance parameter optimization in games, which is a combination of SPSA and RPROP (Riedmiller and Braun, 1993). Bhatnagar et al. (2003) proposed two-timescale simultaneous perturbation stochastic approximation using deterministic perturbation sequences. Bhatnagar and Borkar (2003) also proposed multiscale chaotic SPSA and smoothed functional algorithms for simulation optimization. Bhatnagar (2005) proposed adaptive multivariate three-timescale stochastic approximation algorithm for simulation based optimization.

Despite abundant research on SPSA algorithms, wan et al. (2005) has found that SPSA algorithm is noisy in supply chain management. On some problems, it converges.

However, on others, it does not. Therefore, to improve the performance of SPSA, more research is needed to be explored.

3.1.1.2 Likelihood Ratios (LR)

In the likelihood ratio method, also called the score function, the gradient of the expected value of an output variable with respect to an input variable is expressed as the expected value of a function of a) input parameters, and b) simulation parameters e.g. simulation run length, output variable value etc. For instance, for a Poisson process with rate λ, if N_T is the number of events in time interval $(0, T)$, and y is an output variable, then

$$\frac{\partial E(y)}{\partial \lambda} = E\left[\left(\frac{N_T}{\lambda} - T\right)\lambda\right] \qquad \textbf{3-6}$$

The right hand side of the equation above can be computed by keeping track of a statistic during a simulation run. Better estimates can be obtained by as a ratio of two expected values – the likelihood ratio. The construction of a likelihood ratio that has desirable computaional and variability characteristics is an important issue in the development of LR gradient estimators (Glynn, 1989b). LR methods are discussed in Glynn (1989a), and Reiman and Weiss (1986).

3.1.1.3 Perturbation Analysis (PA)

In infinitesimal perturbation analysis (IPA) all partial gradients of an objective function are estimated from a single simulation run. The idea is that in a system, if an input variable is perturbed by an infinitesimal amount, the sensitivity of the output variable to the parameter can be estimated by tracing its pattern of propagation. This will be a function of the fraction of propagations that die before having a significant effect on the response of interest. IPA assumes that an infinitesimal perturbation in an input variable does not affect the sequence of events but only makes their occurrence times slide smoothly. The fact that all derivatives can be derived from a single simulation run, represents a significant advantage in terms of computational efficiency. On the other hand, the estimators derived using IPA are often biased and inconsistent. According to Glynn (1989b), when both IPA and LR method apply to a given problem, the IPA gradient estimator is more efficient. Other PA methods include smoothed perturbation analysis (SPA), and IPA variants. In a relatively short time since the introduction of PA to the simulation optimization field, a significant volume work on this topic has been reported (see, e.g., Suri, 1983; Ho and Cao, 1991; Kapuscinski and Tayur, 1999). For a comparative study of finite difference, LR and IPA, see L'Ecuyer (1991).

3.1.1.4 Frequency Domain Method (FDM)

A frequency domain experiment is one in which selected input parameters are oscillated sinusoidally at different frequencies during one long simulation run. The output variable values are subject to spectral (Fourier) analysis, i.e. regressed against sinusoid at the input driving frequencies (Morrice and Schruben, 1989). If the output variable is sensitive to an input parameter, the sinusoidal oscillation of that parameter should induce corresponding (amplified) oscillations in the response.

Table 3-1: Stochastic Approximation Methods (Fu, 2005)

Approach	#Simulations	Key features	Disadvantages
IPA	1	High efficient, easy to implement	Limited applicability
PA	>1	Model-specific implementations	Difficult to apply
LR	1	Requires only model input distributions	Possibly high variance
FD	$2p$	Widely applicable, model-free	Large #simulations
SD	$p+1$	Widely applicable, model-free	Nosier, biased
SPSA	2	Widely applicable, model-free	Nosier, biased

Frequency domain experiments involve addressing three questions: how does one determine the unit of the experimental or oscillation index, how does one select the driving frequencies, and how does one set the oscillation amplitudes? These questions have been addressed in Jacobson et al. (1988), and Jacobson (1989). Frequency domain methodology was first introduced as a screening tool for continuous input factors in discrete-event simulations in Schruben and Cogliaon (1987). Jacobson and Schruben (1988) extended the approach to gradient direction estimation.

Finally, Table 3-1 gives a brief summary of the main approaches in the estimating the gradient for stochastic approximation algorithms (Fu, 2005).

3.1.2 Response Surface Methodology (RSM)

The goal of RSM is to obtain an approximate functional relationship between the input variables and the output (response) objective function. When this is done on the entire (global) domain of interest, the result is often called a metamodel. This metamodel can be obtained in various ways, and two of the most common ways are using regression and neural networks. Once a metamodel is obtained, in principle, appropriate deterministic optimization procedures can be applied to obtain an estimate of the optimum. However, in general, optimization is usually not the primary purpose for constructing a metamodel and, in practice, when optimization is the focus, some form of sequential RSM is used (Kleijnen, 1998). In practice mtamodel is got in a more localized way. The "local response surface" is used to determine a search strategy (e.g., moving to the estimated gradient direction) and the process is repeated. In other words, the metamodels do not attempt to characterize the objective function in the entire solution space but rather concentrate in the local area that the search is currently exploring.

The following outlines a simple two-stage version of sequential RSM using regression. Phase I involves an iterative gradient search procedure by which a set of points around the current point are simulated (e.g., a 2^p factorial design, where p is the dimension of the input vector), and a linear regression is performed to characterize the response surface around the current iterate. A line search is carried out in the direction of steepest descent to determine the next point in the iteration. This process is repeated until linear fit is deemed inadequate, which signals the end of Phase I. In Phase II, additional points are simulated in the surrounding region of the current point, and a higher order (usually) quadratic regression is carried out to estimate the optimum from the resulting fit.

3.1.3 Statistical Methods

Statistical global optimization algorithms employ a statistical model of the objective function to bias the selection of new sample points. These methods are justified with Bayesian arguments that suppose that the particular objective function that is being optimized comes from a class of functions that is modeled by a particular stochastic function. Information from previous samples of the objective function can be used to estimate parameters of the stochastic function, and this refined model can subsequently be used to bias the selection of points in the search domain.

This framework is designed to cover average conditions of optimization. One of the challenges of using statistical methods is the verification that the statistical model is appropriate for the class of problems to which they are applied. Additionally, it has proved difficult to devise computationally interesting versions of these algorithms for high dimensional optimization problems.

3.1.3.1 Importance Sampling Methods

Importance sampling (IS) is a variance reduction technique that can be used in the Monte Carlo method. IS has been used effectively to achieve significant speed ups in simulations involving rare events, such as failure in a reliable computer system or ATM communication network (Shahabuddin, 1995). The idea behind IS is that certain values of the input random variables in a simulation have more impact on the parameter being estimated than others. If these "important" values are emphasized by sampling more frequently, then the estimator variance can be reduced. Hence, the basic methodology in

IS is to choose a distribution which "encourages" the important values. This use of "biased" distributions will result in a biased estimator if it is applied directly in the simulation. However, the simulation outputs are weighted to correct for the use of the biased distribution, and this ensures that the new IS estimator is unbiased. The weight is given by the likelihood ratio, that is, the Radon-Nikodym derivative of the true underlying distribution with respect to the biased simulation distribution.

The fundamental issue in implementing IS simulation is the choice of the biased distribution which encourages the important regions of the input variables. Choosing or designing a good biased distribution is the "art" of IS. The rewards for a good distribution can be huge run-time savings; the penalty for a bad distribution can be longer run times than for a general Monte Carlo simulation without any special techniques.

3.1.3.2 Ranking and Selection

Ranking and Selection (R&S) is usually used to solve small discrete optimization problems. It focuses on selecting the optimal input parameter values θ^* over a finite set $\Theta = \{\theta^1, \quad \theta^2, \quad \mathrm{K} \quad ,\theta^k\}, k < +\infty$, where k is very small(i.e., 2 to 20). By applying a two-stage procedure (Dudewicz and Dalal, 1975; Rinott, 1978), the output from simulation runs at the k input parameter values can be used to determine the most likely input parameters that minimize $F(\theta)$ (objective function). By defining differences in $F(\theta)$ that are less than $\delta > 0$ to be insignificant, we can assure the probability of making the correct selection P^* by choosing the length of our simulation run carefully. In general, as P^* approaches 1, or alternatively, as δ approaches 0, the length of the k simulation runs must approach infinity. Procedures of this type are referred to as indifferent zone ranking and selection(R&S) procedures. To apply these procedures, the simulation runs must be independently seeded to ensure that the simulation outputs from each run are independent. Koenig and Law (1985) extend the indifference zone approach for use as a screening procedure. And they presented a method for selecting procedure. They present a method for selecting a subset of size m of the k systems so that with probability at least P^*, the selected subset will contain the best system. Morrice et al. (1998, 1999) proposes an indifference zone R&S procedure that allows multiple performance measures through the use of a multiple attribute utility function.

Like R&S, multiple comparison procedures (MCPs) attempt to identify the optimal input from a finite set. MCPs approaches the optimization problem as statistical inference problem and, unlike R&S, does not guarantee a decision. Three main classes of MCPs are used in practice: all pairwise multiple comparisons (MCA) (Tukey, 1953), multiple comparisons with the best (MCB) (Hsu, 1984; Hsu and Nelson, 1988), and multiple comparisons with a control (MCC) (Dunnett, 1995; Bofinger and Lewis, 1992; Damerji and Nakayama, 1999). The most popular approach among these is MCB.

When the set of possible input parameter values is discrete, but very large, ordinal optimization approach (Ho et al., 1992, 2000) should be adopted. This method focuses on finding good solutions, rather than trying to find the very best solution (i.e., goal softening). In doing this, it reduces the search for an optimal solution from sampling over a very large set of solutions to sampling over a smaller, more manageable set of good solutions. It should be pointed out that ordinal comparison approach is exponential convergent to optimal (Dai, 1995; Xie, 1997).

3.1.3.3 Sample Path Optimization

Sample path optimization (SPO) (Gürkan et al., 1994; Ferris et al., 2000) is a method application that attempts to exploit the powerful machinery of existing deterministic optimization methods for continuous variable problems. The framework is as follows: $\omega = (\omega_1, \omega_2, \text{K} \ ,\omega_n ,\text{K})$ as the set of all possible sample paths for $L(\theta, \omega_i)$. Define the sample mean over the first n sample paths: $\overline{L}_n(\theta) = \frac{L(\theta, \omega_i)}{n}$. Once $\omega = (\omega_1, \omega_2, \text{K} \ ,\omega_n ,\text{K})$ has been generated, the approximate objective function $\overline{L}_n(\theta)$ is a deterministic function of the parameter θ. Therefore, we can approximate the original simulation optimization problem with the deterministic optimization problem $\min_{\theta \in \Theta} \overline{L}_n(\theta)$, now a standard mathematical programming algorithm can be applied to solve the approximate deterministic optimization problem (Andradóttir, 1998a).

3.1.3.4 Ordinal Optimization (OO)

Traditional optimization approaches focus on iteratively searching the design universe and converging to the best one. However, these approaches can be time consuming. Even the simulation of a single design can be expensive because accurately estimating performance measures usually requires long simulation. Thus, finding the best design is often infeasible for large discrete-event systems.

Instead of insisting on picking the best design, Ordinal Optimization (Ho et al. 1992, 2000) concentrates on finding good, or better, designs and reduces the required simulation time dramatically. The key idea behind ordinal optimization is that it is much easier to approximately sort out relative order than to precisely estimate (absolute) value. Monte Carlo estimate is limited by the canonical $1/\sqrt{n}$ convergence rate (where n is the length of simulation runs), whereas the probability of correctly selecting the best among a set of alternative often exhibit convergence rates that are asymptotically exponential ($1-e^{-\Lambda n}$, for some constant Λ). Additional significant computational savings can be achieved by goal softening instead. Ordinal Optimization has been applied on a 10-node network, where it is shown that we can isolate a good design with high probability with relatively short simulations, instead of long simulations. They also demonstrate many orders of magnitude of speedup.

A crucial issue to apply Ordinal Optimization is the ability of knowing when we are confident or satisfied enough with the ordinal results, e.g., a good subset has been determined with high probability. To fully utilize the advantages of ordinal Optimization, in this subject, some research (see, e.g., Chen et al. 2000a and 2000b) developing effective approaches to quantify the simulation confidence level, particularly for large discrete-event systems.

3.1.4 Heuristic Methods

The following heuristic methods represent the latest developments in the field of direct search methods, and only function evaluations are needed (Carson and Maria, 1997). Many of these techniques balance exploration with exploitation thereby resulting in efficient global search strategies.

3.1.4.1 Random Search Method

Random search method moves through the solution space by randomly selecting a point from the neighborhood of the current point (see, e.g., Pitsoulis and Resende, 2001; Resende and Ribeiro, 2004). This implies that a neighborhood must be defined as part of developing a random search algorithm. The advantage of random search methods is their generality and the existence of theoretical convergence proofs. They have been primarily applied to discrete problems, although, in principle, they could be applied to continuous optimization problems, as well. Its appeal is based on the existence of theoretical convergence proofs. Unfortunately, these theoretical results mean little in practice where it is more important to find high quality solutions within a reasonable length of time than to guarantee convergence to the optimum in an infinite number of steps.

Differences in random algorithms manifest themselves in tow main fashions: (a) how the next point is chose; and (b) what is the estimate for the optimal design. For (b), the choice is usually between taking the current design point versus choosing the one that has been visited the most often.

Let $N(\theta)$ denote the neighborhood set of $\theta \in \Theta$. One version of random search that gives the general flavor is the following:

Step 1: Initialize:

Select initial point θ_*.

Set $n_{\theta_*} = 1$ and $n_\theta = 0 \ \forall \theta_* \neq \theta$.

Step 2: Iterate:

Select another $\theta_i \in N(\theta_*)$ according to some pre-specified probability distribution.

Perform simulations to obtain estimates $J(\theta_*)$ and $J(\theta_i)$.

Increase counter for point with best estimate and update current point: (**1** denotes indictor function)

$n_{\theta_*} = n_{\theta_*} + 1\{J(\theta_*) \le J(\theta_i)\}$;

$n_{\theta_i} = n_{\theta_i} + 1\{J(\theta_i) \le J(\theta_*)\}$;

Step 3: Final answer:

When stopping rule satisfied, return

$$\theta^* = \arg \max_{\theta \in \Theta} n_\theta$$

3.1.4.2 Evolutionary Algorithms (EAs)

Evolutionary algorithms are stochastic search methods that mimic the metaphor of natural biological evolution. Evolutionary algorithms operate on a population of potential solutions applying the principle of survival of the fittest to produce better and better approximations to a solution. At each generation, a new set of approximations is created by the process of selecting individuals according to their level of fitness in the problem domain and breeding them together using operators borrowed from natural genetics. This process leads to the evolution of populations of individuals that are better suited to their environment than the individuals that they were created from, just as in natural adaptation.

Evolutionary algorithms model natural processes, such as selection, recombination, mutation, migration, locality and neighborhood. Figure 2-2 shows the structure of a simple evolutionary algorithm. Evolutionary algorithms work on populations of individuals instead of single solutions. In this way the search is performed in a parallel manner.

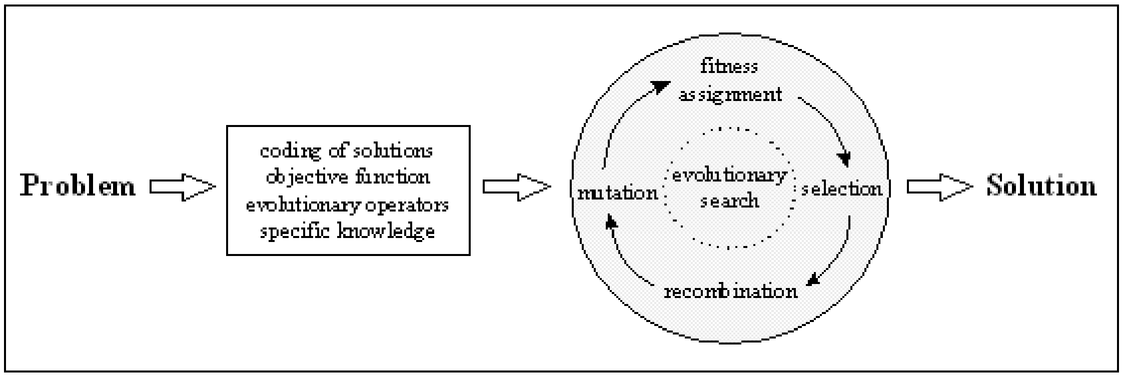


Figure **3-2**: Problem solution using evolutionary algorithms

At the beginning of the computation a number of individuals (the population) are randomly initialized. The objective function is then evaluated for these individuals. The first/initial generation is produced.

If the optimization criteria are not met the creation of a new generation starts. Individuals are selected according to their fitness for the production of offspring. Parents are recombined to produce offspring. All offspring will be mutated with a certain probability. The fitness of the offspring is then computed. The offspring are inserted into the population replacing the parents, producing a new generation. This cycle is performed until the optimization criteria are reached.

Such a single population evolutionary algorithm is powerful and performs well on a wide variety of problems. However, better results can be obtained by introducing multiple subpopulations. Every subpopulation evolves over a few generations isolated (like the single population evolutionary algorithm) before one or more individuals are exchanged between the subpopulation. The multi-population evolutionary algorithm models the evolution of a species in a way more similar to nature than the single population evolutionary algorithm.

From the above discussion, it can be seen that evolutionary algorithms differ substantially from more traditional search and optimization methods. The most significant differences are:

- Evolutionary algorithms search a population of points in parallel, not just a single point.
- Evolutionary algorithms do not require derivative information or other auxiliary knowledge; only the objective function and corresponding fitness levels influence the directions of search.
- Evolutionary algorithms use probabilistic transition rules, not deterministic ones.
- Evolutionary algorithms are generally more straightforward to apply, because no restrictions for the definition of the objective function exist.

- Evolutionary algorithms can provide a number of potential solutions to a given problem. The final choice is left to the user. (Thus, in cases where the particular problem does not have one individual solution, for example a family of pareto-optimal solutions, as in the case of multi-objective optimization and scheduling problems, then the evolutionary algorithm is potentially useful for identifying these alternative solutions simultaneously.)

Different main schools of evolutionary algorithms have evolved during the last 40 years: genetic algorithms mainly developed in the USA by J. H. Holland (1975), evolutionary strategies developed in Germany by I. Rechenberg (1973) and H.-P. Schwefel (1981), and evolutionary programming proposed by Fogel et al. (1966). Each of these constitutes a different approach; however, they are inspired by the same principles of natural evolution. A good introductory survey can be found in Fogel (1994).

Specific examples of EAs are given below. Most of these techniques are similar in spirit, but differ in the details of their implementation and the nature of the particular problem to which they have been applied.

- **Genetic algorithm** - This is the most popular type of EA (see, e.g., Holland 1992, Goldberg 1989). One seeks the solution of a problem in the form of strings of numbers (traditionally binary, although the best representations are usually those that reflect something about the problem being solved - these are not normally binary), virtually always applying recombination operators in addition to selection and mutation;
- **Evolutionary programming** - Like genetic programming, only the structure of the program is fixed and its numerical parameters are allowed to evolve;

- **Evolution strategy** - Works with vectors of real numbers as representations of solutions, and typically uses self-adaptive mutation rates (see Schwefel 1995, Maria 1995 for more details);
- **Genetic programming** - Here the solutions are in the form of computer programs, and their fitness is determined by their ability to solve a computational problem.
- **Learning classifier system** - Instead of a using fitness function, rule utility is decided by a reinforcement learning technique.

Because they do not make any assumption about the underlying fitness landscape, it is generally believed that evolutionary algorithms perform consistently well across all types of problems (see, however, the no-free-lunch theorem). This is evidenced by their success in fields as diverse as engineering, art, biology, economics, genetics, operations research, robotics, social sciences, physics, chemistry, and others.

Apart from their use as mathematical optimizers, evolutionary computation and algorithms have also been used as an experimental framework within which to validate theories about biological evolution and natural selection, particularly through work in the field of artificial life. Techniques from evolutionary algorithms applied to the modeling of biological evolution are generally limited to explorations of micro-evolutionary processes, however some computer simulations, such as Tierra and Avida, attempt to model macro-evolutionary dynamics.

A limitation of evolutionary algorithms is their lack of a clear genotype-phenotype distinction. In nature, the fertilized egg cell undergoes a complex process known as embryogenesis to become a mature phenotype. This indirect encoding is believed to make the genetic search more robust (i.e. reduce the probability of fatal mutations), and also may improve the evolvability of the organism. Recent work in the field of artificial embryogeny, or artificial developmental systems, seeks to address these concerns.

3.1.4.3 Simulated Annealing

Simulated annealing is a generalization of a Monte Carlo method for examining the equations of state and frozen states of n-body systems (Metropolis et al. 1953). The concept is based on the manner in which liquids freeze or metals recrystalize in the process of annealing. In an annealing process a melt, initially at high temperature and disordered, is slowly cooled so that the system at any time is approximately in thermodynamic equilibrium. As cooling proceeds, the system becomes more ordered and approaches a "frozen" ground state at T=0. Hence the process can be thought of as an adiabatic approach to the lowest energy state. If the initial temperature of the system is too low or cooling is done insufficiently slowly the system may become quenched forming defects or freezing out in metastable states (ie. trapped in a local minimum energy state).

The original Metropolis scheme was that an initial state of a thermodynamic system was chosen at energy E and temperature T, holding T constant the initial configuration is perturbed and the change in energy dE is computed. If the change in energy is negative the new configuration is accepted. If the change in energy is positive it is accepted with a probability given by the Boltzmann factor $e^{-\frac{dE}{T}}$. This processes is then repeated sufficient times to give good sampling statistics for the current temperature, and then the temperature is decremented and the entire process repeated until a frozen state is achieved at T=0.

By analogy the generalization of this Monte Carlo approach to combinatorial problems is straight forward (Kirkpatrick et al., 1983; Cerny, 1985). The current state of the thermodynamic system is analogous to the current solution to the combinatorial problem, the energy equation for the thermodynamic system is analogous to at the objective function, and ground state is analogous to the global minimum. The major difficulty (art) in implementation of the algorithm is that there is no obvious analogy for the temperature T with respect to a free parameter in the combinatorial problem. Furthermore, avoidance of entrainment in local minima (quenching) is dependent on the "annealing schedule", the choice of initial temperature, how many iterations are performed at each temperature, and how much the temperature is decremented at each step as cooling proceeds.

Simulated annealing has been used in various combinatorial optimization problems and has been particularly successful in circuit design problems (see Kirkpatrick et al., 1983).

3.1.4.4 Tabu Search

Tabu search is a mathematical optimization method, belonging to the class of local search techniques. Tabu search enhances the performance of a local search method by using memory structures. Tabu search is generally attributed to Fred Glover (1989, 1990).

Tabu search uses a local or neighborhood search procedure to iteratively move from a solution x to a solution x' in the neighborhood of x, until some stopping criterion has been satisfied. To explore regions of the search space that would be left unexplored by the local search procedure and—by doing this—escape local optimality, tabu search modifies the neighborhood structure of each solution as the search progresses. The solutions admitted to N * (x), the new neighborhood, are determined through the use of special memory structures. The search now progresses by iteratively moving from a solution x to a solution x' in N * (x).

Perhaps the most important type of short-term memory to determine the solutions in N * (x), is the use of a tabu list. In its simplest form, a tabu list contains the solutions that have been visited in the recent past (less than n moves ago, where n is the tabu tenure). Solutions in the tabu list are excluded from N * (x). Other tabu list structures prohibit solutions that have certain attributes (e.g. traveling salesman problem (TSP) solutions that include certain arcs) or prevent certain moves (e.g. an arc that was added to a TSP tour cannot be removed in the next n moves). Selected attributes in solutions recently visited are labeled tabu-active. Solutions that contain tabu-active elements are tabu. This type of short-term memory is also called recency-based memory.

Tabu lists containing attributes are much more effective, although they raise a new problem. With forbidding an attribute as tabu, typically more than one solution is declared as tabu. Some of these solutions that must now be avoided might be of excellent quality and have not yet been visited. To overcome this problem, aspiration criteria are introduced which allow overriding the tabu state of a solution and thus include it in the allowed set. A commonly used aspiration criterion is to allow solutions which are better than the currently best known solution.

3.1.5 A-Team

It should be mentioned A-Team (asynchronous team) is a process that involves combining various problem solving strategies so that they can interact synergistically. De Souza and Talukdar (1991) viewed an A-team as a process that is both fast and robust. They have demonstrated that A-teams consisting of GA and conventional algorithms, such as Newton's Method and Levenberg-Marquardt algorithms, for solving sets of nonlinear algebraic equations, result in considerable savings in the amount of computational effort (number of function evaluations) necessary for finding solutions. A-team is inherently suitable for multi-criteria simulation optimization problems, and therefore, represents one of the fastest growing areas of simulation optimization research. For optimizing a kanban sizing problem, Hall and Bowden (1996) utilized a two-phase approach- Evolutionary Strategies (ES) followed by Hooke-Jeeves search method--to obtain "good" solutions with 60% fewer simulation runs than with ES alone.

3.1.6 Optimization for Simulation Software

While the above five approaches account for most of the literature in simulation optimization, they have not been used to develop optimization for simulation software. Table 3-2 lists the software package for optimization over simulation.

Table 3-2: Commercial Implementations of Simulation Optimization (April et al. 2003)

Optimization Package	Technology	Simulation Software
OptQuest	Scatter Search	AnyLogic Arena Crystal Ball CSIM19 Enterprise Dynamics Micro Saint ProModel Quest SimFlex SIMPROCESS SIMUL8 TERAS
Evolutionary Optimizer	Genetic Algorithms	Extend
Evolver	Genetic Algorithms	@Risk
AutoStat	Evolutionary Strategies	AutoMod

Fu (2002) has pointed out that the current commercial software is a good start, but fails to exploit the research in simulation optimization, from which there are many useful results that have potential to dramatically improve the efficiency of the process.

3.2 Supply Chain Management

Supply chain management (SCM) is the process of planning, implementing, and controlling the operations of the supply chain with the purpose to satisfy customer requirements as efficiently as possible. Supply chain management spans all movement and storage of raw materials, work-in-process inventory, and finished goods from point-of-origin to point-of-consumption.

To manage a supply chain well, the following problems must be addressed:

- Distribution Network Configuration: Number and location of suppliers, production facilities, distribution centers, warehouses and customers.
- Distribution Strategy: Centralized versus decentralized, direct shipment, cross docking, pull or push strategies, third party logistics.
- Information: Integrate systems and processes through the supply chain to share valuable information, including demand signals, forecasts, inventory and transportation.
- Inventory Management: Quantity and location of inventory including raw materials, work-in-process and finished goods.

This research will focus on inventory management. Therefore, in the next, related knowledge about inventory management and related research will be introduced.

3.2.1 Supply Chain Inventory Replenishment Policy

Before we introduce the inventory problems, we should give a short introduction about replenishment policy, which consists of decisions regarding when to order and how much to reorder. These decisions determine replenishment frequency and safety inventories along with the service level. There are several forms of replenishment policies, and they are listed below:

a) **Basic stock continuous review policy**: When the inventory position (i.e., on-hand plus on-order minus backorder) at falls below some specific base-stock level, R , a replenishment order is placed.

b) **(r, Q) continuous review policy**: When the inventory drops to r, the manufacture places order of size Q to supplier.

c) **(s, S) continuous review policy**: When the inventory drops to s, the manufacture places order of size up to S to supplier.

d) **(R, s, Q, c) periodic review policy:** Review the inventory level every R units of time, if the inventory is less than or equal to s you must order Q, if the inventory is less than or equal to c you can order($Q - c$).

Once the enterprise decides their replenishment policy, the corresponding decision variables will be decided, too. If the demand is stationary, the decision variable of inventory level will not change through the whole simulation.

3.2.2 Literature Review of SCM

A good review of new trends of research in SCM has been reported by Stadtler and Kilger (2005). Some studies indicated that buyer-supplier relationships are becoming more dependent on factors like quality, delivery performance, and flexibility in contract commitment to work together, as opposed to traditional relationships based on cost. Electronic Data Interchange (EDI) and Distributed Database have been identified as important technological advancements that may benefit supply chain performance in a significant manner (Srinivasan et al. 1994). While providing general guidelines and identifying elements of best practice, the benchmarking approach has been of limited help to managers who are looking for specific quantitative solutions to everyday problems.

On the analytical front, research on multi-echelon inventory problems has a long history (Clark and Scarf, 1958; Svoronous and Zipkin, 1991). A multi-echelon systems is one in which there are multiple tiers in the supply chain. This line of work typically assumes centralized control of the supply network, thus overlooking the possibility of decentralized decision making. More recent supply chain models in this area also include Cohen and Lee (1988), Cohen and Moon (1990), and Newhart et al. (1993), Graves et al. (2000).

In the past decades, abundant research has been done for the supply chain inventory optimization problems. For example, Ettl et al. (2000) developed an analytical method for a supply chain model that takes as input the bill of materials, the (nominal) lead times, the demand and cost data, and the required customer service levels, so as to generate base-stock level for each stocking location, in order to minimize the overall inventory capital throughout the network and to guarantee customer service requirement. They used conjugate gradient routine to search for the optimal solution. Agrell (1995) proposed a multi-criteria framework for inventory control for a single stage supply chain where three objectives are targeted. Graves (1998) developed an adaptive base-stock policy for a single-item inventory system, where the demand process is non-stationary. Arslan et al. (2005) developed a heuristics for a single-product inventory model for multiple demand classes. Cakanyidirim et al. (2005) considered the case of (r, Q) policy with lead time options. Nagurney et al. (2005) studied supply chain networks, electronic commerce, and supply side and demand side risk. Daniel (2005a) et al. proposed a hybrid genetic algorithm to determine base stock levels in a serial supply chain with a single objective and with multiple objectives, where minimizing total holding cost (THC) and minimizing total shortage cost (TSC) are targeted. Daniel (2005b) et al. also proposed Simulated Annealing (SA) for deciding base-stock levels in the supply chain. The above work has contributed in a significant manner to managerial decision making. However, these models are limited in handling general supply chain networks like fork-join supply chain.

