A Novel Non-Gradient Optimization Methodology with Application to Mechanical Design and Scheduling in Production and Healthcare

by
Behnam Sharif

A Thesis
Submitted To The Faculty of Graduate Studies
In Partial Fulfillment of The Requirements
For The Degree of

Master of Science

Department of Mechanical and Manufacturing Engineering
University of Manitoba
Canada

THE UNIVERSITY OF MANITOBA

FACULTY OF GRADUATE STUDIES

*****

COPYRIGHT PERMISSION

A Novel Non-Gradient Optimization Methodology with Application to Mechanical Design and Scheduling in Production and Healthcare

BY

Behnam Sharif

A Thesis/Practicum submitted to the Faculty of Graduate Studies of The University of

Manitoba in partial fulfillment of the requirement of the degree

MASTER OF SCIENCE

Acknowledgments

I would like to take this opportunity to thank my advisors, Dr. Tarek ElMekkawy and Dr. Gary Wang for providing the opportunity to conduct this research and for their guidance and moral support during the course of this research project.

I would like to thank Health sciences Centre staff for providing information and data during our visits to the hospital. A special thank to Helga Bryant and Leanne Bernaerdt for their cooperation and support.

I give thanks to god for giving me wisdom and strength. Lastly, but certainly not least, I would like to thank my dearest parents for their support and encouragement throughout my work.

Abstract

Modern engineering design optimization, scheduling in industrial engineering featuring combinatorial optimization, and appointment optimization in healthcare share some common challenges, i.e., they all demand methods that can handle often expensive black-box objective functions, work without function gradients, and locate the optima efficiently and robustly. Meta-modeling techniques arise from the field of design optimization, as well as meta-heuristics or stochastic search approaches in combinatorial optimization are non-gradient methodologies that have been developed to solve simulation (black-box)-based optimization problems. There lacks of a methodology, however, that is capable of efficiently solving problems of no-gradient and (expensive) black-box functions originated from above-mentioned different areas. This thesis proposes such a methodology that consists of a series of methods rooted in Mode Pursuing Sampling (MPS) (Wang et al. (2004a)). In its original form, MPS only solves continuous global optimization problems. In this work, MPS has been extended to design problems involving discrete variables, as well as combinatorial problems with a large search space. Correspondingly four variants of algorithms have been developed, namely, D-MPS for discrete-variable global optimization, Co-MPS for combinatorial optimization, TSP-MPS for the well-known Traveling Salesman Problem, and AS-MPS for appointment scheduling in healthcare. All of the algorithms have achieved better or comparable results with the state-of-the-art. The work contributes significantly by bringing the core concept of MPS into the discrete and combinational optimization domains, and by developing a novel Double Sphere Method that is common in all the algorithms. The developed methods have high potential to be used in industrial practice.

Table of Contents

List of Figures

List of Tables

Nomenclature

• Chapter 3

f: Expensive black-box function to be optimized by D-MPS method.

S[f]: Domain of objective function *f.*

N: Number of points X_k which are uniformly distributed base points in *S[f].*

N_d: Number of indices for dimension *d.*

$S_N[f]$: Discrete space of *S[f]* generated by X_k, $k=1,...,N$.

m: Number of expensive points generated at each iteration .

E[f]: Set of all expensive points and their expensive function values generated so far.

X:* Global optimum found by D-MPS algorithm.

Iter: Current number of iteration.

f^(x): Approximated value of *f(x).*

p[k]: Approximated value of all X_k, $k=1,..,N$.

g[k]: Sampling guidance function for D-MPS algorithm.

G[k]: Cumulative density functions for D-MPS algorithm.

DSM method: Double Sphere Method used in D-MPS.

α: A control parameter for DSM method..

n_α: Number of steps for no improvement to activate hyperspheres of DSM method.

Hypersphere S: One of the hyperspheres used in DSM. It tends to shrink when there are n_α iterations of non-improvement.

Hypersphere B: One of the hyperspheres used in DSM. It tends to shrink when there is an improvement.

$R_{s,iter}$: Radius of hypersphere *S* at iteration *iter.*

$R_{b,iter}$: Radius of hypersphere B at iteration $iter$.

R_s: Initial radius of hypersphere S.

R_{min}: the minimum radius that both hyperspheres can reach.

D_1: Domain provided by hypersphere S.

D_2: Domain provided by hypersphere B.

b_{iter}: Flag showing the improvement or non-improvement.

n-Eval: Number of function evaluations.

Maxiter: Maximum number of iterations.

• Chapter 4

p_i: Processing time for each operation i.

c_i: Completion time of job i.

d_i: Due date of job i.

w_i: Weight of job i.

K: Number of agents, u=1,...,K

n: The number of jobs to be sequenced.

m_c: A parameter denoting the number of neighbors of the seed to be generated at any iteration.

P: Number of elements in the elite list

c_{iter}: A parameter to control generating a new-born agent.

σ_u^*: A job sequence or the seed of agent "u" which is the local optima found by agent k so far. It will be used by the agent to generate neighborhood solutions.

S and B: The controllers that define the domain of the new neighborhood solutions.

LS: Component of the local memory of each agent. It is a "$2*m_c*n$" matrix composed of new neighbors of the current seed with regard to controller S. It has "m_c" neighborhood solutions of σ_u^* which are based on "insert" moves and "m_c" neighborhood solutions of σ_u^* which are based on the "exchange" moves.

LB: Component of the local memory of each agent. It is a "$2*m_c*n$" matrix composed of new neighbors of the current seed with regard to controller B. It has "m_c" neighborhood solutions of σ_u^* which are based on "insert" moves and "m_c" neighborhood solutions of σ_u^* which are based on the "exchange" moves.

VS: Component of Local memory of each agent. It is a "$2*m_c$" matrix composed of weights of the elements of LS.

VB: Component of Local memory of each agent. It is a "$2*m_c$" matrix composed of weights of the elements of LB.

m_b: Selected new solutions from *LB* which will be sent for evaluation by objective function..

m_s: Selected new solutions from *LS* which will be sent for evaluation by objective function..

$E_{w,t}$: Weights of neighborhood solutions in VS and VB.

pr_i: Probability assigned to each element if VS and VB.

e_p: Elements in the elite list, p=1,...,P.

G_1: Component of global memory. It is a domain controller matrix, whose elements are changed by the behavior of controller *B* and *S*. $G_1 = \{(s_{ij}, b_{ij}), i = 1,..., n,\ j = 1,..., n\}$, (s_{ij}, b_{ij}) shows the domain of neighborhood solution which can be generated from

any sequence if job "i" at position "j" is selected to be the neighborhood generator.(i.e. If job "i" at position "j" is a tardy job and it is randomly selected to be inserted or exchanged with its previous jobs in the sequence)

α_c: A control parameter to update G_1.

G_2: Component of the global memory. It is a weight matrix that that is used for assigning an approximation value to each neighborhood of an agent. It is composed of O_{ij}'s which shows the weight of having job "i" at position "j". It is being updated based on the f_{iter}^{k} and F_{iter}.

$f_{iter}^{k,i}$: The value which is used to update G_2. It is the relative measure of the current seed of agent "k"'s objective function to other elite list objective function multiplied by another measure which is a relative contribution of job "i" to the current seed objective function.

F_{iter}: The value which is used to update G_2, if the current seed of any agent improves the global optima.

n_{ij}: Job i being in position j for every e_p.

N_e: Matrix for generating new-born solutions by SA method.

N'_e: Matrix for generating new-born solutions by Assignment problem method.

E_i: List of jobs before tardy job t_i.

π_i: Probability for R&M method.

a: Parameter in R&M method.

ARPD: Average relative percentage for the Co-MPS method reaching the global optima.

n_{hit}: Percentage of finding the optimal solution by Co-MPS algorithm.

- **Chapter 5**

n_t: Size of TSP problem solved by TSP-MPS.

c_{ij}: Cost of ij-edge of TSP graph.

V: Set of vertices for TSP graph.

E: Set of edges for TSP graph.

$G(V,E)$: Graph including (V,E).

T: Tour in the graph $G=(V,E)$.

x: Characteristic vector of a tour in $G=(V,E)$.

x_e: Binary variable representing edge "e" is in the tour with characteristic vector x.

burma14: Test problem of TSP-MPS with 14 cities.

Gr21: Test problem of TSP-MPS with 21 cities.

Gr96: Test problem of TSP-MPS with 96 cities.

- **Chapter 6**

T_i: Length of service time for patient i.

b_i: Start of service time for patient i.

e_i: End of service time for patient i.

A_i: Arrival of patient i.

p_i: Patient i's idle time.

M_i: Physicians' idle time during patient's "i" visit.

r: Unit idle cost of patients.

s: Unit idle cost of physicians.

n_i: Number of patients to e scheduled in i^{th} block.

n_I: Number of patients given identical appointment time at start of the session.

a_i: Job allowance.

μ : Mean of consultation time.

C(1): Cardiac patients.

T(2): Thoracic patients.

V(3): Vascular patients.

W(4): Women patients.

O(5): Other patients.

C_a: Number of classes of agents in ASMPS.

m_a: Number of neighbors to be generated in ASMPS algorithm.

t_a: Time resolution of ASMPS algorithm.

α_a : Parameter of G_1 in ASMPS algorithm.

N_a: Matrix of ASMPS algorithm for generating new-born solutions.

Chapter 1

Introduction

1.1 Motivations

In today's world, certain types of optimization challenges arise in several industries. In the field of mechanical design, one of the main challenges is to develop methodologies which can be applied to expensive black-box objective functions which include computation intensive simulation models such as finite element analysis (FEA) and computational fluid dynamics (CFD). Gradient information from these models are often expensive to obtain or un-reliable due to noises. In order to approach these types of optimization problems, direct and non-gradient optimization methodologies need to be developed.

In the field of production scheduling, the main challenge is to develop algorithms that can practically find the optimal or near-optimal solutions in a huge combinatorial solution space. Generally efficient meta-heuristic approaches that do not depend on the numerical properties of the objective function need to be developed for combinatorial optimization problems.

In the field of healthcare scheduling or healthcare re-engineering, which has drawn intensive attention from operation researchers during past decades, one of the main challenges is to develop optimization methodologies which can be adapted to any complex system or subsystem of a healthcare organization. Developing simulation-based

optimization methods that treat a healthcare system as a black-box function can be a significant contribution to the field of healthcare optimization.

All in all, in the three above mentioned industries, mechanical design optimization, and scheduling in production and healthcare, developing direct, non-gradient, and simulation-based optimization methodologies which approach the objective function or a simulation model as a black-box function is needed.

From the literature, meta-modeling, stochastic search algorithms, meta-heuristics, or simulation-based optimization methods are some of the solution methodologies that have received numerous thoughts and may suit for the complex system optimization. In fact, all these methods treat the objective function of an optimization model as an external procedure or black-box function and search for the optimal solution inside a solution space. However, there lacks of a methodology that is capable of efficiently solving problems of non-gradient and expensive simulation (black-box) based optimization problems arise from different fields.

1.2 Objectives

In this research, a series of optimization methodologies that do not use gradient-based information and efficiently solve simulation-based optimization problems are to be developed to tackles various challenges in the fields of product design and scheduling in production and healthcare.

1.3 Thesis Outline

In this work, four different versions of Mode Sampling Pursuing (MPS) algorithms have been developed for the above-mentioned challenges in real-world applications. These four optimization methodologies provide solutions to the optimization problems in mechanical design optimization, single combinatorial sequencing problems, and healthcare optimization problems. All of the methods are rooted in a previously developed MPS method by Wang et al. (2004a) for optimization of black-box design problems with continuous variables ('C-MPS'). All of the methods developed in this work are direct search methods without using gradient information, and targeted for simulation-based problems.

Chapter 2 provides a literature review on related studies that include optimization algorithms in general, classical optimization algorithms, stochastic optimization algorithms (meta-heuristics), metamodeling algorithms, and simulation-based optimization algorithms. In Chapter 3, a 'D-MPS' (Discrete Mode Pursuing Sampling) method will be elaborated for product design with discrete variables or mixed discrete and continuous variables. Results of the 'D-MPS' for several test cases will be discussed and compared to other algorithms. Based on 'D-MPS', a 'Co-MPS' (Combinatorial-MPS) algorithm is developed and applied to solve the single machine total tardiness problem, which is described in Chapter 4. Chapter 5 will discuss the 'TSP-MPS' (Traveling Salesman Problem-MPS) algorithm, which is a natural extension of 'Co-MPS' to solve the well-known scheduling problem, the Traveling Salesman Problem. The results of TSP-MPS will be compared to other methods in the literature. In Chapter 6, a

variation of Co-MPS for appointment scheduling, namely 'AS-MPS' (Appointment Scheduling-MPS), will be introduced. Conceptual result of this approach for Pre-admission clinic in Health Science Center (HSC) in Winnipeg will be discussed. Finally, Chapter 7 will conclude the overall research and discuss about future research.

Chapter 2

Literature Review

As discussed in Chapter 1, a new non-gradient optimization methodology has been proposed in this thesis, which has application in mechanical design (D-MPS, Chapter 3), combinatorial optimization problems, namely single machine sequencing (Co-MPS, Chapter 4) and Traveling salesman problem (TSP-MPS, Chapter 5) in addition to appointment scheduling in healthcare system (AS-MPS, Chapter 6). The main concept of all the proposed algorithms is rooted in Mode Pursuing Sampling (MPS) method, which was previously developed by Wang et al. (2004a).

All of the proposed methods in this thesis are meta-model based or meta-heuristic optimization algorithms, and are non-gradient methods. In fact, classical approaches of non-gradient optimization or direct search methods establish a basis for the optimization algorithms developed in this thesis. Therefore approaches of direct search methods of optimization will be briefly reviewed in Section 2.1.

MPS on continuous variables (C-MPS) was introduced for mechanical design problems by Wang et al. (2004). In Chapter 3, a generalization of C-MPS has been proposed for discrete domain mechanical design optimization problems. The so called D-MPS method can be classified as a meta-model based design optimization (MBDO) algorithm. MBDO is a new category of non-gradient optimization methods, which are different from direct

search methods. In this sense, I will discuss the literature on MBDO techniques in Section 2.2.

In Chapter 4 and 5, the concept of MPS will be generalized for combinatorial problems, and Co-MPS and TSP-MPS are developed, respectively. Although the key concepts of the two algorithms are similar to MPS, Co-MPS and TSP-MPS have more similarities with meta-heuristic approaches rather than meta-modeling approaches. Meta-heuristic approaches are often called stochastic search methods, and are part of non-gradient optimization methods. Hence, I will discuss the literature review on meta-heuristic approaches in Section 2.3.

In Chapter 6, a new version of MPS-based optimization algorithm has been proposed, which has an application to clinic scheduling. The AS-MPS algorithm uses discrete-event simulation as a black-box for its objective function evaluation. Simulation-based optimization algorithms can be approached by all the non-gradient optimization techniques which include direct search methods, MBDO methods, and meta-heuristics. Hence, I will describe the literature review on simulation-based objective function with a focus on discrete-event simulation in Section 2.4.

As discussed earlier, the MPS-based optimization algorithms have been tested on several applications: mechanical design problems, Single Machine Total Tardiness Scheduling problem, Traveling salesman problem, and appointment scheduling. The literature review of every application will be discussed in each chapter.

2.1 Direct search optimization algorithms

In some cases of an unconstrained optimization problem over $f(x)$, derivatives of $f(x)$ are unavailable or untrustworthy. Although a necessary condition for a minimizer or maximizer is that the gradient equals to zero, there are many optimization methods that do not explicitly use derivatives. Direct search methods are one of the basic groups of non-gradient optimization methods. In this section, I will give a brief review of direct search methods. In Section 2.1.1, I will describe direct search methods with a review of the backgrounds of some of these methods. Section 2.1.2 is a discussion of the practicality of direct search methods. Section 2.1.3 discusses the convergence analysis of the direct search methods.

2.1.1 Direct search methods: Background review

Since 1950s, direct search methods have been known (Lewis et al. (2000)) gives a detailed historical review on direct search methods). However, by the early 1970s, these methods were ignored by the mathematical optimization community and disappeared from its literature for reasons summarized in Swann's (1971):

". . . [direct search] methods have been developed heuristically, . . . no proofs of convergence have been derived for them, . . . [and] sometimes the rate of convergence can be very slow."

As described in Rosenbrock (1960), early direct search methods were based on heuristics often derived from examples drawn in two dimensions. In addition, as discussed in Gill et

al. (1981), many users preferred not to involve in the calculation of gradients, which for a long time has been the biggest source of error in applying optimization software libraries. In 1991, direct search methods again became an interesting topic in research, especially in the context of parallel computing and an accompanying convergence analysis (Troczon (1991)). Since then, two things have become clear:

1. Direct search methods remain an effective option, and sometimes the only option, for several varieties of difficult optimization problems, e.g., the pattern search algorithms by Torczon (1997) and grid-based framework of Coope et al. (2000).