On the other hand, the use of simulation as a vehicle for understanding issues of organizational decision making has gained considerable attention and momentum in recent years. For instance, Towill et al. (1992) used simulation to evaluate various inventory policies. Fu and Healy (1992) used simulation optimization for (s, S) inventory systems. Lee and Billington (1993) developed a supply chain model operating under a periodic-review base-stock system at Hewlett Packard, and employed a search heuristic to find optimal inventory levels across the supply chain. Glasserman and Tayur (1995) developed simulation-based methods (called infinitesimal perturbation analysis) for estimating sensitivities of inventory costs with respect to policy parameters. Swaminathan et al. (1998) proposed multi-agent based simulation for modeling supply chain dynamics. Rao et al. (2000) developed an integrated model to analyze different supply chain configurations for Caterpillar’s new line of compact construction equipment by using decomposition techniques, network optimization, inventory modeling and simulation. Jung et al. (2004) developed a simulation based optimization approach to

SCM under demand uncertainty. Wan et al. (2005) proposed simulation-based optimization with surrogate models to supply chain management. And they applied their methods to continuous supply chain inventory optimization problems. Therefore, for discrete manufacturing systems, the application of this method is not sure.

Despite this vast of research on supply chain inventory optimization, there is still a lot of work to be done. For example, for discrete supply chain inventory optimization problems, current researches mostly adopt GA and SA as methodologies, which are not computation efficient. In addition, it is beyond the analytical methods' ability to solve inventory optimization problems in a very general (fork-join) network. According to author's knowledge, no research has adopted simulation optimization to solve constrained discrete supply chain optimization problems. Therefore, in this research, an efficient simulation optimizatione method, Augmented Spontaneous Stochastic Approximation (ASPSA), would be proposed to solve discrete general (fork-join) supply chain network inventory optimization problems. Here, the problems can be constrained. In next chapter, ASPSA-I, which targets unconstrained optimization problems, is discussed in detail.

Chapter 4

Research Methodology

This chapter describes a new simulation optimization method - Augmented Spontaneous Perturbation Stochastic Approximation (ASPSA) algorithm over discrete set. The method will be an asynchronous team type heuristics which integrate SPSA algorithm, random pre-search method, ordinal optimization, and line search method together.

Traditionally, for discrete inventory optimization problems, GA and SA are employed to solve the problems. However, these two methodologies are computational expensive. Therefore, the first task of the research is to develop a new efficient simulation optimization method, ASPSA, for discrete unconstrained inventory optimization problems, and then test its performance by published inventory supply chain problems.

In the reality, there always exist constraints. For example, inventory problems are subject to required customer service levels, and call center problems are often of the form of maximizing throughput subject to customer waiting time is more than some limits. Therefore, the second task of this research is building another version of ASPSA for constrained optimization problems.

This chapter is organized as follows: the first section will provide a brief overview of SPSA algorithm. Based on SPSA, a new simulation optimization method-ASPSA, is proposed. After that, convergence rate of ASPSA algorithm will be proved.

4.1 SPSA

SPSA has been intensively studied in recent years because of the following reasons:

- SPSA can solve large complex optimization problems without increasing computation complexity;
- It is easy to implement;
- It needs smaller number of function evaluations than Genetic Algorithm, Tabu Search methods;
- SPSA have been also proved to be applied to discrete optimization problems (Gerencser et al., 1999).

Although in Chapter 2, we have given a short introduction of SPSA. Here, we present SPSA in detail. The followings are notations used in SPSA.

4.1.1 Basic Algorithm

<u>Notation</u>

- k : Index for iteration number.
- $k_{\max}$: The maximum iteration number to run the algorithm.
- θ: Decision variable vectors.
- $J(\theta)$: The objective function.
- $g(\theta)$: The gradient at θ.
- $y(\theta) = J(\theta) + noise$
- c_k : The perturbation magnitude
- a_k : The step size

- $\Delta_k = [\Delta_{k1} \quad \Delta_{k2} \quad \ldots \quad \Delta_{kp}]$: A random perturbation vector at iteration k .

SPSA Algorithms

Step 1. Initialization and coefficient selection

Set counter index $k = 0$.

Randomly choose an initial guess θ_0 .

Set nonnegative a_k and c_k

Step 2. Generate simultaneous perturbation vector Δ_k

Step 3. Loss function evaluation

Evaluate $y(\theta_k + c_k\Delta_k)$ and $y(\theta_k - c_k\Delta_k)$.

Step 4. Gradient approximation

$$\hat{g}_k = \frac{y(\theta_k + c_k\Delta_k) - y(\theta_k - c_k\Delta_k)}{2c_k} \begin{bmatrix} \frac{1}{\Delta_{k1}} \\ \frac{1}{\Delta_{k2}} \\ \vdots \\ \frac{1}{\Delta_{kp}} \end{bmatrix} \qquad 4\text{-}1$$

Step 5. Update θ **estimate**

$\theta_{k+1} = \theta_k - a_k \hat{g}_k$.

Step 6. Iteration or termination

$k \leftarrow k+1$. If stop criteria is satisfied, stop. Otherwise, go to step 2.

Figure **4-1**: SPSA Algorithm (Spall, 2002)

SPSA tries to solve minimization a loss functions $J(\theta)$, where the optimization problem can be translated into finding the minimizing θ^* such that $\frac{\partial J}{\partial \theta} = 0$. This is the classical formulation of (local) optimization for loss functions. It is assumed that measurements of $J(\theta)$ are available at various values of θ. The SPSA processes in general recursive form: $\hat{\theta}_{k+1} = \hat{\theta}_k - a_k \hat{g}_k(\hat{\theta}_k)$. Its detail procedures can be stated as in Figure 4-1.

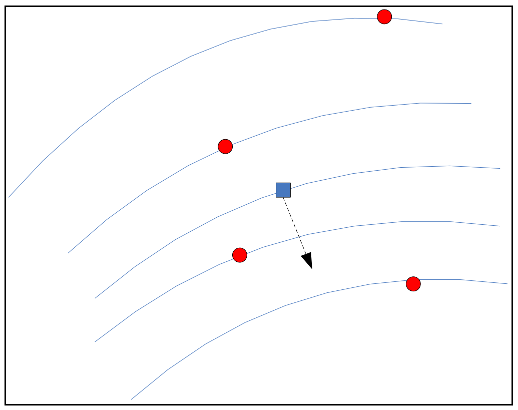
It should be highlighted that usually Δ_k is achieved by Bernoulli distribution. That is Δ_{ki} can be ± 1 with 50% probability respectively. For continuous optimization problems, a_k and c_k usually are very small, and are less than 1. From the above description of SPSA, we can see that SPSA is simple and easy to implement, and the gradient is free of system size, which is great for large systems.

4.1.2 Relationship of Gradient Estimate to True Gradient

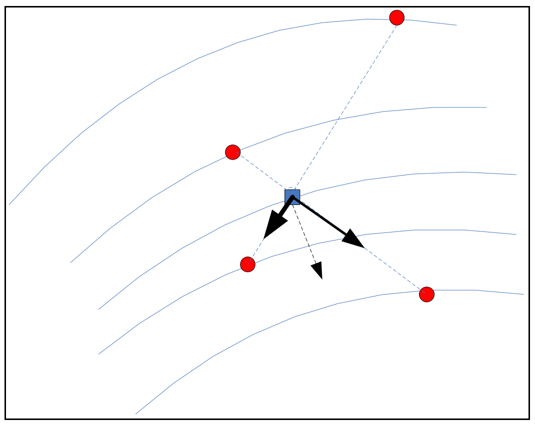
Spall (2002) found the relationship between the estimated gradient and true gradient. The informal rationale for the strange-looking gradient approximation in Eq. 4-1 is quite simple. Consider the m^{th} element of the approximation. Let us sketch how this element is an "almost unbiased" estimator of the m^{th} element of the true gradient. Suppose that the measurement noise $\varepsilon(\theta)$ has mean zero and that J is several times differentiable at $\theta = \theta_k$. Then, using a simple first-order Taylor expansion (Eq. 4-2)

$$
\begin{aligned}
E[\hat{g}_{km}(\theta_k) \mid \theta_k] &= E[\frac{y(\theta_k + c_k\Delta_k) - y(\theta_k - c_k\Delta_k)}{2c_k\Delta_{km}} \mid \theta_k] \\
&= E[\frac{J(\theta_k + c_k\Delta_k) - J(\theta_k - c_k\Delta_k)}{2c_k\Delta_{km}} \mid \theta_k] \\
&= E[\frac{J(\theta_k) + c_k g_k(\theta_k)^T\Delta_k - [J(\theta_k) - c_k g(\theta_k)^T\Delta_k]}{2c_k\Delta_{km}} \mid \theta_k] \\
&= E[\frac{2c_k g(\theta_k)^T\Delta_k}{2c_k\Delta_{km}} \mid \theta_k] \\
&= g_{km}(\theta_k) + \sum_{i\neq m} g_{ki}(\theta_k) E[\frac{\Delta_{ki}}{\Delta_{km}}]
\end{aligned}
\qquad 4\text{-}2
$$

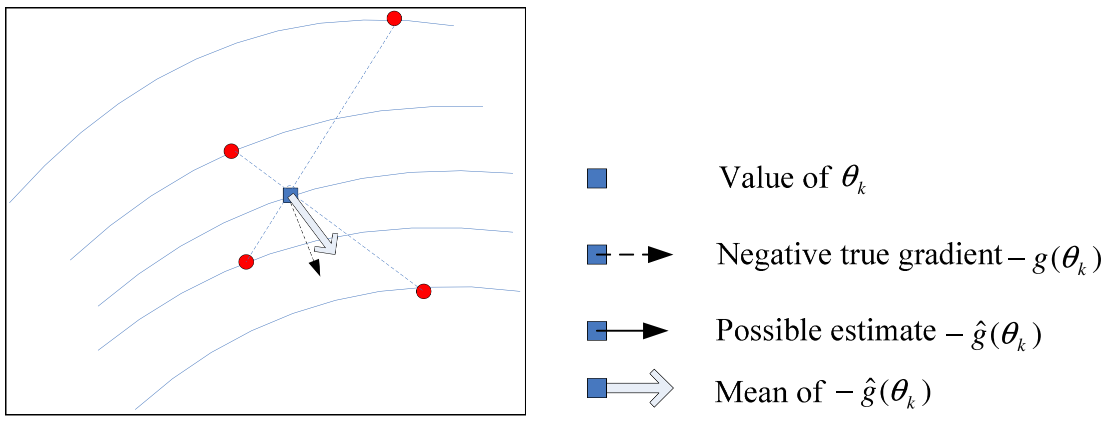
Assume that Δ_{ki} has mean zero and is independent of Δ_{km} for $i \neq m$, $E[\frac{\Delta_{ki}}{\Delta_{km}}] = 0$ for all $i \neq m$. Then Eq. 4-2 shows us that $E[\hat{g}_{km}(\theta_k)] \approx g_{km}(\theta_k)$. That is, approximate gradient is convergent to true gradient (for example, see Figure 4-2).



(a) $-g(\theta_k)$ **and four possible sample points around** θ_k



(b) Two possible search direction and magnitude given by $-\hat{g}(\theta_k)$


Value of θ_k
Negative true gradient $-g(\theta_k)$
Possible estimate $-\hat{g}(\theta_k)$
Mean of $-\hat{g}(\theta_k)$

(c) Mean of the two possible $-\hat{g}(\theta_k)$

Figure **4-2:** Comparisons of true gradient and SPSA estimated gradient in a low-noise setting with $p = 2$ (Spall, 2002)

4.2 Augmented SPSA (ASPSA-I) For Unconstrained Optimization Problems

To improve the performance of SPSA, ASPSA is proposed in this section. Five functions are integrated into SPSA to improve its performance. And they are:

1) Ordinal Optimization;
2) Pre-search using random search method;
3) Uncommon step size a_{ki} for decision variable $\theta_{ki}, i = 1,\ 2,\ \mathrm{K}\ \ p$;
4) Long term memory of best solution;
5) Line Search.

Next, we will in detail explain the functions of the above functions.

4.2.1 Ordinal Optimization

While simulation is the only general performance tool of choice, it is not satisfactory for many of these problems. The main reason is the time consuming runs, especially when parametric search for good designs are involved with a combinatorial large search space. There are fundamental limitations to improve the simulation speed due to fact that confidence interval of performance estimate decreases at best at the rate of $1/\sqrt{n}$ where n denotes the length of simulation.

Ordinal optimization is an attempt to circumvent these difficulties at least for the initial stages of optimization search. It rests on two ideas: (i) Order is much easier to determine than value --- It is easier to determine if $A > B$ than to determine $A - B = ?$ To see this, just consider the case of two numbers A and B observed under i.i.d. normal noise. Assume $A \sim N(u_A, \sigma_A^2)$ and $B \sim N(u_B, \sigma_B^2)$. Let $C = A - B$, then we have C is also following normal distribution ($C \sim N(u_A - u_B, \sigma_A^2 + \sigma_B^2)$). It is easy to see $P(A > B) = P(C > 0) > P(C > ?) = P(A - B > ?)$ (Figure 4-3) (ii) Softening the goal makes hard problem easier -- Instead of asking the best for sure let us settle for the good enough with high probability. Often times several orders of magnitude speed up can be achieved in simulation experiments when we adopt these two viewpoints.

It is now clear as to how OO can help in speed up simulation for performance evaluation. If we insist on using performance value to weed out poor designs, the $1/\sqrt{n}$ decrease of the confidence interval (CFI) of the performance value estimate will often not be fast enough. Instead we should simply use performance order to separate the good enough from the bad. This can be done with much shorter simulation and large estimation noise. The twin effect of softening the goal and distinguish order rather than value can introduce several orders of magnitude improvement in the computation burden. Therefore, we can get Property 1.

Property 1: Ordering needs shorter simulation length than valuing.

Assume that we usually use full simulation length (n) in common situation to get value (absolute difference between two numbers), using Property 1, we know that we only need $n_s (n_s < n)$ for the ordering. Thus, we can greatly improve searching speed in simulation optimization. To show this point, we will use a good example for gradient calculation in ASPSA-I. The gradient in SPSA can be expressed as Eq. 4-3. If dividing $| \frac{(y(\theta_k + c_k \Delta_k) - y(\theta_k - c_k \Delta_k))}{2c_k} |$ for both left and right sides of Eq. 4-3, we can get the revised gradient (Eq. 4-4). From Eq. 4-4, we can see that we only care about the order between $y(\theta + c_k\Delta_k)$ and $y(\theta - c_k\Delta_k)$, instead of how much $y(\theta + c_k\Delta_k) - y(\theta - c_k\Delta_k)$. That means we can run shorter simulation length (n_s) to sort order between $y(\theta + c_k\Delta_k)$ and $y(\theta - c_k\Delta_k)$ in ASPSA-I, instead of running full simulation length (n) like SPSA does. Thus, we can save simulation resource and speed up search process. In addition, using revised gradient $\hat{g}_k$, it makes step size a_k easier to set in ASPSA-I than that in SPSA (Eq. 4-4) because the range of $\hat{g}_k$, which is related to perturbation vector Δ_k (known in advanced), is more predictive than that of g_k, which is related to objective y_k (unknown in advance).

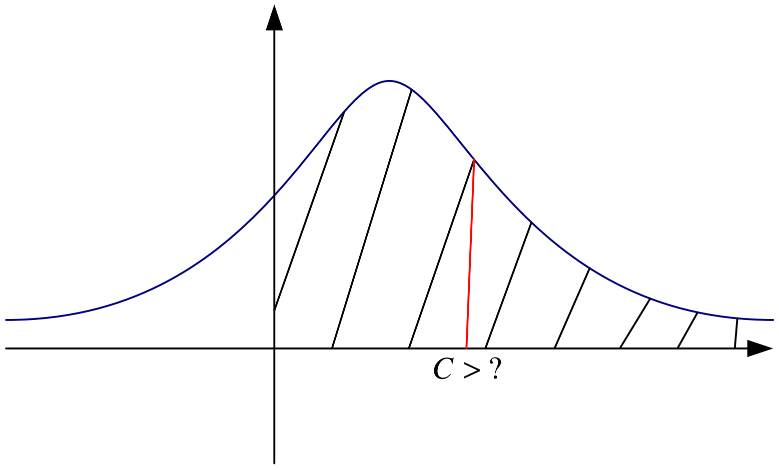


Figure **4-3:** Ranking is easier than valuing

$$g_k(\theta_k) = \frac{(y(\theta_k + c_k\Delta_k) - y(\theta_k - c_k\Delta_k))}{2c_k}\begin{bmatrix} \frac{1}{\Delta_{k1}} \\ \mathrm{M} \\ \frac{1}{\Delta_{kp}} \end{bmatrix} \qquad 4\text{-}3$$

$$\hat{g}_k(\theta_k) = \begin{cases} 1/\Delta_k & \mathit{if} \quad y(\theta + c_k\Delta_k) > y(\theta - c_k\Delta_k) \\ -1/\Delta_k & \mathit{otherwise} \end{cases} \qquad 4\text{-}4$$

4.2.2 Presearch

It is well known that a good starting point will speed up the convergence rate of gradient based method. Therefore, in ASPSA, we integrate random search to find a better starting point.

Our pre-search method is randomly choosing several decision points from the decision space (Figure 4-4), and then rank them in short simulation runs and chose the best one as the starting point. Thus, we avoid increasing the computation time because of pre-search.

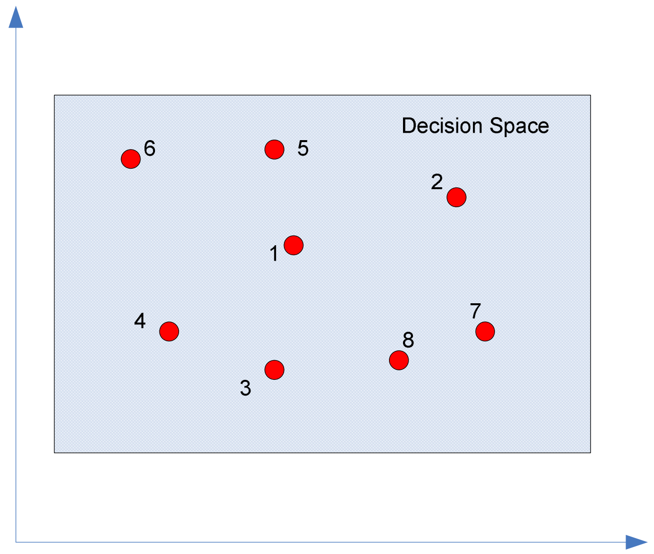


Figure 4-4: Use ranking to select the best solution

4.2.3 Non-uniform Step Size

SPSA has been shown to be noisy and the choice of parameters (a_k and c_k) are critical on its performance (see, e.g., Gerencser et al. 1999, Wan et al. 2005). When function evaluation is expensive, as if often the case in simulation, small sample behavior of the algorithm becomes important. In that case the proper tuning of the parameters becomes critical. To simplify implementation, in our ASPSA-I over discrete set, we simply set $c_k = 1$ for all iterations. Therefore, only one parameter a_k is left to choose.

In practice, the learning rate of a_k and c_k are often kept a common value for calculating gradients for all dimensions. Further, in all previous works on SPSA known to us it was assumed that the perturbations Δ_{ki} ($i = 1, \ 2, \ \ldots \ p$) have the same distribution. When different dimensions have different scales or range then it does not make too much sense to use the same step size a_{ki} ($i = 1, \ 2, \ \ldots \ p$) for all dimensions. Therefore, to be more reasonable, different a_{ki} will be chosen for different dimension according to some random distribution. This randomization is also similar to mutation in GAs, which can prevent ASPSA-I from falling into local extremes.

4.2.4 Line Search

In SPSA, we can see another type of noise is from step size a_k. Improper a_k may lead to worse solution. Therefore, in this part, line search method is used to find best step size on the searching direction $-\hat{g}_k$.

To simplify line searching and avoid over increasing simulation load too much, we simply choose several different step sizes, and then compare their results θ_{k+1} to find best step size (Figure 4-5). Similarly, ordering is enough for finding the best step size. Therefore, still based on Property 1, short simulation length (n_s) is used here to choose best step size to shorten simulation time.

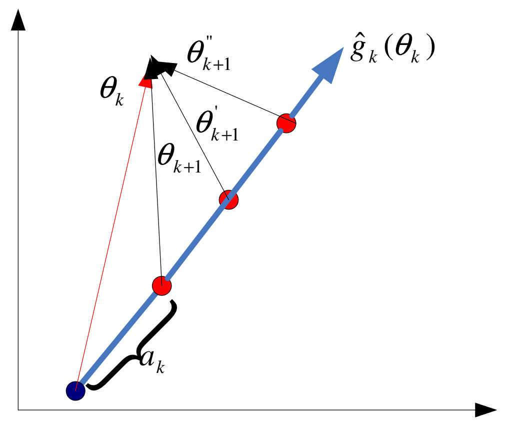


Figure 4-5: Line search method in ASPSA-I

4.2.5 Long Term Memory

One shortcoming of SPSA method is memoryless. Keeping memory of elite solutions can speedup searching process and improve solution quality (see, e.g., Fleurent and Glover, 1999). In ASPSA-I, we will keep memory of best decision point θ_{best} found until currently. If $J(\theta_{k+1}) < J(\theta_{best})$, $\theta_{best} = \theta_{k+1}$. Otherwise, $\theta_{k+1} = \theta_{best} + \Lambda_{k+1}$, where Λ_{k+1} is a perturbation vector following some random distribution.

After explaining the proposed five functions, we will give a detail implementation of ASPSA-I algorithm (Figure 4-6) which targets on unconstrained optimization problems.

ASPSA-I algorithm

```
Step 1. Initialization and coefficient selection
        Set counter index $k = 0$.
        Set nonnegative $c_k$ and distribution for $a_k$
Step 2. Pre-search
        For $j = 1, \quad K \quad m$
            Randomly choose $\theta_j$ from the decision space;
            Run simulation in $n_s$ length to evaluate $J(\theta_j)$.
        Let $J_b = \min\{J_j, j = 1, \quad K, \quad m\}$;
        Run full simulation length to evaluate $J(\theta_b)$;
        Set $\theta_{best} = \theta_k = \theta_b$, $J_{best} = J_b$.
Step 3. Generate simultaneous perturbation vector $\Delta_k$
Step 4. Loss function evaluation
        Evaluate $y(\theta_k + c_k\Delta_k)$ and $y(\theta_k - c_k\Delta_k)$ in short simulation run $n_s$
Step 5. Gradient approximation
        $\hat{g}_k = \begin{cases} 1/\Delta_k & if \quad y(\theta + c_k\Delta_k) > y(\theta - c_k\Delta_k) \\ -1/\Delta_k & otherwise \end{cases}$
Step 6. Line search
      For $j = 1, \quad K \quad l$
            1). Randomly $a_i^j \, (i = 1, \quad K, \quad p)$ from some distribution
            2). Calculate $\theta_{k+1,i}^j = \theta_{k,i} - a_i^j \hat{g}_{ki}$
            3). Run short simulation ($n_s$) length to evaluate $J(\theta^j)$
      Let $J_b = \min\{J_j, j = 1, \quad K, \quad l\}$;
      Run full simulation length to evaluate $J(\theta_b)$;
      If $J_{best} > J_b$, $J_{best} = J_b$ and $\theta_{best} = \theta_{k+1} = \theta_b$;
      Otherwise, $\theta_{k+1} = \theta_{best} + \Lambda_k$ where $\Lambda_k$ is a perturbation vector
Step 7. Iteration or termination
        $k \leftarrow k + 1$. If stop criteria is satisfied, stop. Otherwise, go to step 2.
```

Figure 4-6: ASPSA-I algorithm

4.3 Convergence Proof of ASPSA-I Algorithm

To show ASPSA-I is convergent, we should show $\lim_{k\to\infty}\theta_{best} \to \theta^*$. To prove this, we can use Theorem 2.1 published in Spall (2003, p. 40) which is for proving convergence of random search method. First from the statement of ASPSA algorithm, we can see that it is a special random search algorithm with a specific searching direction $g(\theta)$ (like neighborhood in random search algorithm). And Theorem 2.1 can be stated as followings:

Theorem 2.1: Suppose that θ^* is the unique minimizer of J on the domain Θ in the sense that $J(\theta^*) = \inf_{\theta\in\Theta} J(\theta)$, $J(\theta^*) > -\infty$, and

$$\inf_{\theta\in\Theta, \|\theta-\theta^*\|\geq\eta} J(\theta) > J(\theta^*) \qquad \textbf{4-5}$$

For all $\eta > 0$. Suppose further that for any $\eta > 0$ and for all k, there exists a function $\delta(\eta) > 0$ such as

$$P[\theta_{best} : J(\theta_{best}) < J(\theta^*) + \eta] \geq \delta(\eta) \qquad \textbf{4-6}$$

Then, for random search algorithm with noise-free loss measurements, $\theta_{best} \to \theta^*$ as $k \to \infty$.

Proof: Because $J(\theta^*) > -\infty$ and $J(\theta_{best})$ is monotonically nonincreasing due to having noise-free loss measurements, $\lim_{k\to\infty} J(\theta_{best}(\omega))$ exists for each underlying sample point ω, as in (e.g., Apostol, 1974, p.185). Let $S_\eta = \{\theta : J(\theta) < J(\theta^*) + \eta\}$. By Eq. 4-6, $P(\theta_{best} \in S_\eta) \geq \delta(\eta)$ for any $\eta > 0$. Hence, by the independence of the sampling distribution, $P(\theta_{best} \notin S_{\eta} \forall k) \leq [1-\delta(\eta)]^{k} \to 0$ as $k \to \infty$ .So $\lim_{k\to\infty} J(\theta_{best}) < J(\theta^{*}) + \eta$ in probability. Because $\eta > 0$ is arbitrarily small, we know that $\lim_{k\to\infty} J(\theta_{best}) \to J(\theta^{*})$ in probability as $k \to \infty$. Then by a standard result in probability theory (e.g., Laha and Rohatgi, 1979, Proposition 1.3.4), this implies that $J(\theta_{best}) \to J(\theta^{*})$ as $k \to \infty$. Hence, proof is over.

Chapter 5

Apply ASPSA-I on Unconstrained Discrete Supply Chain Inventory Optimization Problems

This part is mainly testing the performance of the proposed ASPSA-I by a set of published numerical experiments.

5.1 Supply Chain Model

The supply chain model considered in this study is a four stage serial supply chain consisting of a retailer, a distributor; a manufacturer and a supplier (see Figure 5-1).

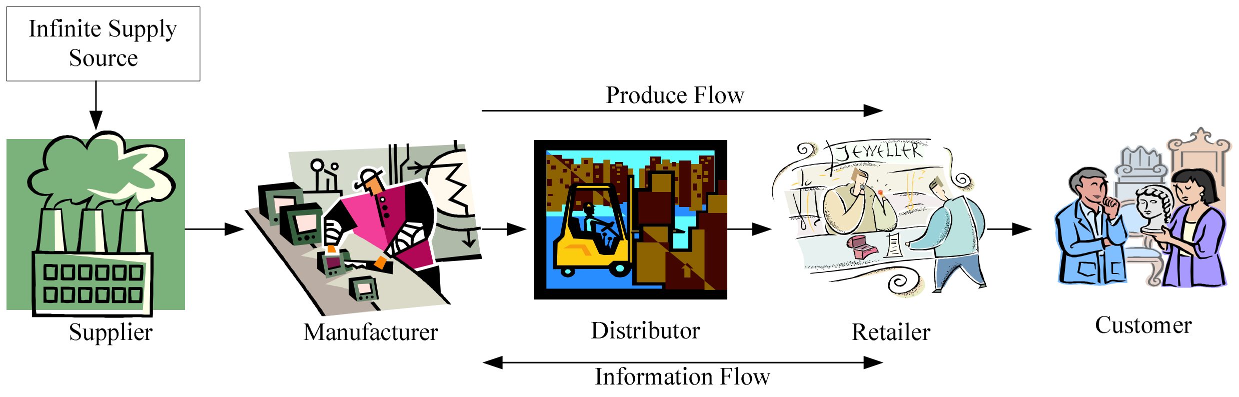


Figure 5-1: Supply chain network

5.1.1 Model Assumptions

- A single product flows through the supply chain
- Information lead time is negligible or zero
- Every store has a replenishment lead time which includes the production and transportation lead times
- Retailer faces random customer demand and the random distribution is stationary
- Base-stock level at every store takes discrete integer values
- There is no lot-size or discount policy for any store
- Excess demand is backlogged at every store until necessary stock becomes available
- Linear store holding and shortage cost-rates exist for all the stores in the supply chain
- Transportation cost is directly proportional to the quantity shipped
- All store have infinite capacity
- The most upstream store is linked to an infinite supply source.

5.1.2 Inventory Control Policy: a Base-stock Policy

All stores in the supply chain work with a periodic-review base-stock policy, where the review period is assumed to be one time unit. A base-stock policy is preferred in view of the simplicity, ease of operation and the ordering costs being assumed to be one time unite. A base-stock policy is preferred in view of its simplicity, ease of operation and the ordering costs being assumed to be negligible. The inventory system in the chain begins with an initial inventory of s units. Whenever a customer order for x units is received, an inventory replenishment order for x units is placed immediately to the upstream store. Replenishment orders are filled after the lead time L. The customer order is filled as far as possible, from the supply on-hand. Should the total unfilled customer demand exceed the inventory on-hand, then it is assumed that those customers will not cancel any orders, but await the arrival of stock (i.e. backlogging is allowed). Thus the sum of inventory on-hand and on-order (minus backlog) is constant in time and equal to s, the so-called base-stock (Hanssmann, 1962).

5.1.3 Measure of Performance

The measure of performance chosen in this study is the total supply chain cost which is defined as given below. The notations used are as follows:

j	Store index
n	Number of stores in the supply chain
s_j	Store base-stock level at store j
s_j^{UB}	Upper bound for s_j
s_j^{LB}	Lower bound for s_j
h_j	Store inventory holding cost-rate at store j
b_j	Store inventory backorder cost-rate at store j
L_j	Replenishment lead time with respect to store j
$L_j^{\max}$	Maximum replenishment lead time with respect to store j
$L_j^{\min}$	Minimum replenishment lead time with respect to store j

T_s	Short simulation run length
T	Full simulation time run length ($T_s < T$)
$I_{j,t}$	Inventory on-hand at store j at the end of time period t
$B_{j,t}$	Backorder at store j at the end of time period t
$R_{j+1,j,t}$	Quantity shipped by store $j+1$ to store j at time period t
$\tau_{j+1,j}$	Transportation cost-rate to transport one unit of material from store $j+1$ to j

Therefore the TSCC over all stores in the supply chain over T time periods is given by Eq. 5-1:

$$TSCC = \sum_{t=1}^{T} \sum_{j=1}^{n} (I_{j,t} \times h_j + B_{j,t} \times b_j + R_{j+1,j,t} \times h_{j+1} \times L_j + R_{j+1,j,t} \times \tau_{j+1,t}) \qquad \textbf{5-1}$$

The objective of this work is to compute optimal base-stock policy $s^* = \{s_1^*, s_2^*, s_3^*, \ldots, s_n^*\}$, which minimize TSCC. The last two terms in by Eq. 5-1 will not affect the optimal base-stock policy s^* due to the assumptions of backlogging, and linear or variable transport cost with no fixed or overhead transport cost. Hence, Eq. 5-1 reduces to Eq. 5-2:

$$TSCC = \sum_{t=1}^{T} \sum_{j=1}^{n} (I_{j,t} \times h_j + B_{j,t} \times b_j) \qquad \textbf{5-2}$$

5.1.4 Model Description

The supply chain considered in this study consists of four stores, namely, retailer, distributor, manufacturer and supplier. The supplier is linked to an infinite supply source. Random customer demand originates in the supply chain at the lowest store (store-1), the retailer (see Figure 5-1). Every store operates with pre-specified base-stock level, s_j as prescribed by the base-stock policy. When the store-1 receives the customer order, it meets the customer demand to the extent of on-hand inventory. Store-1 now places an order to store-2 (distributor) so that store-1 can increase its on-hand inventory level to its pre-specified base-stock level. Each store in the supply chain has a replenishment lead time to receive replenishment from the upstream store, provided that the upstream store has sufficient on-hand inventory to meet the downstream store's demand. Therefore, the actual replenishment for store-1 is determined not only by the replenishment lead time between these two stores, but also by the availability of on-hand inventory at the second store. If store-2 doesn't have enough on-hand inventories, then store-2 raises a replenishment order or store-3. Assuming store-3 has enough on-hand inventories to meet the demand from store-2, store-3 replenishes store-2, which, in turn, would meet the demand of store-1. If so, the actual replenishment lead time for store-1 is the sum of the replenishment lead times of store-1 and store-2. It is observed from this replenishment process that the replenishment lead time for any store in the supply chain will lie anywhere from its respective replenishment lead time to the maximum replenishment lead time that is equal to the sum of replenishment lead times of all upstream stores, with the actual lead time depending upon the on-hand inventory levels of the upstream stores, It is to be noted that the store on-hand inventory levels are in turn, dependent on the store base-stock levels.

5.2 Multi-agent Based Supply Chain Simulation

In this work, multi-agent based simulation is used as a performance-evaluation tool to evaluate the candidate base-stock policies. Compared to traditional DES techniques, Multi-agent based simulation (MABS) has the following important advantages:

- Supporting simulation of pro-active behavior ability, which is important when simulating humans (and animals) who are able to take initiatives without external stimuli.
- Supporting distributed computation in a very natural way. Since each agent is typically implemented as a separate piece of software corresponding to a process, it is straight-forward to let different agents run on different machines. This allows for better performance and scalability.
- Since each agent typically is implemented as a separate process and is able to communicate with any other agent using a common language, it is possible to add or remove agents during a simulation without interruption. And as a consequence of this and the structure preserving mapping between simulation software and the reality, it is even possible to swap an agent for the corresponding simulated entity. This enables extremely dynamical simulation scenarios.

Modularity of multi-agent simulation can lead to simpler programming. Rather than tackling the whole task with a centralized agent, programmers can identify subtasks and assign control of those tasks to different agents. In addition, it is possible to program the simulation model and software on a very high level, e.g., in terms of beliefs, intentions, etc., and making it easier for non-programmers to understand and even participates in the software development process.

We will build the MAS for Supply Chain Network, in which each agent is modeling a functional entity of supply chain such as supplier, manufacturer, retailer, and customer etc. Therefore, there are 5 types of agents in our supply chain library. The function and goal of each type of agent are identified in Table **5-1** .

It is very known that agent needs to communicate with each other to finish some functions. Therefore, we should identify the message types flown in the supply chain network. There are three types of messages through the supply chain: material, information, and financial. For simulation, now we only define two types of messages (Table 5-2). In addition, the format of message flow is shown in Table 5-3.

Table **5-1**: The function and goal of each agent type in supply chain library

Agent Type	**Function**	**Goals**
Supplier Agent	This one models the supplier. The supplier agent provides resources, components, or assemblies to manufacturing agent.	It mainly focuses on inventory of finished goods.
Manufacturing Agent	This one simulates a manufacturing plant, where the components are assembled and products are manufactured. In general, orders can be from retailers or customers directly.	The main concern here optimal procurement of components (particular common components) and on efficient management of inventory and manufacturing process.
Distribution agent	It receiving products from manufacturer and send them to retailers.	The main focus here is to reduce the inventory carried and maximize throughput
Retail agent	A retail agent is where customers buy products.	The main focus here is on reducing the cycle time for delivery of a customer order and minimizing stockouts.
Customer agent	customer agent is where the demand for products is generated	The main concern here is to get the products before the expected due date or immediately.

Table 5-2: Message types definition in MABS

Message Type ID	Message Type	Function
1	Material flow	Delivery of goods by one agent to another
2	Information flow	Model the exchange of information between supply chain agents. It includes request for goods, demand-forecast information, expected delivery due dates, order cancellation, and order modification, discount etc.

Table 5-3: Message Format

Message Type ID	Sender ID	Receiver ID	Sending Time	Content

However, until now there have been no proper programming languages to well support agent programming naturally. So in this work, we try to enhance the capability of the object-oriented language, C++, to build this agent system. Two types of classes are designed for the supply chain network. One is the store class, and the other is customer class. Then these classes will be integrated as basic units for agents (Figure 5-2). In store class, each implementation of agent has suppliers and customers, and processing time. However, in customer class, the implementation of agent only has suppliers and random demands.

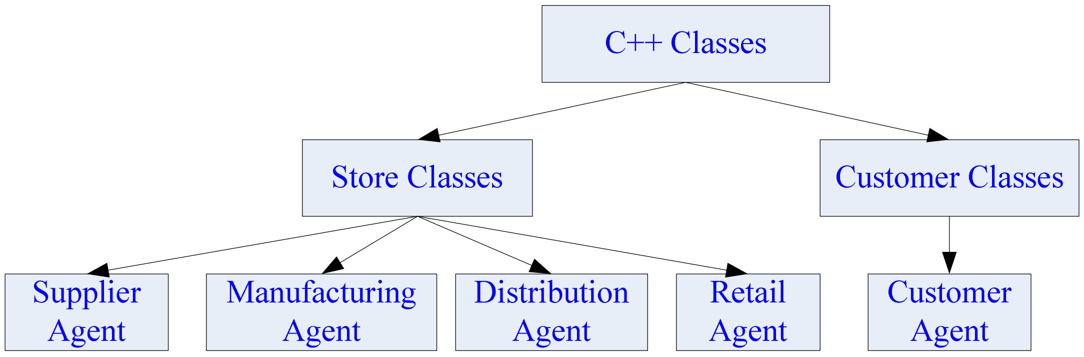


Figure 5-2: Agents of supply chain implemented by C++ classes

The sequence of events that take place at store j, where $j = 1$ to 4, at t is given below (Daniel and Rajendran, 2005b):

- The receipt of material at store j takes place if the material is due to arrive from store $j+1$ at the current time t. The store's inventory information (on-hand inventory and on-order inventory) is updated.
- If there is any backlog at store j, then the backlog is filled either partially or fully, depending on the on-hand inventory. The downstream store $j-1$ will receive this shipment after the corresponding replenishment lead time, and store $j's$ inventory is updated again.
- Store j receives the current-instant demand from downstream store $j-1$.
- Store j replenishes store $j-1$ for the current-instant demand, depending upon the on-hand inventory. If demand exceeds on-hand inventory, then the excess demand is backlogged. Store $j-1$ will receive this replenishment after the fixed replenishment lead time.
- Base-stock policy at store j triggers an order to the upstream store $j+1$, and this order is realized by store $j+1$ immediately as information lead time is zero.
- Store j 's inventory information such as on-hand inventory, backlog, and on-order inventory are updated finally.
- The holding and shortage costs are computed accordingly at the end of time period t.

It is assumed in this study that a time period corresponds to a unit time. The time period t is incremented and the same sequence of events are repeated. These sequences of events take place at every store over the entire simulation run length (see Eq. 5-2 for the computation of total supply chain costs over T periods).

5.3 Calculation of Upper and Lower Bound for Base Levels

An intial solution for ASPSA-I algorithm can be generated at random. Due to the nature of the problem under investigation, base-stock level at every store is bound by an upper bound s_j^{UB} and lower bound s_j^{LB} . These bounds are assumed at by considering the minimum and maximum customer demand, and the respective store's minimum and maximum replenishment lead time. Therefore the upper and lower bound of base stock for all stores are computed as given below (see Eq. 5-3 and Eq. 5-4).

$$s_j^{UB} = D_j^{\max} \times \sum_{i=j}^{N} L_i^{\max} \qquad \text{5-3}$$

$$s_j^{LB} = D_j^{\min} \times L_j^{\min} \qquad \text{5-4}$$

For example, the minimum and maximum demands per unit time are 10 unites and 100 unites respectively. Table 5-4 lists the maximum and minimum lead time for each stage. In this case, the maximum possible replenishment lead time for retailer is sum of the preparation time of retailers, distributors, manufacturer and supplier, i.e., 18 days, and the minimum possible replenishment lead time for retailer is 4 day. s_j^{UB} and s_j^{LB} for these four stores are computed using Eq. 5-5 , Eq. 5-6 , Eq. 5-7 , and Eq. 5-8.

Table 5-4: An example of lead time settings for four stages

Stage ID j	$L_j^{\max}$	$L_j^{\min}$
1	9	4
2	6	3
3	2	1
4	1	1

$s_1^{UB} = 100 \times 18 = 1800$;	$s_1^{LB} = 10 \times 4 = 40$;	5-5
$s_2^{UB} = 100 \times 9 = 900$;	$s_2^{LB} = 10 \times 3 = 30$;	5-6
$s_3^{UB} = 100 \times 3 = 300$;	$s_3^{LB} = 10 \times 1 = 10$; and	5-7
$s_1^{UB} = 100 \times 1 = 100$;	$s_1^{LB} = 10 \times 1 = 10$.	5-8

5.4 Experiment Design

Experiments are conducted to study the performance of the proposed ASPSA-I algorithm by considering a variety of supply chain problems.

5.4.1 Supply Chain Settings

The experiment settings considered here are published in Daniel and Rajendran (2005b). Five cost-rate settings are considered with different holding and shortage cost-rates at various stores in the supply chain by reflecting the value addition the product gains at it moves downstream (see Table 5-5). Two lead time settings for every store are considered to take care of the process that every store performs in adding value to the produce or service. Put together, the total settings comprise ten supply chain test settings which are used to evaluate the effectiveness and robustness of the proposed heuristic techniques.

5.4.2 Simulation Based Evaluation of TSCC for a Given Set of Base-stock Levels

Simulation is used in this study to evaluate the instatllation base-stock levels procided by the proposed ASPSA-I by measuring their TSCC. In the simulation experiments, customer demand is generated from a uniform distribution in the range $[20, \quad 60]$ units per unit time. The supply chain is simulated for the store base-stock levels yielded by a solution technique corresponding to a given supply chain test problem. The supply chian is simulated for the store base-stock levels yielded by a solution technique corresponding to a given supply chain test problem. The supply chain is simulated for a given full length 1200 days and each simulation is replicated 30 times with 15 different pairs of uniform random number streams and their antithetic uniform random number streams to generate customer demands. The TSCC over full length $T = 1200$ days is computed, and the mean TSCC over 30 replications is finally computed corresponding to a given set of store base-stock levels. These values of run-length and replications are fixed in order to minimize the standord error of sample mean (i.e. TSCC).

5.4.3 Performance Evaluation

To evaluate the performance of ASPSA-I algorithm against other stochastic algorithm such as SPSA, GA etc., 20 different runs are carried out using 20 different random seeds. And then $\text{TSCC}_{\text{average,}}$ $\text{TSCC}_{\text{min,}}$ $\text{TSCC}_{\text{max,}}$ of these 20 runs will be used to evaluate solution quality of algorithms. To evaluate computation effort, the number of evaluations (N) is used. In this research, we set $T \times 30$ as one unit of evaluation. Therefore, in ASPSA-I algorithm, for short simulation runs, it will use $\frac{Ts \times 30}{T \times 30} = \frac{Ts}{T}$ units of evaluation, which is a fraction number. Then, $N_{average}$, $N_{\max}$, $N_{\min}$ of 20 runs will be used to evaluate computation efforts of algorithms.

Table 5-5: Supply chain setting

Setting	Components	Store -4	Store -3	Store -2	Store -1	Ratio of b_j/h_j
CS1_LT1	h_j	1	2	4	8	2
	b_j	2	4	8	16	
	L_j	4	5	3	1	
CS2_LT1	h_j	1	2	4	8	4
	b_j	4	8	16	32	
	L_j	4	5	3	1	
CS3_LT1	h_j	1	2	4	8	6
	b_j	6	12	24	48	
	L_j	4	5	3	1	
CS4_LT1	h_j	1	2	4	8	8
	b_j	8	16	32	64	
	L_j	4	5	3	1	
CS5_LT1	h_j	1	2	4	8	10
	b_j	10	20	40	80	
	L_j	4	5	3	1	
CS1_LT2	h_j	1	2	4	8	2
	b_j	2	4	8	16	
	L_j	4	6	3	2	
CS2_LT2	h_j	1	2	4	8	4
	b_j	4	8	16	32	
	L_j	4	6	3	2	
CS3_LT2	h_j	1	2	4	8	6
	b_j	6	12	24	48	
	L_j	4	6	3	2	
CS4_LT2	h_j	1	2	4	8	8
	b_j	8	16	32	64	
	L_j	4	6	3	2	
CS5_LT2	h_j	1	2	4	8	10
	b_j	10	20	40	80	
	L_j	4	6	3	2	

5.5 Solution Methodologies under Evaluation

5.5.1 ASPSA-I algorithm

To employ the proposed ASPSA-I algorithm solving the discrete optimization problems, parameters like a_k , pre-search number m, line search number l, short simulation run length etc. should be specified. Table 5-6 lists parameter settings to run ASPSA-I algorithm for all experiments.

Table 5-6: Parameter setting of ASPSA-I algorithm

Name	Settings
Δ_k	Bernoulli distribution ($\pm$ 1 with 50% probability)
c_k	Constant, and $c_k = 1$
a_{kj}	A uniform Distributed integer chosen from [0, $\frac{10j}{\sqrt{k+10}}+1$]
T_s	50
T	1200
m	10
l	4
Λ_k	An integer uniformly chosen from[-10, 10]
Stopping criteria	$k \geq 1000$ or θ_{best} is not improved in 10 consecutive iterations

Table 5-7: Parameter setting of SPSA algorithm

Name	Distribution and Parameters
Δ_k	Bernoulli distribution
c_k	Constant, and $c_k = 1$
a_k	An integer rounded from $\left[\frac{100}{\sqrt{k+10}}+1\right]$
Stopping criteria	$k \geq 1000$ or θ_{best} is not improved in 10 consecutive iterations

5.5.2 SPSA algorithm

To adapt SPSA algorithm solve the discrete optimization problems, the gradient will be also used Eq. 5-9 to achieve for easily setting a_k. However, $y(\theta + c_k\Delta_k)$ and $y(\theta - c_k\Delta_k)$ are achieved by using full length simulation instead of using short simulation run like ASPSA-I. The parameter settings are listed in Table 5-7.

$$\hat{g}_k = \begin{cases} 1/\Delta_k & \textit{if} \quad y(\theta + c_k\Delta_k) > y(\theta - c_k\Delta_k) \\ -1/\Delta_k & \textit{otherwise} \end{cases} \qquad \text{5-9}$$

5.5.3 Genetic Algorithm (GA)

GA is a naïve implementation, primarily to test solution quality of the proposed ASPSA-I algorithm and SPSA algorithm, and indicates relative computation effort. The GA code was downloaded from http://www.iitk.ac.in/kangal/codes.shtml. This is a GA implementation using binary and real coded variables. Mixed variables can be used. Constraints can also be handled. And the parameters to run the code are listed in Table 5-8.

Table 5-8: Parameters setting for Genetic Algorithm

Name	**Settings**
Population size	20
Are decision variables bound rigid?	Yes
Parameter-space niching to be done?	Yes
Niching parameter value	10
Runs number	20
Crossover probability	0.9
Mutation probability	0.25
Random seed	0.123
Distribution index for SBX and mutation	2 and 100
Stopping criteria	100 generations

5.6 Complete Enumeration (CE)

All combinations of base-stock levels corresponding to every store in the range $\left[s_j^{LB} \quad s_j^{UB} \right]$, where $j = 1$ to 4, are enumerated and evaluated through simulation to obtain the optimal set of base-stock levels yielding the minimum TSCC. This is done in order to compare the solutions generated by various procedures with the optimal solution.

5.7 Results and Discussions

The performance of heuristic methodologies is evaluated in 20 runs using 20 difference random seeds by considering the supply chain test problems. Optimal Solutions are obtained for the all the supply chain test problems through a complete enumeration of the solution space. The performance of the heuristic methodology is analyzed by the quality of the solutions yielded and the computational effort required to obtain the solution.

To evaluate solution quality, we use the best store base-stock levels, minimum of TSCC (TSCCmin), maximum of TSCC (TSCCmax), mean of TSCC (TSCCmean), and standard deviation of TSCC in 20 runs by each of the solution methodologies. Table **5-9** exhibited the running results. Figure **5-3** compares TSCC$_{min}$ obtained by ASPSA-I, SPSA and GA against optimal solution obtained by complete enumeration method. TSCC$_{min}$ obtained by ASPSA-I and GA is very near optimal solution, however, those obtained by SPSA algorithm is not so good. In addition, Figure **5-4(a)** shows TSCC of 20 runs obtained by SPSA algorithm varying in a very large range, yet those obtained by ASPSA-I and GA varies in small range (Figure **5-4(b)** and Figure **5-4(c)**). Moreover, Figure **5-5** shows the relative improvement (*RI*) of TSCC of GA over ASPSA-I, this is calculated by Eq. **5-10**. From Figure **5-5**, we can see that the relative improvement of GA against ASPSA-I is not more than 4.5%, and in CS1_LT2 case, GA performs much worse than ASPSA-I on TSCC$_{\max}$. Therefore, we can get conclusion that ASPSA-I algorithm perform almost as well as GA algorithm on solution quality aspect for the tested problems. In addition, compared with published results (Table 4 in Daniel and Rajendran, 2005b, p. 174-176), we can see that ASPSA-I achieved near optimization solution as Simulated Annealing (SA) does.

$$RI_{TSCC} = \frac{TSCC_{ASPSA} - TSCC_{GA}}{TSCC_{GA}} \times 100\% \qquad \textbf{5-10}$$

In addition, computation effort, which is evaluated by $N_{\min}$, $N_{\max}$, and N_{mean} of 20 runs are listed in Table **5-10**, from which we can see, GA use much more simulation runs (thousands) than SPSA and ASPSA-I algorithm (hundreds) (over 6 times more). In addition, compared with published results (Table 4 in Daniel and Rajendran, 2005b, p. 177), we can see that ASPSA-I and SPSA also uses much less computation resources than Simulated Annealing (SA), which needs over one thousands function evaluations.

In addition, ASPSA-I algorithm uses fewer number of simulation runs than SPSA algorithm. Figure 5-6 also clearly shows that computation effort of ASPSA-I algorithm varies in a smaller range than that of SPSA algorithm. To see that ASPSA-I algorithm uses much less computation effort than GA, the relative computation effort (R_N) is used, and it is calculated by Eq. 5-11 . Figure **5-7** shows relative computation effort of GA over ASPSA-I on all 10 cases, and it can easily notice that GA spends over 20 times computation effort in average (on N_{mean}). Therefore, it indicates that ASPSA-I algorithm is much more efficient than GA algorithm.

$$R_N = \frac{N_{GA}}{N_{ASPSA}} \qquad \textbf{5-11}$$

In summary, we can get the following conclusions from the above experiments:

1). GA algorithm performs best in solution quality, however, worst in computation effort.

2). ASPSA-I algorithm performs a little worse than GA algorithm in solution quality, but best in computation effort (using less than 1/20 of simulation runs by GA algorithm).

3). SPSA algorithm performs worst in solution quality. In computation effort aspect, it is also worse than ASPSA-I algorithm.

4). Considering both solution quality and computation effort, ASPSA-I algorithm performs best.

Table 5-9: Performance evaluation of solution methodologies with deterministic lead times

Case ID	Cases	Method	$\theta_{best} = \{S, M, D, R\}$	$TSCC_{best}$	$TSCC_{worst}$	$TSCC_{mean}$	Std(TSCC)
1	CS1_LT1	CE	{179, 227, 139, 50}	387347	387347	387347	0
		SPSA	{173, 238, 140, 48}	394424	1074632	482404	150281
		ASPSA-I	{181, 226, 140, 50}	387509	391276	387956	844
		GA	{181, 226, 139, 50}	387406	392011	387903	1055
2	CS2_LT1	CE	{186, 233, 145, 54}	458431	458431	458431	0
		SPSA	{194, 237, 149, 54}	467375	2414666	762363	512176
		ASPSA-I	{186, 233, 145, 54}	458431	463324	459274	1229
		GA	{185, 233, 145, 54}	458463	461155	458805	608
3	CS3_LT1	CE	{189, 237, 148, 56}	495787	495787	495787	0
		SPSA	{180, 235, 151, 56}	502356	2486352	1150549	747828
		ASPSA-I	{190, 235, 149, 56}	495886	503282	497101	1851
		GA	{189, 237, 148, 56}	495787	496882	495947	255
4	CS4_LT1	CE	{192, 239, 151, 57}	519953	519953	519953	0
		SPSA	{190, 233, 151, 59}	529508	2650598	1519282	801109
		ASPSA-I	{193, 238, 151, 57}	520037	523866	521149	1053
		GA	{193, 239, 151, 57}	519963	520795	520275	226
5	CS5_LT1	CE	{194, 241, 153, 57}	537259	537259	537259	0
		SPSA	{196, 247, 163, 55}	567542	2864779	1453765	732445
		ASPSA-I	{195, 242, 153, 57}	537344	560325	539531	5023
		GA	{194, 241, 153, 57}	537259	538184	537567	263
6	CS1_LT2	CE	{179, 269, 139, 91}	443444	443444	443444	0
		SPSA	{181, 270, 140, 89}	444723	951349	519471	118753
		ASPSA-I	{178, 270, 139, 91}	443480	444537	443804	348
		GA	{179, 269, 139, 91}	443444	481332	445534	8430
7	CS2_LT2	CE	{186, 276, 144, 97}	536984	536984	536984	0
		SPSA	{187, 274, 142, 97}	538652	888081	578325	76954
		ASPSA-I	{186, 276, 145, 97}	537021	540340	537564	862
		GA	{186, 276, 144, 97}	536984	541755	537387	1045
8	CS3_LT2	CE	{189, 280, 147, 101}	588911	588911	588911	0
		SPSA	{183, 286, 146, 102}	593223	2465616	739014	422974
		ASPSA-I	{190, 280, 147, 101}	588938	594058	589956	1424
		GA	{189, 280, 147, 101}	588911	589351	589033	114
9	CS4_LT2	CE	{191, 283, 150, 103}	623705	623705	623705	0
		SPSA	{194, 285, 154, 103}	628286	2473220	888264	495520
		ASPSA-I	{191, 283, 150, 103}	623705	630258	624836	1871
		GA	{193, 283, 150, 103}	623718	625313	624018	409
10	CS5_LT2	CE	{194, 285, 152, 104}	649403	649403	649403	0
		SPSA	{192, 287, 152, 105}	650194	2403013	871846	426809
		ASPSA-I	{194, 285, 151, 104}	649514	653766	650503	1066
		GA	{194, 285, 152, 104}	649403	650161	649672	232

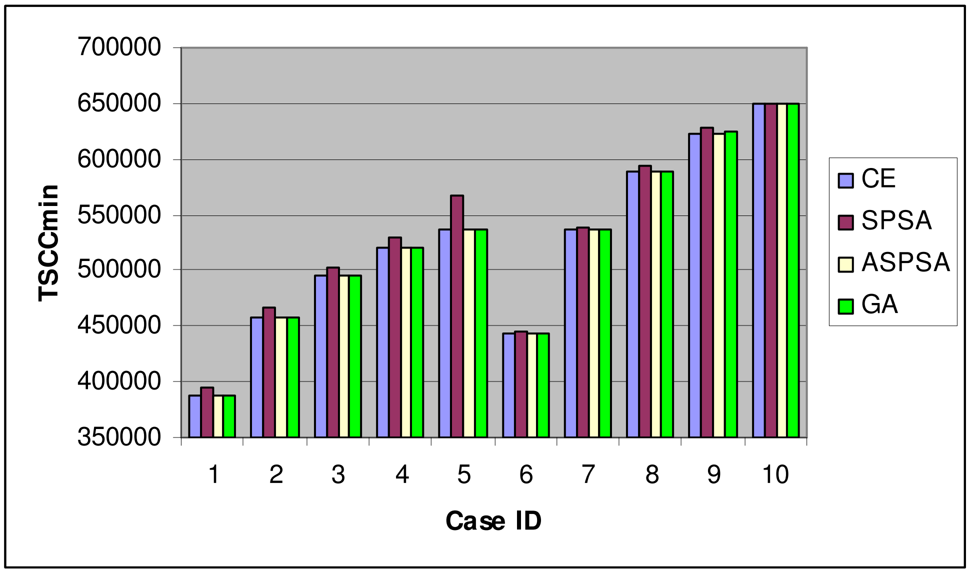
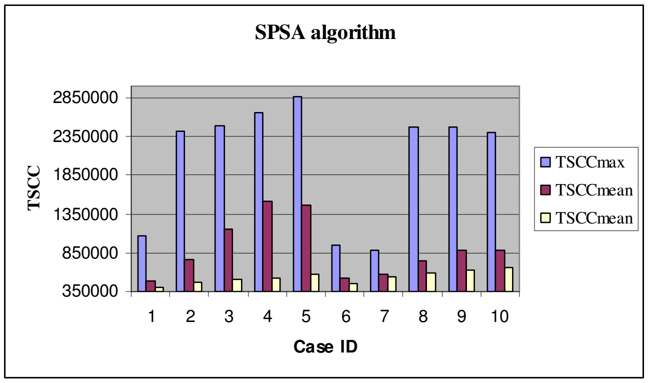
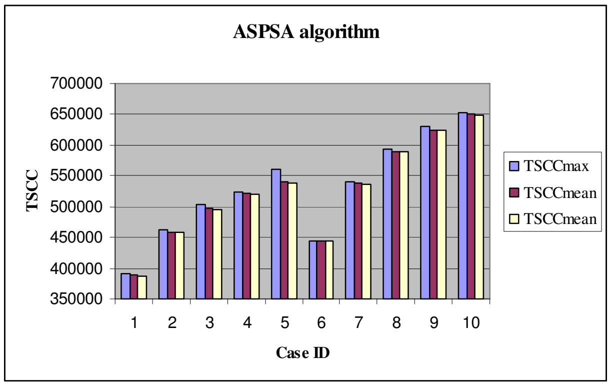
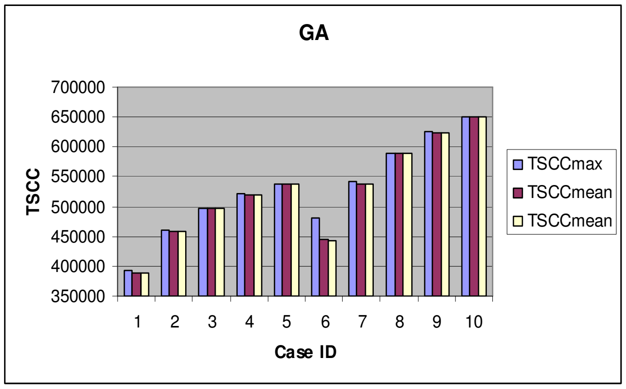


Figure 5-3: Comparing $TSCC_{min}$ of 20 runs for four algorithms



(a) Solution quality of SPSA algorithm





(b) Solution quality of ASPSA-I algorithm

(c) Solution quality of GA algorithm

Figure 5-4: Solution quality of TSCC of 20 runs for SPSA, ASPSA-I and GA algorithms and tested by case CS4_LT2