2. For a large number of direct search methods, it is possible to provide rigorous guarantee of convergence, e.g., the convergence analysis of grid based methods in Coope et al. (2001), convergence analysis of pattern search algorithms in Torczon (1997), and convergence analysis of the class of direct search methods in Kolda et al. (2003).

Here, I will mention one of the earliest examples of a direct search method as described by Davidon (1959):

"Enrico Fermi and Nicholas Metropolis used one of the first digital computers, the Los Alamos Maniac, to determine which values of certain theoretical parameters (phase shifts) best fit experimental data (scattering cross sections). They varied one theoretical parameter at a time by steps of the same magnitude, and when no such increase or decrease in any one parameter further improved the fit to the experimental data, they halved the step size and repeated the process until the steps were deemed sufficiently small. Their simple procedure was slow but sure, and several of us used it on the Avidac computer at the Argonne National Laboratory for adjusting six theoretical parameters to

fit the pion-proton scattering data we had gathered using the University of Chicago synchrocyclotron."

2.1.2 Direct search method in practice

As mentioned earlier, direct methods have been used during 1950-1970 due to their simplicity in implementation. In today's situation, complicated use of derivative based methods and existence of options to generate approximations to the gradient (the vector of first partial derivatives) and/or the Hessian (the matrix of second partial derivatives)help users to use derivative-based methods. As discussed in Bischof et al. (1996), Bischof et al. (1992) and Griewank (1989) automatic differentiation tools are now widely available.

The important question in today's optimization communities is when should we use the direct search methods? As will be described in next chapters of this thesis, for expensive black-box functions whose derivatives can not be obtained, or the objective value is based on the simulation software, or one is dealing with discrete or even combinatorial domain with a huge solution space and a NP-hard proven problem, the direct search method or the variation of theses methods is the best choice.

Although there are several definitions of direct search methods that can be found in the literature, most of them are close to each other. I will bring the more generalized definition of direct search methods as follows:

In Gill, Murray, and Wright (1981), the authors remarked that, "Methods based on function comparison are often called *direct search methods*." Likewise, Trosset (1997) gave the definition, "A direct search method for numerical optimization is any algorithm that depends on the objective function only through the ranks of a countable set of function values."

2.1.3 Analyzing convergence of direct search methods

In the classical line search methods such as steepest descent, Newton, or quasi Newton methods, it is impossible to prove second-order convergence results (i.e. convergence to a minimizer (Hessian matrix being positive or semi definite)). Line search methods are considered as first-order methods, while direct search methods are zeroth-order methods. Although in practice, first-order methods, reliably find local minimizers, the global convergence analysis of such methods only establishes first-order convergence. (First order convergence of an optimization method means that the solution found in any iteration will converge to a local optima of f).

In a review of Kolda et al. (2003), the authors have given theorems for first-order convergence results for a particular class of zeroth-order methods . Kolda et al. (2003) generalized and illustrated a framework in which some other direct methods could be classified into one category: *generating set search* (GSS). GSS includes the generalized pattern search methods previously analyzed by Torczon (1997)], which cover special cases of the pattern search algorithm of Hooke and Jeeves (1961), multidirectional search variants of the EVOP technique introduced by Box (1957), compass search (Described

from Davidon (1959) as well as the extensions introduced by Lewis and Torczon (1996) to handle positive bases. Additionally, GSS includes a subset of the algorithms encapsulated in the grid-based framework of Coope and Price (2000, 2001).

Kolda et al. (2003) proved that within a certain assumption, any convergent sequence of unsuccessful iterates must converge to a point where the gradient vanishes (or equals to zero). In fact, they showed that even though there was no explicit knowledge of the gradient, *at any unsuccessful iteration* there was an implicit bound on the norm of the gradient in terms of the step-length control parameter Δk (where Δk is the step size at any iteration).

It should be noted that all the direct search methods can be used as a domain controller for other search strategies such as stochastic or meta-heuristic searches, MBDO, etc.

One of the main contributions of this thesis is the DSM (Double Sphere Method), which has been used in all of the algorithms proposed in Chapters 3-6. This method incorporates two hyper-spheres that enlarge and shrink the search space based on successful or unsuccessful (improving or non-improving) iterations in the past. It brings a new perspective to direct search methods. Although I haven't proved its convergence, future studies can be directed toward giving the proof of DSM convergence under the Lipchitz condition for its derivative and on a continuous function which is differentiable on its domain.

In the next section, I will describe another class of non-gradient optimization methods, called as meta-model based design optimization (MBDO) methods. The D-MPS algorithm which will be described in Chapter 3 is a MBDO Approach.

2.2 MBDO approaches

One of the main categories which is related to this thesis is MBDO techniques. Although, these methods have been used specially for mechanical design problems, the concept of MBDO can be used in other domains.

As discussed in Conn et al. (1997), there are some non-gradient optimization methods that are not based on Taylor's series approximation of objective/constraint functions constructed by using analytical or accurate finite-difference estimates of derivatives. Instead, the function models are constructed via least squares fits (Glad et al. (1977)) or interpolation techniques. I will give a definition of the meta-model in Section 2.2.1. Section 2.2.2 will describe the applications of meta-modeling techniques. Section 2.2.3 will focus on one of the most important applications of meta-modeling, which are MBDO strategies.

2.2.1 Definition of meta-model

Most of the engineering optimization problems include computation intensive simulation models such as finite element analysis (FEA) and computational fluid dynamics (CFD). Instead of being an explicit function, the expensive simulation processes are treated as a black-box function and a function without a known or reliable derivative. Researchers

are attracted to using approximation methods to model these expensive processes. The approximation models are then called metamodels. Metamodels evolve from the classical Design of Experiments (DOE) theory, in which one of the simplest metamodels, polynomial function, is used and called as response surface. As described in Wang et al. (2007), metamodel is a simplified model of a computation-intensive or complex model;. metamodelling is the process of creating a metamodel. Having a metamodel, optimization methods can then be applied to find global or local optima, which is referred as metamodel-based design optimization (MBDO). In this sense, there is a difference in global meta-modeling (or global approximation) and MBDO. Although, the main focus is on MBDO since it is directly related to the thesis work, global meta-modeling as a key technique of MBDO will be discussed in the next section.

2.2.2 Meta-modeling techniques

As described in Ulman (2002) and Wang et al. (2007), global metamodeling provides a decision-support role for design engineers. One of the main categories of the supporting functions is helping MBDO procedures. I will briefly describe functionalities of metamodeling and try to summarize a broad range of literature on MBDO in particular.

1. Model approximation: As described earlier, having a global approximation for any computation-intensive process across the design space is one of the main functionalities of meta-model. As described by Wang et al. (2007), model approximation is composed of sampling, meta-model choice, model fitting, and model validation. Finding an accurate global meta-model with a reasonable cost will be useful for MBDO, which I will discuss later.

2. Sampling mechanisms: As discussed earlier, metamodeling is originated form the theory of Design of experiments. Simple sampling mechanisms which can be found in the literature are mostly the same as what have been used in planning the experiments so that the random error will have the minimum effect on the parameter estimation from the experiments. By assuming every parameter of the experiments such as temperature as one dimension, planning the experiment is the same as sampling points in a space.

In addition to the experimental design sampling schemes such as fractional factorial (Myers et al. (1995)), central composite design (CCD) (Chen (1995)), Box-Behnken (Myers et al. (1995)), alphabetical optimal (Giunta et al. (1997)), and Plackett-Burman designs (Myers et al. (1995)) , Koehler and Owen (Koehler et al. (1996)) described several Bayesian and Frequentist "Space Filling" designs, including maximum entropy design (Currin et al. (1991)), mean squared-error designs, minimax and maximin designs (Johnson et al. (1990)), Latin Hypercube designs, orthogonal arrays, and scrambled nets.

Uniformity and the sample size are main criteria for comparing these sampling mechanisms. A comparison of these sampling methods is in Simpson et al. (2001).
Also, due to the difficulty of forecasting the sample size or even its density, sequential and adaptive sampling has gained popularity in recent years. Some of them are the models which have been proposed by Lin (2004), Sasena et al. (2002) and Wang (2003).

3. Meta-model choice: After sampling in the design space, a meta-model should be constructed based on these sample points. It could be a simple linear spline, or a complex adaptive neural network. Some of the models are as follows:

    - Radial Basis Functions(RBF) (Dyn et al. (1986))
    - Multivariate Adaptive Regression Splines (MARS) (Friedman (1991))
    - Least Interpolating Polynomials (De boor et al. (1990))
    - Inductive learning (Langley et al. (1995))

    In recent years, Kriging models and related Guassian processes are intensively studied (Martin et al. (2005)). For comparison between these meta-model choices, a comprehensive study has been done in Wang et al. (2007).

4. Model validation: Several validation model has been discussed in the literature such as: Meckesheimer et al. (2002) studied the cross-validation method, mean square error (RMSE), the maximum absolute error (MAX), and "R" square.

5. Design space exploration: Sensitivity analysis and "what if" questions are good examples which can be classified into design space exploration. Having a meta-model, one can understand the effect of any variable on the design space (solution space). Also, visualization of the data on hand is another application of meta-models, which help the user explore the design space. Winer and Bloebaum (2002) developed a visual design steering method based on the concept of Graph Morphing. Eddy and Kemper proposed cloud visualization for the same purpose (Eddy et al. (2002)).

6. Problem formulation: Along as other practical tools of metamodel, one can use a meta-model to formulate an optimization problem that is easier to handle. For example, some of the objectives or constraints can be eliminated, or modified. Box and Draper

(1969) introduced a method to gradually refine the response surface to capture the real function by eliminating unimportant variables. Welch et al. (1992) documented a systematic approach in order to screen the variables.

As discussed earlier, one of the main applications of metamodeling is helping the optimization problems. In the next section I will discuss this issue.

2.2.3 Meta-model based design optimization strategies (MBDO)

As categorized by Wang et al. (2007), three types of MBDO strategies can be found in the literature.

1. The first group is sampling all the solution space and fitting the meta-model to all the points. Then, after validating the model, perform a global optimization strategy. Refs. (Gu (2001)), Myers (1995)) are some of the examples of this type of MBDO.

2. The second group involves the validation and/or optimization in the loop in order to make a decision on re-sampling and re-modeling. Although re-sampling is done iteratively, the meta-model does not have a direct effect on the sampling scheme. As discussed in Osio and Amon (1996), a multi-stage kriging model can be used to sequentially improve the accuracy of approximation as additional sample points were generated.

3. The third group of methods directly sample to desired areas as guided by the meta-model. C-MPS developed by Wang et al. (2004) and its extension for discrete domain (D-MPS) that will be discussed in Chapter 3 belong to this category. Shan et al. (2005) is another example that used this group of MBDO.

MBDO algorithms have many advantages. The concept of having approximation over all the solution space and gradually using the meta-model to sample more points on the interesting regions of the objective function (the third group of MBDO), is an interesting idea which has not been used in other optimization algorithms. Meta-heuristic algorithms or stochastic search methods do not have this top-down perspective toward the objective function. They use nature-inspired algorithms which have a bottom-up perspective and operate on single solutions instead of a global approximation or meta-model.

In the next section, I will describe meta-heuristic algorithms which have been discussed in the literature. Co-MPS and TSP-MPS algorithms which we developed in Chapters 4 and 5 have similarities to these algorithms in addition to their basic concept rooted in MPS (the third group of MBDO) .

2.3 Meta-heuristic approaches

"Stochastic searches", "simulation-based optimization algorithms", "nature-inspired approaches to optimization" and "meta-heuristic algorithms" are names of the often same group of optimization approaches. Their different names are derived when they are used in different areas of optimization. In the combinatorial optimization community, "Genetic algorithms," "Tabu search," "Simulated annealing," and "Ants algorithms" are called meta-heuristics. Although different in nature, the above mentioned approaches have some similarities and to some extent, they are complicated non-gradient or direct search approaches. In this section, I will give a literature survey on the definition and history of meta-heuristic in combinatorial optimization in Section 2.3.1. Main four

branches of meta-heuristics that have similarities to the Co-MPS and TSP-MPS (Chapters 4 and 5) will be discussed in Section 2.3.2. Since I believe Co-MPS and TSP-MPS are a mixture of meta-modelling and meta-heuristics, qualitative comparison between meta-model approaches and meta-heuristics will be discussed in Section 2.3.3.

2.3.1 Definition and history of meta-heuristic algorithms in combinatorial optimization

First introduced by Glover (1986), the term "meta-heuristic" is a very complicated concept and its hard to find a straight-forward definition for it. Since 1980's, several approaches have been proposed for NP-hard combinatorial problems, which were neither heuristics (Heuristics are special algorithms for specific problems) nor exact algorithms which guarantee the optimum. They were complex forms of direct search methods, which did not use any derivative information from objective function. Moreover, they use the objective function only as the measure for ranking the solutions which change through the run time toward the global optima or near-optima. These meta-heuristic algorithms, as described in Osman et al. (1996), incorporate concepts from biological evolution, intelligent problem solving, mathematical and physical sciences, nervous systems, and statistical mechanics.

As described in Gendreau et al. (2005), after the era of specialized heuristics which were typically developed to solve complex combinatorial optimization problems, the emergence of more general solution schemes changed the picture drastically. Now, the challenge is to adapt a metaheuristic to a particular problem or problem class, which usually requires much less work than developing a specialized heuristic from scratch. As described in Glover et al. (2003), metaheuristics, in their original definition, "are solution

methods that orchestrate an interaction between local improvement procedures and higher level strategies to create a process capable of escaping from local optima and performing a robust search of a solution space." One of the main features which Glover et al. (2003) has identified is the existence of procedures which can be separated from any meta-heuristic and put into another. In fact, meta-heuristics include any procedures that employ strategies to overcome being trapped in local optima in complex solution spaces. More importantly, those procedures that utilize one or more neighborhood structures as a means of defining admissible moves to transition from one solution to another are used more frequently in new metaheuristics or adaptive memory programming (Glover (1997)). As described in Osman et al. (1996), "A meta-heuristic is an iterative generation process which guides a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search spaces using learning strategies to structure information in order to find efficiently near-optimal solutions."

In most of the literature, meta-heuristic approaches are categorized in two groups (Gendreau et al. (2005)) and Glover et al. (2003)). The first group is population based methods, among which Genetic Algorithms and Ants algorithms are the most important ones. The second group is composed of "single thread" solution methods or point-to-point methods such as Tabu search or simulated annealing. In both groups, neighborhood exploration is one of the most important concepts. As describe by Glover et al. (2003), in the case of certain population-based procedures, neighborhoods are implicitly (and somewhat restrictively) defined by reference to replacing components of one solution with those of another. In some other approaches, neighborhood structures are used in

their full generality, including constructive and destructive neighborhoods as well as those for transitioning between (complete) solutions. On the other hand, in "single thread" solution approaches, which do not undertake to manipulate multiple solutions simultaneously, they use a wide range of available solutions that not only manipulate diverse neighborhoods but incorporate numerous forms of strategies ranging from thoroughly randomized to thoroughly deterministic ones, depending on the elements such as the phase of search or (in the case of memory-based methods) the history of the solution process (Gendreau et al. (2005)).

As will be described in Chapters 4 and 5, Co-MPS and TSP-MPS that have been developed in this thesis are a mixture of population based and single thread meta-heuristics. They are memory-based approaches which use a history of solution process not only for selecting a new neighborhood but also generating new neighborhood solutions. To our knowledge, none of the current meta-heuristics have used a memory-based strategy for generating new neighborhoods. DSM (Double sphere methodology), which is a complex form of direct search, is used for restricting the neighborhood generation in Co-MPS and TSP-MPS.

In the next section, I will describe four main population-based methods and single thread methods which share some features with Co-MPS and TSP-MPS.

2.3.2 Four main meta-heuristics for combinatorial optimization

As discussed in Section 2.3.1, meta-heuristics can be categorized into two main groups:

1. Population-based methods
2. Single thread methods

In the following sections, I will briefly describe the history and concept of four main meta-heuristics: Genetic algorithm, Ants systems (population-based methods), as well as Tabu search and Simulated annealing (single thread solution), which have similarities with the proposed TSP-MPS and Co-MPS in Chapters 4 and 5.

2.3.2.1 Genetic algorithm

In the historical review by Gendreau et al. (2005) and Glover et al. (2003), it is mentioned that although there have been other indications of evolutionary strategies by Rechenberg (1973) or idea of evolutionary programming by Fogel (1998), the first book which introduced the concept of Genetic algorithms was from John Holland "*Adaptation in Natural and Artificial Systems*" (Holland (1975)).

After 1975, Holland's graduate students, Dejung (1975) and Goldberg had focused more on optimization role of genetic algorithms. In 1989, an influential book by Glodberg (1989) "*Genetic Algorithms in Search, Optimization and Machine Learning*" made GA one of the dominant meta-heuristics.