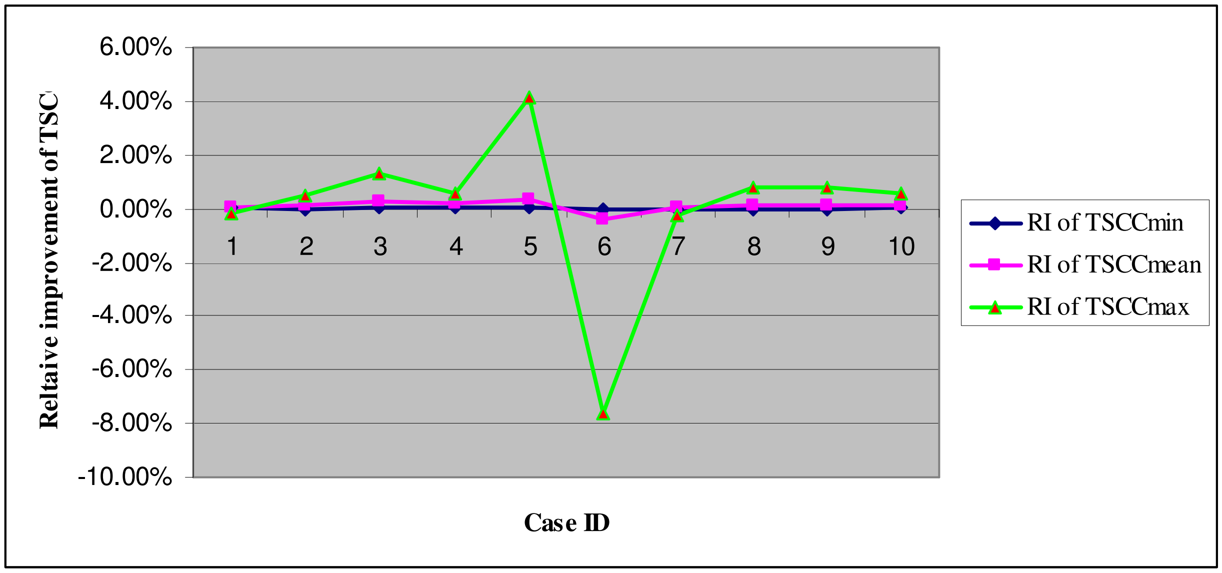
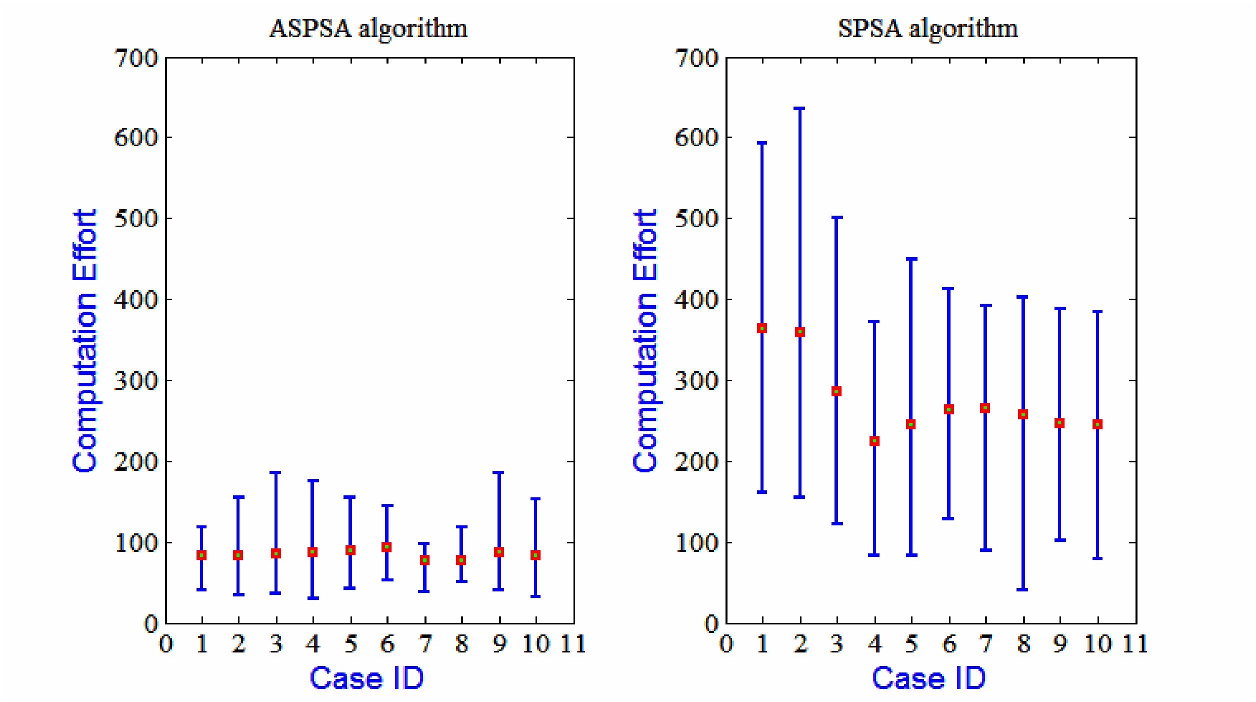


Figure 5-5: The relative improvement of TSCC of GA against ASPSA-I for 10 cases



(a) Computation effort of ASPSA-I algorithm (b) Computation effort of SPSA algorithm

Figure 5-6: Comparing computation effort of ASPSA-I algorithm against SPSA algorithm in 20 runs

Table 5-10: Computation effort required by different solution procedures

Case ID	Cases	Methods	$N_{\min}$	$N_{\max}$	$N_{\max}$	Std(N)
1	CS1_LT1	SPSA	161	593	364	135
		ASPSA-I	39	118	83	22
		GA	2000	2000	2000	0
2	CS2_LT1	SPSA	155	635	360	126
		ASPSA-I	33	155	82	27
		GA	2000	2000	2000	0
3	CS3_LT1	SPSA	122	500	286	103
		ASPSA-I	35	186	85	37
		GA	2000	2000	2000	0
4	CS4_LT1	SPSA	83	371	224	82
		ASPSA-I	30	176	88	33
		GA	2000	2000	2000	0
5	CS5_LT1	SPSA	83	449	245	96
		ASPSA-I	41	155	90	34
		GA	2000	2000	2000	0
6	CS1_LT2	SPSA	127	413	263	94
		ASPSA-I	53	145	93	24
		GA	2000	2000	2000	0
7	CS2_LT2	SPSA	89	391	266	88
		ASPSA-I	38	98	77	17
		GA	2000	2000	2000	0
8	CS3_LT2	SPSA	39	403	256	102
		ASPSA-I	51	118	77	20
		GA	2000	2000	2000	0
9	CS4_LT2	SPSA	101	387	247	87
		ASPSA-I	40	185	87	41
		GA	2000	2000	2000	0
10	CS5_LT2	SPSA	79	383	245	88
		ASPSA-I	31	152	82	32
		GA	2000	2000	2000	0

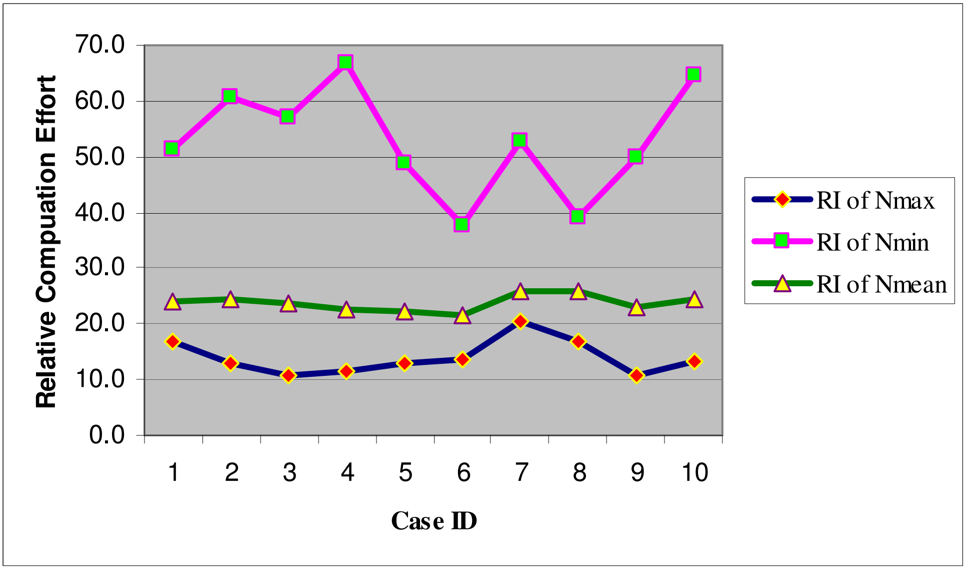


Figure 5-7: Relative computation effort of GA to ASPSA-I

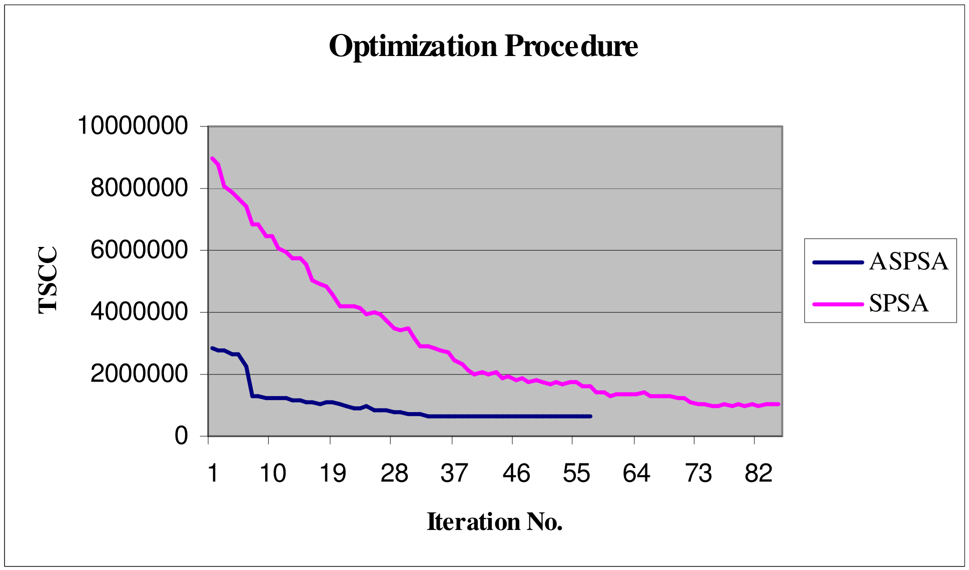


Figure 5-8: Optimization trace of ASPSA-I and SPSA algorithms for CS4_LT2 case

Figure 5-8 demonstrates the process of optimization of SPSA algorithm and ASPSA-I algorithm for CS4_LT2 case. From that, we can see ASPSA-I algorithm begins with a better solution to search because of its added pre-search function. Moreover, ASPSA-I algorithm achieves better solution in fewer simulation runs than SPSA, which can be credited to other added functions. Figure 5-9 shows the decision variable evolving process during optimization, and indicates the capability of ASPSA-I algorithm converges to the optimal solution.

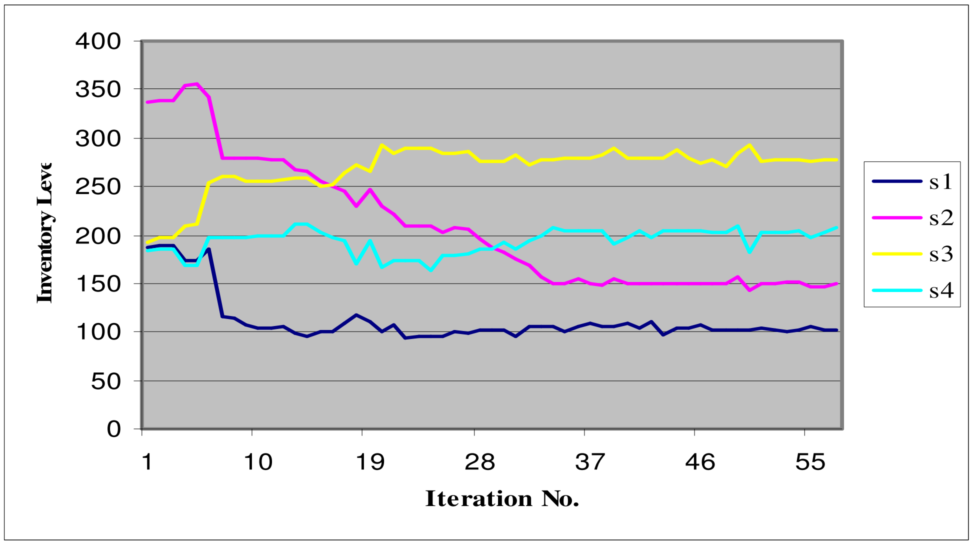


Figure 5-9: Optimization trace of decision variables of ASPSA-I algorithm for CS4_LT2 case

5.8 Experiment 2-Inventory optimization in the presence of stochastic replenishment lead times

The second set of experiment is also from Daniel and Rajendran (2005). And the second set of experiments consider the stochastic lead times. So far, this study has addressed uncertainty only in terms of demand and the performance of the heuristic solution methodologies is analyzed. Uncertainties are the main drivers to store inventories at various locations in a supply chain. Uncertainties can be present in demand, supply or lead time. Even when future demand can be accurately predicated, the supply chain needs to hold additional inventory to ensure smooth flow of production or service to customers, when the replenishment lead times are stochastic or uncertain. It is more realistic to find in a supply chain that both the external demand and store's replenishment lead times are stochastic in nature. In a supply chain with demand and replenishment lead times are stochastic; it is quite logical to believe that the supply chain would tend to hold more inventory than the earlier case of considering only the existence of random customer demand. At this juncture, it is important to study and figure out how well the solution procedures perform in optimizing inventory in a supply chain operating in an environment of stochastic demand and stochastic replenishment lead time.

Table **5-11** shows the supply chain settings with stochastic replenishment lead times. For example, for the supply chain setting CS1_LT1, the replenishment lead time associated with store-2 can vary between 2 time unites to 4 time units.

The performance of the proposed ASPSA-I algorithm in terms of solution quality and computational effort is studied with different supply chain settings (refer to Table **5-11**). Optimal store base-stock levels in the class of base-stock policies are generated by complete enumeration of the search space. The performance of the proposed ASPSA-I is evaluated. The performance of ASPSA-I is compared against GA and SPSA algorithm. All the rest of the experiment design is same as that of the supply chain operating with deterministic lead times.

Table **5-12** summaries the best store base-stock levels, minimum of TSCC (TSCCmin), maximum of TSCC (TSCCmax), mean of TSCC (TSCCmean), and standard deviation of TSCC in 20 runs with respect to each of the solution methodology. From Table **5-12**, we can see that the store base-stock levels provided by SPSA algorithm are significantly far from the optimal solutions. For $\text{TSCC}_{\text{mean}}$, the worse case at CS1_LT2 is as far as 989.06% from optimal solution. The computation effort required by each of the methodologies is shown in Table **5-13** . Compare

Table 5-13 and Table 5-10, we can see that stochastic replenishment cases take more simulation effort than deterministic replenishment lead time cases for SPSA and ASPSA-I algorithms.

Table 5-11: Supply chain setting with the consideration of stochastic replenishment lead time

Setting	Components	Store -4	Store -3	Store -2	Store -1	Ratio of b_j/h_j
CS1_LT1	h_j	1	2	4	8	2
	b_j	2	4	8	16	
	L_j	[3, 5]	[4, 6]	[2, 4]	1	
CS2_LT1	h_j	1	2	4	8	4
	b_j	4	8	16	32	
	L_j	[3, 5]	[4, 6]	[2, 4]	1	
CS3_LT1	h_j	1	2	4	8	6
	b_j	6	12	24	48	
	L_j	[3, 5]	[4, 6]	[2, 4]	1	
CS4_LT1	h_j	1	2	4	8	8
	b_j	8	16	32	64	
	L_j	[3, 5]	[4, 6]	[2, 4]	1	
CS5_LT1	h_j	1	2	4	8	10
	b_j	10	20	40	80	
	L_j	[3, 5]	[4, 6]	[2, 4]	1	
CS1_LT2	h_j	1	2	4	8	2
	b_j	2	4	8	16	
	L_j	[3, 5]	[5, 7]	[2, 4]	[1, 3]	
CS2_LT2	h_j	1	2	4	8	4
	b_j	4	8	16	32	
	L_j	[3, 5]	[5, 7]	[2, 4]	[1, 3]	
CS3_LT2	h_j	1	2	4	8	6
	b_j	6	12	24	48	
	L_j	[3, 5]	[5, 7]	[2, 4]	[1, 3]	
CS4_LT2	h_j	1	2	4	8	8
	b_j	8	16	32	64	
	L_j	[3, 5]	[5, 7]	[2, 4]	[1, 3]	
CS5_LT2	h_j	1	2	4	8	10
	b_j	10	20	40	80	
	L_j	[3, 5]	[5, 7]	[2, 4]	[1, 3]	

Table 5-12: Performance evaluation of solution methodologies with the consideration of stochastic replenishment lead times

Case ID	Cases	Method	$\theta_{best} = \{S, M, D, R\}$	TSCC$_{min}$	TSCC$_{max}$	TSCC$_{mean}$	Std(TSCC)
1	CS1_LT1	CE	{180, 231, 155, 50}	522420	522420	522420	0
		SPSA	{186, 224, 159, 48}	526676	854603	594592	94107
		ASPSA-I	{180, 231, 154, 51}	522637	536963	527527	3719
		GA	{180, 231, 156, 50}	522421	528682	523789	1455
2	CS2_LT1	CE	{190, 244, 164, 54}	629410	629410	629410	0
		SPSA	{197, 236, 169, 53}	633695	1527498	756907	213720
		ASPSA-I	{196, 244, 163, 54}	629998	636502	632184	1880
		GA	{190, 244, 164, 54}	629410	631024	630027	441
3	CS3_LT1	CE	{206, 245, 170, 56}	686190	686190	686190	0
		SPSA	{207, 247, 176, 56}	694699	3420438	1453863	942142
		ASPSA-I	{206, 245, 170, 56}	686190	699054	691910	3812
		GA	{206, 245, 170, 56}	686190	691937	687285	1233
4	CS4_LT1	CE	{206, 255, 174, 57}	724937	724937	724937	0
		SPSA	{203, 251, 180, 56}	734462	3538469	2013322	946922
		ASPSA-I	{210, 253, 173, 57}	726049	770298	733289	11403
		GA	{206, 255, 174, 57}	724937	728581	725845	896
5	CS5_LT1	CE	{209, 258, 177, 57}	752167	752167	752167	0
		SPSA	{204, 252, 160, 59}	816862	3753780	2260790	879552
		ASPSA-I	{209, 258, 176, 57}	752206	762622	758047	3358
		GA	{209, 258, 176, 57}	752206	756150	753430	794
6	CS1_LT2	CE	{182, 273, 144, 100}	695010	69510	69510	0
		SPSA	{183, 273, 143, 103}	697741	1171465	757009	107408
		ASPSA-I	{182, 273, 145, 100}	695020	726432	701349	7092
		GA	{182, 273, 144, 100}	695010	704996	696628	2398
7	CS2_LT2	CE	{190, 286, 157, 109}	879471	879471	879471	0
		SPSA	{196, 287, 153, 111}	885735	1454234	964274	158797
		ASPSA-I	{184, 282, 156, 111}	884329	892406	887033	2191
		GA	{190, 286, 157, 109}	879471	882858	880929	967
8	CS3_LT2	CE	{204, 289, 164, 116}	990783	990783	990783	0
		SPSA	{192, 298, 159, 117}	998260	1365497	1062982	113542
		ASPSA-I	{204, 289, 164, 118}	991557	1012818	997042	5001
		GA	{204, 289, 164, 116}	990783	995186	992073	1002
9	CS4_LT2	CE	{206, 295, 166, 121}	1065480	1065480	1065480	0
		SPSA	{201, 289, 166, 119}	1073814	1138822	1093510	18409
		ASPSA-I	{203, 291, 167, 122}	1066344	1094365	1073147	5870
		GA	{206, 295, 166, 121}	1065480	1068178	1066479	868
10	CS5_LT2	CE	{208, 302, 166, 126}	1120190	1120190	1120190	0
		SPSA	{220, 292, 170, 125}	1126275	1242542	1148116	29693
		ASPSA-I	{208, 299, 167, 127}	1123021	1148294	1129058	5106
		GA	{208, 302, 166, 128}	1121047	1123839	1122503	806

Table 5-13:Computation effort of solution methodologies with the consideration of stochastic replenishment lead times

Case ID	Cases	Method	CEmin	CEmax	CEmean	Std(CE)
1	CS1_LT1	SPSA	164	854	489	168
		ASPSA-I	43	148	78	30
		GA	2000	2000	2000	0
2	CS2_LT1	SPSA	236	746	491	157
		ASPSA-I	49	154	83	29
		GA	2000	2000	2000	0
3	CS3_LT1	SPSA	170	755	374	169
		ASPSA-I	49	181	92	39
		GA	2000	2000	2000	0
4	CS4_LT1	SPSA	50	512	293	134
		ASPSA-I	33	140	86	27
		GA	2000	2000	2000	0
5	CS5_LT1	SPSA	41	434	260	106
		ASPSA-I	49	160	88	32
		GA	2000	2000	2000	0
6	CS1_LT2	SPSA	155	722	482	153
		ASPSA-I	38	203	76	38
		GA	2000	2000	2000	0
7	CS2_LT2	SPSA	125	719	459	164
		ASPSA-I	38	203	76	38
		GA	2000	2000	2000	0
8	CS3_LT2	SPSA	167	689	454	140
		ASPSA-I	43	131	67	22
		GA	2000	2000	2000	0
9	CS4_LT2	SPSA	143	683	484	140
		ASPSA-I	38	118	71	23
		GA	2000	2000	2000	0
10	CS5_LT2	SPSA	176	728	477	155
		ASPSA-I	43	139	81	32
		GA	2000	2000	2000	0

5.9 Experiment 3-Inventory Optimization in Fork-Join Supply Chain

The third set of experiments is carried on a more complicated network, fork-join supply chain network (Figure 5-10). And the third set of experiments considers both the stochastic elements and network structure. Table 5-14 shows two types of cost setting for the network.

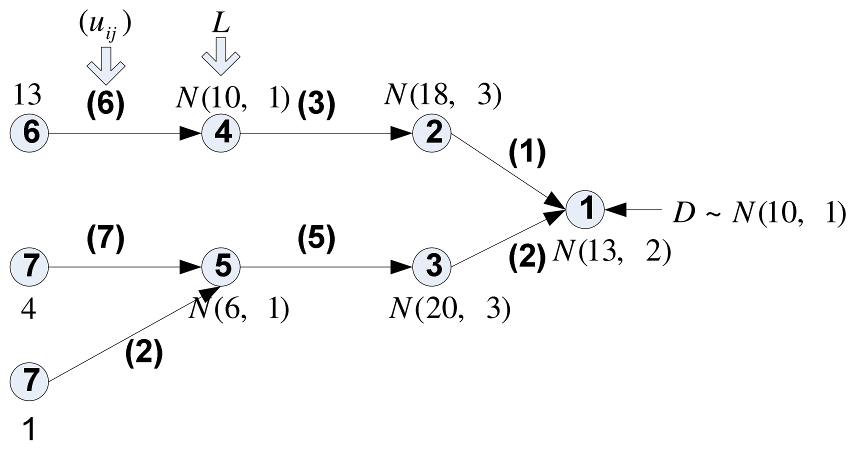


Figure 5-10: Network structure of experiment 3

Table 5-14: Two types of supply chain cost settings in example 3

Setting	Components	Store -8	Store -7	Store -6	Store -5	Store -4	Store -3	Store -2	Store -1	Ratio of b_j/h_j
CS1	h_j	1	1	2	3	4	5	6	13	2
	b_j	2	2	4	6	8	10	12	26	
CS2	h_j	1	1	2	3	4	5	6	13	5
	b_j	5	5	10	15	20	25	30	65	
CS3	h_j	1	1	2	3	4	5	6	13	10
	b_j	10	10	20	30	50	50	60	130	

The performance of ASPSA-I is evaluated and compared GA, and the results are shown in Table 5-15, from which we can see for the complicated supply chain network, ASPSA-I can also achieve comparable results to GAs with must less computation effort.

Table 5-15:Performance evaluation of solution methodologies on a fork-join supply chain network

Case ID	Cases	Method	$\theta_{best} = \{s_1, \quad K, \quad s_2\}$	TSCC	N
1	CS1	ASPSA-I	{130, 213, 1508, 177, 3745, 2344, 5741, 1422}	$6.3\text{x}10^6$	411
		GA	{128, 213, 1435, 180, 3435, 2341, 5881, 1371}	$5.4\ \text{x}10^6$	5000
2	CS2	ASPSA-I	{139, 204, 1470, 219, 3902, 2340, 5888, 1476}	$7.14\ \text{x}10^6$	404
		GA	{137, 210, 1451, 183, 3536, 2358, 5888, 1413}	$6.26\ \text{x}10^6$	5000
3	CS3	ASPSA-I	{139, 210, 1477, 198, 3831, 2359, 6125, 1470}	$7.65\text{x}10^6$	245
		GA	{136, 216, 1478, 207, 3675, 2377, 6024, 1372}	$6.95\text{x}10^6$	5000

5.10 Design of Experiment

From above experiments, we can see ASPSA-I algorithm performs much better than SPSA algorithm. This can be due to added functions Therefore, the objective of this section is:

- To identify the factors that affecting performance of ASPSA-I algorithm
- To come out with a clear recommendation regarding the parameter settings that gives best performance of ASPSA-I algorithm.

Table 5-16 lists the factors we want to investigate, and response variable will be objective TSCC and computation effort N . 1/4 factorial design will be used to analyze the influence of factors on the responses. Four replications are used to collect data. After getting the data, running factorial design gives us. Figure 5-11 shows that presearch number, line search number and non-uniformalty of step size all influence the objective TSCC, and Figure 5-12 shows that the presearch number, the interaction of line search number and memory, the interaction of line search number and short simulation run length, and short simulation run length are influencing computation effort.

Table 5-16: Factors of design experiment

Factor	Description	Type	High Level	Low Level
A	Presearch number	numerical	10	1
B	Line search number	numerical	4	1
C	Short simulation length	numerical	50	1200
D	Step size	categorical	uniform	nonuniform
E	Memory	categorical	Having memory	None

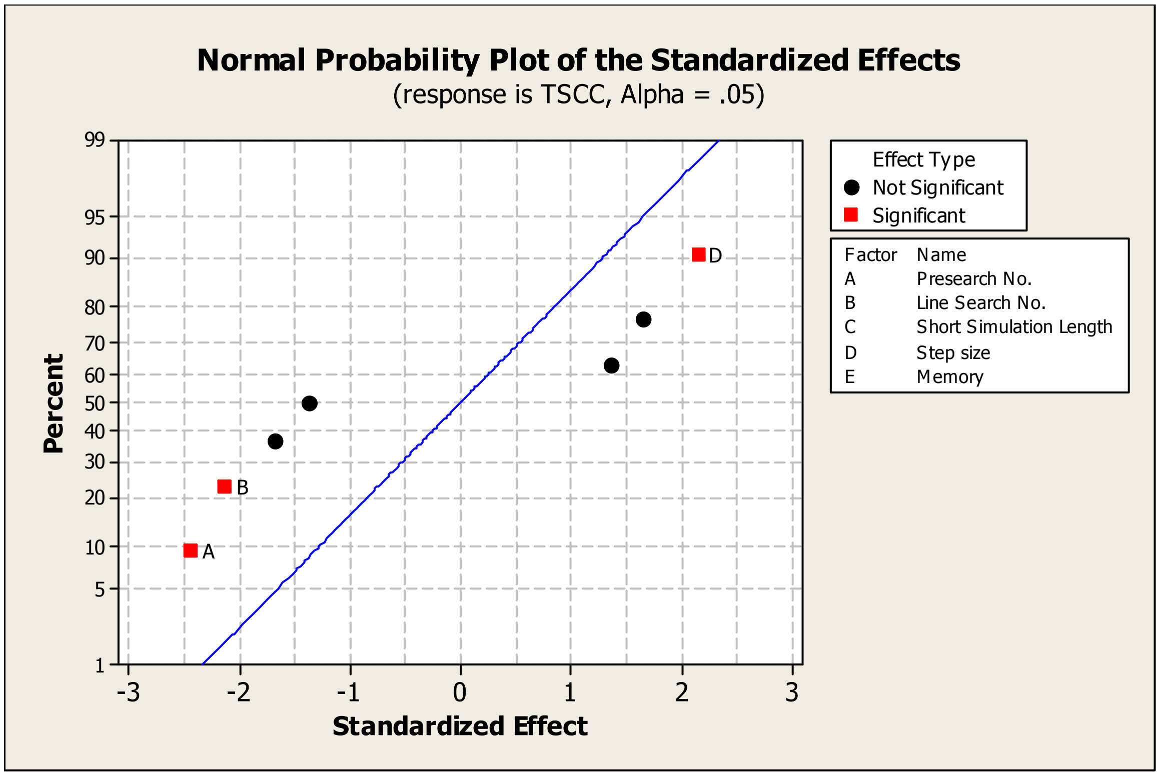


Figure 5-11: Major factors to TSCC

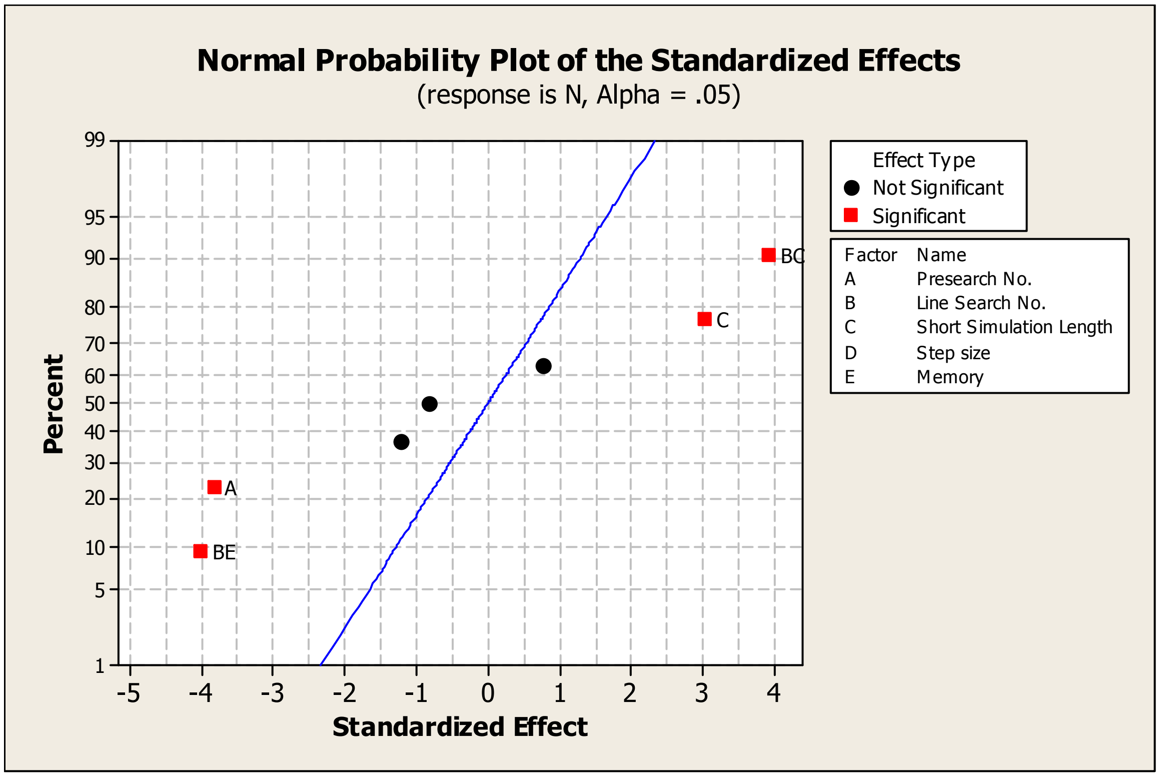


Figure 5-12: Major factors for N

Three important factors (pre-search number, line search number, and short simulation run length) will be studied further. To study every importance factor, other factors such as a_k, c_k, and Δ_k distribution will be held constant. The response variable will be solution quality (TSCC$_{\min}$, TSCC$_{\max}$, TSCC$_{\text{mean}}$ in 20 runs) and computation effort ($N_{\min}$, $N_{\max}$, N_{mean} in 20 runs).