At the same time as reported by Glover et al. (2003) in OR (Operations Research) community, the neighborhood search was introduced to help solve the NP-Hard problems to near-optimality (Lin (1965)). GA optimization procedures had some features which could adaptively incorporate those neighborhood search algorithms. The concept of "Selection" and "Mutation" were a completely new and practical idea in combinatorial optimization, which prevented the search from trapping into local optima. The first GA

introduced by Holland (1975) searched a function of solution space rather than the solution space directly. As mentioned by (Holland (1975)) they used a binary string which correspond to genotype and have intrinsic characteristic of phenotype which is a solution space. Although, Holland (1975) specifies that having binary coding is an "optimal" way of searching the neighborhoods, but there have been some ideas against that (Vose (1994)).

Nowadays, there have been so many new modifications of Genetic algorithms, which have been gathered under the name "Evolutionary computing" or "Evolutionary programming" (Fogel (1999)).

2.3.2.2 Tabu search

Introduced by Glover (1989), Tabu search in its original is basically a single thread metaheuristics. It is a local search which allows for non-improvement moves, while recording the moves which produce the visited solutions in a fixed short term memory for preventing the cycle of going back to visited solutions. Keeping track of tabu moves are done by local memory, while in most of the early tabu search papers, a long term memory or "aspiration list" is used to reinforce the attractive components. (For example, keeping the elite list and allowing tabu moves if they produce better results than before (Hertz et al. (1991)). Two main components of basic tabu search are the definition of solution space and closely linked to it, neighborhood structure. In fact, if "S" is the current solution, $N(S)$ will be a neighborhood of S, by the definition:

$N(S)$={Solutions obtained by applying local transformation to S}

As described in Glover et al. (2003), having a good knowledge of the problem at hand should make the tabu search designer to choose the best solution space and neighborhood structure (including local transformation or "movements"). A very important concept of Tabu search for combinatorial optimization problems, which can be implemented in any meta-heuristics, is making the use of local memory to control a tabu move. For example, in terms of "exchange" movements, a tabu move can be controlled to not exchange the second job with any other jobs in a sequence, or not exchange the second job with the third job, and so on. The tabu move is thus in the hand of the designer of the tabu search. In the Co-MPS method which will be explained in Chapter 4, I also make use of local memories to implement DSM. Such a strategy is a generalization of tabu moves. In fact, DSM is a dynamic function which restricts the moves based on previous iterations. In fact, instead of having a tabu move which is 0 and 1, I have a more flexible tabu move (any number between [0 1]), which allows smaller neighborhoods to be changed not all the neighborhood. Also, the local memory in Co-MPS has the task of assigning weights to neighbors in order to make them a candidate for the selection procedure.

Tabu search has some procedures that can be directly used in other algorithms. For example, as explained in Glover and Laguna (1997), making use of candidate lists can be helpful for implementing new conditions for changing a tabu move to a permissible one. Another procedure, which is mainly used in Tabu search strategies, is the use of intermediate-term memory, such as a recency memory, in which one records the number of consecutive iterations that various "solution components" have been presented in the current solution without interruption. This procedure is called "intensification". A typical

approach to intensification is to restart the search from the best currently known solution and to "freeze" (fix) in it the components that seem more attractive. In the Co-MPS and TSP-MPS, I refer to this procedure as "exploitation" by making the neighborhood small. Another procedure in tabu search is "diversification," which tries to move the search toward the places that have not been explored by the search procedure. One of the major diversification techniques as reported by Gendreau et al. (1993) is to force a few rarely used components in the current solution (or the best known solution) and restart the search from this point. In Co-MPS and TSP-MPS, I refer to this procedure "exploration" by expanding the neighborhood to avoid trapping into a local optimum..

In recent years, tabu searches are more focused on enabling parallel searches and making use of the elite list more and more (Crainic et al. (1997)). As described by Glover (1997) and Glover and Laguna (1997), "fragments" (elements) of excellent solutions are often identified quite early in the search process, but the challenge is to complete these fragments or to recombine them. In Co-MPS and TSP-MPS, I make use of a concept called "new-born" agent, which tries to combine the good fragments of solutions found so far. I make use of the cooling concept from "Simulated annealing" to recombine the solutions. In the next section, I will discuss the concepts and history of "Simulated annealing."

2.3.2.3 Simulated annealing

As described in Glover et al. (2003), the world of combinatorial optimization was shattered in 1983 by the appearance of a paper in which the authors (Kirkpatrick et al.

(1983) were describing a new heuristic approach called *Simulated Annealing* (SA). The important fact was the existence of a proof of convergence to an optimal solution of a combinatorial problem with Markov chain, albeit in infinite computing time.

Simulated annealing is a "single thread" meta-heuristic and is named because of its analogy to the process of physical annealing with solids, in which a crystalline solid is heated and then allowed to cool very slowly until it achieves its most regular possible crystal lattice configuration (i.e., its minimum lattice energy state), and thus is free of crystal defects. The way in which simulated annealing differs from other local search procedures is by escaping the local optima by allowing hill-climbing moves (i.e. moves which worsen the objective function value). As the number of iterations reaches the maximum number of iteration, the temperature parameter is decreased to zero and hill climbing moves occur less frequently.

At any iteration, if we have w as the current solution, and if w' is the neighborhood solution of w, then the algorithm accept w' with the probability:

$$P(\text{Accept } w' \text{ as next solution}) = \begin{cases} \exp[-f(w') - f(w))/t_k] & \text{if } f(w') - f(w) > 0 \\ 1 & \text{if } f(w') - f(w) \le 0 \end{cases} \qquad \textbf{Equation 2-1}$$

where t_k is the temperature parameter at iteration k, such that

$$t_k > 0 \quad \text{for all } k \text{ and } \lim_{k \to +\infty} t_k = 0 \qquad \textbf{Equation 2-2}$$

For studying the behavior of "Simulated annealing," many studies model the procedure as a Markov chain, and find the steady sate probabilities. The proof of convergence is given in infinite solution time (Davis (1991)).

In Co-MPS and TSP-MPS, I used the concept of cooling in generating new-born solutions. For combining the features of best known solutions, I compare them with each other and if the probability of having a job at a given position is higher than a dynamic probability function value, I accept and freeze the job at that position. As the number of iterations reaches the maximum, a job is fixed at a specific position if many occurrences of this job at the specific position have been seen in elite solutions.

2.3.2.4 The Ant colony optimization metaheuristic

From the historical point of view, the initial ant colony optimization method was first proposed by the name of ant systems (AS) for the well-known Traveling salesman problem(TSP) (Dorigo et al. (1997a, 1997b)). As described by Glover et al. (2003), despite the encouraging initial solutions, AS could not compete with the state-of –the art algorithms for TSP. Later on, extensions of "Ant system algorithms" resulted in Ant colony optimization algorithms (ACO) and other modifications which resulted in world class results for NP-hard problems.

Ant systems algorithms are inspired by pheromone trails of ants for finding the shortest way to their nest. As described by Deneuburg et al. (1990), in many ant species, individual ants may deposit a pheromone (a particular chemical that ants can smell) on

the ground while walking. In fact, by sensing pheromone trails, ants can follow the path to food discovered by other ants. Also, they are capable of exploiting pheromone trails to choose the shortest among the available paths leading to the food.

In the optimization world, as described in Colorni et al. (1992), which was one of the earliest publications on Ants colony optimization algorithm, ACO algorithms are population-based meta-heuristic and are composed of artificial ants. Artificial ants are stochastic solution construction procedures that probabilistically build a solution by iteratively adding solution components to partial solutions by taking into account (i) heuristic information on the problem instance being solved, if available, and (ii) (artificial) pheromone trails which change dynamically at run-time to reflect the agents' acquired search experience. The important factor about ACO algorithms is that they use construction procedures and not local search procedures.

Here I will discuss every agent's behavior in more detail.

Consider an optimization problem of (S, f, Ω), where "S" is the set of candidate solutions, "f" is the objective function which assigns any $s \in S$ an objective value of $f(s)$ (in case of static objective function), and Ω is the set of constraints. Now, as the Ants system algorithm is the constructive meta-heuristic, suppose:

1. A finite set of $C=\{c_1,c_2,..,c_n\}$ of components is given. In case of single sequencing, these components can be : C={1-2-4, 1-2, 3, 1, 2 , ..}, where the activities or jobs to be sequenced are:{1,2,3,4}

2. The state of the problem are defined in terms of sequences $x=<c_1,c_2,..c_k,..>$. The set of all possible sequences is denoted by X.

3. The set of candidate solutions "S" is a subset of X. For example, sequence "1-2-1-3-4" or "1-2" is in X, but may not be a candidate solution.

4. The finite set of constraints Ω defines the set of feasible states X', with $X' \subset X$.

Given the above representation, artificial ants build solutions by moving on the construction graph $G=(C, L)$, in which the vertices are the components "C" and the set "L" fully connects C (elements of L are called "connections"). The problem constraints are implemented in the policy followed by the artificial ants when building solutions. In general, ants try to build feasible solutions, but if necessary, they can generate infeasible solutions. Components and connections have associated with them a *pheromone trail* (τ_i if associated with components, and τ_{ij} if associated with connections) encoding a long-term memory about the whole ant search process that is updated by the ants themselves, and a *heuristic value* "y" (y_i and y_{ij} respectively) representing *a priori* information about the problem instance definition or run-time information provided by a source different from the ants. In many cases, "y" is the cost, or an estimate of the cost, of extending the current state. These values are used by the ants' heuristic rule to make probabilistic decisions on how to move on the graph.

Two main features of each ant are as follows:

- It has a memory that it uses to store information about the path it followed so far. Memory can be used (i) to build feasible solutions (i.e., to implement constraints), (ii)

to evaluate the solution found, and (iii) to retrace the path backward to deposit pheromone.

- It selects the move by applying a probabilistic decision rule. Its probabilistic decision rule is a function of (i) locally available pheromone trails and heuristic values, (ii) the ant's private memory storing its past history, and (iii) the problem constraints.

Here, I will mention how ant k at state i will choose to go to state j, and also how each ant updates the pheromone trails which connect with other ants.

- Probabilistic decision rule for each ant to move to the new state

At each iteration of the Ant system algorithm, an ant k will choose to go from state i to j with a probability given by:

$$p^{k}_{ij}(t) = \frac{[\tau_{ij}(t)]^{\alpha}.[y_{ij}]^{\beta}}{\sum_{l \in N_i^k}[\tau_{il}(t)]^{\alpha}.[y_{il}]^{\beta}} \quad if \ j \in N_i^k \qquad \textbf{Equation 2-3}$$

Where t shows the number of the current iteration and n_{ij} is *a priori* available heuristic information. For example for TSP problem it can be $1/d_{ij}$, where d_{ij} shows the distance from city i to j, and states correspond to the cities. In the next step, ants move from an initial city to other cities for constructing candidate solutions. In this case, sequences will be set of all the cities. N_i^k is the set of all the possible neighbor solutions which ant k can go from state i.

- Updating the pheromone trail

In AS this task is done by first lowering the pheromone trails by a constant factor (this is called "pheromone evaporation") and then allowing each ant to deposit pheromone on the edges that belong to its tour:

$$\forall (i, j)\ \tau_{ij}(t+1) = (1-\rho).\tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau_{ij}^{k}(t) \qquad \textbf{Equation 2-4}$$

The parameter ρ is used to avoid unlimited accumulation of the pheromone trails and enables the algorithm to "forget" previous "bad" decisions. Also, suppose one has m agents and $\Delta\tau_{ij}^{k}(t)$ is the amount of pheromone ant k deposits on the edges i, j at iteration t; it is defined as

$$\Delta\tau_{ij}^{k}(t) = \begin{cases} 1/L^{k}(t) & \text{if edge}(i,j)\text{is used by ant "}k\text{"} \\ 0 & \text{otherwise} \end{cases} \qquad \textbf{Equation 2-5}$$

In the above equation, L^k is the objective function of ant k.

Since, Co-MPS and TSP-MPS both have agents that try to search for an optimum; they have more similarity to Ant algorithms than other meta-heuristics. I will describe the main differences of Co-MPS and Ant systems algorithm as follows:

- Firstly, Co-MPS and TSP-MPS came from a meta-model concept of design optimizations, which tries to have a control over all the solution space. In fact, these algorithms are iteratively improving heuristics and not constructive methods such as Ant systems or any ACO algorithm.
- Global memory of Co-MPS and TSP-MPS has almost the same structure as pheromone trails of connections in Ants systems. The main difference is that τ_{ij} shows the weigh for having j^{th} component after i^{th} component, without considering the position of any of i^{th} and j^{th} component in the main solution. In Co-MPS, Q_{ij} in the global memory denotes the weight of having i^{th} component at the j^{th} position. Also, the way that the global memory is being updated is dependent of the relative measure

of the objective function value of every agent and the summation of elite solutions' objective function values.

- In Co-MPS, moving to a new neighborhood solution is done in the local memory of each agent, which assigns the value from global memory to every neighborhood of the current seed of any agent and then performs a selection to find the new solution. In fact, Co-MPS makes an approximation over the neighborhoods of the current solution. This approximation evolves over the optimization run-time and becomes more exact at the end of the algorithm. In ACO, the probability distribution is about selecting a new sequence to enter the agents' current solution, not changing the whole neighborhood. This probability also becomes more exact in the last iteration of the ACO due to pheromone trails of all the agents in previous solutions.
- Generating neighborhood is restricted by DSM (Double Sphere Method) which is a complex direct search method or a generalized tabu-move.

2.3.3 Comparison between MBDO and meta-heuristic approaches

Although meta-model based optimization algorithms are in the design optimization domain and thus called meta-model based design optimization (MBDO) and meta-heuristics are in the combinatorial domain, both have some concepts that can be modified and help designing new optimization algorithms.

Here I describe some differences between these two groups of methods.:

- MBDO and especially the third class of MBDO as described in Section 2.2, have a top-down perspective toward the optimization problem. The

sample points are not intelligent in the way that they do not change to become new sample points. The meta-model is helpful in giving the whole approximation over the objective function. The intelligence of third group of MBDO such as MPS (Wang et al. (2004)), is to make the meta-model more exact around the interesting region of the objective function. On the other hand, meta-heuristic approaches have the bottom-up perspective and the solutions are intelligently changed to each other to find the better solutions.

- MBDO strategies are practical for regular objective functions (those do not have so many ups and downs in a relatively small domain). On the other hand, combinatorial objective functions, are not regular, and thus focus should be on the moves of changing the solutions and not the solutions themselves.
- Intelligence of the third class of MBDO is in the use of past-iterations information to sample more around the mode of the function and make the meta-model more accurate over those areas, while intelligence of meta-heuristics is in finding good sections of sequence and try to combine them.

In this sense, in Co-MPS and TSP-MPS adopted the main advantage of MBDO, which is having a rough approximation of the whole objective function and try to evolve or modify it over the optimization run. At the same time, I used the concepts of selection from genetic algorithms, concept of temperate cooling from simulated annealing for "new-born" agents, and concept of cooperative agents from Ant colony optimizations.

In Co-MPS and TSP-MPS, having a global memory that stores the weights of having any job or city on any position plays the role of objective function approximation. Agents, then try to locally search the mode of the "movement function", instead of solutions directly.

2.4 Simulation-based optimization algorithms

Simulation-based optimization algorithms have been discussed in several areas of the literature such as design optimization or complex combinatorial situations. Simulation-based optimization algorithms are basically optimization procedures when there is no explicit formula for the objective function and the output value of the objective function is a stochastic function of decision variables. In fact, the objective function is in the black-box format, whether it is computationally expensive such as finite element analysis software (FEA) or the combinatorial interaction of factors is huge such as service simulation in healthcare system where discrete-event software simulation is used.

Simulation-based optimization problems can be approached by all the methods we described in previous sections: direct search methods, MBDO algorithms, or meta-heuristics optimization algorithms. In this section, I describe the general model of the simulation-optimization problem in Section 2.4.1. In Section 2.4.2, I will categorize the approaches which have been discussed in the literature by referencing to the methods described in previous Sections 2.1-2.3. Although all the methods which have been developed in this thesis can be applied to simulation-optimization problems, the AS-MPS algorithm discussed in Chapter 6 is specifically designed with the discrete-event

simulation of the pre-admission clinic in the Health Sciences Center in Winnipeg. I will discuss some issues related to the differences of AS-MPS algorithm and other discrete-event simulation-based optimization algorithms in the literature in Section 2.4.3.

2.4.1 General model of simulation-based optimization problem

The formulation of simulation-based optimization problems is often done for maximization or minimization of the expected value of the objective function of a system. Although as reported by Azadivar (1999), operation of a system might be considered optimal if the risk of exceeding a certain threshold is minimized, as described by Guikema et al. (2004) in other situations, one might be interested in minimizing the dispersion of the response rather than its expected value. Here, let's limit to optimization of the expected values.