5.10.1 Line Search Number on Performance of ASPSA-I

Figure 5-13 shows how the line search number influences solution quality of ASPSA-I algorithm, from which we can see increasing line search number will improve solution quality. When the line search number is 1, as SPSA algorithm does, solution quality is worst.

Figure 5-14 shows how the line search number influences computation effort of ASPSA-I algorithm. It can be noticed that at the beginning computation effort decreases with line search number, and then increases. Based on the graphs, 3-6 line search number will be recommended.

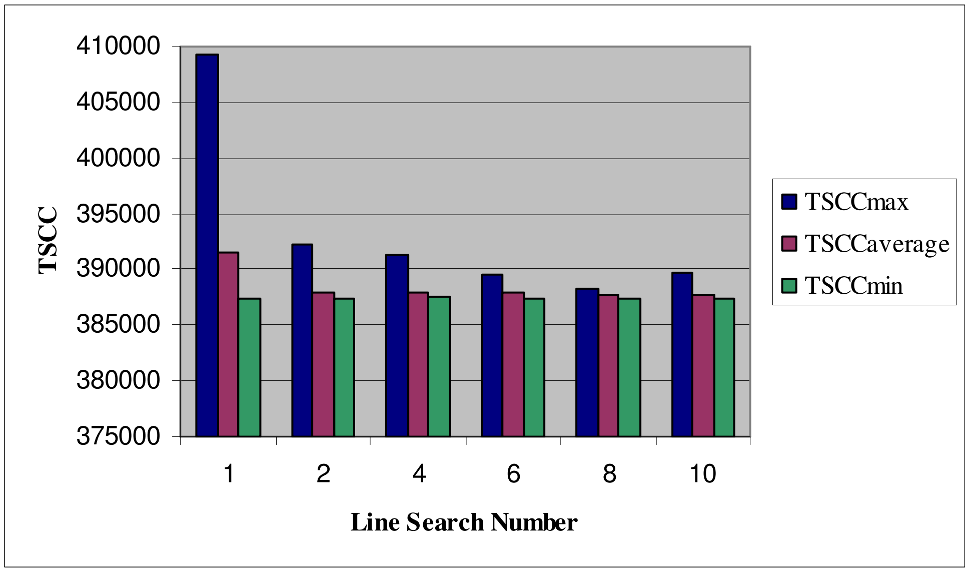


Figure 5-13: The influence of local search number on objective TSCC

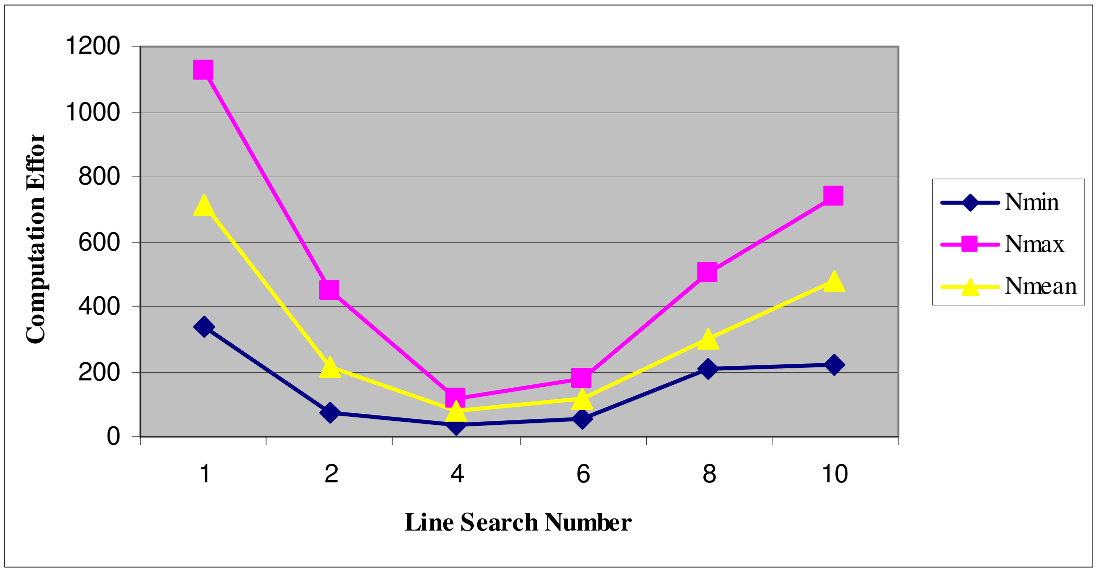


Figure 5-14: The influence of local search number on computation effort of ASPSA-I algorithm

5.10.2 Pre-search Number on Performance of ASPSA-I

Figure 5-15 shows how the pre-search number influences solution quality of ASPSA-I algorithm, from which we can see pre-search number doesn't influence solution quality too much. Too many pre-search numbers will lead to pre-convergence and worsen solution quality.

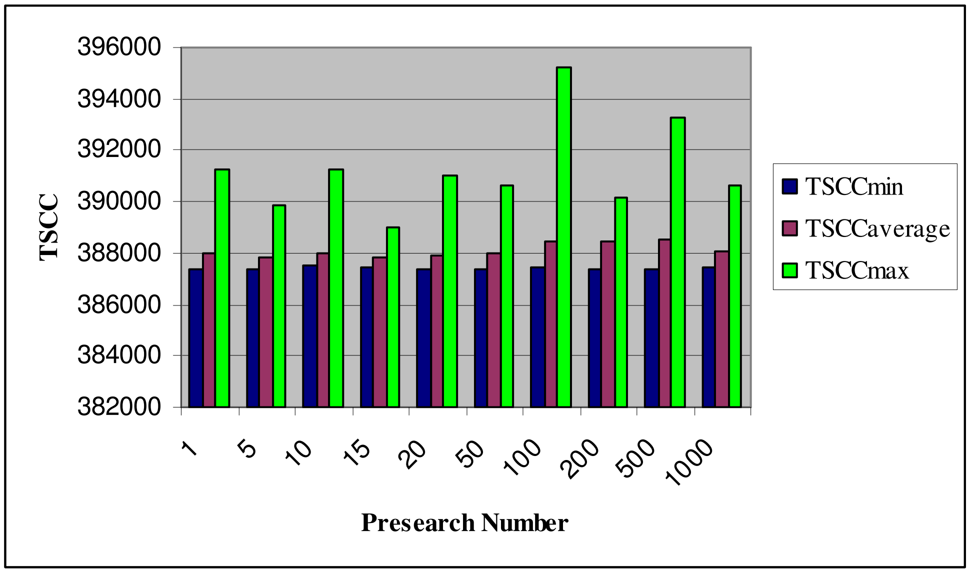


Figure 5-15: The influence of Pre-search number on objective TSCC

Figure 5-16 shows how the pre-search number influences computation effort of ASPSA-I algorithm. It can be noticed that at the beginning computation effort decreases with presearch number, and then slowly increases. Based on the figures, setting pre-search number between 10 and 200 will be recommended.

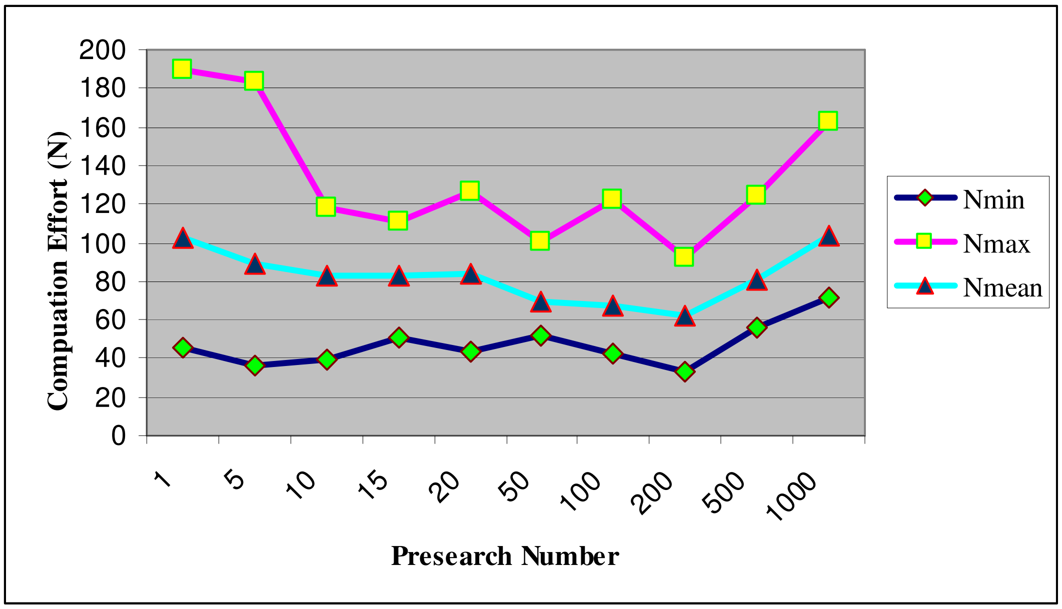


Figure 5-16:The influence of pre-search number on computation effort of ASPSA-I algorithm

5.10.3 Short Simulation Length on Performance of ASPSA-I

As it is explained in Chapter 3, short simulation length will be used to evaluate ranks between different decision variables. This has been adopted in presearch, evaluating gradients and line searching in ASPSA-I algorithm.

In this section, we will study how short simulation length influences the performance of ASPSA-I algorithm. Figure 5-17 shows short simulation run length does not have decisive relationship with solution quality. However, Figure 5-18 indicates that simulation length greatly influence the computation effort of ASPSA-I algorithm. The smaller simulation run length is, the less computation effort will be needed. Therefore,

$\frac{T}{20} \sim \frac{T}{10}$ will be recommended for the short simulation length, where T is the full simulation length.

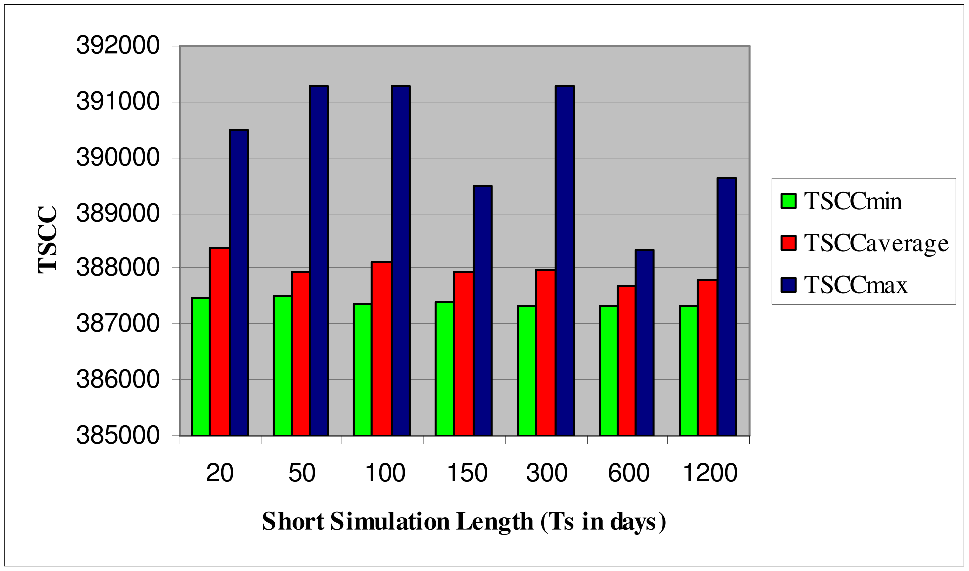


Figure 5-17: The influence of short simulation length on objective TSCC

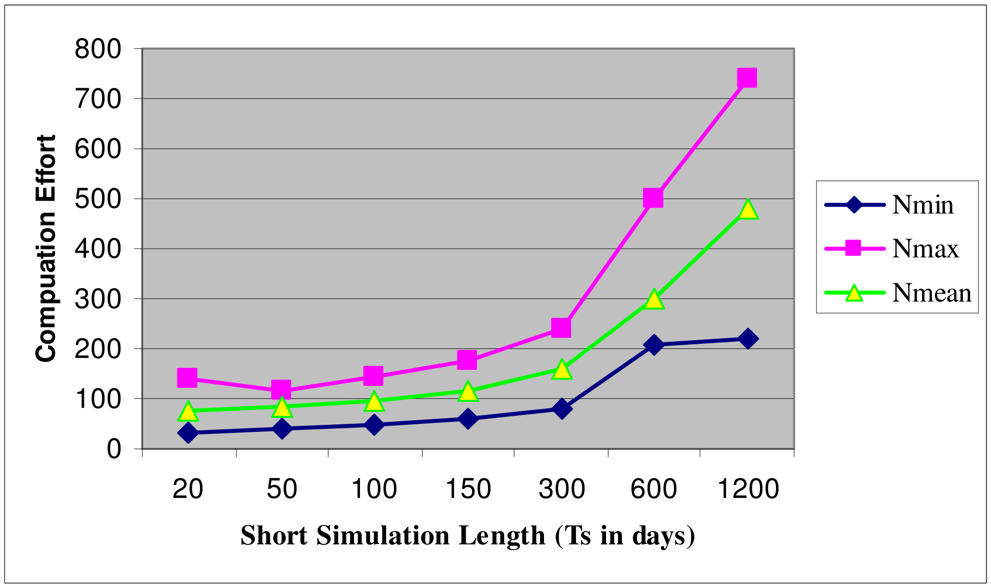


Figure 5-18: The influence of short simulation length on computation effort

Chapter 6

ASPSA-II for Constrained Supply Chain Inventory Optimization Problems

The ultimate purpose of this chapter is to design ASPSA-II algorithm for constrained optimization problems and apply it on discrete constrained supply chain inventory optimization problems. The penalty function method will be used to transfer the constrained optimization problems into unconstrained optimization problems.

6.1 Available Techniques for Constrained Optimization Problems

Although a lot of researches have been done in constrained deterministic optimization (see, e.g., Bertsekas, 1995; Chong and Zak, 1996), little progress can be found for constrained optimization in the stochastic domain. In the area of stochastic approximation (SA), most of the available results are based on the simple idea of projective the estimate θ_n back to the nearest point in G, where G is constraint set (Wang and Spall, 1999). These projection-based SA algorithms are typically showed in Eq. 6-1, where $\pi_G : R^p \to G$ is the set projection operator, and $\hat{g}_n(\theta_n)$ is an estimate of the gradient $g(\theta_n)$. The main difficulty for this projection approach lies in the implementation (calculation) of the projection operator π_G. Except for simple constraints like interval or linear constraints, calculation of $\pi_G(\theta)$ for an arbitrary vector θ is very tough task.

$$\theta_{n+1} = \pi_G[\theta_n - a_n \hat{g}_n(\theta_n)] \qquad \textbf{6-1}$$

There are also some other techniques proposed for dealing with constraints. For example, Hiriat-Urruty (1977) and Pflug (1981) present and analyze a SA algorithm based on the penalty function method for stochastic optimization of a convex function with convex inequality constraints; Kushner and Clark (1978) present several SA algorithms based on the Lagrange multiplier method, the penalty function method and a combination of both. However, the convergence of these SA algorithms based on "non-projection" techniques generally requires complicated assumptions on the cost function L and the constraint set G (Wang and Spall, 1999). Furthermore, the implementation of these algorithms involves choosing various design parameters that greatly affect the efficiency of these algorithms. It is necessary to carry on more systematic studies of the convergence for these algorithms to make them practical. Next, we first discuss the supply chain network we are interested in.

6.2 The Supply Network

6.2.1 Network Topology

The supply chain network considered here is same as the one published by Etll et al. (2000). The network consists of a collection φ of stocking locations, or stores, each of which stocks one type of s.k.u. (Figure **6-2**).

Each store $i \in \varphi$ has a set of supply stores, denoted by $\varphi_{>i}$, which consists of all those stores that directly supply stroe i. Moreover, this one-level BOM also specifies the usage counts, u_{ji}, $j \in \varphi_{>i}$, where u_{ji} denotes the number of s.k.u.s from store j needed to produce one s.k.u. at store i.

By way of symmetry, for each store $i \in \varphi$, denote by $\varphi_{i>}$ consists of at least one output store at the same site. If i is an output store, either $\varphi_{i>}$ is empty, in which case i an end store. The set of end stores is denoted by φ_0. In our study, we will make the following model assumptions:

<u>Model Assumptions</u>

- A single product flows through the supply chain
- Information lead time is negligible or zero
- Every store has a replenishment lead time which includes the production and transportation lead times
- Retailer store faces random customer demand
- Base-stock level at every store takes discrete integer values
- There is no lot-size or discount policy for any store
- When the store has not enough inventory for the order, the unsatisfied order is backlogged until necessary stock becomes available
- An assembly store can begin to make production only if all suppliers has enough

inventory satisfying its requirement

- Linear store holding and shortage cost-rates exist for all the stores in the supply chain
- Transportation cost is directly proportional to the quantity shipped
- All store have infinite capacity
- The most upstream store is linked to an infinite supply source.

6.2.2 Base-Stock Control and Nominal Lead Times

Each store i follows a base-stock control policy for managing inventory. The policy works as follows. When the inventory position (i.e., on-hand plus on-order minus backorder) at store i falls below some specified base-stock level, R_i, a replenishment order is placed.

If store i which has only one supplier and the upstream supplier has on-hand inventory to satisfy store i's order requirement, the order is filled immediately and shipped to store i. The shipping time constitutes the nominal lead time of store i, denoted by L_i. All orders placed by store i have i.i.d. nominal lead times. If the upstream store has a stockout, the order joins a backorder queue at the store, i.e., the order has to wait to get filled.

If store i has more than one supplier, the replenishment order is essentially a production order whereby one s.k.u. of the store i is assembled from the s.k.u.s of the upstream stores, according to the one-level BOM at store i. If all the input stores have stock on-hand, producing starts immediately. (Upon completion, the s.k.u. is placed in the store as part of its on-hand inventory.) The production time constitutes the nominal lead time of the output store (also denoted by L_i). If any one of the supplier stores has a stockout, the production order joins a backorder queue and is not executed until all components needed for assembly are available.

Note that the nominal lead time is different from, but related to, the actual lead time, $\tilde{L}_i$, which is the lead time taking into consideration the possibility of a stockout at the supply stores($\varphi_{>i}$). To be more specific, the actual lead time associated with a store is the difference between the times an order arrives at the store. Clearly, the actual lead time (a random variable) is at least the nominal lead time but may be significantly more, depending on whether a stockout occurs at any of its supply stores.

6.2.3 End Customers, Demands, and Service Levels

Customer classes are identified according to different product types and different service-level requirements. Hence, if two orders are for the same product but with different service-level requirements, they will be classified as two different classes, denoted by M the set of all customer classes. Associated with each customer class $m \in M$ are the end store that supplies the demand, and the transit (delivery) time, T_m. In addition, let W_m denote the waiting time to receive an order for a typical class m customer; then the service-level requirement for class m customers is given by $P(W_m \le \beta_m) \ge \alpha_m$, where α_m and β_m are given parameters: β_m is a possibly random due date for class m orders, while α_m is the fraction of class m orders that are filled before the due date.

There is also a demand stream (forecast or real), $\{D(m,1), D(m,2),....\}$, associated with each class m, where the second argument of $D(m,.\}$, 1, 2,…, denotes the time periods; e.g., day 1, day 2, etc. (Throughout, we use discretized time units to characterize demand.) Note that here we allow the demand data to be random and nonstationary. Assume independence among the demand classes and independence between the demands and nominal lead times.

A base-stock control scheme requires translating the end-customer demand streams into the effective demand stream at each store i, denoted $\{D_i(1), D_i(2), D_i(3),...\}$. Next, for each end store $i \in \varphi_0$, define the effective demand stream as: $D_i(t) := \sum_{m:str(m)=i} D(m, o_m + t) \quad t = 1, \quad 2, \quad 3, \quad ...$, where $str(m)$ denotes the (end) store that supplies the demand class m. Now we can proceed recursively, moving upstream in the BOM, for all other stores in the network. Then, for all stores $i \notin \varphi_0$, define the effective demand stream as: $D_i(t) := \sum_{j \in \varphi_{i>}} u_{ij} D_j(o_j + t) \quad t = 1, \quad 2, \quad 3, \quad ....$

6.3 Problem Formulation

Notations:

i	Index for store number

n	Number of stores in the supply chain
D_i	Demand per period for store i
σ_{D_i}	Standard deviation of demand per period for store i
L_i	Lead time for replenishment for store i
σ_{L_i}	Standard deviation of lead time for store i
N_i	The demands during lead time for store i
u_i	The mean of N_i
σ_i	The standard deviation of N_i
θ_i	Installation base-stock level at store i
θ_i^{UB}	Upper bound for θ_i
θ_i^{LB}	Lower bound for θ_i
h_i	Installation inventory holding cost-rate at store i
b_i	Installation inventory backorder cost-rate at store i
L_i	Replenishment lead time with respect to store i
T_s	Short simulation period
T	Full simulation time period ($T_s < T$)
$I_{j,t}$	Inventory on-hand at store j at the end of time period t
$B_{j,t}$	Backorder at instore j at the end of time period t
m	Index for class m customer

M	Customer class set
α_m	Required customer service level for class m customer, where $m \in M$
β_m	Required due dates for class m customer
W_m	The waiting time to receive an order for a typical class m customer

Therefore the average daily cost (ADC) over all stores in the supply chain over T time periods is given by Eq. 6-2 when we omit transportation cost:

$$ADC = (\sum_{t=1}^{T}\sum_{i=1}^{n}(I_{i,t} \times h_i + B_{i,t} \times b_i))/T \tag{6-2}$$

The objective of this work is to compute optimal base-stock policy $\theta^* = \{\theta_1^*, \theta_2^*, \ldots, \theta_n^*\}$, which is to minimize the expected average inventory capital (ADC) through the network while satisfying customer service-level requirements, as specified by Eq. 6-3 and Eq. 6-4.

$$\min\ ADC = (\sum_{t=1}^{T}\sum_{i=1}^{n}(I_{i,t} \times h_i + B_{i,t} \times b_i))/T \tag{6-3}$$

$$\text{s.t.}\quad P(W_m \le \beta_m) \ge \alpha_m;\ \ \forall m \in M \tag{6-4}$$

6.4 Penalty Function

To solve the proposed constrained optimization problems, we will use the penalty method to transfor constrained optimization problems to nonconstrained optimiztion problems, and then apply the proposed ASPSA-II to solve. Let $f_m = P(W_m \le \beta_m) - \alpha_m$, and the penalty function we used in this resarch is shown in Eq. 6-5.

$$L = \begin{cases} ADC & if\ f_m \geq 0 \quad \forall m \\ ADC + ADC \times penalty\_scaler \times e^{-\sum_{m:f_m<0} f_m} & otherwise \end{cases} \qquad 6\text{-}5$$

6.5 ASPSA-II for Constrained Inventory Optimization Problems

Because constrained optimization problems are more complicated than unconstrained problems, some revision on ASPSA-I is adopted for solving constrained optimization problems.The only difference between ASPSA-I and ASPSA-II is in the presearch function, where ASPSA-II will search feasible solution.

The complete ASPSA-II algorithm for constrained algorithm can be stated in Figure 6-1.

6.6 Experiments and Results

The purpose of this section is to test the performance of ASPSA-II algorithm through numerical examples. In particular, we want to show numerical comparisons between results from the performance evaluation routine and those obtained from GA algorithm. There are two set of experiments will be tested here. The first one is with stationary demand, and the second one is with nonstationary (seasonal) demand.

```
Step 1:   Presearch for feasible solution

          1). Randomly choose a decision point $\theta$
          2). Find feasible solution
              do{
                  Evaluate $\theta$ in full simulation length and check if it is feasible;
                  If $\theta$ is unfeasible
                          $\theta_i = \theta_i + c_i$, where $c_i$ is a constant number for all $i$.
                  } while( $\theta$ is unfeasible);
Step 2:   Generate simultaneous perturbation vector $\Delta_k$
Step 3:   Loss function evaluation
          Evaluate $y(\theta_k + c_k\Delta_k)$ and $y(\theta_k - c_k\Delta_k)$ in short simulation run $n_s$.
Step 4:   Gradient approximation

          $\hat{g}_k = \begin{cases} 1/\Delta_k & \text{if} \quad y(\theta + c_k\Delta_k) > y(\theta - c_k\Delta_k) \\ -1/\Delta_k & \text{otherwise} \end{cases}$

Step 5:   Line search

          For $j = 1, \quad K \quad l$

              4). Randomly $a_i^j$ $(i = 1, \quad K, \quad p)$ from some distribution

              5). Calculate $\theta_{k+1,i}^j = \theta_{k,i} - a_i^j \hat{g}_{ki}$

              6). Run short simulation ($n_s$) length to evaluate $J(\theta^j)$

           Let $J_b = \min\{J_j, j = 1, \quad K, \quad l\}$;
           Run full simulation length to evaluate $J(\theta_b)$;
           If $J_{best} > J_b$, $J_{best} = J_b$ and $\theta_{best} = \theta_{k+1} = \theta_b$;
            Otherwise, $\theta_{k+1} = \theta_{best} + \Lambda_k$ where $\Lambda_k$ is a perturbation vector
Step 6:   Iteration or termination
          $k \leftarrow k + 1$. If stop criteria is satisfied, stop. Otherwise, go to step 3.
```

Figure 6-1: ASPSA-II for constrained optimization problems

6.6.1 Experiment 1 (with stable demand)

The supply chain network of experiment 1 is described in Figure 6-2. The lead time settings are listed in Table 6-1. $\mu_{ij} = 1$ for all i and j. Table 6-2 shows the two setting of holding cost and backorder cost. In addition, five different demand settings will be tested in this experiment (Table 6-3).

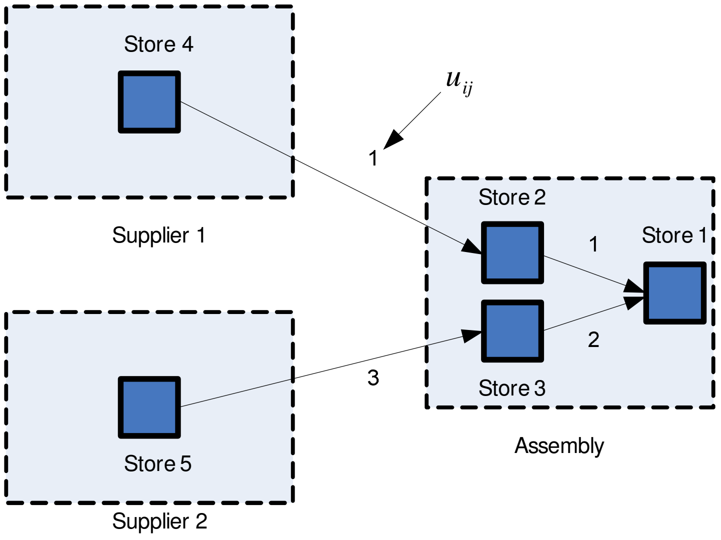


Figure 6-2: Supply chain network of example 1

Shipments of finished products to customers follow a normal distribution with mean 4 and standard deviation 0.5. There is a single type of finished produce (at store 1) and a single customer demand stream associated with it. For the customer service level, $\beta = 5$ and $\alpha = 99\%$ are chosen.

The first set of experiments investigate the issue of whether the solutions, in particular the base-stock levels, produced by ASPSA-II algorithm are comparable to those found by GA. The parameters setting of ASPSA-II algorithm and GA algorithm are listed in Table 6-4 and Table 6-5 respectively.

Table 6-1: Nominal lead time distributions used in example 1

Store *i*	**Lead Time Distribution**		
	Distribution Type	Mean	Std. Dev.
1	Normal	3.0	1.0
2	Normal	8.0	1.0
3	Normal	10.0	1.0
4	Constant	6.0	0
5	Constant	10.0	0

Table 6-2: Demand settings used in example 1

Settings	**Distribution Type**	**Mean**	**Std. Dev**
D1	Normal	10	4.6
D2	Normal	20	9.1
D3	Normal	30	13.7
D4	Normal	40	18.3
D5	Normal	50	22.8

Table 6-3: Supply chain cost settings

Setting	**costs**	**store -5**	**store -4**	**store -3**	**store -2**	**store -1**	**Ratio of b_j/h_j**
CS1	h_j	1	2	4	8	15	2
	b_j	2	4	8	16	30	
CS2	h_j	1	2	4	8	15	10
	b_j	10	20	40	80	150	

Table 6-4: Parameters setting of ASPSA-II algorithm for experiment 1

Name	Settings
Δ_k	Bernoulli distribution
c_k	Constant, and $c_k = 1$
a_{ki}^{j}	A uniform distributed integer chosen from $(0, \left\lceil \frac{10}{\sqrt{10+k}} \right\rceil (j+1))$
T_s	50
T	1000
m	4
Stopping criteria	$k \geq 1000$ or θ_{best} is not improved in 20 consecutive iterations

Table 6-5: Parameters setting of GA for experiment 1

Name	Settings
Population size	30
Are decision variables bound rigid?	Yes
Parameter-space niching to be done?	Yes
Niching parameter value	10
Runs number	1
Crossover probability	0.9
Mutation probability	0.2
Random seed	0.2
Distribution index for SBX and mutation	20 and 100
Stopping criteria	100 generations

The supply chian is simulated for the store base-stock levels yielded by a solution technique corresponding to a given supply chain test problem. The supply chain is simulated for a given full length 1200 days and each simulation is replicated 10 times with 10 different random number streams of customer demands. The ADC over full length $T = 1200$ days is computed, and the mean ADC over 10 replications is finally computed corresponding to a given set of store base-stock levels.