As denoted by Prudius et al. (2004), an optimization problem of the form

$$\max_{\theta \in \Theta} f(\theta) = E(X_\theta) \qquad \textbf{Equation 2-6}$$

where $f : \Theta \rightarrow$ R is the objective function, Θ is the feasible region, and X_θ is a random variable whose distribution depends on the decision parameter θ, for all $\theta \in \Theta$. Assume that for any given solution $\theta \in \Theta$, the expected value in the above equation cannot be computed exactly, but instead can only be estimated via a "black-box" simulation procedure. This feature, together with the necessity to optimize, makes the above mentioned problem especially difficult (Fu (2002)).

In some other references such as Avadivar (1999), the existence of constraints sometimes changes the form of a model. In practice, many decision makers prefer to deal with their constraints at the risk of violation of a particular constraint rather than being within a certain range for the expected value of the feasible region. In other words, since the variables are stochastic and are reported from the simulation, there are two ways to deal with the constraints: either constraints can be violated within a certain probability or they can be in the specific range with the expected value of the variables. The second approach will result in deleting god solutions while the first one will give a reasonable solution in addition to the deeper perspective of the solution space to the designer. Normally, dealing with constraints is different from objective functions. The constrained simulation-optimization will have the form:

$$\begin{aligned} &\text{Maximize (Minimize) } f(\theta) = E[X_\theta] \\ &\text{Subject to: } \quad P\{Y_\theta < 0\} > 1 - \alpha \end{aligned} \qquad \textbf{Equation 2-7}$$

where $\boldsymbol{P}$ is the vector of probabilities of violation of constraints and α is the vector of risks of these violations that the decision maker is prepared to accept.

2.4.2 Solution approaches for simulation-based optimization problems

As discussed earlier, all the direct search methods, MBDO techniques, and meta-heuristics can be used for simulation-based optimization, since they do not use the knowledge of explicit objective function. One approach is considered better than the other approach if it has less function valuations or simulation procedure calls during the optimization run-time.

Here, I will categorize some recent approaches of simulation-based optimization algorithms. The first group of algorithms developed in early 80's includes algorithms that used to estimate the gradient of the objective function. As reported by Azadivar (1999), two major factors in determining the success of these methods are the reliability and efficiency. Reliability is important because simulation responses are stochastic and a large error in gradient estimation may result in a movement in an entirely wrong direction. The efficiency is a major factor because simulation experiments are expensive and it is desirable to estimate gradients with a minimum number of function evaluations. One of the methods which has been used to estimate the gradient is "Infinitesimal Perturbation Analysis" (Ho et al. (1983)). Perturbation analysis, when applied properly to models that satisfy certain conditions, estimates all gradients of the objective function from a single simulation experiment. The main principle behind perturbation analysis is that if a decision parameter of a system is perturbed by an infinitesimal amount, the sensitivity of the response of the system to that parameter can be estimated by tracing its pattern of propagation through the system. A complete discussion of all issues in IPA has been published in a book by Ho and Cao (1991).

In addition to gradient estimators algorithms, direct search methods such as "complex search" has been used widely in simulation-based optimization domain. Complex search is an extension of Nelder and Mead's (1965) Simplex search that has been modified for constrained problems. As described in Azadivar and Lee (1988), the search starts with evaluation of points in a simplex consisting of $p+1$ vertices in the feasible region. It proceeds by continuously dropping the worst point from among the points in the simplex

and adding a new point determined by the reflection of this point through the centroid of the remaining vertices.

Along with direct search methods, meta-modeling optimization techniques have been widely used in simulation-based optimization problems. From the early work of Biles (1974) in using response surface methodology to the intelligent way of mode pursuing sampling from Wang et al. (2004), meta-modeling has been a promising area of research in simulation-based optimization domain.

Meta-heuristic approaches have been also used in simulation-based optimization problems and specially discrete-event simulation models. Joshi et al. (1996) used Genetic algorithm as a means of optimization for simulation of operation and support activities of launch vehicles during conceptual design in NASA.

Ghiani et al. (2007) used a mixture of MBDO and also simulated annealing, which are connected to the simulation model. Their simulation model is a discrete-event simulation procedure called *Central Server Model* (CSM), in which users get into (and get out of) the system through a central station while processing is made at some peripheral stations. A CSM is suitable for representing phenomena of congestion which appear within flexible manufacturing systems and computer systems.

The following figure shows the relation of discrete event simulation and optimization.

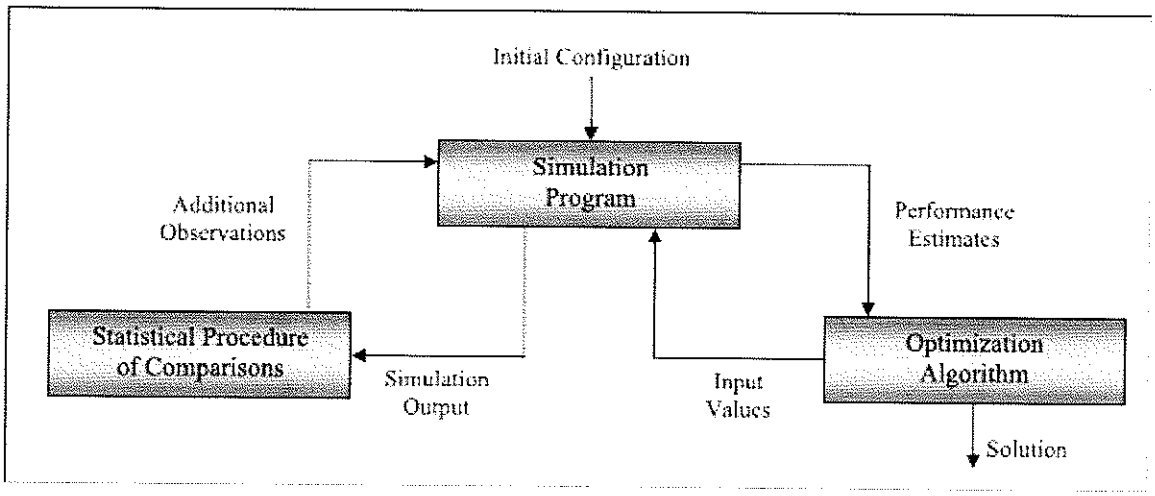


Figure 2-1. Relation of optimization algorithm and statistical procedure for a discrete-event simulation-based optimization from Ghiani et al. (2007)

2.4.3 Discussion on AS-MPS and its comparison to other discrete-event simulation optimization methodologies

As discussed in the introduction of this chapter, AS-MPS is a modified version of Co-MPS which uses a discrete-event simulation as its objective function. The simulation model is the pre-admission clinic of the Health Sciences Center in Winnipeg. With reference to other simulation-based optimization methods described in Section 2.4.2, I will discuss some issues about AS-MPS here:

- AS-MPS is a meta-heuristic approach which uses a global memory of all the patients and their schedule to approximate a neighborhood of any agent's seed or current schedule. This approximation becomes more accurate over time.
- Every agent of AS-MPS has its own structure and follows a heuristic rule of scheduling which has been found in the literature. For example, one agent will schedule patients every 30 minutes. At the same time, new-born agents do not have specific structure and thus will inherit from all the other agents good partial sequences of the whole schedule.

- Since the approximation is used instead of evaluating all the neighbors' objective function, the number of function evaluations will be decreased significantly.
- DSM (Double sphere method) will dynamically guide the search toward the more interesting region of the objective function.

2.5 Summary

The optimization methodology which developed in this thesis is a non-gradient approach to problems in different areas. In this chapter, several types of non-gradient optimization algorithms have been reviewed. First, I discussed the direct search methods which have been introduced as first methods of non-gradient optimization algorithms. The DSM (Double Sphere Method) which has been developed in this thesis as a domain provider for optimization algorithms is a complex version of direct search methods. In the second section of this chapter I reviewed the MBDO algorithms, which are meta-model based optimization algorithms and are one of the non-gradient optimization methods. The D-MPS algorithm developed in Chapter 3 of this thesis is a meta-model based optimization algorithm for discrete domain problems. In Section 3 of this chapter, I reviewed meta-heuristic algorithms in combinatorial optimization domains, since Chapter 4, 5 are about the combinatorial methods which have been developed upon the MPS-based methodology. In the last section of this chapter, I discussed the simulation-based optimization algorithms which are another group of non-gradient optimization methods. Chapter 6 of this thesis will discuss the simulation-based optimization algorithm which has been developed for appointment scheduling in healthcare area and uses a simulation procedure as its objective function.

Chapter 3

Mode Pursuing Sampling Method for Discrete Variable Design Optimization on Expensive Black-Box Functions

3.1 Overview of the proposed algorithm

Based on previously developed Mode Pursuing Sampling (MPS) approach for continuous variables, a variation of MPS for discrete variable global optimization problems on expensive black-box functions is developed in this chapter. The proposed method, namely the discrete variable MPS (D-MPS) method, differs from its continuous variable version (C-MPS) not only on sampling in a discrete space, but moreover, on a novel double-sphere strategy. The double-sphere strategy features two hyper-spheres whose radii are dynamically enlarged or shrunk in control of respectively the degree of "exploration" and "exploitation" in the search of the optimum. Through testing and application to design problems, the proposed D-MPS method demonstrates excellent efficiency and accuracy as compared to the best results in literature on the test problems. The proposed method is believed a promising global optimization strategy for expensive black-box functions with discrete design variables. The double-sphere strategy provides an original search control mechanism and has potential to be used in other search algorithms.

3.2 Introduction

Although most of the classic approaches of design optimization consider continuous variables, many practical design problems deal with discrete or integer variables. For example, the diameters of a pipe, thickness of a structural member or size of a screw are discrete design variables since they may have to be selected from commercially available or standard sizes. Also, many other design variables such as the number of bolts, number of teeth of a gear, or number of coils of a spring must be integers. If there is a discrete variable in an engineering optimization problem, it is often a Mixed Discrete-Continuous Non-linear Programming problem (MDNLP). Khompatraporn et al. (2005) reviewed different software tools that are currently used for solving MDNLP. It has been agreed on that the existence of any non-continues variable considerably increases the difficulty of finding the optimal solution (Lampinen et al. (1999)). This is due to the fact that MDNLP combines the difficulties of both of its subclasses: the combinatorial nature of mixed integer programs (MIP) and the difficulty in solving non-convex (and even convex) nonlinear programs (NLP) with nonlinear (or even linear) constraints (Bussieck et al. (2003)). Huang et al. (1997) categorized the mixed continuous-discrete variable problems into six types depending on the type of the variables, type of the objective function (differentiable, non-differentiable, combinatorial), and whether or not the discrete variables can have non-discrete values during the solution process. This categorization is based on the assumption that the objective function is presented explicitly. In most of the engineering design problems, this is not the case because engineers often deal with computer analysis or simulation processes, such as finite element analysis (FEA) and computational fluid dynamics (CFD) processes. These

computational processes are usually referred as black-box functions, which can be non-convex, non-differentiable, and even discontinuous. Considering that only the input and output for these types of functions are available, characteristics of the function such as the gradient and Hessian matrix can be numerically approximated, but they are usually unreliable Zabinsky (1998). Although in most of the practical engineering problems, a sub-optimal or even feasible solution is satisfactory, the global optimum is always preferred. This work addresses the challenge of discrete variable global optimization with expensive black-box functions such as FEA and CFD processes.

In general, global optimization approaches can be classified into two groups: deterministic and stochastic methods. Horst (1995) gives a review of global optimization methods for MDNLP. Recently, metamodeling optimization techniques have been studied in the literature in support of engineering design optimization. Wang and Shan (2006) gave a review of metamodeling techniques. Some of the deterministic, stochastic and metamodeling methods are as the followings:

3.2.1 Deterministic approaches

When the objective/constraint functions are explicitly expressed and known, exact methods can be applied. These methods include the well-known Branch and Bound (B&B), sequential linear programming (SLP), cutting plane techniques, outer approximation (OA), Lagrange relaxation (duality) approaches, and so on. Another group of methods are called the rounding methods, which use simple dynamic rounding-off techniques (Arora (1989)). Most of these methods such as B&B either use the closely

related non-linear problem (NLP) to reach to the main MDNLP (Gupta (1985)), or rely on the successive solution of the related MIP (mixed integer programming) such as outer approximation (OA) (Fletcher (1994)).

For optimization problems with nonlinear constraints, linearly constrained Lagrangian methods (LCL) solve a sequence of sub-problems that minimize an augmented Lagrangian function subject to linearization of constraints, as described in Branin (1972). These deterministic approaches are based on explicitly expressed objective and constraint functions. Thus, they are not directly applicable to black-box functions.

3.2.2 Stochastic methods

Some of the stochastic methods for global optimizations include the sequential random search, pure adaptive search (PAS) (Reaume (2001)), and various evolutionary programming methods such as simulated annealing (SA) (Hedar (2004)) and genetic algorithms (GA) (Lemonge (2004)). Most of these stochastic methods do not need *a priori* knowledge about the objective function, which makes them good candidates for optimization on black-box functions. These methods, however, usually require a large number of function evaluations, often in the magnitude of ten thousands even for a two-variable problem, which makes these methods impractical in modern design involving computational intensive processes. For example, it is reported that it takes Ford Motor Company about 36-160 hrs to execute one crash simulation (Gu (2001)). Assuming only 50 crash simulations are needed to solve a two-variable optimization problem, the total computation time would be 75 days to 11 months, which is practically unacceptable.

Several sequential random searches such as the pure adaptive search (PAS) have been developed to ease the computational demand (Kiatsupailbul (2001)). In addition, Jones (1993) introduced a Lipschitz global optimization algorithm called DIRECT for black-box functions, whose disadvantage is its slow convergence (Sergeyev (2006)). Different modifications of DIRECT have been proposed by (Sergeyev (2006)) to solve the slow convergence problem and provide modifications for higher dimensions. Another approach for discrete variable optimization on black-box function is to use soft computing-based approaches such as neural networks (NN) as a modeling paradigm because of its universal approximation property. Moreover, combining NN with evolutionary algorithms and genetic quantum algorithms has also been studied in Zhang et al. (2002) and Akbarzadeh (2005).

3.2.3 Metamodel based optimization algorithms

Metamodel based optimization algorithms are developed specifically for expensive black-box functions in order to reduce the number of function evaluations in stochastic and direct optimization. Most of these methods are based on the idea of sampling in the design space, building approximation models from these sampling points, and then performing optimization on the approximation function. I have discussed the meta-modeling approaches in Chapter 2.

This chapter adapts the C-MPS method which has been proposed in Wang et al. (2004a) to the discrete domain. The proposed method is called discrete variable MPS (D-MPS), distinguishing from the original MPS in Wang et al. (2004a) which is referred as

continuous variable MPS (C-MPS). Due to the discrete variable nature, the convergence scheme of C-MPS is no longer applicable. A novel double-sphere strategy is developed to control the convergence of D-MPS. In Section 3.3, MPS in its original form Wang et al. (2004a) and Fu et al. (2002) will be presented. In Section 3.4, the basic D-MPS method will be described. Section 3.5 is devoted to explaining D-MPS with the double-sphere strategy. Parameters of the D-MPS method will also be discussed in Section 3.6. Section 3.7 will show test problems and comparison with results from the literature.

3.3 Review of Mode Pursuing Sampling Method

The MPS method as described in Wang et al. (2004a) is referred as the continuous variable MPS (C-MPS) to differentiate from this work. The C-MPS algorithm integrates the technique of metamodeling and a novel discriminative sampling method. It generates more sample points in the neighborhood of the function mode (optimal) and fewer points in other areas as guided by a special sampling guidance function. Basically, C-MPS consists of four steps. The first step is the initial random sampling in which m points are generated, where m is an arbitrary integer that usually increases as the dimension of the problem increases. These m points are called "expensive points" since their function value should be evaluated by the black-box function. Using the points generated in the first step, a linear spline function is fitted:

$$\hat{f}(x) = \sum_{i=1}^{m} c_i \parallel x - x^{(i)} \parallel \qquad \textbf{Equation 3-1}$$

where $\hat{f}(x)$ is an approximation of the original black-box function $f(x)$ with coefficients c; $x^{(i)}$, $i=1,\ldots,m$ are sample points. Then another function is defined, $g(x) = c_0 - \hat{f}(x)$, as

the sampling guidance function, where c_0 is a constant. In the second step, N (usually large) number of points are randomly sampled and re-ordered according to their guidance function value. These points are called "cheap points" since their function values are evaluated by the linear guidance function, not the black-box function. Their function values will be referred as "approximation values." The third step is discriminative sampling of m points from the "cheap points". The inverse Cumulative Density Function (CDF) sampling method is used for discriminative sampling, in which the CDF is constructed based on the "approximation values". Hence, the new m sample points have the tendency to concentrate around the minimum of $\hat{f}(x)$. For better convergence, a speed-up factor is used in this step to increase the probability of sampling better points. The fourth step involves a quadratic regression in a sub-area around the current best solution. When the approximation in the sub-area is sufficiently accurate, local optimization can be performed in this sub-area to obtain the optimum.

In short, C-MPS is an algorithm which uses discriminative sampling as its engine and has an intelligent mechanism to use the information from past iterations to lead the search toward the global optima. These mechanisms are based on the following approximations:

1. Approximation of the entire function: by fitting the metamodel described in Eq. (3-1) with all the previous "expensive points", C-MPS tries to improve the metamodel (approximation) accuracy.
2. Approximation around the attractive sub-areas: due to the discriminative sampling, more and more points are generated around attractive regions, so the approximation accuracy of theses regions gradually increases.

3.4 Development of Basic Discrete Variable MPS (D-MPS)

In this chapter, it is assumed that all the optimization variables are discrete in nature. This section describes the basic D-MPS without the double-sphere strategy.

3.4.1 Basic D-MPS algorithm

Suppose an n-dimensional black-box function, $f(x)$, has to be minimized over a domain $S[f]$. As in most of the discrete engineering problems, it is assumed that each variable x_i has a pre-defined index set I_i. This can be shown as $S[f]=I_1*I_2*...*I_n$. Without loss of generality and with the assumption that each set I_i has i_j members, $S[f]=\{X_1,X_2,...,X_I\}$ can be written, where $I=\prod_{k=1}^{n} i_k$ and X_i is a vector of length n, representing a point in the solution space.

Assume that $f(x)$ is positive on $S[f]$. In general, if $f(x)$ is negative for some $X \in S[f]$, then a positive number can be always added to $f(x)$, so that it becomes positive on $S[f]$. Note that minimizing $f(x)$ is equivalent to maximizing $-f(x)$. The proposed algorithm consists of three main tasks: generating cheap points, approximation, and discriminative sampling. Specifically,

1. Generating cheap points: $S[f]$ is the domain from which cheap points will be generated. In the first step, a discrete space $S_N[f] \subset S[f]$ is generated consisting of N uniformly distributed base points in $S[f]$, and $S_N[f]=\{X_1,..., X_N\}$, which consists of the "cheap points" in the current iteration. The infeasible points are deleted before proceeding to the next step. In the proposed algorithm, the domain for generating

$S_N[f]$ is not always $S[f]$. The double-sphere strategy, which will be introduced in Section 3.4, controls the domain for generating cheap points.

2. Approximation: The second task is the linear spline interpolation step. At any iteration, the interpolation is done based on all the previous expensive points selected by discriminative sampling except that at the first step m points are randomly generated. All of the expensive points form a set $E[f]$. Then, the linear spline function in Eq. (3-1) is used to fit the points and $\hat{f}(x)$ is derived. Note that $f(x)$ is defined over a discrete set $S[f]$, whereas $\hat{f}(x)$ is continuous. Therefore, there is a need to discretize $\hat{f}(x)$ before continuing to the next step. For all the discrete points in $S_N[f]$, their approximated function values are obtained as $p[k] = \hat{f}(X_k)$ for k=1, 2,...,N.
3. Discriminative sampling: First, all the points in $S_N[f]$ are sorted by their approximated function values $p[k]$. Because of the desire to raise the sampling probability for points of lower function value for the purpose of minimization, $g[k]=c_0-p[k]$ is defined as the sampling guidance function, where c_0 is a constant larger than or equal to the maximum of $p[k]$ to ensure that $g[k]$ is always positive. Then, $g[k]$ for each point is divided by the sum of all $g[k]$ values to generate a value analogous to a probability density. Then, these "probability density" values are added up to generate an analogous Cumulative Density Function (CDF) with respect to the sorted sample points named as $G[k]$. At the end, "m" random numbers inside the interval (0, 1) are generated, and m new samples are selected at which the corresponding CDF values equal to these random numbers along the sorted sample. As a result, more points of lower f value will be selected due to the convex and monotonically increasing shape of CDF. All the new samples are then added to the E[f] with their actual function values. In the next iteration, these new

sample points will be used in addition to all previous expensive points for interpolation. The next section explains the basic D-MPS algorithm with an example.

3.4.2 Sampling example for basic D-MPS

For the ease of understanding, the basic D-MPS (without the double-sphere strategy) is illustrated with the well known six-hump camel back (SC) function, as described in Branin et al. (1972). The mathematical expression of SC is

$$f_{sc}(x) = 4x_1^2 - \frac{21}{10}x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4 \qquad \textbf{Equation 3-2}$$

Although this function in its original form has two continuous variables, I discretize the variables with "0.01" distance between each index as follows: $x_1, x_2 \in \{-2, -1.99, \cdots, 2\}$. A contour plot of the continuous SC function is shown in Figure 3-1, where the H's represent local minima (6 local minima) and H_2 and H_5 indicate two global optima at (-0.09, 0.71) and (0.09, -0.71) respectively with f_{min}=-1.0316. With discretization, $I_1=I_2=\{-2,-1.99,\ldots,2\}$ and the domain set $S[f]=I_1*I_2=\{(-2,-2),(-1.99,-2),\ldots,(2,2)\}$ can be defined. The three main steps of D-MPS algorithm are described below:

1. Generating cheap points: By assuming N=100, $S_N[f]$ is uniformly constructed by selecting N members of $S[f]$. By assuming m=6, X_1, X_2, …, X_6 are then randomly selected from $S_N(f)$. These points will be added to $E(f) = \{(X_1, f(X_1)), \cdots, (X_6, f(X_6))\}$. $E[f]$ is the set for all expensive points.

2. Approximation: In this step, $\hat{f}$ is constructed by fitting Eq. (3-1) to the m sample points.

   By having m sample points, $x^{(i)}$'s are known and constants. Then, by fitting m sample points and their function value to Eq. (3-1), one will have m equations and m

unknowns, which can be solved to find coefficients.(c_i's) . By having all the c_i's, $\hat{f}$ is constructed and can be used as an approximation of the objective function based on the m sample points.

3. Discriminative sampling: By having $\hat{f}$, one can calculate all $p[k] = \hat{f}(X_k)$ for k=1,2,...,N. Now, A discrete function $g[k]=c_0-p[k]$ for k=1,2,...,N is constructed for all of the members of $S_N[f]$. "c_0" can be set to the maximum of $p[k]$ so that $g[k]$ will be positive. Sorting the points in $S_N[f]$ in the descending order of the values of $g[k]$, the sequence of corresponding function values of $g[k]$ is plotted in Figure 3-2(a). Next, the $\frac{g[k]}{\sum_{k=1}^{N} g[k]}$ will be calculated, which result in $\overline{g[k]}$ or "probability" for each cheap point. By using the sorted $S_N[f]$, the $G[k]$ for X_k will be calculated, which is the "cumulative probability density" for each point as shown in Figure 3-2(b). Then m=6 points are drawn with replacement according to the distribution of $G[k]$. All such points form the new sample points X_7, X_8, ..., X_{12}. These new samples will be passed to the black-box function and receive the corresponding function values. Accordingly, the number of function evaluation is increased by m and the set of expensive points will be updated as $E(f) = \{(X_1, f(X_1)), \cdots, (X_{12}, f(X_{12}))\}$. Continuing the process until convergence, the final set of sample points are plotted in Figure 3-6(a), which are compared with the complete D-MPS (with the double-sphere strategy.) The convergence criterion is set by the maximum number of iterations.

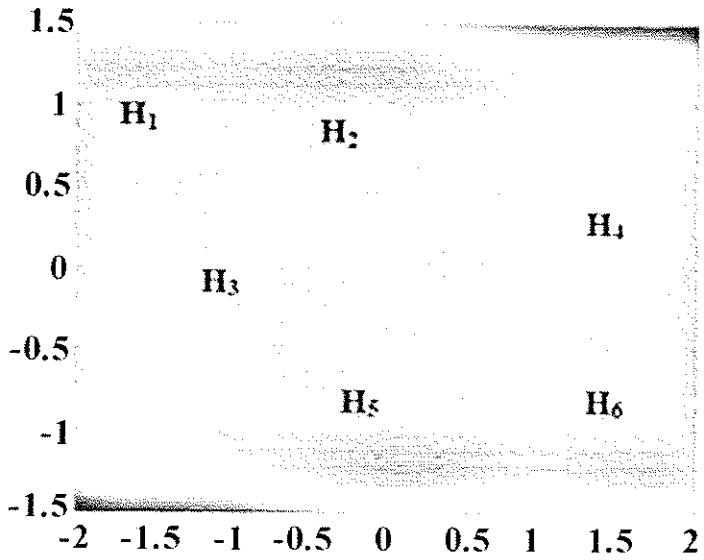
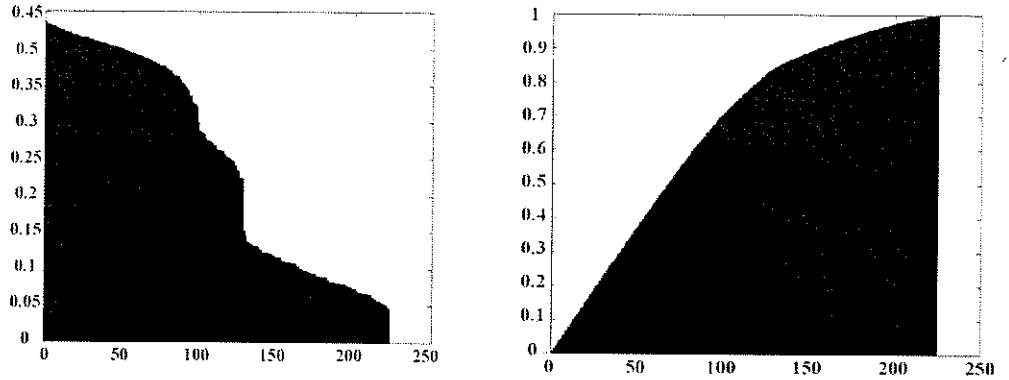


Figure 3-1. Contour plot of the SC function



(a) g[k], for k=1,..,100 **(b) for k=1,..100**

Figure 3-2. Plot of the ranked point distributions of $g[k]$ and $G[k]$

3.5 Development of Complete D-MPS with Double-sphere Strategy

In C-MPS, Wang et al. (2004a) used quadratic fitting with a speed control factor to control the convergence of MPS. For D-MPS, since the optimization region is not continuous, the quadratic fitting approach no longer applies. Moreover, it is more difficult to reach the exact global optimum in a discrete domain than in a continuous domain, for which the

optimum can be closely approached. In this paper, a novel double-sphere strategy has been developed for D-MPS. To distinguish from the basic D-MPS, D-MPS with the double-sphere strategy is called the complete D-MPS.

3.5.1 Double-sphere Concept

Double-sphere consists of two hyper-spheres with the center on the current optimum (the best solution which has been found in the algorithm at the current iteration) and their radii are dynamically changing based on the information from previous iterations. Sampling should be done inside these hyper-spheres. The two hyper-spheres dynamically control the "exploration" and "exploitation" functions of the search algorithm. The descriptions of the two hyper-spheres are as follows.

1. Hyper-sphere S: This hyper-sphere with a small initial radius shrinks to a new radius $R_s*\alpha$ if there is no improvement after n_a iterations, where R_s is the current radius of S and α is a coefficient within [0, 1] for size control. If there is an improvement in the current iteration, i.e., a better solution is found by the algorithm, hyper-sphere S increases in size with a new radius R_s/α.

2. Hyper-sphere B: This hyper-sphere starts with a large initial radius R_b and acts as a dual of hyper-sphere S. It grows to a new radius R_b/α if there is no improvement after n_a iterations and shrinks to $R_b*\alpha$ if there is an improvement in the current iteration. At iteration "i" of the double-sphere strategy, neither R_s nor R_b can be larger than the original design space, $R_{b,0}$.

The two hyper-spheres, B and S, are designed in order to achieve the balance between "exploration" and "exploitation" in the search process. "Exploration" aims to extend the search scope while "exploitation" intensifies the search in a local area thus speeds up the search. There is no definite task assignment of the two hyper-spheres as which controls "exploration" or "exploitation," as one might expect. In an iteration of the search, S could have a bigger radius than B or vice versa, and they may even have equal radii. In the case when the search yields marginal or no improvement, hyper-sphere S shrinks for "exploitation" in the attractive region, while hyper-sphere B enlarges for "exploration" of new promising regions. On the other hand, if the search procedure is inside an attractive region in which consecutive improvements occur, then S tends to grow to avoid being trapped intro a local optimum, i.e., "exploration," and B tends to shrink for "exploitation." The dynamic coupled behavior of the two hyper-spheres is the core of the double-sphere strategy, which will be further discussed in following sections.

3.5.2 D-MPS with Double-sphere Strategy

In the first iteration, the initial radius for B and S is set as $R_{b,0}$ and $R_{s,0}$, respectively, where $R_{b,0} \geq R_{s,0} \geq R_{\min}$. $R_{s,0}$ is one of the parameters of the proposed algorithm, which denotes the initial radius of the smaller hyper-sphere. R_{min} is set to be the smallest radius that encloses m discrete points in $S[f]$, noting for discrete variables there are only a finite number of valid points in a given region. Therefore, when a discrete variable space is given, R_{min} can then be determined. $R_{b,0}$ is set to be the smallest real number that contains all the points inside the feasible region $S[f]$.

Recall that the basic D-MPS is composed of three main steps, i.e., generating cheap points, approximation, and discriminative sampling. In the complete D-MPS, these main steps will be performed on the domains provided by the double-sphere. For the objective function $f(x)$ on domain $S[f]$, the double-sphere strategy dynamically provides a domain $D_1 \cup D_2 \subset S[f]$; D_1 is the domain inside the smaller hyper-sphere; and D_2 is the domain between the smaller hyper-sphere and bigger hyper-sphere. The three main steps of D-MPS are performed on both D_1 and D_2. In other words, generating cheap points, approximation, and discriminative sampling are called two times at any iteration of the algorithm. Therefore, at any iteration, one will have $m/2$ new samples or "expensive points" from D_1, and $m/2$ new samples or "expensive points" from D_2. Although, the number of expensive points from each subset can be also dynamically changed based on the improvement, but having one constant instead of another parameter can help the simplicity of the algorithm in terms of parameter setting for each problem. These two groups of sample points are generated on the basis of two approximations:

1. Approximation inside the smaller hyper-sphere (D_1): This approximation will in general be of higher quality than the one in the original space, since it is performed inside a smaller hyper-sphere. As it was discussed in Section 3.2, this approximation provides rich information about the black-box function in the attractive region.
2. Approximation for the space between the smaller hyper-sphere and bigger hyper-sphere (D_2): This approximation results in an approximation of a solution space inside the bigger sphere excluding the smaller hyper-sphere.

The double-sphere strategy uses a Boolean input b_{iter}=0 or 1 to indicate if the new sample point in the current iteration has a better objective function value than the previously found optimum. Steps of the sampling procedure of the complete D-MPS algorithm are as follows.

Input:

- $S[f]=\{X_1,X_2,...,X_I\}$: Domain or the discrete set over which $f(x)$ is to be minimized
- N : Number of cheap points to be generated in each iteration
- m : Number of expensive points or sample points to be generated in each iteration
- α : The coefficient for increasing/decreasing the radius of hyper-spheres
- n_α : The number of consecutive iterations to activate the double-sphere strategy
- $R_{s,0}$: The initial radius of hyper-sphere S
- $f(x)$: The (expensive) black-box objective function

Output:

- The optimum X^* and $f(X^*)$ from the Set $E[f]$, which has all the expensive points with their function values.
- *nEval:* number of expensive function evaluations.

Step 1. Initialize the five parameters N, m, α, n_α, and $R_{s,0}$; define D_1, D_2; and set Counter=0, where "Counter" counts the number of consecutive iterations at which no improvement is made.

Step 2. Generate a uniform distribution of $N/2$ points from D_1 and D_2 respectively, to form discrete space sets $S_{N,1}[f]$, and $S_{N,2}[f]$. Delete the infeasible points.

Step 3. Perform basic D-MPS in D_1 and D_2 respectively to generate $m/2$ points from $S_{N,1}[f]$ and $S_{N,2}[f]$ respectively. Evaluate the m points.

Step 4. Add the m points and their function values to $E[f]$; $E(f) = \{(X_1, f(X_1)), \cdots, (X_{Iter*m}, f(X_{Iter*m}))\}$, where "Iter" is the number of iterations thus far. Find the lowest function value and update X^*_{Iter} as the best solution. If the stopping criterion is satisfied, i.e., the maximum number of iterations is reached, go to Step 7. Otherwise, if there is an improvement, set $b_{iter}=1$; else $b_{iter}=0$.