The performance of heuristic methodologies is evaluated by proposed 10 supply chain settings. The performance of the heuristic methodology is analyzed by the quality of the solutions yielded and the computational effort required to obtain the solution. Table **6-6** listed the running results for the 10 supply chain settings. From Table **6-6**, we can see that solution quality of ASPSA-II algorithm is not so good as those achieved by GA, but GA takes much more computation effort (over 5 times). It should be pointed out that the inferior of solution quality of ASPSA-II to GA is not so large (between 0.63% and 9.12%), on some cases, ASPSA-II performs a litter better than GA such as for CS1_D5 case (0.64% better).

Figure 6-3, Figure 6-4 and Figure 6-5 show the optimization process of best ADC, penalized ADC and decision variables. From them, we can see that ASPSA-II algorithm still converge to the optimal solution for constrained optimization problems. In addition, we can see that during presearch line search phrase, the objective ADC decreases very quickly. However, in ASPSA-II search phrase, ADC decreases not so quickly. This can be due to complicacy from constraints. Another finding will be in the final phase, searching is jumping between feasible space and infeasible space in a large frequency (Figure 6-4).

Table 6-6: Comparison between proposed ASPSA-II algorithm and GA algorithm

Settings	Methods	$R_i, i = 1, \quad K, \quad 5$	ADC	Ser. Lev.	Com. Eff.
CS1_D1	ASPSA-II	{45, 95, 229, 152, 301}	976.4	0.989	168
	GA	{45, 102, 238, 126, 231}	930.3	0.992	3000
CS1_D2	ASPSA-II	{89, 221, 534, 227, 604}	2205.6	0.99	224.5
	GA	{93, 198, 560, 282, 403}	2021.1	0.99	3000
CS1_D3	ASPSA-II	{127, 359, 750, 377, 828}	3146.7	0.989	266
	GA	{139, 311, 879, 418, 532}	3133.1	0.989	3000
CS1_D4	ASPSA-II	{203,376, 965, 565, 929}	3874.6	0.989	451
	GA	{192, 417, 1007, 540, 914}	3880.8	0.991	3000
CS1_D5	ASPSA-II	{242, 512, 1264, 612, 1275}	5034.8	0.99	619
	GA	{237, 518, 1390, 665, 1024}	5067	0.99	3000
CS2_D1	ASPSA-II	{47, 107, 244, 162, 325}	1203.8	0.995	93
	GA	{43, 101, 248, 152, 241}	1126.3	0.990	3000
CS2_D2	ASPSA-II	{93, 209, 506, 322, 726}	2471.7	0.994	133
	GA	{92, 199, 496, 347, 513}	2265.1	0.99	3000
CS2_D3	ASPSA-II	{141, 315, 746, 499, 903}	3540.6	0.993	281
	GA	{143, 307, 754, 499, 751}	3430.5	0.990	3000
CS2_D4	ASPSA-II	{180, 425, 1025, 653, 955}	4675.1	0.991	335
	GA	{188, 414, 1003, 659, 1019}	4529.7	0.993	3000
CS2_D5	ASPSA-II	{245, 512, 1306, 817, 1386}	5871.6	0.994	677
	GA	{227, 513, 1248, 834, 1317}	5737.98	0.991	3000

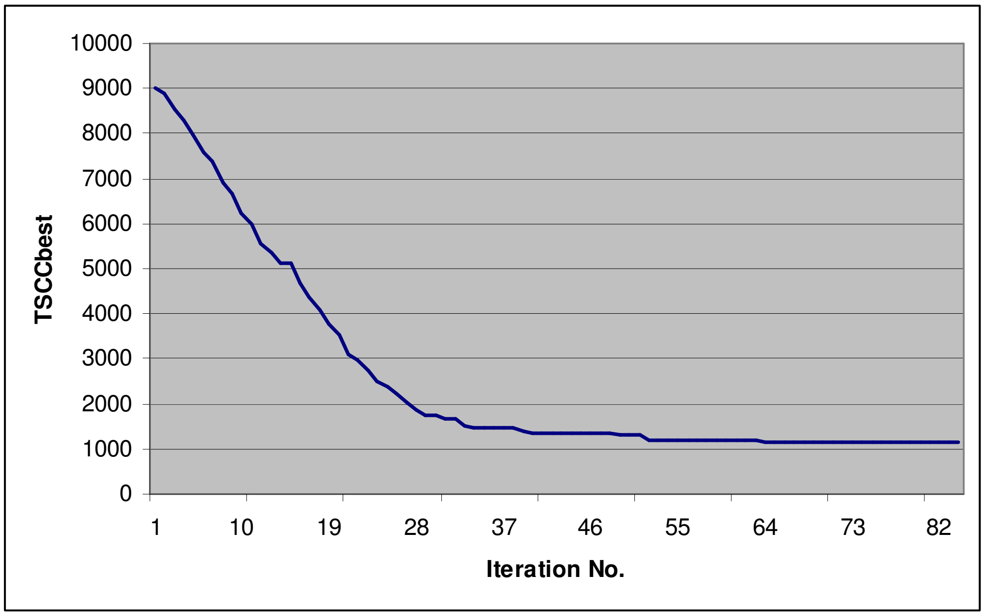


Figure 6-3: Best ADC optimization trace of ASPSA-II algorithms for CS1_D1 case

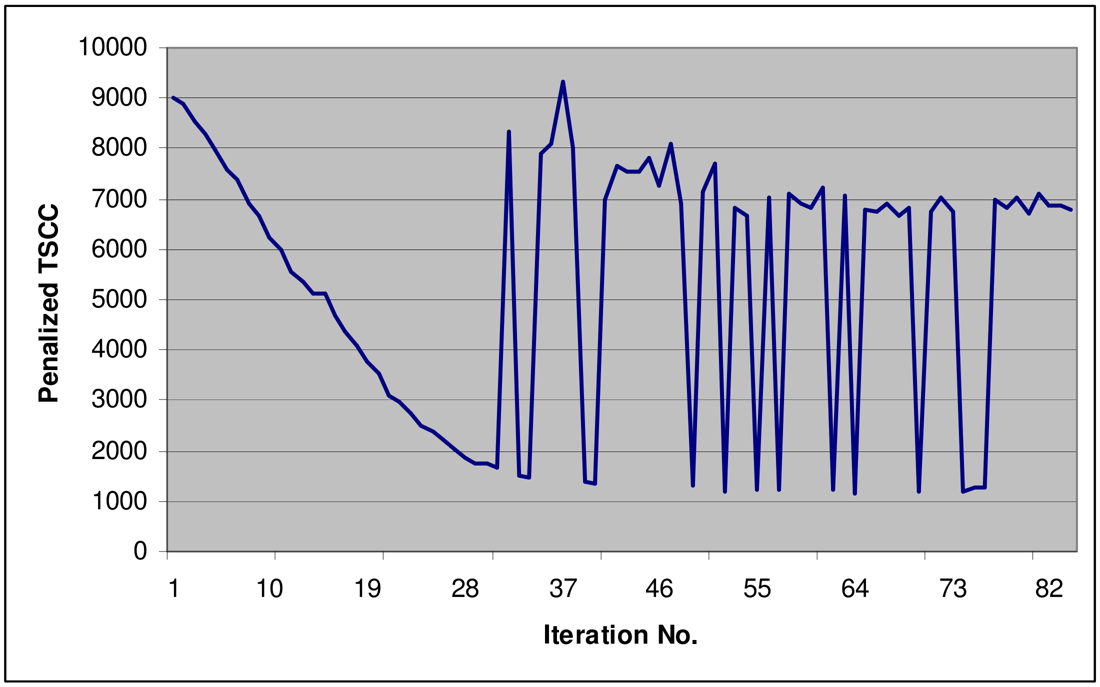


Figure 6-4: Penalized ADC optimization trace of ASPSA-II algorithms for CS1_D1 case

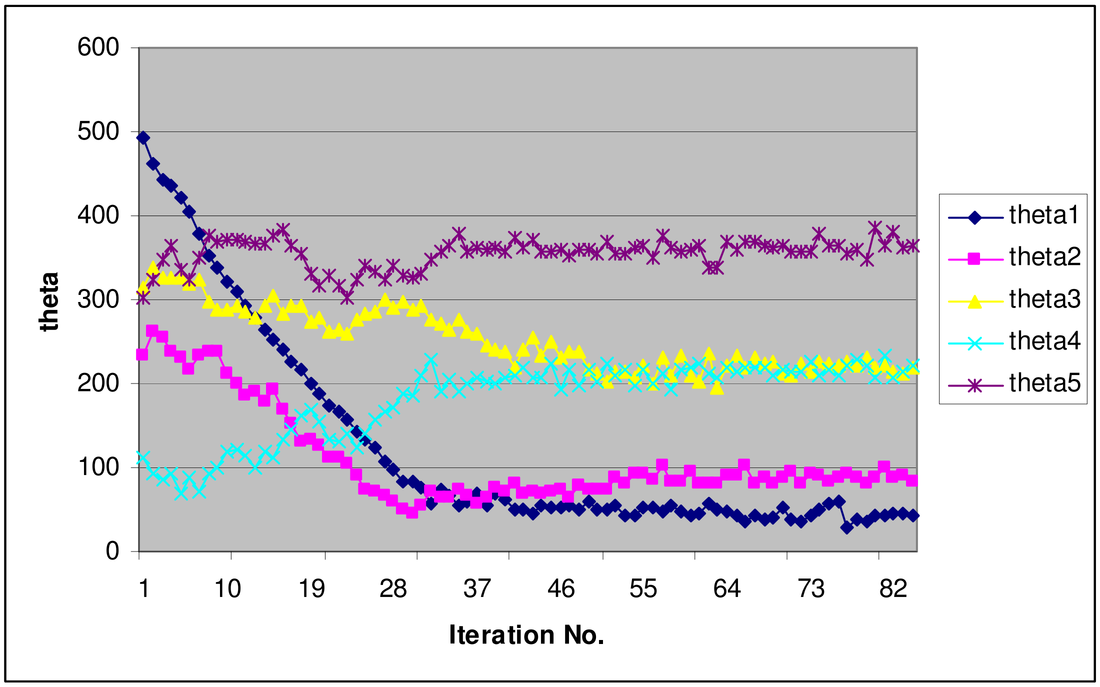


Figure 6-5: Decision variables optimization trace of ASPSA-II algorithms for CS1_D1

6.6.2 Experiment 2 (with seasonal demand)

We are also interested in the performance of ASPSA-II algorithm for supply chain inventory optimization problems with non-stationary demand. Table 6-7 lists three different demand situations, and the average demand for each season is 10. We still use the supply chain network given in experiment 1 with CS2 cost settings. Table 6-8 shows the parameter settings of ASPSA-II algorithm for three cases.

Table 6-9 shows the running results for the above three demand settings. We can see that the bigger of demand fluctuates, the higher of inventory (ADC) cost. Figure 6-6 also shows how the base-stock inventory level varies with the season at store 1, and we can see demand fluctuation directly influences base-stock level at that store.

Table 6-7: Mean demand used in experiment 2; standard deviation $\sigma = u/10$

Demand Scenario	**Mean Demand per Season(u)**			
	Season 1	Season 2	Season 3	Season 4
Stationary	10	10	10	10
High Fluctuation	10	5	15	10
Low Fluctuation	10.5	11	9.5	9

Table 6-8: Parameters setting of ASPSA-II algorithm for experiment 2

Name	**Settings**
Δ_k	Bernoulli distribution
c_k	Constant, and $c_k = 1$
a_i^j	A uniform distributed integer chosen from $(0, (j+2))$
T_s	1000
T	4000
Stopping criteria	$k \geq 1000$ or θ_{best} is not improved in 20 consecutive iterations

Table 6-9: Running results found by ASPSA-II algorithm for experiment 2

Demand Scenario	**ADC**	**Computation Effort**	**Customer Service Level**
Stationary	482.2	149	0.998
Low Fluctuation	620.1	271	0.999
High Fluctuation	1115.2	228	0.997

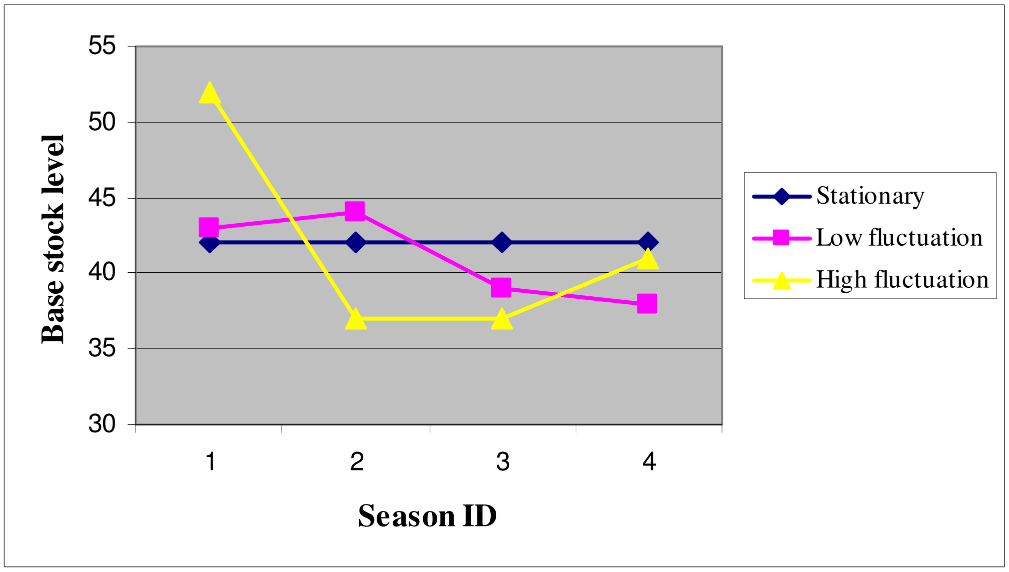


Figure 6-6: Base-stock level of store 1 during four seasons

In conclusion, this chapter proposes ASPSA-II for constrained supply chain inventory optimization problems. And the algorithm is tested in two set of experiments. The first set of experiments is with stationary demands. And the second one is with seasonal demands. Compared against GA, ASPSA-II can also achieve same level solution quality with much fewer simulation efforts. Moreover, this new simulation optimization method- ASPSA-II, can also works for seasonal demands, which is very common in reality. Until now, we have finished numerical experiments. Hence, we will obtain our conclusions in next chapter.

Chapter 7

Conclusion and Future Research Direction

We get our conclusion in this chapter. In addition, it also summarizes the contributions of the research. Some further research directions are also discussed.

7.1 Conclusion

In this study, a new and efficient simulation optimization method-Augmented Spontaneous Perturbation Stochastic Approximation (ASPSA) algorithm is proposed. ASPSA is based on a gradient-based method-SPSA (Spall, 1992), but it also integrates presearch, some idea in ordinal optimization (ranking is easier than valuing), line search, and uncommon gain into SPSA algorithm. In detail, through manipulating simulation length during optimization process, ASPSA achieves speedup; and through added functions, ASPSA improve solution quality. Convergence of ASPSA algorithm is also proved in Chapter 4 from random search theory standpoint.

Two versions of ASPSA are proposed here. ASPSA-I is designed for unconstrained discrete optimization problems, and ASPSA-II is for constrained discrete optimization problems. The difference between ASPSA-I and ASPSA-II is on the prsearch phrase. The former adopts random search algorithm in presearch, however, the later employs line search in presearch because of complexity induced by constraints.

The performance of ASPSA is tested by discrete supply chain inventory optimization problems (serial supply chain network without constraints and fork-join supply chain network with customer service level constraints). The numerical experiments show that ASPSA is competitive to Genetic Algorithms and Simulated Annealing in solution quality (only at most 6% worse in tested problems). However, ASPSA uses much less computation effort (20 times less than GAs in the tested problems). In addition, we find that ASPSA-II converges not so fast as ASPSA-I for the unconstrained problems.

Moreover, this new and efficient algorithm-ASPSA, also answered two fundamental questions in optimization area: 1) If the gradient-based algorithm can be applied to realistic discrete optimization problems; 2) If the simulation length is important to solution quality in simulation optimization algorithms, considering the confidence level of simulation results is proportion to $\frac{1}{\sqrt{n}}$. The answer to first question is "yes", and the answer to the second one is "no". Based on what has been found in this study, we can conclude that this research will impact the simulation optimization field.

7.2 Research Contributions

The outcome of this research has contributed o two major fields: 1) Simulation Optimization; 2) Application of ASPSA on solving realistic complex discrete supply chain inventory optimization problems (unconstrained and constrained).

7.2.1 Contributions to Simulation Optimization

ASPSA contributes to simulation optimization in two aspects: good solution quality and less computation effort. This can be credited to the integrated techniques, and they are:

1). Presearch to find better initial solution

2). Ordinal Optimization to save simulation resources (ranking is more easier than valuing)

3). Uncommon gain to correct gradient approximation

4). Line search to speed up searching process

5). Long memory of best solution to guarantee convergence

Numerical studies found that gradient-like ASPSA-I can not only solve the discrete unconstrained inventory problems with high solution quality (comparable to GAs), but also spends less computation effort than GAs (20 times less). In addition, compared with SPSA, ASPSA-I achieves better solution quality with less computation effort. The improvement of ASPSA against SPSA can be credited to added functions. This is also the first research to combine ordinal optimization with SPSA. Although the confidence level of simulation is proportion to $\frac{1}{\sqrt{n}}$, this research proves that the final solution quality doesn't have direct relationship with simulation length.

In Chapter 6, ASPSA-II is proposed to solve a complex discrete constrained general (fork-join) supply chain optimization problem with customer service level requirement, where demands could be unstationary (seasonal). Also the numerical experiments show that ASPSA-II is comparable to GAs (not worsen than 6.01%) in solution quality using much less computation effort. In addition, for nonstationary (seasonal) demands, ASPSA-II doesn't require any assumptions as analytical methods do.

7.2.2 Contribution to Discrete Supply Chain Optimization Problems

The contributions of this research for supply chain optimization can be found in three aspects:

1) A new efficient simulation optimization method, ASPSA, is found for discrete supply chain optimization problems.

2) This new method, ASPSA, can solve the constrained supply chain optimization problems for a general (fork-join) network without stringent assumptions.

3) ASPSA can also solve the supply chain optimization problems with nonstationary demands.

Numerical experiments have shown that ASPSA is much more efficient than popular methods like GAs and SA. In addition, the solution quality got by ASPSA is comparable to GAs and SA. This makes ASPSA is more attractive for the large supply chain network problems.

Moreover, compared with dominant analytical methods in supply chain inventory optimization; ASPSA doesn't need any assumptions about supply chain network. It can handle the general (fork-join) network even with unstationary demands.

7.3 Future Research

From the study, we found the followings questions needed to answer in the future:

1. ASPSA algorithm convergences much faster in unconstrained optimization problems than on constrained optimization problems. Therefore, the first question in the future needed to answer is how to speed up ASPSA convergence rate for constrained optimization problems.
2. Now the memory only remembers the best solution found until now, not fully using the previous search results. Therefore, how to use fully explore the previous search results is another important research problem.
3. ASPSA explores ordinal optimization to speed up searching process. Similarly, it will be interesting to see how GA explores ordinal optimization to speed up searching process.
4. In proposed ASPSA, decision variables bounds are very influential for final solution quality. Another question in the future needed to answer is how to make ASPSA still work very well without knowing decision variable bounds.

Bibliography

Agrell, P. J. 1995. A Multicriteria Framework for Inventory Control, International Journal of Production Economics, 41, 59-70.

Alessandri, A., and T. Parisini. 1997. Nonlinear Modeling of Complex Large-Scale Plants Using Neural Networks and Stochastic Approximation. *IEEE Transactions on Systems*, *Man, Cybernetics*, 27, 750-757.

Andradóttir, S. 1998a. A Review of Simulation Optimization Techniques, *Proceedings of the 1998 Winter Simulation Conference*, 151-158.

Andradóttir, S. 1998b. "Simulation Optimization," in *Handbook of Simulation: Principles, Methodology, Advances, Applications, and Practice* (J. Banks, ed.), Wiley, New York, Chapter 9.

April, J. 2003. Practical Introduction to Simulation Optimization, *Proceedings of the 2003 Winter Simulation Conference*, 71-78.

Arslan, H., S. C. Graves and T. Roemer. 2005. A Single-Product Inventory Model for Multiple Demand Classes. Working Paper, MIT.

Baltcheva and Cristea 2000.

Bertsekas, D. P. 1995. *Nonlinear Programming*, Belmont, Massachusetts: Athena Scientific.

Bhatnagar, S., M. C. Fu, S. Marcus, and S., Bhatnagar. 2001. Two-timescale Algorithm for Simulation Optimization of Hidden Markov Models. *IIE Transactions*, 33, 245-258.

Bhatnagar, S. and V. S. Borkar. 2003. Multiscale Chaotic SPSA and Smoothed Functional Algorithms for Simulation Optimization. *Simulation*, 79(10), 568-580.

Bonfinger, E. and G. J. LEWIS. 1992. Two-stage Procedures for Multiple Comparisons with a Control, *American Journal of Mathematical and Management Sciences*, 12, 253-275.

Burnett, R., 2004. Application of Stochastic Optimization to Collision Avoidance, *Proceedings of the 2004 American Control Conference*, 2789-2794.

Buzacott, J. A. and G. J. SHANTHIKUMAR. 1993. *Stochastic Models of Manufacturing Systems*, Englewood Cliffs, NJ: Prentice Hall.

Cakanyidirim, M. and S. Luo. 2005. (R, Q) Policy with Lead Time Options, Available athttp://som.utdallas.edu/faculty/working_papers/SOM200538.pdf.

Carson, J. S. 1996. AutoStat Output Statistical Analysis for AutoMod Users, *Proceedings of the 1996 Winter Simulation Conference*, 492-499.

Carson, Y. and A. Maria. 1997. Simulation Optimization: methods and applications, *Proceedings of the 1997 Winter Simulation Conference*, 118-126.

Cauwenberghs, G. 1994. *Analog VLSI Autonomous Systems for Learning and Optimization*, Ph. D Dessertation, California Institute of Technology.

Cerny, V. 1985. Thermodynamical Approach to the Traveling Salesman Problem: an Efficient Simulation Algorithm, *Journal of Optimization Theory and Applications*, 45, 41-51.

Chen, C. H., J. Lin, E. Yücesan, and S. E. Chick. 2000a. Simulation Budget Allocation for Further Enhancing the Efficiency of Ordinal Optimization, *Journal of Discrete Event Dynamic Systems: Theory and Applications*, 10(3), 251-270.

Chen, C. H., H. C. Chen, and E. Yucesan. 2000b. Computing Efforts Allocation for Ordinal Optimization and Discrete Event Simulation, *IEEE Transactions on Automatic Control*, 45(5), 960-964.

Chin, D. C. 1999. The Simultaneous Perturbation Method for Processing Magnetospheric Images, *Optical Engineering*, 38(4), 606-611.

Chong, E. K. P. and S. H. Zak. 1996. *An Introduction to Optimization*, New York: John Wiley and Sons.

Clark, A. J., and H. Scarf. 1958. Optimal Policies for a Multiechelon Inventory Problem, *Management Science*, 6, 475-490.

Cohen, M. A. and H. L. Lee. 1988. Strategic Analysis of Integrated Production-distribution Systems: Models and Methods. *Journal of Operations Research*, 36(2), 216-228.

Cohen, M. A. and S. Moon. 1990. Impact of Production Scale Economies, Manufacturing Complexity and Transportation Costs on Supply Chain Facility Networks, *Journal of Manufacturing and Operations Management*, 3 (4), 269-292.

Dai, L. 1995. Convergence Properties of Ordinal Comparison in the Simulation of Discrete Event Dynamic Systems. *Proceeding of the IEEE conference on Decision and Control*, 2604-2609.

Damerdji, H., and M. K. Nakayama. 1999. Two-stage Multi-comparison Procedure for Steady-sate Simulations, *ACM Transaction on Modeling and Computer Simulations*, 9, 1-30.

Daniel, J. S. R. and C. Rajendran. 2005a. Heuristic Approaches to Determine Base-stock Levels in a Serial Supply Chain with a Single objective and with Multiple Objectives, *European Journal of Operational Research* (in press).

Daniel, J. S. R. and C. Rajendran. 2005b. Determination of Base-stock Levels in a Serial Supply Chain: a Simulation-based Simulated Annealing Heuristics, *International Journal of Logistics Systems and Management,* 1, 149-186.

Dennis, J. E. and D. J. Schnabel. 1989, "A view of Unconstrained Optimization" in Optimization (Handbooks in OR and MS, vol. 1, G. L. Nemhauser et al., eds.), Elsevier, New York, 1-72.

De Souza, P. S. and S. N. Talukdar. 1991. Genetic Algorithms in Asynchronous Teams, *Proceedings of the 4th International Conference on Genetic Algorithms*, 392-397.

Dudewicz, E. J., and Dalal, S.R., 1975. Allocation of Observations in Ranking and Selection with Unequal Variance, *The Indian Journal of Statistics*, 37B, 28-78.

Dunnet, C. W., 1955. A Multiple Comparisons Procedure for Comparing Several Treatments with a Control, *Journal of American Statistical Association*, 78, 965-971.

Duenyas, I., 1994, A Simple Release Policy for Networks of Queues with Controllable Inputs, *Operation research,* 42**,** 1162-1171.

Ermakov, S. M. and N. K. Krivulin. 1995, Efficient Parallel Algorithms for Tandem Queues System Simulation, *Proceedings of the 3rd Beijing Conference on System simulation and Scientific Computing*, 8-12.

Ettl, M., G. E., Feigin, G. Y. Lin and D. D. Yao. 2000. A Supply Chain Network Model with Base-stock Control and Service Requirements, *Operation Research*, 48(2), 216-232.

Ferris, M. C. , T. S. Munso and K. Sinapiromsaran. 2000. A Practical Approach to Sample-Path Simulation Optimization, *Proceedings of the 2000 Winter Simulation Conference*, 795-804.

Fogel, D. B., 1994. An Introduction to Simulated Evolutionary Optimization, *IEEE Trans. on Neural Networks: Special Issue on Evolutionary Computation*, 5(1), 3-14.

Fogel, L. J., A. J. Owens and M. J. Walsh. 1966. *Artificial Intelligence through Simulated Evolution* , New York: John Wiley.

Framinan, J. M., R. Ruiz-Usano, and R. Leisten. 2000. Input Control and Dispatching Rules in a Dynamic CONWIP Flow-shop, *International Journal of Production Research,* 38, 4589-4598.

Fu, M. C., 1990. Convergence of a Stochastic Approximation Algorithm for the GI/G/1 Queue Using Infinitesimal Perturbation Analysis, *Journal of Optimization Theory and Applications*, 65, 149-160.

Fu, M. C., 1994. A Tutorial Review of Techniques for Simulation Optimization, *Proceedings of the 1994 Winter Simulation Conference*, 149-156.

Fu, M. C. and K. J. Healy. 1992, Simulation Optimization of (s, S) Inventory Systems, *Proceedings of the 1992 Winter Simulation Conference*, 506-514.

Fu, M. C., and S. D. Hill. 1997. Optimization of Discrete Event Systems via Simultaneous Perturbation Stochastic Approximation, *IIE Transactions*, 29, 233-243.

Fu, M. C. and J.-Q. Hu. 1997. *Conditional Monte Carlo: Gradient Estimation and Optimization Applications*, Kluwer Academic, MA: Boston.

Fu, M. C. 2002. Optimization for Simulation: Theory vs. Practice. *INFORMS Journal on Computing*, 14(3), 192-215.

Fu, M. C. 2005. Some Topics in Optimization for Simulation. Available at http://www.ece.northwestern.edu/~nocedal/acnw/slides/ACNW05_mfu.pdf.

Gerencsėr, L. 1998. The Use of the SPSA Method in ECG Analysis, *IEEE Transactions on Biomedical Engineering* (in press).

Gerencsėr, L. and S. D. Hill. 1999. Optimization over Discrete Sets via SPSA, *Proceedings of the 38th Conference on Decisions & Control*, 1791-1794.

Gerencsėr, L., S. D. Hill and Z. Vágó. 2001. Optimization over Discrete Sets via SPSA, *Proceedings of the American Control Conference*, 1503-1504.

Gerencsėr, L., S. D. Hill, Z. Vágó and V. Z. Vince. 2004. Discrete optimization, SPSA and Markov Chain Monte Carlo methods, *Proceedings of the 2004 American Control Conference*, 3814-3819.

Glasserman, P. and S. Tayur. 1994. The Stability of a Capacitated Multi-echelon Production Inventory System under a Base-stock Policy, *Operations Research,*42, 913-924.

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

Glover, F. 1990. Tabu Search – Part II, *ORSA Journal on Computing*, 2 (3), 4-32.

Glover, F. J. P. Kelly and M. Laguna. 1996. New Advances and Applications of Combining Simulation and Optimization, *Proceedings of the 1996 Winter Simulation Conference*, 144-152.

Glynn, P. W. 1989a. Likelihood Ratio Derivative Estimators for Stochastic Systems, *Proceedings of the 1989 Winter Simulation Conference,* 374-380.

Glynn, P. W. 1989b. Optimization of Stochastic Systems Via Simulation, *Proceedings of the 1989 Winter Simulation Conference,* 90-105.

Goldberg, D. 1989. *Genetic Algorithms in Search, Optimization & Machine Learning*, Addison-Wesley Publishing Company, MA: Reading.

Graves, S. 1999. A Single-Item Inventory Model for a Non-Stationary Demand Process, *Manufacturing & Service Operations Management*, 1(1), 50-61.

Graves, S. and S. P. Willems. 2000. Optimizing Strategic Safety Stock Placement in Supply Chains, *Manufacturing & Service Operations Management*, 2(1), 68-83.