Step 5. Double-sphere Strategy:

Step 5.1. If $b_{iter}=1$ go to Step 5.2, else Counter=Counter+1, go to Step 5.3.

Step 5.2:

- Move the center of both hyper spheres to the current optimum
- Increase the radius of smaller hyper-sphere to R_s/α
- Reduce the bigger hyper-sphere to αR_b
- Check if the inequalities $R_{min}<=R_b$ and $R_s<=R_{b,0}$ holds, if not set them to the closest bound (R_{min} or $R_{b,0}$).

Step 5.3. If "Counter"< n_α, go to Step 6.

If "Counter"= n_α

If ($R_s > R_{min}$ or $R_b < R_{b,0}$)

- Reduce the smaller radius to $R_s * \alpha$
- Increase bigger radius to R_b/α ,
- Check for the bounds as in Step 5.2
- Set "Counter"=0

If ($R_s = R_{min}$ and $R_b = R_{b,0}$)

- Set $R_b = R_{min}$
- Set "Counter"=0

Step 6. *iter=iter+1*; define D_1, D_2 based on the new radii (R_s, R_b); generate a uniform distribution of N points from D_1, D_2 to form a discrete space set $S_{N,1}[f], S_{N,2}[f]$. Delete the infeasible points. Go to step 3.

Step 7. Report the X^*_{Iter}, and its corresponding variable as the global minimum. Report the number of function evaluations *nEval=Iter*m*.

Following figure shows the flowchart of the proposed method.

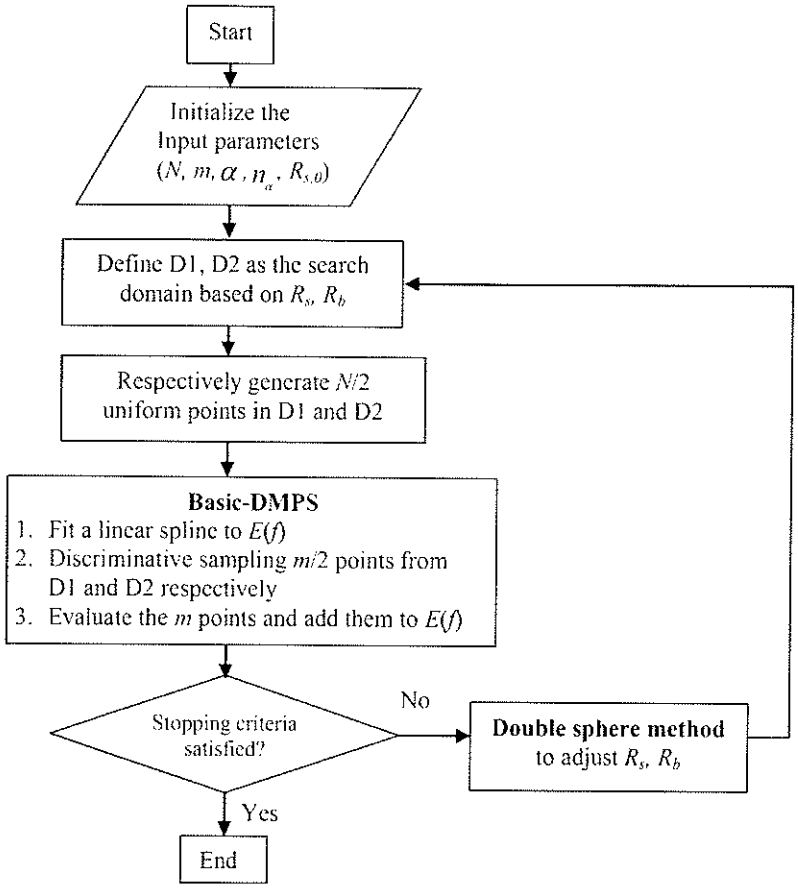


Figure 3-3. Flowchart of the proposed DMPS algorithm

During an optimization process, if there are consecutive improvements, the double-sphere method may lead to a situation at which $R_b=R_{min}$ and $R_s=R_{b,0}$. This situation usually happens at the beginning of the search when it is desirable to exploit regions that lead to improvements. On the other hand, if there are no consecutive improvements, R_s may reach its minimum R_{min} and R_b may reach its maximum $R_{b,0}$. This situation is called a stable state. The stable state happens usually at last iterations of the D-MPS algorithm. Figure 3-4 shows the variation of both radii versus the iterations for the gear train test problem (to be discussed in Section 3.5). The stars on the horizontal line indicate the iterations when an improvement is observed.

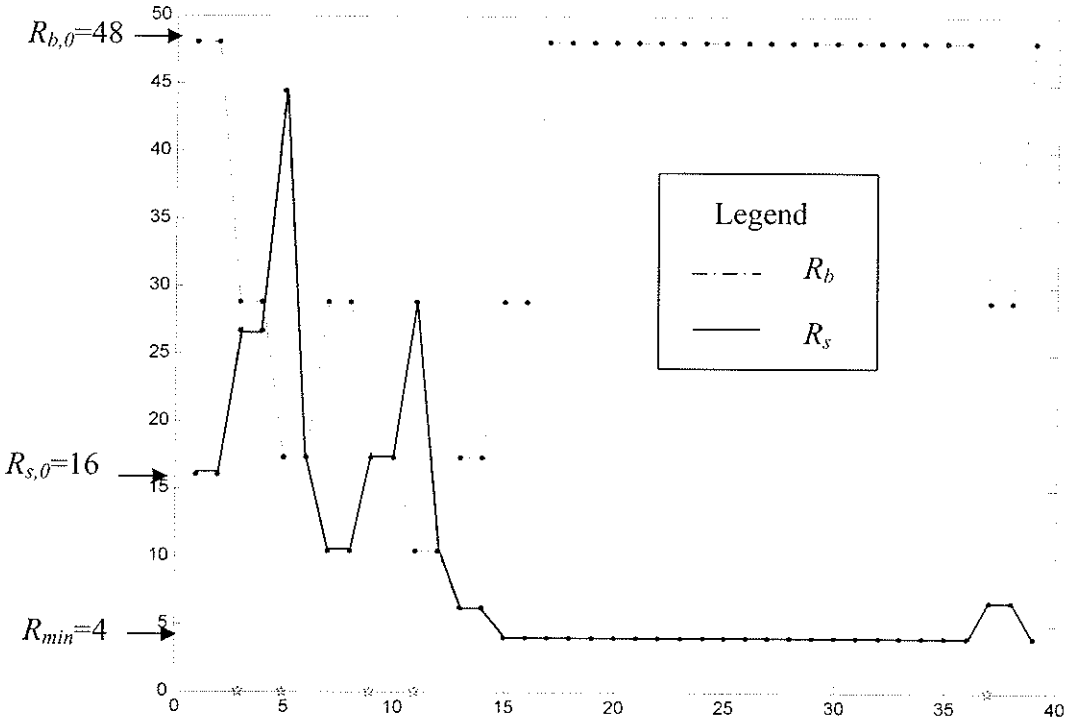


Figure 3-4. Hyper-sphere radii over iterations

From Figure 3-4 one can see that the search reaches the stable condition at which $R_b=R_{b,0}$ and $R_s=R_{min}$ at Iteration 15 and continues to Iteration 37, largely due to the fact that

further shrinking S or enlarging B is not possible in these iterations. Therefore in Step 5.3 of the complete D-MPS algorithm, a "bouncing-back" feature is designed. That is, when the stable condition is reached (R_b=$R_{b,0}$ and R_s=R_{min} and Counter=n_a), hyper-sphere B is forced to be of the same radius of R_{min}. As it continues with no improvement, hyper-sphere B will enlarge gradually to $R_{b,0}$ (while S takes R_{min}) so that the entire space is searched again in a layer-by-layer manner for potential improvements.

3.5.3 Example for the Complete D-MPS

Consider the SC function defined in Eq. (3-2) in Section 3.3.2. The parameters of D-MPS are as follows:

- The number of expensive points at each iteration, m=4.
- The number of cheap points generated at each iteration, N=32.
- The radius of the small hyper-sphere, $R_{s,0}$=0.5.
- The coefficient for increasing/decreasing the radius, α =0.7.
- The number of steps for the double-sphere strategy, n_a =2.

The solution space for SC function is $[-2, 2]^2$; the center for both hyper-spheres S and B is (0, 0), and $R_{b,0} = \sqrt{8}$. Since both variables are discretized to an interval of size 0.01, R_{min}=$0.01 * \sqrt{2}/2$. The initial state is shown in Figure 3-5(a), where the circles show the twin global optima and the cross shows the center for hyper-spheres B and S. Hyper-sphere S is shown with the dotted line and hyper-sphere B is shown with the solid line.

The D-MPS algorithm runs for 30 iterations. Hyper-spheres at Iterations 2, 4, 8, and 20 are shown in Figure 3-5(b)-(f) respectively.

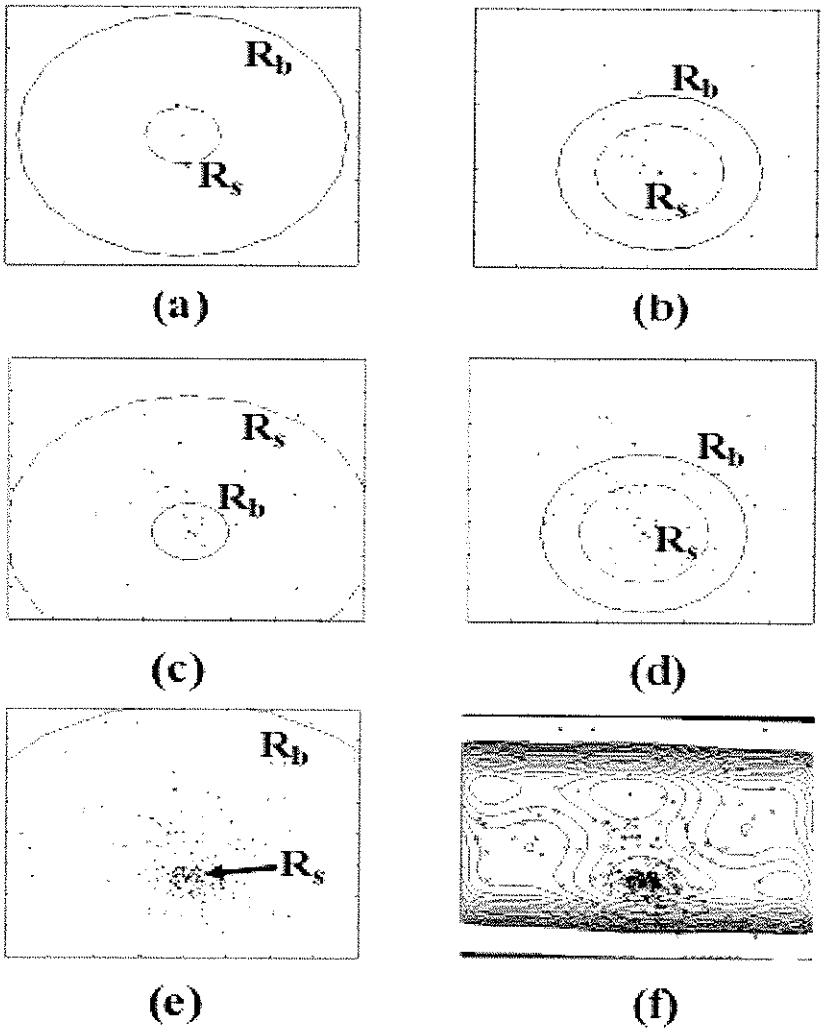


Figure 3-5. Example for D-MPS with the double-sphere strategy for SC function

In the second iteration of D-MPS as shown in Figure 3-5(b), a new best solution improves the current optima, and the center of both hyper-spheres moves to the new optimum. In addition, the radius of hyper-sphere S is increased to avoid being trapped to a local optimum while B is decreased for potential exploitation. In the next iterations, the algorithm continues to find better solutions and hyper-sphere B becomes smaller than S, as it can be noted from Figure 3-5(c). From the fourth iteration to the eighth iteration, no improvement is made. Therefore, hyper-sphere S starts to shrink for exploitation around the region of current optima while B starts to grow to explore other regions in the search

space. As a result S becomes smaller than B, as shown in Figure 3-5(d). At the iteration 20, as it is shown in Figure 3-5(e), most of the sample or expensive points are around the mode of the function, while in the meantime the entire solution space is explored. Figure 3-5(f) shows the contour plot of the SC function with all the sample points generated by the D-MPS algorithm.

The basic D-MPS algorithm (without the double-sphere strategy) is compared with the complete D-MPS (with the double-sphere strategy) for the SC problem. Figure 3-6 shows the expensive points with small circles. In both cases, 150 expensive points have been generated. Figure 3-6(b) generated by the complete D-MPS shows a concentration of points around one of the global optima, while no obvious pattern is observed in Figure 3-6(a) for the basic D-MPS.

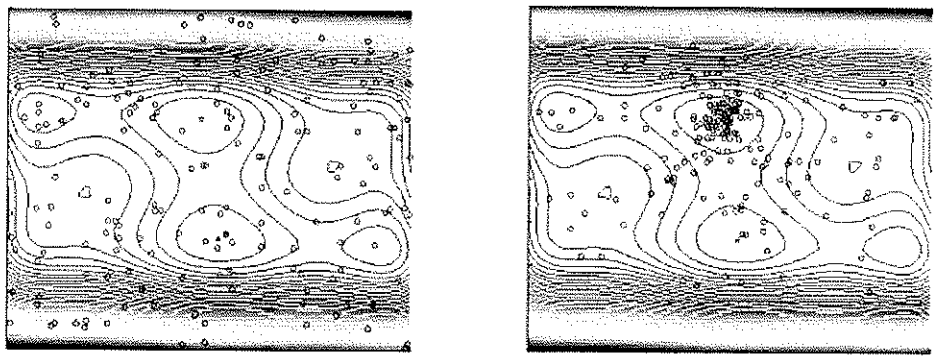


Figure 3-6 Result comparison between (a) the basic D-MPS (b) and complete D-MPS

3.5.4 ANOVA Study of D-MPS Parameters

As it was discussed in Section 3.4.1, there are five parameters that have to be set for the proposed method. In this section, the result of Analysis of Variance (ANOVA) study of

the parameters on the SC function will be discussed. As it can be seen from Table 3-1, each parameter has three different levels, and ten runs are executed for each setting. For a full factorial design, 2430 runs are executed in total. Table 3-1 shows the different level values for the variables. For a two-variable problem, N is taken as the square of the number of indices for each variable, N_d.

Table 3-1. Variables for ANNOVA study

Factor	Values
N	16^2, 32^2, 64^2
m	4, 8, 10
n_α	1, 4, 8
α	0.5, 0.75, 0.95
R_s	0.2, 0.7, 1.4

The total number of function evaluations needed to reach the known global optimum is the output factor of concern. The average function value for every ten runs is used for the ANOVA study. The ANOVA results are listed in Table 3-2 and the parameter interactions are plotted in Figure 3-7. The first column of Table 3-2 shows the source of variability with some interaction terms omitted due to negligible effects. The second shows the mean squares due to each source; the third is the F statistics, and the fourth is the p value for the F statistics. If p is sufficiently small (<0.01), the associated factor has a non-trivial effect on the result.

It can be seen from Table 3-2 that N is not sensitive with respect to the number of function evaluations (nEval). The interaction effect between N and other parameters is also negligible (see Figure 3-7). It thus suggests that N can be set to a low level to decrease the computation cost. Parameter α is the most sensitive variable. When α is

high (i.e., little changes in radii), the interactions between α and other parameters, especially R_s and n_α, have high influence on the number of function evaluations. This is understood that when there is little change in radii of the hyper-spheres, the double-sphere strategy does not function actively and therefore D-MPS performs as a basic D-MPS, for which R_s and n_α play a more important role in algorithm control. Therefore, parameter α is recommended at a low setting, e.g., 0.5. If α is too low, however, the algorithm will be too aggressive and may converge prematurely. From Figure 3-7, R_s, n_α, and m can also be chosen at a low level to reduce the total number of function evaluations. One is cautioned that when parallel computation is possible and the goal is to reduce the total computing time at the expense of more function evaluations, these parameters can be set at higher levels. Although the parameter study is based on the SC function, it helps one understand the influence of the parameters and their interactions, and provides some guidelines for application as in the test cases in Section 3.5.

Source	Mean Square	F	*p*
N	1681	0.34	0.716
m	561406	112.81	0.000
n_α	1461420	293.66	0.000
α	2694001	541.34	0.000
R_s	1280365	257.28	0.000
$m*n_\alpha$	40663	8.17	0.000
$m*\alpha$	79712	16.02	0.000
$m*R_s$	71000	14.27	0.000
$n_\alpha*\alpha$	241748	48.58	0.000
$n_\alpha*R_s$	165153	33.19	0.000
$\alpha*R_s$	571855	114.91	0.000
$m*\alpha*R_s$	39489	7.94	0.000
$n_\alpha*\alpha*R_s$	67093	13.48	0.000

Table 3-1. ANOVA results for all parameters

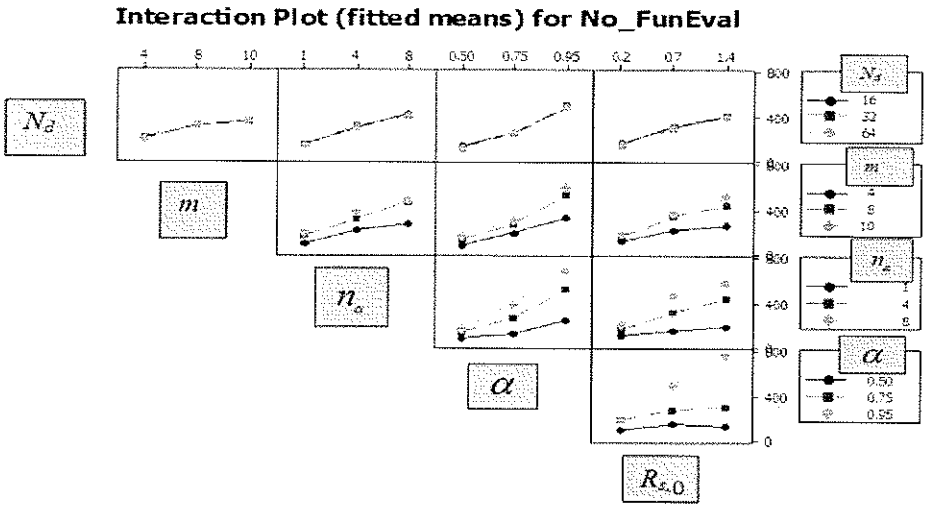


Figure 3-7. Parameter interaction plot

Since N is not a sensitive parameter, it is better to set it as low as possible. Also, depending on the problem and the search strategy, setting α near 0.5 makes the search strategy aggressive while setting it to higher value near 1 makes the search strategy a non-aggressive one. On the other hand, R_s and α has high interaction to each other, which indicates that for aggressive setting it is better not to make R_s relatively small. n_α should be set in the way that double-sphere gets to the steady state approximately after maximum number of iterations.

Further research can focus on the more systematic and generalized way for setting the parameters depending on the optimization problem.

3.6 Test of D-MPS

Three test problems have been solved using the proposed D-MPS algorithm. These problems are: 1) Discrete Six-hump camel back (SC) function (Branin et al. (1972)), 2)

Gear train problem (Huang et al. (1997)), and 3) Pressure vessel problem (Cao et al. (1997)). In the following, these problems will be discussed and D-MPS will be compared with other methods from the literature on the results. As described in Wang et al. (2004b) and Droste et al. (2001) for black-box functions, the number of function evaluations is a more appropriate indicator of computation efficiency than CPU time. Therefore this work also uses the number of function evaluations for efficiency comparison.

3.6.1 SC problem

The SC function is defined in Eq. (3-2) and discussed in Section 3.3.1. The variables are assumed to be discretized as follows: $I_1 = I_2 = [-2,-1.99,..., 1.99,2]$. Parameters are set to: $(N, m, R_s, \alpha, n_a) = (256, 4, 0.2, 0.5, 1)$. Ten runs have been performed. Results of D-MPS are given in Table 3-3, where *'nIter'* stands for the number of iterations and *'nEval'* stands for the number of function evaluations. The known analytical optimum for the discretized problem is (-0.09, 0.71) and (0.09, -0.71) with the function value f*=-1.0316. From Table 3-3, D-MPS consistently find a close-to-optimum solution with a modest number of function evaluations.

3.6.2 Gear train (GT) problem

The objective of the gear train problem is to optimize the gear ratio for the compound gear train as shown in Figure 3-8. The gear ratio for a reduction gear train is defined as the ratio of the angular velocity of the output shaft to that of the input shaft.

Table 3-3. Results of D-MPS on SC

Run No.	nIter	nEval	X1*	X2*	Y*
1	16	64	0.1	-0.72	-1.0309
2	16	64	0.11	-0.72	-1.0298
3	8	32	0.09	-0.7	-1.0303
4	12	48	-0.07	0.7	-1.0291
5	16	64	0.07	-0.7	-1.0291
6	22	88	-0.08	0.7	-1.0301
7	15	60	-0.07	0.7	-1.0291
8	19	76	0.1	-0.71	-1.0311
9	15	60	0.1	-0.7	-1.0298
10	11	44	0.08	-0.71	-1.0312
Average	**15**	**60**	**-**	**-**	**-1.0300**

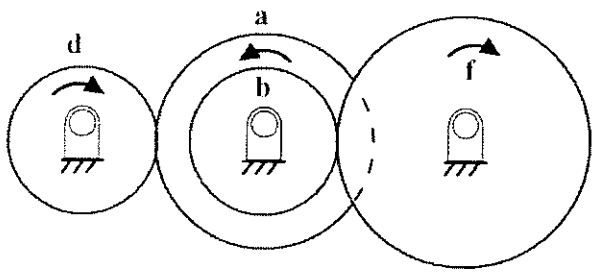


Figure 3-8. Gear train problem

In order to produce the desired gear ratio, the compound gear train is constructed out of two pairs of the gearwheels, "d–a" and "b–f". The gear ratio i_{tot} between the input and output shafts can be expressed as:

$$i_{tot} = \frac{w_o}{w_i} = \frac{z_d z_b}{z_a z_f} \qquad \textbf{Equation 3-3}$$

where w_o, w_i are the angular velocities of the output and input shafts, respectively, and z denotes the number of teeth on each gearwheel. It is desirable to produce a gear ratio as close as possible to 1/6.931. For each gear, the number of teeth must be from 12 to 60. The design variables are denoted by a vector $X = [x_1, x_2, x_3, x_4] = [z_d, z_b, z_a, z_f]$. Hence

x_1, x_2, x_3 and x_4 are the numbers of teeth of gears d, b, a, and f, respectively, which must be integers. The optimization problem is expressed below with a known optimum of zero:

$$\min f = \left(\frac{1}{6.931} - \frac{x_1 x_2}{x_3 x_4}\right)^2 \qquad \textbf{Equation 3-4}$$

$$s.t. \quad 12 \le x_i \le 60 \; i = 1,\ldots,4$$

For the GT problem ten runs have been executed with the following parameter setting, m=*4*, N=1250, α=0.7, n_a=7, and R_s=0.33 (all variables are normalized to [0, 1]). The following table compares the best solution of D-MPS with those from gradient-based approaches and different evolutionary algorithms.

Table 3-4. Results of D-MPS on GT

Run No.	nIter	nEval	Function Value	X^*
1	500	2000	2.3078e-011	15 26 53 51
2	215	860	2.7009e-012	19 16 49 43
3	21	84	2.7009e-012	19 16 49 43
4	500	2000	1.8274e-008	14 16 42 37
5	500	2000	9.9216e-010	24 13 46 47
6	500	2000	1.0936e-009	17 15 57 31
7	290	1160	2.7009e-012	16 19 43 49
8	500	2000	1.3616e-009	17 14 55 30
9	500	2000	1.1661e-010	17 22 54 48
10	500	2000	2.3576e-009	15 18 39 48
Average/ Median	Ave:402	Ave:1610	Median: 5.5439e-010	

Table 3-5. Performance comparison between D-MPS and other algorithms on GT

Items	MIBBSQP [27]	IDCNLP [28]	SA [29]	MVEP [25]	MIHDE [27]	D-MPS
x_1	18	14	30	30	19	16
x_2	22	29	15	15	16	19
x_3	45	47	52	52	43	43
x_4	60	59	60	60	49	49
$f(X)$	5.7*10^-6	4.5*10^-6	2.36*10^-9	2.36*10^-9	2.7*10^-12	2.7*10^-12
Gear ratio	0.146666	0.146411	0.14423	0.14423	0.14428	0.14428
Error(%)	1.65%	1.47%	0.033%	0.033%	0.00047%	0.00047%

The methods that have been compared with D-MPS in Table 3-5 are as follows: MIBBSQP (Mixed Integer Branch and Bound using the Sequential Quadratic Programming algorithm) (Lin et al.(1999)), IDCNLPC (Integer-discrete-Continuous Nonlinear Programming algorithm) (Fu et al. (1991)), SA (Simulated Annealing) (Zhang et al.(1993)), MVEP (Mixed-variable Evolutionary Programming)(Cao et al.(1997)), MIHDE (Mixed-integer Hybrid Evolutionary Algorithm) (Lin et al.(1999)). All the results from references are based on 50,000 function evaluations, as compared to 1,610 needed by D-MPS.

Table 3-6. Performance comparison between D-MPS and Ant algorithms

	AS	ACS	MMAS	RBAS	BWAS	D-MPS
Best	2.3578e-09	2.7071e-12	2.4597e-07	2.7071e-12	2.7267e-08	2.7009e-12
Average	6.4482e-04	5.5613e-04	1.0700e-1	4.5709e-03	3.2102e-02	5.5439e-010
Worst	1.9196e-02	1.6139e-02	1.4877e+00	6.8187e-02	2.5088e-01	1.8274e-08
n.Eval	10000	10000	10000	10000	10000	1610

Authors in Capriles et al.(2005) tested different ant algorithms for the gear train problem. Their results are compared to the D-MPS in Table 3-6. Their results are based on an average of 30 runs with 10,000 function evaluations per run. The methods which have been compared to D-MPS in Table 3-6 include: AS (Ant System), ACS (Ant Colony System), MMAS (Max-Min Ant System), RBAS (Rank Based Ant System), and BWAS (Best-Worst Ant System) (Capriles et al.(2005)). As can be seen from Tables 3-5 and 3-6, D-MPS leads to better solutions than most of the methods, while the number of function evaluations is much lower.

3.6.3 Design of a pressure vessel

The pressure vessel problem is to design a compressed air storage tank with a working pressure of 3000 psi and a minimum volume of 750 ft^3. The schematic of a pressure

vessel is shown in Figure 3-9. The cylindrical pressure vessel is capped at both ends by hemispherical heads. Using rolled steel plate, the shell is to be made in two halves that are joined by two longitudinal welds to form a cylinder. Each head is forged and then welded to the shell. Let the design variables be denoted by the vector $X=[x_1,x_2,x_3,x_4]$. x_1 is the spherical head thickness, x_2 is the shell thickness, x_3 and x_4 are the radius and length of the shell, respectively.

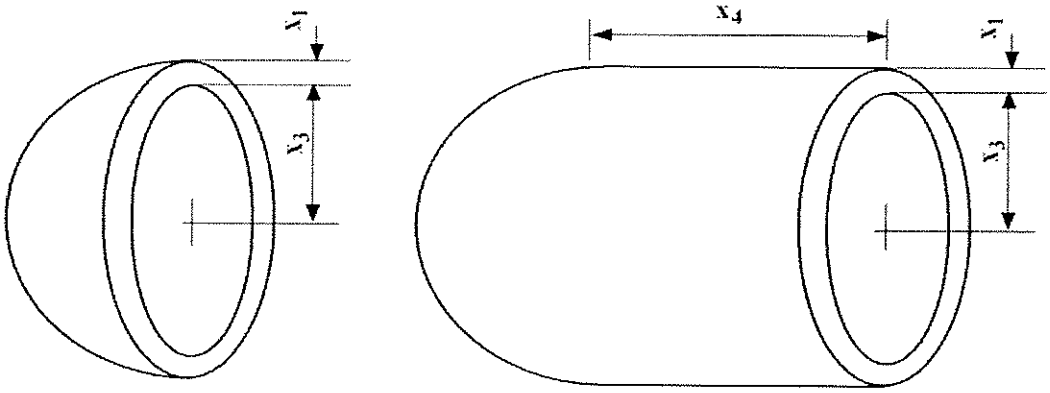


Figure 3-9. Pressure vessel

The objective function is to reduce the combined costs of materials, forming and welding of the pressure vessel. The constraints are set in accordance to ASME codes. The mathematical model of the problem is:

$$\min f(X) = 0.6224x_1x_3x_4 + 1.7781x_2x_3^2 + 3.1661x_1^2x_4 + 19.84x_1^2x_3$$

$$s.t.\ \ g_1(X) = 0.0193x_3 - x_1 \le 0 \qquad \text{Equation 3-5}$$

$$g_2(X) = 0.00954x_3 - x_2 \le 0$$

$$g_3(X) = 750\times 1728 - \pi x_3^2 x_4 - \frac{4}{3}\pi x_3^3 \le 0$$

$$g_4(X) = x_4 - 240 \le 0$$

where the design variables x_3 and x_4, are continuous and x_1, x_2 are integer multiplies of 0.0625. I consider continuous variables as discrete ones with 0.1 increment. All the variables are in the following ranges: $x_1 \in [1.0, 1.375]$, $x_2 \in [0.625, 1.0]$, $x_3 \in [25, 150]$,

and $x_4 \in [25, 240]$. Table 3-7 shows the result of D-MPS on pressure vessel for 10 runs. For this test, different parameters are set for each run so that the performance of D-MPS can be observed for a wide scope of parameter settings ($n_\alpha \in [1,3], R_S \in [0.6, 0.7], \alpha \in [0.5, 0.6]$). The best results from different methods are compared with that of D-MPS for this problem, as listed in Table 3-8. As it can be noted, D-MPS leads to better quality results when compared with other methods. The number of function evaluations required by D-MPS on the problem is 400, in contrast to 50,000 in other methods shown in Table 3-8.

Table 3-7. Results of D-MPS for the pressure vessel problem

Run No.	nIter	nEval	Function Value	X^*
1	100	400	7197.6932	1, 0.625, 49.5, 102.8
2	100	400	7249.9739	1.0625,0.625,55,66.5
3	100	400	7140.2116	1,0.625,50.9,93.3
4	100	400	7351.9418	1.0625,0.625,52.4,81.8
5	100	400	7279.9775	1.0625,0.625,54.8,68.2
6	100	400	7178.2013	1, 0.625, 51.3, 92.2
7	100	400	7241.1644	1.0625,0.625,53.7, 72.5
8	100	400	7372.8138	1.0625,0.625,54,74.4
9	100	400	7155.836	1,0.625,51,93.2
10	100	400	7072.9156	1,0.625,51.3,89.2
Average/Median	-	-	**7223.421**	

Table 3-8. Comparison of D-MPS and other methods for pressure vessel problem

Items	MIBBSQP [27]	IDCNLP [28]	SA [29]	MVEP [25]	D-MPS
x_1	1.125	1.125	1.125	1.000	1.000
x_2	0.625	0.625	0.625	0.625	0.625
x_3	47.7008	48.38070	58.290	51.1958	51.3
x_4	117.701	111.7449	43.693	90.7821	89.2
$f(X)$	8129.8	8048.619	7197.7	7108.6160	7072.9156

3.7 Discussion on Generalized Double-Sphere Method

In the current DMPS algorithm for discrete variables, without loss of generality, I normalize the discrete indices into the domain [0, 1], as for many practical problems, different variables have different set of indices. Double-sphere method in its original

form can handle all different indices unless there is a significant difference in the number of indices. For example, for handling binary variables as well as continues variable in a mixed variable problem, binary variables have only two indices, while a continuous variable can be represented as a discrete variable with infinite number of indices. In this situation, a d-dimensional hyper sphere which is symmetrical in regard to all the d variables can not be a good measure for providing a domain for optimization algorithm.

In its original form for a discrete d-dimensional function "f" with support $S[f] \subset I_1 * I_2 * \ldots I_d$, where I_j is the index set for variable x_j, at iteration "i" of the double-sphere, hyper sphere B will provide the domain $S_1(i) = \{x \mid x \in S[f], \sum_{j=1}^{d} (X(j) - X_c(j))^2 \leq R^2_{b,(i-1)}\}$ which X_c represents the current optima(d-dimensional vector).An extension for the mixed variables situation would be : $i^* = \max\{i_1, i_2, \ldots, i_d\}$, where i_j represents the cardinal number of the set I_j.

$$S_2(i) = \{x \mid x \in S[f], \sum_{j=1}^{d} \frac{i_j}{i^*} (X(j) - X_c(j))^2 \leq R^2_{b,(i-1)}\}$$ **Equation 3-6**

If there exist a continuous variable, I consider i^* as solution precision of the continuous variable. In other words, instead of having a hyper-sphere, one will have a hyper-ellipse. In fact in a generalized double-sphere method, one dimension of the sphere can shrink faster than the other because of the difference in number of indices.