Gürkan, G., A. Y. Özge and S. M. Robinson. 1994. Sample-Path Optimization in Simulation, *Proceedings of the 1994 Winter Simulation Conference*, 247-254.

Hall, J. D., R. O. Bowden and J. M. Usher. 1997. Using Evolution Strategies and Simulation to Optimize a Pull Production System, *Journal of Materials Processing Technology*, 61, 47-52.

Hall, J. D., and R. O. Bowden. 1997. Simulation Optimization by Direct Search: a Comparative Study. *Proceedings of the Sixth International Industrial Engineering Research Conference*, 298-303.

Hannssmann, F. 1962. *Operations Research in Production and Inventory Control*, New York: John Wiley & Sons Inc.

Hill, S. D., L. Gerencsèr and V. Z. Vince. 2003. Stochastic Approximation on Discrete Sets Using Simultaneous Perturbation Difference Approximations, *2003 conference on Information Science and Systems*.

Hiriart-Urruty, J. 1977. "Algorithms of penalization type and of dual type for the solution of stochastic optimization problems with stochastic constraints," in Recent Developments in Statistics (J. R. Barra, ed.), North Holland Publishing Company.

Ho, Y. C. and X. R. Cao. 1991. *Perturbation Analysis of Discrete Event Dynamic Systems,* Kluwer Academic Publishers.

Ho, Y. C., R. Sreenivas and P. Vakili. 1992. Ordinal Optimization of DEDS, *Discrete Event Dynamic Systems: Theory and Applications*, 2, 61-88.

Ho, Y. C., Cassandras, C. G., Chen, C.H., and Dai, L.H., 2000. Ordinal Optimization and Simulation. Journal of Operations Research Society, 51, 490-500.

Holland, J. H. 1975. *Adaptation in Natural and Artificial Systems*, Ann Arbor: The University of Michigan Press.

Holland, J. H. 1992. Genetic Algorithms, *Scientific American*, July, 66-72.

Hopkins, H. S. 1997. *Experimental Measurement of 4-D Phase Space Map of a Heavy Ion Beam*, Ph. D thesis, Dept. of Nuclear Engineering, University of California-Berkeley.

Hopp, W. J. and M. L. Spearman. 1995, *Factory Physics*, Chicago, IL: Irwin.

Hopp, W. J. and M. L. Roof. 1998, Setting WIP levels with Statistical Throughput Control (STC) in CONWIP Production Lines, *International Journal of Production Research,* 36**,** 867-882.

Hsu, J. C. 1984. Constrained Simultaneous Confidence Intervals for Multiple Comparisons with the Best, *Annals of Statistics*, 12, 1136-1144.

Hsu, J. C. and B. L. Nelson. 1988. Optimization over a Finite Number of System Designs with One-stage Sampling and Multiple Comparisons with the Best, *Proceedings of the 1988 Winter Simulation Conference*, 451-457.

Hutchison, D. W. and S. D. Hill. 2001. Simulation Optimization of Airplane Delay with Delay, *Proceedings of the 2001 Winter Simulation Conference*, 1017-1022.

Hutchison, D. W. 2002. On an Efficient Distribution of Perturbations for Simulation Optimization using Simultaneous Perturbation Stochastic Approximation, *Proceedings of the IASTED International Conference APPLIED MODELING AND SIMULATION*, 440-445.

Jacobson, S. H. and L. W. Schruben. 1988. Simulation Response Gradient Direction Estimation in the Frequency Domain*, Operations Research.*

Jacobson, S. H. and 1989. Oscillation Amplitude Considerations in Frequency Domain Experiments, *Proceedings of the 1989 Winter Simulation Conference,* 406-410.

Jacobson, S. H., D. J. Morrice and L. W. Schruben. 1988. The Global Simulation Clock as Frequency Domain Experiment Index, *Proceedings of the 1988 Winter Simulation Conference,* 558-563.

Jung, J. Y., G. Blau, J. F. Pekny, G. V. Reklaitis and D. Eversdyk. 2004. A Simulation based Optimization Approach to Supply Chain Management under Demand Uncertainty, *Computers and Chemical Engineering*, 28, 2087-2106.

Kapuscinski, R. and S. R. Tayur. 1999. "Optimal Policies and Simulation based Optimization for Capacitated Production Inventory Systems" in Quantitative Models for Supply Chain Management (Chapter 2 in S. R. Tayur, R. Ganeshan, M.J. Magazine, eds.), Klumer Academic, Boston, MA.

Kiefer, J. and J. Wolfowitz. 1952. Stochastic Estimation of a Regression Function, *Annals of Mathematical Statistics*, 23, 462-466.

Kirkpatrick, S., C. D. Gelatt, and M. P. Vecchi. 1983. Optimization by Simulated Annealing. *Science*, 220, 671-680.

Kleijner, J. P. C., 1998. "Experimental Design for Sensitivity Analysis, Optimization, and Validation of Simulation Models." in Handbook of Simulation: Principles, Methodology, Advances, Applications, and Practice (Chapter 6 in J. Banks, ed.), John Wiley & Sons, New York.

Kleinman, N. L., S. D. Hill and V. A. Ilenda. 1997. SPSA/SIMMOD Optimization of Air Traffic Delay Cost, *Proceedings of American Control Conference*, 1121-1125.

Koening, L. W. and A. M. Law. 1985. A Procedure for Selecting a Subset of Size m Containing the 1 Best of k Independent Normal Populations, with Applications to Simulation, *Communications in Statistics,* B14, 719-734.

Kocsis, L., Szepesvári, C., and Winands, M. H. M., 2003. RSPSA: Enhanced Parameter Optimization in Games.

Kushner, H. J. and G. G. Yin. 1997. Stochastic Approximation with Averaging and Feedback: Rapidly Convergence On-Line Algorithms, *IEEE Transactions on Automatic Control*, 40, 24-34.

Jung, J. Y., G. Blau, J. F. Pekny, G. V. Reklaitis and D. Eversdyk. 2004. A Simulation based Optimization Approach to Supply Chain Management under Demand Uncertainty, *Computers and Chemical Engineering*, 28, 2087-2106.

Kirkpatrick, S., C. D. Gelatt Jr. and M. P. Vecchi. 1983. Optimization by Simulated Annealing, *Science*, 220, 671-680.

Lee, H. L. and C. Billington. 1993. Material Management in Decentralized Supply Chains, *Operations Research,* 41, 835-847.

L'Ecuyer, P. 1991. An Overview of Derivative Estimation, *Proceedings of the 1991 Winter Simulation Conference,* 207-217.

L'Ecuyer, P., N. Giroux and P. W. Glynn. 1994. Stochastic Optimization by Simulation: Numerical Experiments with the M/M/1 Queue in Steady State, *Management Science*, 40, 1245-1261.

Maeda, Y. and R. J. P. De Figueiredo, 1997. Learning Rules for Neuro-Controller via Simultaneous Perturbation, *IEEE Transactions on Neural Networks*, 8, 1119-1130.

Maedal, Y., H. Hirano and Y. Kanata. 1995. A Learning Rule of Neural Networks via Simultaneous Perturbation and its Hardware Implementation, *Neural Networks*, 8, 251-259.

Maria, A. 1995. *Genetic Algorithm for Multimodal Continuous Optimization Problems*. PhD dissertation, University of Oklahoma, Norman, OK.

Metropolis, N., A. Rosenbluth, M. Rosenbluth, A. Teller and E. Teller. 1953. Equation of State Calculations by Fast Computing Machines, *Journal of Chemistry and Physics*, 21, 1087-1091.

Mitra, G., I. Hai and M. T. Hajian, 1997. A Distributed Processing Algorithm for Solving Integer Programs using a Cluster of Workstations, *Parallel Computing,* 23**,** 733-753.

Morrice, D. J. and L. W. Schruben. 1989. Simulation Sensitivity Analysis Using Frequency Domain Experiments, *Proceedings of the 1989 Winter Simulation Conference,* 367-373.

Morrice, D. J., J. Butler and P. W. Mullarkey. 1998. An Approach to Ranking and Selection for Multiple Performance Measures, *Proceedings of the 1998 Winter Simulation Conference,* 719-725.

Morrice, D. J., J. Butler, P. W. Mullarkey and S. Gavireni. 1999. Sensitivity Analysis in Ranking and Selection for Multiple Performance Measures, *Proceedings of the 1999 Winter Simulation Conference,* 618-624.

Nagurney, A., J. Cruz, J. Dong and D. Zhang. 2005. Supply Chain Networks, Electronic Commerce, and Supply Side and Demand Side Risk, *European Journal of Operational Research,* 164, 120-142.

Nechyba, M. C. and Y. Xu. 1997. Human-Control Strategy: Abstraction, Verification, and Replication, *IEEE Control Systems Magazine*, 17(5), 48-61.

Newhart, D. D., K. L. Scott and F. J. Vasko. 1993. Consolidating Product Sizes to Minimize Inventory Levels for a Multi-stage Production and Distribution Systems, *Journal of the Operational Research Society*, 44(7), 637-644.

Phua, K. H., W. Fan and Y. Zeng. 1998. Parallel Algorithms for Large-Scale Nonlinear Optimization, *Journal of International Transactions in Operational Research (ITOR),* 51**,** 67-77**.**

Pitsoulis, L. S. and G. C. Resende. 2001. Greedy Randomized Adaptive Search Procedures, *AT&T Labs Research Technical Report*.

Pflug, G. C. 1981. On the Convergence of a Penalty-type Stochastic Optimization Procedure, *Journal of Information and Optimization Sciences*, 2, 249-258.

Ralphs, T. K. 2003. Parallel Branch and Cut for Capacitated Vehicle Routing, *Parallel Computing,* 29**,** 607-629.

Rao, U., A. Scheller-Wolf and S. Tayur. 2000. Development of a Rapid-Response Supply Chain at Caterpillar, *Operations Research*, 48(2), 189-204.

Rechenberg, I. 1973. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*,Stuttgart: Fromman-Holzboog.

Reiman, M. I. and A. Weiss. 1986. Sensitivity Analysis Via Likelihood Ratios, *Proceedings of the 1986 Winter Simulation Conference,* 285-289.

Reidmiller, M. and H. Braun. 1993. A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP algorithm, *Proceedings of the IEEE International Conference on Neural Networks*, 586-591.

Reiser, M. and S. S. Lavenberg. 1980. Mean-value-analysis of Closed Multi-chain Queueing Networks, *J. ACM,* 27**,** 310-322.

Resende, M. G. C. and C. Ribeiro. 2004. Parallel Greedy Randomized Adaptive Search Procedures, *AT&T Labs Research Technical Report TD-67EKXH*. To appear in Parallel Metaheuristics. E. Alba (ed.). John Wiley and Sons, 2005.

Rezayat, F. 1995. On the Use of an SPSA-Based Model-Free Controller in Quality Improvement, *Automatica*, 913-915.

Rinott, Y. 1978. On Two-stage Selection Procedures and Related Probability Inequalities, *Communications in Statistics*, A7, 799-811.

Robins, H. and S. Monro. 1951. A Stochastic Approximation Method, *Annals of Mathematical Statistics*, 22, 400-407.

Robinson, S. 2002. A Statistical Process Control Approach for Estimating the Warm-up Period, *Proceedings of the 2002 Winter Simulation Conference*, 439-445.

Ryan, S. M., B. Bayant and F. F. Choobineh. 2000. Determining Inventory Levels in a CONWIP Controlled Job Shop, *IIE Transactions,* 32**,** 105-104.

Sadegh, P. 1997. Constrained Optimization via Stochastic Approximation with a Simultaneous Perturbation Gradient Approximation, *Automatica*, 33, 889-892.

Sadegh, P. and J. C. Spall. 1998a. Optimal Random Perturbation for Stochastic Approximation with a Simultaneous Perturbation Gradient Approximation, *Automatic Control*, 43, 1480-1484.

Sadegh, P. and J. C. Spall. 1998b. Optimal Sensor Configuration for Complex Systems, *Proceedings of American Control Conference*, 3575-3579.

Schruben, L. W. and V. J. Cogliano. 1987. An Experimental Procedure for Simulation Response Surface Model Identification, *Communications of the Association for Computing Machinery,* 30, 716-730.

Schwefel, H.-P. 1981. *Numerical optimization of computer models*, Chichester: Wiley & Sons.

Schwefel, H.-P. 1995. *Evolution and Optimum Seeking*, New York: John Wiley & Sons.

Shahabuddin, P. 1995. Rare Event Simulation in Stochastic Models, *Proceedings of the 1995 Winter Simulation Conference,* 178-185.

Shapiro, A. 1996. Simulation-Based Optimization-Convergence Analysis and Statistical Inference, *Communications in Statistics-Stochastic Models*, 12, 425-454.

Spall, J. C. 1987. A Stochastic Approximation Technique for Generating Maximum Likelihood Parameter Estimates, *Proceedings of the American Control Conference*, 1161-1167.

Spall, J. C. 1988. A Stochastic Approximation Algorithm for Large-Dimensional Systems in the Keifer-Wolfowitz Setting, *Proceedings of the IEEE Conference on Decision and Control*, 1544-1548.

Spall, J. C. 1992. Multivariate Stochastic Approximation Using a Simultaneous Perturbation Gradient Approximation, *IEEE Transactions on Automatic Control*, 37, 332-341.

Spall, J. C. and J. A. Cristion. 1994. Nonlinear Adaptive Control Using Neural Networks: Estimation Based on a Smoothed Form of Simultaneous Perturbation Gradient Approximation, *Statistical Sinica*, 4. 1-27.

Spall, J. C. 1997. Accelerated Second-Order Stochastic Optimization Using Only Function Measurements, *Proceedings of the 36th Conference on Decision & Control*, 1417-1424.

Spall, J. C. 1998a. An Overview of the Simultaneous Perturbation Method for Efficient Optimization, *Johns Hopinks APL Technical Digest*, 19 (4), 482-492.

Spall, J. C. 1998b. Implementation of the Simultaneous Perturbation Algorithm for Stochastic Optimization, *IEEE Transactions on Automatic Control*, 37, 332-341.

Spall, J. C. 1999. Stochastic Optimization and the Simultaneous Perturbation Method, *Proceedings of the 1999 Winter Simulation Conference*, 101-109.

Spall, J. C. 2000. Adaptive Stochastic Approximation by the Simultaneous Perturbation Method, *IEEE Transactions on Automatic Control*, 45, 1839-1853.

Spall, J. C. 2002. *Introduction to Stochastic Search and Optimization*, John Wiley & Sons, Inc., Hoboken, New Jersey.

Spearman, M. L., D. L. Woodruff and W. L. Hopp. 1990. CONWIP: a Pull Alternative to Kanban, *International Journal of Production Research,* 28**,** 879-894.

Srinivasan, K., S. Kekre and T. Mukhopadhyay. 1994. Impact of Electronic Data Interchange Technology on JIT Shipments, *Management Science*, 40 (10), 1291-1305.

Stadtler, H. and C. Kilger. 2005. Supply Chain Management and Advanced Planning: Concepts, Models, Software and Case Studies.

Suri, R. 1983. Infinitesimal Perturbation Analysis of Discrete Event Dynamic Systems: a General Theory, *Proceedings of the 22nd IEEE Conference on Decision and Control,* 1030-1038.

Svoronos, A. and P. Zipkin. 1991. Evaluation of One-for-one Replenishment Policies for Multiechelon Inventory Systems, *Management Science*, 37(1). 68-83.

Swaminathan, J., S. Smith, and N. Saleh. 1998. Modeling Supply Chain Dynamics: a Multi-agent Approach, *Decision Science*, 29(3), 607-632.

Swisher J. R. and P. D. Hyden. 2000. A Survey of Simulation Optimization Techniques and Procedures, *Proceedings of the 2000 Winter Simulation Conference*, 119-128.

Swisher, J., P. D. Hyden, S. H. Jacobson and L. W. Schruben. 2004. A Survey of Recent Advances in Discrete Input Parameter Discrete-Event Simulation Optimization, *IIE Transactions*, 36, 591-600.

Tang, H. P. and T. N. Wong. 2005. Reactive Multi-agent System for Assembly Cell Control, *Robotics and Computer-Integrated Manufacturing*, 21, 87-98.

Towill, D. R., M. M. Naim and J. Winker. 1992. Industrial Dynamics Simulation Models in the Design of Supply Chains, *International Journal of Physical Distribution and Logistics Management*, 22, 3-13.

Tukey, J. W. 1953. *The Problem of Multiple Comparisons*. Unpublished manuscript.

Xie, X. 1997. Dynamics and Convergence Rate of Ordinal Comparison of Stochastic Discrete-Event Systems, *IEEE Transactions on Automatic Control*, 42, 586-590.

Wan, X., P. E. Pekny and V. Reklaitis. 2005. Simulation-based Optimization with Surrogate Models-Application to Supply Chain Management, *Computers and Chemical Engineering*, 29, 1317-1328.

Wang, I.-J. and J. C. Spall. 1999. A Constrained Simultaneous Perturbation Stochastic Approximation Algorithm based on Penalty Functions, *Proceedings of the American Control Conference*, 393-399.

Wolpert, D. H. and W. G. Macready. 1995. No Free Lunch Theorems for Search, *Technical Report SFI-TR-95-02-010* , Santa Fe Institute.

Appendix A

Codes for Unconstrained Supply Chain Inventory Optimization Problems

/*Simulation codes for serial supply chain */

Simlation.cpp

```
#include "Simulation.h"
#include "mersenne.h"


double Simulation(int* x, int Seed,int n, int Days, int Warm_Up)
{
            struct Event_Node* Execute_Node;
            struct Event_Node *New_Node;
            struct Event_Node *New_Node1;

            int Demand_Size;
            int Order_Size;
            int Backorder_Size;
            int Inventory_Level;
            int Tag;
            int i=0;
            int j,k;
            int Destination_ID;
            int Source_ID;
            double Total_Cost=0;
            FILE * iFile;
   int Order_Quantity[4];

   iFile=fopen ("Inventory.txt","w+");
   if (iFile==NULL) perror ("Error opening file");

     fprintf(iFile,"Demand************Inventory Backorder OnOrder************* \n");

  Clock=0;
  Phead=NULL;
  Bhead=NULL;

  TRandomMersenne rg;          // make instance of random number generator
```

```
   rg.RandomInit(Seed);
   for(i=0;i<Number_Of_DistributionShop_Number;i++)
           {
             Dis[i].Set_Q(x[i]);
          Dis[i].Initialization();
                    Dis[i].Set_LeadTime_Seed((i+1)*500+Seed);
           }

  for(i=0;i<Days;i++)
  {
            ++ Clock;


            if(Phead!=NULL)
            {
                     Execute_Node=Phead;

       Again:if(Execute_Node->Time_Stamp==Clock)
                     {

                      Destination_ID=Execute_Node->node.Destination_ID;
                      Source_ID=Execute_Node->node.Source_ID;
                              if(Execute_Node->Event_Type==1)
                              {
                               if(Destination_ID>0)
                               {
                        Dis[Destination_ID-1].OrderArrived(Execute_Node->node.Order_Size);

                               Dis[Source_ID-1].MinusInTransit(Execute_Node->node.Order_Size);

                               }
                              }

                              if(Execute_Node->next!=NULL)
                       New_Node=Execute_Node->next;
                              else
                                        New_Node=NULL;
                       Phead=Remove_Node(Phead,Execute_Node);
          if(New_Node!=NULL)
                              {
                              Execute_Node=New_Node;
                              goto Again;
                              }
                     }
            }



 if(Bhead!=NULL)
 {
```

```
           Execute_Node=Bhead;
Continue:    while(Execute_Node!=NULL&&Execute_Node->Time_Stamp<=Clock)
          {
                    Source_ID=Execute_Node->node.Source_ID;
                    Destination_ID=Execute_Node->node.Destination_ID;
                    Order_Size=Execute_Node->node.Order_Size;
                    Inventory_Level=Dis[Source_ID-1].GetInventoryLevel();


                    Tag=0;
        if(Inventory_Level>0)
                    {

                     if(Inventory_Level>=Order_Size)
                           {
                      j=Order_Size;

                      if(Execute_Node==Bhead)
                                   {
                                     Tag=1;
                                    Bhead=Execute_Node->next;
                                    if(Bhead!=NULL)
                                    Bhead->previous=NULL;
                                    free(Execute_Node);
            Execute_Node=Bhead;

                                   }
                             else
                                   {
                                    New_Node1=Execute_Node->previous;
                                    New_Node1->next=Execute_Node->next;
                                    if( Execute_Node->next!=NULL)
                                    Execute_Node->next->previous=New_Node1;
                                    free(Execute_Node);
            Execute_Node=New_Node1;
                                   }
                           }
                     else
                           {
                      Backorder_Size=Order_Size-Inventory_Level;
                      j=Inventory_Level;
          Execute_Node->node.Order_Size=Backorder_Size;
                      Tag=0;
                           }
                           Dis[Source_ID-1].MinusBackorderInventoryLevel(j);
                           Dis[Source_ID-1].MinusInventoryLevel(j);
                           if(Destination_ID!=0)
                           {
                           Dis[Source_ID-1].PlusInTransit(j);

                           }
```

```
                        if(Destination_ID!=0)
                        {
                    New_Node=new Event_Node;
                    New_Node->node.Source_ID=Source_ID;
                    New_Node->node.Destination_ID=Destination_ID;
                    New_Node->Event_Type=1;
                    New_Node->Time_Stamp=Clock+Dis[Source_ID-1].GetLeadTime();
        New_Node->node.Order_Size=j;
        New_Node->previous=NULL;
                    New_Node->next=NULL;
                    Phead=Sort_Insert(Phead,New_Node);
                        }
                  }

                 if(Tag==1)
                         goto Continue;
                 else
                 {
                 if(Execute_Node!=NULL)
                 Execute_Node=Execute_Node->next;
                 }


        }
}

   for(j=0;j<Number_Of_CustomerShop_Number;j++)
         {

          Demand_Size=rg.IRandom(20,60);
                if(n%2==0)
                {
                        k=Demand_Size;
                        Demand_Size=80-k;
                }

         fprintf(iFile," %4d  *", Demand_Size);
          Dem[j].OrderPlaced();
          Source_ID=Dem[j].GetSupplierID();
                Dis[Source_ID-1].DemandArrived(Demand_Size);
         }

         for(j=1;j<Number_Of_DistributionShop_Number;j++)
         {
                Order_Quantity[j-1]=Dis[j-1].CalculateOrderQuantity();
                if(Order_Quantity[j-1]>0)
                Dis[j].DemandArrived(Order_Quantity[j-1]);
         }

for(j=0;j<Number_Of_DistributionShop_Number-1;j++)
{
```

```
                if(i>=Warm_Up)
                        {
                  Dis[j].UpdateCost();
        }

 }
 fprintf(iFile,"\n");

}

Backorder_Cost=0;

Holding_Cost=0;
  for(j=0;j<Number_Of_DistributionShop_Number-1;j++)
  {
            Backorder_Cost+=Dis[j].GetBackorderCost();
            Holding_Cost+=Dis[j].GetHoldingCost();
            Total_Cost+=Dis[j].GetTotalCost();
  }

  while(Phead!=NULL)
  {
            Execute_Node=Phead;
            Phead=Remove_Node(Phead, Execute_Node);
  }

 while(Bhead!=NULL)
  {
            Execute_Node=Bhead;
            Bhead=Remove_Node(Bhead, Execute_Node);
  }

fclose(iFile);
return (Total_Cost);
}
```

/* main file including ASPSA optimization algorithm*/

main. cpp file

```
#include <stdlib.h>
#include <math.h>
#include <stdio.h>
#include <malloc.h>
#include <memory.h>
#include "Customer_Agent.h"
#include "Customer_Shop.h"
#include "Distribution_Agent.h"
#include "Distribution_Shop.h"
```

```
#include "List.h"
#include "Order_Satisfied.h"
#include "Demand_Placed.h"
#include "Simulation.h"
#include "Globals.h"
#include "mersenne.h"              // members of class TRandomMersenne
#include "Calculate_Simulation_Results.h"
#include "Calculate_DecisionVariable_Bound.h"


CustomerShop Dem[Number_Of_CustomerShop_Number];
DistributionShop   Dis[Number_Of_DistributionShop_Number];

 bool Two_Vectors_Same(int* x, int* y)
 {
        int i;

        for (i=0; i<Number_Of_DistributionShop_Number-1;i++)
        {
                if(x[i]!=y[i])
                        return 0;
        }
        return 1;

 }

int main()
{

  int   i=1,j, N;
  int  pVector[Number_Of_DistributionShop_Number]; //perturbation vector
  int Temp_x[Number_Of_DistributionShop_Number-1];
  int Old_x[Number_Of_DistributionShop_Number-1];
  int Best_Temp_x[Number_Of_DistributionShop_Number-1];
  int g[Number_Of_DistributionShop_Number];
  double r;
  double yplus;
  double yminus;
  int k;
  double ak,ck;
  double s;
  double old_mean_best;
  int Unimproved_Number=0;
  int b,n;

  bool Improve_Flag=1;
  bool Good_Direction_Flag=0;
  double s_mean[PreSearch_Number];
  int Elite_List_Length=0;
  int Sum_Total_Simulation_Runs=0;
  int Sum_Mean_Best=0;
  double Fraction_Time;
  FILE * iFile;
  FILE * iFile1;


  TRandomMersenne rg;

  iFile=fopen ("output.txt","w+");

  if (iFile==NULL) perror ("Error opening file");

 iFile1=fopen ("result.txt","w+");

  ak=1;
  ck=1;
Total_Simulation_Runs=0;
fprintf(iFile,"Mean   Variance    Standdeviation\n");

for(i=0;i<Number_Of_CustomerShop_Number;i++)
{
  Dem[i].SetDemandType(1);
  Dem[i].SetDemandSize(2);
  Dem[i].SetSupplierID(1);
  Dem[i].SetCustomerShopID(i);

}

for(i=0;i<Number_Of_DistributionShop_Number;i++)
{
 Dis[i].SetLeadTimeType(0);
 Dis[i].Set_s(0);
}
Dis[0].SetLeadTimeType(0);
Dis[1].SetLeadTimeType(0);
    Dis[4].Set_h(0);
    Dis[3].Set_h(1);
            Dis[2].Set_h(2);
            Dis[1].Set_h(4);
            Dis[0].Set_h(8);

            Dis[4].Set_b(0);
            Dis[3].Set_b(2);
            Dis[2].Set_b(4);
            Dis[1].Set_b(8);
            Dis[0].Set_b(16);

/*
    Dis[4].Set_b(0);
            Dis[3].Set_b(4);
            Dis[2].Set_b(8);
            Dis[1].Set_b(16);
            Dis[0].Set_b(32);

    Dis[4].Set_b(0);
            Dis[3].Set_b(6);
            Dis[2].Set_b(12);
            Dis[1].Set_b(24);
            Dis[0].Set_b(48);

            Dis[4].Set_b(0);
            Dis[3].Set_b(8);
            Dis[2].Set_b(16);
            Dis[1].Set_b(32);
            Dis[0].Set_b(64);

/*
   Dis[4].Set_b(0);
            Dis[3].Set_b(10);
            Dis[2].Set_b(20);
            Dis[1].Set_b(40);
            Dis[0].Set_b(80);

*/
    Dis[0].SetLeadTime(0);
            Dis[1].SetLeadTime(1);
            Dis[2].SetLeadTime(3);
            Dis[3].SetLeadTime(5);
            Dis[4].SetLeadTime(4);

  /*   Dis[0].SetLeadTime(0);
            Dis[1].SetLeadTime(2);
            Dis[2].SetLeadTime(3);
            Dis[3].SetLeadTime(6);
            Dis[4].SetLeadTime(4);
*/
            Dis[4].Set_LLT(3);
    Dis[3].Set_LLT(4);
            Dis[2].Set_LLT(2);
            Dis[1].Set_LLT(1);

            Dis[4].Set_ULT(5);
    Dis[3].Set_ULT(6);
            Dis[2].Set_ULT(4);
            Dis[1].Set_ULT(3);


Fraction_Time=(double)(Small_Simulation_Time)/Total_Simulation_Time;
for(i=0;i<Number_Of_DistributionShop_Number;i++)
{
  Dis[i].SetOrderPolicy(3);
  Dis[i].SetCustomerID(i);
  Dis[i].SetDistributionShopID(i+1);
  Dis[i].SetSupplierID(i+2);
  Dis[i].SetEndSupplierFlag(0);
 }

Dis[Number_Of_DistributionShop_Number-1].SetEndSupplierFlag(1);
k=0;

Calucate_Bound();

/*   begining of the codes*/
for(N=0;N<Number_Of_Seeds;N++)
{
 rg.RandomInit(400*(N+1));
 Total_Simulation_Runs=0;
 k=0;

 //cout<<"N="<<N<<endl<<endl;
 for(j=0;j<PreSearch_Number;j++)
 {

            for(i=0;i<Number_Of_DistributionShop_Number-1;i++)

                        x[i]=LB_x[i]+rg.IRandom(0,(UB_x[i]-LB_x[i]));//*rg.TRandom();

            s_mean[j]=Calculate_Simulation_Output(x,Small_Simulation_Time,0, 1000*(N+1));
            Total_Simulation_Runs+=Fraction_Time;
             if(j==0||s_mean[j]<mean_best)
               {
                 mean_best=s_mean[j];
        for(i=0;i<Number_Of_DistributionShop_Number-1;i++)
                          best_x[i]=x[i];

               }


}
Unimproved_Number=0;

//cout<<"\nUI="<<Unimproved_Number<<endl;

cout<<"***********best x**********\n";
for(i=0;i<Number_Of_DistributionShop_Number-1;i++)
{
x[i]=best_x[i];
}

mean_best= Calculate_Simulation_Output(best_x,Total_Simulation_Time,0,1000*(N+1));
cout<<mean_best<<endl;
mean[k]=mean_best;
Total_Simulation_Runs+=1;
```

```
for(i=0;i<Number_Of_DistributionShop_Number-1;i++)
              {
                   cout<<x[i]<<"   ";
                  fprintf(iFile,"%d ", x[i]);
              }
cout<<" meat_best= "<<mean_best<<" mean="<<mean[k]<<endl;
fprintf(iFile,"   %.2f  %.2f \n",mean_best, mean[k]);

do{

 ++k;
  for(i=0;i<Number_Of_DistributionShop_Number-1;i++)
  {
   Old_x[i]=x[i];
    r=rg.TRandom(); //rand()/(RAND_MAX+1.0);

     if(r>0.5)
              pVector[i]=ck;
     else
              pVector[i]=-ck;

  }

  for(i=0;i<Number_Of_DistributionShop_Number-1;i++)
  {
              xplus[i]=x[i]+pVector[i];
              xminus[i]=x[i]-pVector[i];
              if(xplus[i]<LB_x[i])
        xplus[i]=LB_x[i];
              else if(xplus[i]>UB_x[i])
        xplus[i]=UB_x[i];
     if(xminus[i]<LB_x[i])
        xminus[i]=LB_x[i];
              else if(xminus[i]>UB_x[i])
        xminus[i]=UB_x[i];
  }

 yplus=Calculate_Simulation_Output(xplus,Small_Simulation_Time,0,1000*(N+1));
 yminus=Calculate_Simulation_Output(xminus,Small_Simulation_Time,0,1000*(N+1));
 Total_Simulation_Runs+=(2*Fraction_Time);

 if(yplus<yminus)
 {
             for(i=0;i<Number_Of_DistributionShop_Number-1;i++)
                          g[i]=-pVector[i];

 }
 else
 {
             for(i=0;i<Number_Of_DistributionShop_Number-1;i++)
                          g[i]=pVector[i];
 }

 b=0;
 ak=10.0/pow(k+10,0.5)+1;
for(j=0;j<Local_Search_Number;j++)
{

 s=0;
 for(i=0;i<Number_Of_DistributionShop_Number-1;i++)
  {

            ck=rg.IRandom(0,ak*j+1);
            s+=ck;
              Temp_x[i]=x[i]- g[i]*ck;//(int)(g[i]*(10*(1-k/120.0)*j+1));

            if(Temp_x[i]<LB_x[i])
              Temp_x[i]=LB_x[i];
            else if(Temp_x[i]>UB_x[i])
              Temp_x[i]=UB_x[i];
  }

 if(s==0)
 {
             n=rg.IRandom(0,Number_Of_DistributionShop_Number-1);
    for(i=0;i<n;i++)
             Temp_x[i]-=g[i];
 }

  s_mean[j]=Calculate_Simulation_Output(Temp_x,Small_Simulation_Time,0, 1000*(N+1));
  Total_Simulation_Runs+=Fraction_Time;

  if(j==0||s_mean[j]<s_mean[b])
  {
              b=j;
     for(i=0;i<Number_Of_DistributionShop_Number-1;i++)
                          Best_Temp_x[i]=Temp_x[i];
  }


}
//cout<<b<<"***";

for(i=0;i<Number_Of_DistributionShop_Number-1;i++)
              x[i]=Best_Temp_x[i];

if(Two_Vectors_Same(x,Old_x))
{
            for(i=0;i<Number_Of_DistributionShop_Number-1;i++)
            x[i]+=rg.IRandom(-1,1);
}
```

```
mean[k]=Calculate_Simulation_Output(x,Total_Simulation_Time,0, 1000*(N+1));
Total_Simulation_Runs+=1;
 for(i=0;i<Number_Of_DistributionShop_Number-1;i++)
 fprintf(iFile,"%d ", x[i]);


   old_mean_best=mean_best;
   if(mean[k]<mean_best)
            {
              mean_best=mean[k];
       for(i=0;i<Number_Of_DistributionShop_Number-1;i++)
                     best_x[i]=x[i];
            }
          else
          {
                    for(i=0;i<Number_Of_DistributionShop_Number-1;i++)
                    {
                     x[i]=best_x[i]; //+rg.IRandom(-1,1);
                    }
          }

          cout<<" meat_best= "<<mean_best<<" mean="<<mean[k]<<endl;
          fprintf(iFile,"   %.2f  %.2f\n",mean_best, mean[k]);

          if(old_mean_best==mean_best)
          {
                    ++Unimproved_Number;
          }
          else
          {
                    Unimproved_Number=0;
          }

}while(k<Stop_Creteria&&Unimproved_Number<10);

cout<<k<<"*********** best x**********\n";
for(i=0;i<Number_Of_DistributionShop_Number-1;i++)
{
cout<< best_x[i]<<"  ";
fprintf(iFile1,"%d ", best_x[i]);
}
fprintf(iFile1," %.0f  %.2f %d\n", mean_best, Total_Simulation_Runs,Unimproved_Number );
cout<<mean_best<<"    "<<Total_Simulation_Runs<<" "<<Unimproved_Number<<"\n";

Sum_Total_Simulation_Runs+=Total_Simulation_Runs;
Sum_Mean_Best+=mean_best;
mean_best_array[N]=mean_best;

}
mean_best=Sum_Mean_Best/Number_Of_Seeds;
Total_Simulation_Runs=Sum_Total_Simulation_Runs/Number_Of_Seeds;
```

```
cout<<mean_best<<"    "<<Total_Simulation_Runs<<"\n\n";

/*for(i=0;i<Number_Of_DistributionShop_Number-1;i++)
x[i]=0;

            New_Node=Ehead;
            do
            {

                        for(i=0;i<Number_Of_DistributionShop_Number-1;i++)
                        {

                                    x[i]=New_Node->Decision_Vector[i];;

                        }
                        s=New_Node->Objective;
                        BLMS(x,s);
               New_Node=New_Node->next;
            }while(New_Node!=NULL);

//EliteList_Printf_Node(Ehead);

cout<<k<<"***********final best x**********\n";
for(i=0;i<Number_Of_DistributionShop_Number-1;i++)
{
cout<< best_x[i]<<"  ";
fprintf(iFile,"%d ", best_x[i]);
//x[i]=best_x[i];
}
*/
fprintf(iFile,"\n   mean_Best  Best_std THC, TSC, Total_Simulation_Runs\n");
fprintf(iFile,"\n          %.0f       %.0f    %.0f    %.0f    %.0f\n",    mean_best,    Best_Std,    Best_THC,
Best_TSC,Total_Simulation_Runs);

//cout<<mean_best<<"    "<<Total_Simulation_Runs<<"\n\n";
fclose(iFile);
fclose(iFile1);
  return 0;
}
```

Appendix B

Codes for Constrained Supply Chain Inventory Optimization Problems

/* Simulation codes for fork-join supply chain*/

Simulation.cpp

```
#include "Simulation.h"
#include "mersenne.h"              // members of class TRandomMersenne

float Simulation(int* x, int Seed,int n, int Days, int Warm_Up)
{
            struct Event_Node* Execute_Node;
            struct Event_Node *New_Node;
            struct Event_Node *New_Node1;

            int Demand_Size;
            int Order_Size;
            int Inventory_Level;
            int Tag;
            int Enough_Inventory_Flag=1;
            int i=0;
            int j,k,m;
            int Destination_ID;
            int Customer_ID;
            int Supplier_ID;
            int Source_ID;
            float Total_Cost=0;
   int Order_Quantity[4],s;
            Days_In_Each_Season=(Days/Season_Number);

  Clock=0;
  Phead=NULL;
  Bhead=NULL;
  TRandomMersenne rg;           // make instance of random number generator
   rg.RandomInit(Seed+345);
            for(i=0;i<Number_Of_CustomerShop_Number;i++)
            {
            Dem[i].Set_Demand_Seed((i+1)*10+Seed);
            Dem[i].CustomerShop_Initialization();
            Dem[i].Set_TransportationTime_Seed((i+1)*7+Seed);
            }

   for(i=0;i<Number_Of_DistributionShop_Number;i++)
```

```
            {
              Dis[i].Set_Q(x[i]);
                     Dis[i].Initialization();
                     Dis[i].Set_LeadTime_Seed((i+1)*10+Seed);
            }

for(i=0;i<Days;i++)
{
            ++ Clock;

for(j=0;j<Number_Of_DistributionShop_Number;j++)
{
          Dis[j].Set_Q(x[j+(Season_ID-1)*Number_Of_DistributionShop_Number]);

}

          if(Phead!=NULL)
          {
                     Execute_Node=Phead;
                     //cout<<"call this"<<endl;
      while(Execute_Node!=NULL)
                      {
                       if(Execute_Node->Time_Stamp==Clock)
                       {
                         Destination_ID=Execute_Node->node.Destination_ID;

                              if(Destination_ID>0)
                              {
                        Dis[Destination_ID-1].OrderArrived(Execute_Node->node.Order_Size);
                               Dis[Destination_ID-1].MinusInTransit(Execute_Node->node.Order_Size);
                              }
                              else
                              {
          if(Execute_Node->node.Actual_Due_Date<=Execute_Node->node.Required_Due_Date)
                    Dem[Destination_ID].OrderSatisfied();
                              }

                             if(Execute_Node->next!=NULL)
                                New_Node=Execute_Node->next;
                             else
                                New_Node=NULL;
                       Phead=Remove_Node(Phead,Execute_Node);
                             Execute_Node=New_Node;
                       }
                       else
                               break;

                      }

          }
```

```
if(Bhead!=NULL)
{
            Execute_Node=Bhead;
Continue: while(Execute_Node!=NULL&&Execute_Node->Time_Stamp<=Clock)
           {

                        Source_ID=Execute_Node->node.Source_ID;
                        Destination_ID=Execute_Node->node.Destination_ID;

                        Order_Size=Execute_Node->node.Order_Size;
                        Enough_Inventory_Flag=1;

                        if(Destination_ID>0)
                        {

                        if(Dis[Destination_ID-1].GetEndSupplierFlag()!=1)
                                  {

                                   for(k=0;k<Dis[Destination_ID-1].GetSupplierNumber();k++)
                                   {
             Supplier_ID=Dis[Destination_ID-1].GetSupplierID(k);

                                    if(Dis[Supplier_ID-1].GetInventoryLevel()>=Dis[Destination_ID-
1].GetUij(k)*Order_Size)
                                             Enough_Inventory_Flag*=1;
                                    else
               Enough_Inventory_Flag*=0;
                                   }
                                 }

                        }
                        else
                        {
                                  if(Dis[Source_ID-1].GetInventoryLevel()>=Order_Size)
                                             Enough_Inventory_Flag=1;
                                   else
                Enough_Inventory_Flag=0;
                        }

                       Tag=0;

                       if(Enough_Inventory_Flag==1)
                                  {
                           j=Order_Size;
                           New_Node=new Event_Node;
                                   New_Node->Event_Type=1;
                                   New_Node->node=Execute_Node->node;
                                   New_Node->previous=NULL;
                           New_Node->next=NULL;
```

```
                    if(Execute_Node==Bhead)
                                {
                                 Tag=1;
                                 Bhead=Execute_Node->next;
                                   if(Bhead!=NULL)
                                   Bhead->previous=NULL;
                                 free(Execute_Node);
          Execute_Node=Bhead;
                                }
                          else
                                {
                                 New_Node1=Execute_Node->previous;
                                 New_Node1->next=Execute_Node->next;
                                 if( Execute_Node->next!=NULL)
                                 Execute_Node->next->previous=New_Node1;
                                 free(Execute_Node);
          Execute_Node=New_Node1;
                                }

       if(Destination_ID>0)
                  {

                           Dis[Destination_ID-1].PlusInTransit(Order_Size);

                           for(k=0;k<Dis[Destination_ID-1].GetSupplierNumber();k++)
                           {
         Supplier_ID=Dis[Destination_ID-1].GetSupplierID(k);
                                m=(int)(Dis[Destination_ID-1].GetUij(k)*Order_Size);
                                if(Dis[Supplier_ID-1].GetBackorderInventoryLevel()>=m)
                                Dis[Supplier_ID-1].MinusBackorderInventoryLevel(m);
                                Dis[Supplier_ID-1].MinusInventoryLevel(m);

                           }
                            New_Node->Time_Stamp=Clock+Dis[Destination_ID-1].GetLeadTime();

                  }
                 else
                           {
                            Dis[Source_ID-1].MinusBackorderInventoryLevel(Order_Size);
                            Dis[Source_ID-1].MinusInventoryLevel(Order_Size);
                            New_Node-
>Time_Stamp=Clock+Dem[Destination_ID].GetTransportationTime();
                            New_Node->node.Actual_Due_Date=New_Node->Time_Stamp;

                           }
                    Phead=Sort_Insert(Phead,New_Node);
                  }

                 if(Tag==1)
                           goto Continue;
                 else
```

```
                    {
                    if(Execute_Node!=NULL)
                    Execute_Node=Execute_Node->next;
                    }
          }
}

for(j=0;j<Number_Of_DistributionShop_Number;j++)

           Global_Inventory_Level[j]=Dis[j].GetInventoryLevel();

for(j=0;j<Number_Of_CustomerShop_Number;j++)
{

            Demand_Size=Dem[j].GetDemandSize();
                  Source_ID=Dem[j].GetSupplierID();
                  Customer_ID=Dem[j].GetCustomerShopID();
                  Inventory_Level=Dis[Source_ID-1].GetInventoryLevel();
                  if(Demand_Size>0)
                  {
                Dem[j].OrderPlaced();
         New_Node1=new Event_Node;
                New_Node1->next=NULL;
                New_Node1->previous=NULL;
                      New_Node1->Event_Type=1;
         New_Node1->node.Destination_ID=Customer_ID;//Supplier_ID;
         New_Node1->node.Source_ID=Source_ID;
                      New_Node1->node.Order_Size=Demand_Size;
         New_Node1->node.Order_Placed_Time=Clock;
         New_Node1->node.Required_Due_Date=Clock+required_due_date;

                  if(Inventory_Level>=Demand_Size)
                    {
                New_Node1->Time_Stamp=Clock+Dem[j].GetTransportationTime();

         Phead=Sort_Insert(Phead,New_Node1);
                             Dis[Source_ID-1].MinusInventoryLevel(Demand_Size);
                    }
                   else
                    {

                             Dis[Source_ID-1].PlusBackorderInventoryLevel(Demand_Size);
                New_Node1->Time_Stamp=Clock;
         Bhead=Sort_Insert(Bhead,New_Node1);
                    }
       New_Node1->node.Actual_Due_Date=New_Node1->Time_Stamp;
                  }

}
```

```
for(j=0;j<Number_Of_DistributionShop_Number;j++)
            Global_Inventory_Level[j]=Dis[j].GetInventoryLevel();

for(j=0;j<Number_Of_DistributionShop_Number;j++)
{
                    Order_Quantity[j]=Dis[j].CalculateProductionQuantity();
                    Enough_Inventory_Flag=1;

                    if(Order_Quantity[j]>0)
                    {
         if(Dis[j].GetEndSupplierFlag()!=1)
                              {
                               for(k=0;k<Dis[j].GetSupplierNumber();k++)
                               {
           Supplier_ID=Dis[j].GetSupplierID(k);
                                if(Global_Inventory_Level[Supplier_ID-
1]>=Dis[j].GetUij(k)*Order_Quantity[j])
                                         Enough_Inventory_Flag*=1;
                                 else
             Enough_Inventory_Flag*=0;
                               }
                              }

                              if(Enough_Inventory_Flag==1)
                              {
                               if(Dis[j].GetEndSupplierFlag()!=1)
                               {
                                 for(k=0;k<Dis[j].GetSupplierNumber();k++)
                                        {
           Supplier_ID=Dis[j].GetSupplierID(k);
                                       Dis[Supplier_ID-
1].MinusInventoryLevel(Dis[j].GetUij(k)*Order_Quantity[j]);
                                        }
                               }
          New_Node1=new Event_Node;
                 New_Node1->next=NULL;
                 New_Node1->previous=NULL;
                        New_Node1->Event_Type=1; // 0: from customer to supply
                               s=Dis[j].GetLeadTime();
                 New_Node1->Time_Stamp=Clock+s;

          New_Node1->node.Destination_ID=Dis[j].GetDistributionShopID();//Supplier_ID;
                               New_Node1->node.Source_ID=Dis[j].GetSupplierID(0);
                        New_Node1->node.Order_Size=Order_Quantity[j];
          New_Node1->node.Order_Placed_Time=Clock;
          New_Node1->node.Required_Due_Date=Clock+required_due_date;
          Phead=Sort_Insert(Phead,New_Node1);
                               Dis[j].PlusInTransit(Order_Quantity[j]);
                              }
                              else
                              {
                                         New_Node1=new Event_Node;
                    New_Node1->next=NULL;
                    New_Node1->previous=NULL;
                          New_Node1->Event_Type=1; // 0: from customer to supply
                    New_Node1->Time_Stamp=Clock;
             New_Node1->node.Destination_ID=Dis[j].GetDistributionShopID();//Supplier_ID;
                                 New_Node1->node.Source_ID=Dis[j].GetSupplierID(0);;
                          New_Node1->node.Order_Size=Order_Quantity[j];
             New_Node1->node.Order_Placed_Time=Clock;
             New_Node1->node.Required_Due_Date=Clock+5;
             Bhead=Sort_Insert(Bhead,New_Node1);

                               for(k=0;k<Dis[j].GetSupplierNumber();k++)
                               {
           Supplier_ID=Dis[j].GetSupplierID(k);
                                if(Global_Inventory_Level[Supplier_ID-
1]<Dis[j].GetUij(k)*Order_Quantity[j])
                                         Dis[Supplier_ID-
1].PlusBackorderInventoryLevel(Dis[j].GetUij(k)*Order_Quantity[j]);
                               }
                              }
                    }


}

if(Phead!=NULL)
{
                    Execute_Node=Phead;

      while(Execute_Node->Time_Stamp==Clock&&Execute_Node!=NULL)
                    {

                     Destination_ID=Execute_Node->node.Destination_ID;
                            if(Execute_Node->Event_Type==1)
                            {
                             if(Destination_ID>0)
                             {
                        Dis[Destination_ID-1].OrderArrived(Execute_Node->node.Order_Size);
                               Dis[Destination_ID-1].MinusInTransit(Execute_Node->node.Order_Size);
                             }
                             else
                             {
                                       if(Execute_Node->node.Actual_Due_Date<=Execute_Node-
>node.Required_Due_Date)
                                       Dem[Destination_ID].OrderSatisfied();
                             }
                            }
```

```
                                  if(Execute_Node->next!=NULL)
                         New_Node=Execute_Node->next;
                                  else
                                              New_Node=NULL;
                         Phead=Remove_Node(Phead,Execute_Node);
                                  Execute_Node=New_Node;

                        }
}


 for(j=0;j<Number_Of_DistributionShop_Number;j++)
{
             if(i>=Warm_Up)

               Dis[j].UpdateCost();

 }

}


for(j=0;j<Number_Of_CustomerShop_Number;j++)
{
Service_Level[j]=Dem[j].CalculateServiceLevel();
}

BackorderCost=0;

HoldingCost=0;
Total_Cost=0;
  for(j=0;j<Number_Of_DistributionShop_Number;j++)
  {
             BackorderCost+=Dis[j].GetBackorderCost();
             HoldingCost+=Dis[j].GetHoldingCost();
             Total_Cost+=Dis[j].GetTotalCost();

  }

  while(Phead!=NULL)
  {
             Execute_Node=Phead;
             Phead=Remove_Node(Phead, Execute_Node);
  }

 while(Bhead!=NULL)
  {
             Execute_Node=Bhead;
             Bhead=Remove_Node(Bhead, Execute_Node);
  }
```

```
return (Total_Cost);
}
```

/* Main.cpp file include ASPSA-II codes*/

main.cpp file

```
#include <stdlib.h>
#include <math.h>
#include <stdio.h>

#include "Customer_Agent.h"
#include "Customer_Shop.h"
#include "Distribution_Agent.h"
#include "Distribution_Shop.h"
#include "List.h"
#include "Order_Satisfied.h"
#include "Demand_Placed.h"
#include "Simulation.h"
#include "Globals.h"
#include "mersenne.h"              // members of class TRandomMersenne
#include "Calculate_Simulation_Results.h"
#include "Calculate_DecisionVariable_Bound.h"

#include "Elite_List.h"
#include "Supply_Chain_Initialization.h"


CustomerShop Dem[Number_Of_CustomerShop_Number];
DistributionShop   Dis[Number_Of_DistributionShop_Number];

 bool Two_Vectors_Same(int* x, int* y)
 {
	int i;

	for (i=0; i<Decision_Number;i++)
	{
		if(x[i]!=y[i])
			return 0;
	}
	return 1;

 }

 double min( double x, double y)
 {
```

```
            if(x>y)
                      return y;
            else
                      return x;
 }

double penalized_f( double f)
{
 double s;

 double penalty;

 penalty=(Service_Level[0]-Required_Service_Level[0]);

 if(penalty<0)

            s=f+f*Penalty_Scaler*exp(-penalty);
 else
            s=f;

 return s;
}


int main()
{

  long   i=1,j;
  int  pVector[Decision_Number]; //perturbation vector

  int Old_x[Decision_Number];
  int Best_Temp_x[Decision_Number];
  int g[Decision_Number];
  int LB[Decision_Number];

int Temp_x[Decision_Number];
  double r;
  double yplus;
  double yminus;
  int k;
  double ak,ck;
  double s,f;
  double old_mean_best;
  int Unimproved_Number=0;
  int b,d;
  bool Improve_Flag=1;
  bool Good_Direction_Flag=0;
//  int temp_best[1000];
  double s_mean[9000];
  double A=200;
  double ratio=(double)(Short_Simulation_Length)/(double)(Full_Simulation_Length);
  FILE * iFile;
  Elite_Node* Ehead=NULL;
//  Elite_Node* New_Node;
  TRandomMersenne rg;
  rg.RandomInit(100);
  iFile=fopen ("output.txt","w+");
  if (iFile==NULL) perror ("Error opening file");


ck=1;
Total_Simulation_Runs=0;
Supply_Chain_Initialization();
k=0;

Calucate_Bound(1);


for(i=0;i<Decision_Number;i++)
{
  x[i]=(int)((UB_x[i])*0.99);
  LB[i]=LB_x[i];
}

j=0;

do
{

             s_mean[j]=Calculate_Simulation_Output(x,Short_Simulation_Length,0);
             f=s_mean[j];
             s_mean[j]=penalized_f(f);
             //cout<< Service_Level[0]<<endl;
             Total_Simulation_Runs+=ratio;
              if(j==0||s_mean[j]<mean_best)
              {
                  mean_best=s_mean[j];
         for(i=0;i<Decision_Number;i++)
                         best_x[i]=x[i];
              }

              for(i=0;i<Decision_Number;i++)
               x[i]+=2;
              j++;

}while(Service_Level[0]<Required_Service_Level[0]);


cout<<"**********best x**********\n";
for(i=0;i<Decision_Number;i++)
```

```
{

x[i]=best_x[i];
if(i<=4)
cout<<x[i]<<" ";
}

mean_best= Calculate_Simulation_Output(x,Full_Simulation_Length,0);
//Full_Simulation_Length,0);
f=mean_best;
combined_mean_best=penalized_f(f);

mean[k]=mean_best;
combined_mean[k]=combined_mean_best;
serviceLevel[k]=Service_Level[0];
serviceLevel_best=serviceLevel[k];
cout<<"B= "<<combined_mean_best<<" O= "<<mean[k]<<" PO= "<<combined_mean[k];
printf(" SL=%.3f\n", serviceLevel[k]);


Total_Simulation_Runs+=1;
d=Decision_Number-1;

do
{
  ++k;
  for(i=0;i<Decision_Number;i++)
   Temp_x[i]=best_x[i];
  b=0;
  for(j=0;j<10;j++)
  {
           s=0;
           Temp_x[d]=(x[d]-LB[d])*j/10+LB[d];
    s_mean[j]=Calculate_Simulation_Output(Temp_x,Short_Simulation_Length,0);
    f=s_mean[j];
    s_mean[j]=penalized_f(f);;
    Total_Simulation_Runs+=ratio;
    if(j==0||s_mean[j]<s_mean[b])
    {
             b=j;
      for(i=0;i<Decision_Number;i++)
                       Best_Temp_x[i]=Temp_x[i];
    }

}
//cout<<"d= "<<d<<" ";

for(i=0;i<Decision_Number;i++)
x[i]=Best_Temp_x[i];

mean[k]=Calculate_Simulation_Output(x,Full_Simulation_Length,0);
```

```
f=mean[k];
combined_mean[k]=penalized_f(f);
serviceLevel[k]=Service_Level[0];

combined_mean[k]);
 old_mean_best=combined_mean_best;

    if(combined_mean[k]<=combined_mean_best)
            {
                mean_best=mean[k];
                        combined_mean_best=combined_mean[k];
                        serviceLevel_best=serviceLevel[k];
                        for(i=0;i<Decision_Number;i++)
                        best_x[i]=x[i];
            }
for(i=0;i<Number_Of_DistributionShop_Number;i++)
{
  cout<<x[i]<<" ";
}

cout<<"B= "<<combined_mean_best<<" O= "<<mean[k]<<" PO= "<<combined_mean[k];
printf(" SL=%.3f\n", serviceLevel[k]);
d-=1;

}while(k<Stop_Creteria&&d>=0);//&&Unimproved_Number<10);

for(i=0;i<Decision_Number;i++)
{
x[i]=best_x[i];
}

do{

 ++k;
  for(i=0;i<Decision_Number;i++)
  {
   Old_x[i]=x[i];
    r=rg.TRandom(); //rand()/(RAND_MAX+1.0);

     if(r>0.5)
              pVector[i]=1;
     else
              pVector[i]=-1;

  }

  for(i=0;i<Decision_Number;i++)
  {
              xplus[i]=x[i]+pVector[i];
              xminus[i]=x[i]-pVector[i];
              if(xplus[i]<LB_x[i])
```

```
               xplus[i]=LB_x[i];
             else if(xplus[i]>UB_x[i])
         xplus[i]=UB_x[i];
     if(xminus[i]<LB_x[i])
         xminus[i]=LB_x[i];
             else if(xminus[i]>UB_x[i])
         xminus[i]=UB_x[i];
  }

 yplus=Calculate_Simulation_Output(xplus,Short_Simulation_Length,0);
 f=yplus;
 yplus=penalized_f(f);
 yminus=Calculate_Simulation_Output(xminus,Short_Simulation_Length,0);
 f=yminus;
 yminus=penalized_f(f);

 Total_Simulation_Runs+=(2*ratio);

 if(yplus<yminus)
 {
            for(i=0;i<Decision_Number;i++)
                        g[i]=-pVector[i];

 }
 else
 {
            for(i=0;i<Decision_Number;i++)
                        g[i]=pVector[i];
 }

b=0;

for(j=0;j<Local_Search_Number;j++)
{
            s=0;

   for(i=0;i<Decision_Number;i++)
   {
     ak=floor((UB_x[i]-LB_x[i])/(k+10));
            ck=rg.IRandom(0,(int)(ak*(2+j)));
            Temp_x[i]=x[i]-(int)(g[i]*ck);

            if(Temp_x[i]<LB_x[i])
           Temp_x[i]=LB_x[i];
            else if(Temp_x[i]>UB_x[i])
           Temp_x[i]=UB_x[i];
   }


  s_mean[j]=Calculate_Simulation_Output(Temp_x,Short_Simulation_Length,0);
  f=s_mean[j];
  s_mean[j]=penalized_f(f);;

 Total_Simulation_Runs+=ratio;
  if(j==0||s_mean[j]<s_mean[b])
  {
              b=j;
     for(i=0;i<Decision_Number;i++)
                         Best_Temp_x[i]=Temp_x[i];

  }

}

for(i=0;i<Decision_Number;i++)
              x[i]=Best_Temp_x[i];

mean[k]=Calculate_Simulation_Output(x,Full_Simulation_Length,0);
f=mean[k];
combined_mean[k]=penalized_f(f);
serviceLevel[k]=Service_Level[0];
Total_Simulation_Runs+=1;
for(i=0;i<Decision_Number;i++)
{
                        fprintf(iFile,"%d ", x[i]);
}

 old_mean_best=combined_mean_best;
    if(combined_mean[k]<combined_mean_best)
            {
                mean_best=mean[k];
                        combined_mean_best=combined_mean[k];
                        serviceLevel_best=serviceLevel[k];
        for(i=0;i<Decision_Number;i++)
                          best_x[i]=x[i];

            }
            else
            {

                        for(i=0;i<Decision_Number;i++)
                          x[i]=best_x[i]+rg.IRandom(-1,1);

            }


            cout<<" B= "<<combined_mean_best<<" O= "<<mean[k]<<" PO= "<<combined_mean[k];
            printf(" SL=%.3f\n", serviceLevel[k]);
            fprintf(iFile,"%.2f %.2f %.2f %.4f %.2f\n",mean_best, mean[k], stdV[k], serviceLevel[k],
combined_mean[k]);

            if(old_mean_best==combined_mean_best)
        {
                ++Unimproved_Number;
        }
        else
        {
                Unimproved_Number=0;
        }

}while(k<Stop_Creteria&&Unimproved_Number<20);

cout<<k<<"*********** best x**********\n";
for(i=0;i<Decision_Number;i++)
{
cout<< best_x[i]<<"  ";
fprintf(iFile,"%d ", best_x[i]);
}

cout<<mean_best<<"    "<<combined_mean_best<<" "<<Total_Simulation_Runs;
printf(" SL=%.3f\n", serviceLevel_best);
fprintf(iFile,"\n   mean_best=%.0f  combined_mean_best=%.0f  Simulation_Run_Number=%.0f\n",
mean_best,combined_mean_best, Total_Simulation_Runs);

fclose(iFile);
  return 0;
}
```

VITA

Liya Wang

Liya Wang was born in Xi’an, China. She received a Bachelor of Science degree in Mechanical Engineering department from Tsinghua University in 1999. After that, she studied in Xi’an Jiaotong University for her Master of Science degree at System Engineering Institute. Since fall 2002, she has been a graduate student in the department of Industrial & Manufacturing Engineering at Pennsylvania State University. During the time, she worked as a research assistant in Discrete Lab under Associate Professor Vittal Prabhu, focused on developing new simulation optimization algorithms which explores parallel computation and current new techniques. At a more general level, her research interests involve developing efficient simulation optimization methods for solving large optimization problems for complex systems such as manufacturing systems and supply chain systems. From the research presented in this thesis, she has published one journal papers in IJPR, and two others submitted to well-known journals. She expects to get a dual degree of Doctor of Philosophy in Industrial Engineering and Operations Research in December of 2006.