3.8 Summary

In this chapter, an algorithm for optimization on expensive black-box functions with discrete variables is proposed. The proposed approach can be considered as an extension

to the Mode Pursuing Sampling (MPS) method for continuous variables (Wang et al.(2004a)). Moreover, this work proposes a novel double-sphere method that dynamically controls the two often contradictory "exploration" and "exploitation" behaviors of an optimization process. The double-sphere method is found effective and efficient in improving the proposed discrete MPS (D-MPS) algorithm. In a more general sense, the double-sphere method provides an original concept for any direct optimization method. Compared to the well-known Trust Region method, the double-sphere method provides more control and flexibility over a search space. It is believed that the double-sphere concept can be applied to other search processes, or design visualization applications.

Numerical results for three test cases show that D-MPS yields high quality results with a modest number of function evaluations. Because of the double-sphere strategy and approximation only in controlled domains, D-MPS has potential for high-dimensional problems, although this needs to be further studied. It is found in practice that D-MPS often finds a high quality region that contains global optima in a few steps. This is due to the coupling effect of the MPS' discriminative sampling strategy and the double-sphere method.

Chapter 4

Combinatorial Mode Pursuing Sampling Method for Single Machine Total Weighted Tardiness Problem

4.1 Overview of the algorithm

A novel meta-heuristic approach for simulation-based combinatorial problems has been developed based on the Discrete Mode Pursuing Sampling method (D-MPS). D-MPS was successfully applied to problems with discrete variables in design optimization. The proposed method employs a number of agents inside the solution space and each of them is composed of double-spheres that dynamically control the convergence behaviour. This novel technique features two controllers whose radii are dynamically enlarged or shrunk in order to control the degree of "exploration" and "exploitation. In order to create a new-born agent at the beginning of any iteration, all agents communicate with a global memory and also communicate with each other. A local memory of each agent is used to select a new solution, and consequently the global memory is updated based on the achieved improvement at any iteration. The proposed Co-MPS algorithm is a simulation-based optimization algorithm which addresses optimization in complex systems. In fact, reducing the number of function evaluations from the simulation model or black-box function by using the past information from the optimization algorithm in addition to finding the global optima in a relatively short time is the main objective of the proposed Co-MPS. Experimental results of benchmark Single Machine Weighted Total Tardiness instances are promising compared to the reported results in the literature.

4.2 Introduction

Basically, scheduling is concerned with the allocation of limited resources to tasks over time. The investigations on various scheduling problems have been of constant interest to researchers. In the modern day industry, having products produce at a faster rate with no compromise in the quality makes the scheduling more important than it used to be. Scheduling problems have been studied such as the job shop scheduling problem, open shop scheduling problem, flow shop scheduling problem, and single machine scheduling problem. This chapter deals with the total weighted tardiness of the single machine scheduling problem.

First introduced by McNaghton (1959), single machine total weighted tardiness (SMTWT) problem is basically a permutation problem like the well-known Traveling Salesman Problem (TSP), assignment problem, or Knapsack problem. This important similarity is not true for other scheduling problems such as the job-shop or flow shop problem, since they are on multiple machines and more importantly the permutation schedule is not always the optimal one. In other words, pre-empting a job or inserting idle time on a machine may improve the result on more than one machine. McNaughton (1959) proved that in the SMTWT problem, an optimal solution exists in which no job is split; In other words, pre-emption is not allowed. Therefore, he concluded that only permutation schedules of the n jobs must be considered.

It should be noted that the single machine scheduling with minimizing weighted tardiness is not the only type of SMTWT. Different variations of single machine scheduling have

been discussed such as the random processing times or sequence dependent job problems (Hodgson (1977)), Xiaochuan (2006)). In the scheduling literature, SMTWT can be represented as $n/1/\sum w_i T_i$, Which means having "n" jobs on "1" machine with the objective of minimizing the total weighted tardiness, where w_i are the weighting factors for each product's tardiness values T_i..

Having the total weighted tardiness as a measure of comparing several sequences for a single machine is of great interest in the practice since it can be directly related to the cost and time of having delayed products. Having weights for each product is also directly related to differences in products, their demands, and prices. In this chapter, I will describe the single machine problem with the objective of minimizing the total weighted tardiness. The SMTWT problem can be represented as finding a permutation σ, where σ = *([1], [2], . . . , [n])* is a sequence of the jobs, and [i] is the i-th job in the sequence.

Given a sequence σ, let $C_{[i]} = \sum_{j=1}^{i} p_{[j]}$, where p_j denotes the processing time for each operation. Now, let *T[i]=max{0, C[i]-d[i]}*, where *C[i]* and *d[i]* is the completion time of job "i" and the due date of job "i", respectively. The objective is to find a sequence σ that minimizes $Z(\sigma) = \sum_{i=1}^{n} w_i T_i$.

Lenstra et al. (1980) proved that single machine total weighted tardiness problems are strongly NP hard. Lawler et al. (1993) and Coffman (1991) also observed that a complete enumeration to get the exact solution leads to computational requirements that grow as an

exponential or high-degree polynomial in the problem size n. Several solution methods have been proposed for solving the SMTWT problem. Heuristics, meta-heuristics, and several types of branch and bounds are examples of solution approaches to the SMTWT problem. A literature review on SMTWT will be given in Section 4.3.

In this chapter, a novel meta-heuristic algorithm has been proposed. This algorithm is noted as Combinatorial Mode Pursuing Sampling (Co-MPS). The Co-MPS algorithm has a certain characteristic which differentiates it from other meta-heuristic approaches in the literature. Although having features such as the global Memory and local memory for the search procedure, the structure of these memories, the way they interact with each other and update themselves are different from those of Ant algorithms, Tabu search, etc... Moreover, a new-domain controller has been proposed, which has control over generating neighborhood solutions based on the previous results from the search algorithm. The results of the Co-MPS algorithm show high quality solutions compared to the state-of-the-art archived in the literature.

This chapter is organized as follows. In Section 4.3, the literature review on SMTWT (Single Machine Total Weighted Tardiness) will be discussed. In Section 4.4, I will describe the Co-MPS algorithm and its characteristics. Results of the Co-MPS algorithm for the SMTWT problem on benchmark problems will be presented in Section 4.5.

4.3 Literature review on SMTWT problem

As it was described in the Section 4.1, the SMTWT problem is to find a processing order of the jobs that minimizes the sum of total weighted tardiness over all jobs. Solution methods for SMTWT problems can be categorized as the following: implicit enumeration methods, heuristics, and meta-heuristics. These three categories of solution methodologies will be discussed in the following.

Although most classical approaches were only in the first category and they can not compete with the new meta-heuristic approaches for large number of jobs in terms of computation time, but they could guarantee an optimal solution. Most of the classical approaches focused on branch and bound techniques. Works of Shwimer (1972), Abdul-rezaq et al. (1990) and Babu et al. (2004) are some examples of these approaches. Although any branch and bound technique guarantees approaching the optimal solution, minimizing the resource used in a solution process depends on two main issues: branching procedure, and method of defining the lower bound. Shwimer (1972) algorithm required very large computation time because weak lower bounds were used. Babu et al. (2004) integrated a lower bound based on Lagrangian decomposition with their branch-and-bound algorithm. As reported by Bilge et al. (2003), most of the branch and bound techniques can not solve more than a 50-job problem in reasonable time. Parallel to branch and bound techniques, dynamic programming is another type of classical solution methods which has been used in the literature. Same as branch and bound, dynamic programming guarantees the optimal solution while the solution time grows exponentially with the increase of the number of jobs. Baker and Scharge (1978)

developed a dynamic programming algorithm which outperformed other methods at that time with the number of jobs up to 30.

The heuristic approaches are designed with the least number of steps to find near optimal solutions in reasonably short time. In this sense, dispatching rules can be considered as minor heuristics. Some of the dispatching rules, that have been reported in the literature, include the Earliest due date (EDD), Weighted shortest processing time (WSPT), Modified Cost OVER time (MCOVERT), and Apparent Urgency (In some references such as Bilge et al. (2007), Apparent Urgency rule is called "R&M").

It should be noted that if there is no more than one tardy job, EDD is optimal. Hence EDD can be used when one is dealing with lightly loaded machines. In comparison to EDD, WSPT gives the optimal sequence when all jobs are necessarily tardy. Therefore, WSPT can be used when one is dealing with heavily loaded machines. Moreover, MCOVERT and A&M are more complex dispatching rules which show better results compared to other dispatching rules. In addition to the simplicity and giving good solutions in a timely manner, these dispatching rules can also be used as a good initial solution generator for any heuristic or meta-heuristic approach. On the other hand, they mainly have poor worst-case performances. In this chapter, some of these dispatching rules will be used to generate an initial solution.

Meta-heuristics can be considered as a compromise between implicit enumeration methods and heuristics. Although they won't guarantee the optimal solution, they reach

near-optimal solutions in a timely manner. In recent years, researchers have paid more attention to meta-heuristic methods such as the simulated annealing (SA), genetic algorithms (GA), Taboo search (TS), and Ant Colony Optimization (ACO). Crauwels et al. (1998) compared heuristics methods with SA and GA, and concluded that the TS algorithm dominated other methods. Merkle and Middendorf (2003), Besten et al. (2000), and Tasgetiren et al. (2006) used ACO for the SMTWT problem. Laguna et al. (1991) considered a single machine scheduling problem for minimizing the sum of setup costs and linear delay penalties, and proposed a TS algorithm that used hybrid neighborhood consisting of both swap and insertion moves. Besten et al. (2000) presented an iterated local search (ILS) algorithm to solve the SMTWT problem, and their algorithm reaches all optimum solutions of well-known OR-lib benchmarks. Congeram et al. (2002) proposed an iterated "dynasearch" algorithm for the SMTWT problem. The authors developed an approach which allows a series of moves to be performed at each generation while local search makes a single move only. Their algorithm outperformed the best TS algorithm that was reported in Crauwels et al. (1998). Maheswaran and Ponnambalam (2005) proposed an intensive search evolutionary algorithm for the SMTWT problem. Tasgetrin et al. (2006) used the differential evolution (DE) algorithm and TS for the SMTWT problem. They showed that DE is faster than the Particle Swarm Optimization (PSO). Although, they concluded that combining both meta-heuristics with special local search (VNS) can significantly improve the results. In another study by Bilge et al. (2007), a TS approach to SMTWT has been proposed. Their algorithm systematically varies the tenure to overcome some difficulties presented by the topology

of the search space. In addition, they made use of various candidate list strategies. Their result showed very high quality solutions for benchmarks in the OR-lib.

4.4 Description of Co-MPS Algorithm

4.4.1 Background of Co-MPS

Co-MPS is basically an extension of the D-MPS algorithm proposed by Sharif et al. (2007). D-MPS is an optimization algorithm for discrete variables in mechanical design problems with expensive black-box functions. This algorithm generates some uniform random points and evaluates their objective values. These points are called "expensive points" since the usually computationally intensive black-box function should be called for evaluating them. Next, a linear spline function is fitted to current expensive points to get an approximation of the original objective function. In the next step, discriminative sampling is performed on a large amount of points generated from the spline function inside the feasible region. These points are called "cheap points". As a result, a few selected points from the discriminative sampling are then evaluated by the black-box function and then added to "expensive points" for next generations' approximation. The approximation of the expensive black-box function evolves over the optimization run-time which helps to sample better points around the mode of the black-box function. A concept called Double Sphere Method (DSM) was built into the algorithm to dynamically control the search neighborhood. DSM includes two hyper-spheres (S and B) that control the neighborhood of the search based on the improvement during the optimization procedure. Both hyper-spheres act as a dual to each other. Hyper-sphere S becomes bigger in radius if there are consecutive improvements while Hyper-sphere B becomes

smaller. On the other hand, hyper-sphere S tends to become smaller if there are consecutive non-improvements while hyper sphere B tends to become bigger. The dynamic behavior of the hyper-spheres controls the degree of the exploration and exploitation.

In summary, D-MPS is basically an intelligent search strategy which gains the following knowledge during the search process:

1.Approximation (or memory) of the region of the current optima (optimum found in the previous iteration).

2.Rough approximation (or memory) of the entire objective function.

3.Having control over the neighborhood within which new generation will be found.

In the D-MPS optimization algorithm, these three intelligent processes are used along with a discriminative sampling mechanism that is comparable to the selection in evolutionary algorithms. D-MPS-based algorithms have the advantage of solving global optimization problems involving expensive objective functions or black-box functions.

In this context, having combinatorial optimization based on the D-MPS idea is not rational, since most of the combinatorial problem has a cheap objective function, but a large solution space and many constraints. Taking that into consideration in MPS-based algorithms for design optimization (expensive function, mixed variable), the approximation around the current optima becomes exact as the optimization process goes on. In the combinatorial-MPS, the same thing should happen.

In D-MPS, the points having a lower approximation value have a higher chance to be selected as new samples, but in the combinatorial version, the operation that improves the current solution has more chance to be chosen. In the following sections, the overall algorithm structure will be briefly discussed. Then, all components of the Co-MPS algorithm will be elaborated.

4.4.2 Co-MPS algorithm

Co-MPS algorithm is composed of "K" agents and a global memory which sends the appropriate values to the agents during the run time. At any iteration, every agent performs certain tasks.

In the sequel, the steps of each agent and its relationship with the global memory will be explained, followed by detailed description of each step in the later sections:

1. Generating neighbourhood solutions: At the beginning of any iteration, every agent starts to generate the neighbourhood solutions around the current agent/seed. There are two controllers, S and B, dynamically control the size of two neighbourhoods around each agent. Depending on the circumstances, one neighbourhood can be for "exploration" and the other one for "exploitation." Accordingly, the two neighbourhood solutions are stored in two matrices LS and LB, respectively for every agent as its local memory. Generating neighbourhood solutions will be discussed in more details in Section 4.4.3.

2. Connecting to global memory for approximating the neighbourhood: In the second step, each agent connects to a global memory, which consists of G_1 and G_2. Each element

of G_1 is composed of (s_{ij},b_{ij}), which shows the degree of freedom which the new neighbourhood can be generated in *LS* and *LB* respectively. (s_{ij},b_{ij}) shows the scope of new neighbourhood solutions if job "*i*" at position "*j*" is selected as neighbourhood generator. (s_{ij}, b_{ij}) will be updated based on the behaviour of the controller *S* and *B* respectively with regards to the information form the past iteration of the optimization algorithm. After generating neighbourhoods, every element is assigned a value from elements of G_2. I denote the elements of G_2 by O_{ij}'s, which show how good is the job "*i*" being at position "*j*", with regards to previous iterations' information. Assigned values to every element of *LS* and *LB* will be saved in *VS* and *VB*, which are components of local memory of every agent. The structure of global memory and how the agents connect to it for approximating their current neighbourhood solutions will be discussed in Section 4.4.5.

3. The discriminative sampling: Discriminative sampling is performed for each agent according to *VS* and *VB*. Discriminative sampling or selection will be discussed in Section 4.4.6. After selecting "m_s" new solutions for each agent from *LS* and "m_b" new solutions from *LB*, the objective function is called to find the value of these new solutions. The best solution will replace the seed of an agent if it improves the current local optima.

4. Updating global memory:

   4.1. The global memory G_1 will be updated based on the improvement reported from each agent. G_1 is composed of s_{ij} and b_{ij} for all jobs and positions. If there is an

improvement reported with the new found seed by any agent, the value of b_{ij} for all the corresponding jobs "i" in position "j" in the seed will be decreased. "s_{ij}" shows the scope of generating next neighbourhoods if job "i" at position "j" is selected as the neighbourhood generator. I denote b_{ij} from the behaviour of the controller B. Controller B in fact makes the distance of generating neighbourhoods smaller and smaller in the case of having improvements, which helps us have more local control over solutions with high potential of turning into local optima. On the other hand, if there are n_α consecutive no-improvement is reported from any agents, s_{ij} for all job i's at position j's will be decreased . I denote s_{ij} from the behaviour of controller S. Controller S in fact makes the distance of generating neighbourhoods smaller and smaller in the case of having no-improvement for a while, which helps to have local search around the neighbourhood solutions that have high potential of being local optima. It should be noted that controllers B and S will act as a dual to each other, preventing all the agents from trapping into a local optima. In other words, whenever controller B starts to decrease b_{ij}'s , controller S will increase the corresponding s_{ij}'s.

4.2. The global memory G_2 will be updated based on the value of the best new solutions or the new seeds of agents. For every new seed, the value of jobs in their position will be updated by a certain value. This value is the job's tardiness divided by the summation of the entire job's tardiness in the current seed of the agent, then multiplied by another relative measure which is the objective function of this agent divided by the summation of all the objective function of elite agents. Since, the

objective function is the summation of all the jobs tardiness, the updating value is a relative measure of the job's contribution to the total tardiness. Updating global memory will be discussed in Section 4.4.7.

5. Until a certain finishing criteria is met, replace the new found seed for all the agents and go back to step 1 to start generating new neighborhood solutions around the new seed. Figure 4-1 shows the main structure of an agent of the Co-MPS algorithm. All agents follow the same procedure as shown in Figure 4-1.

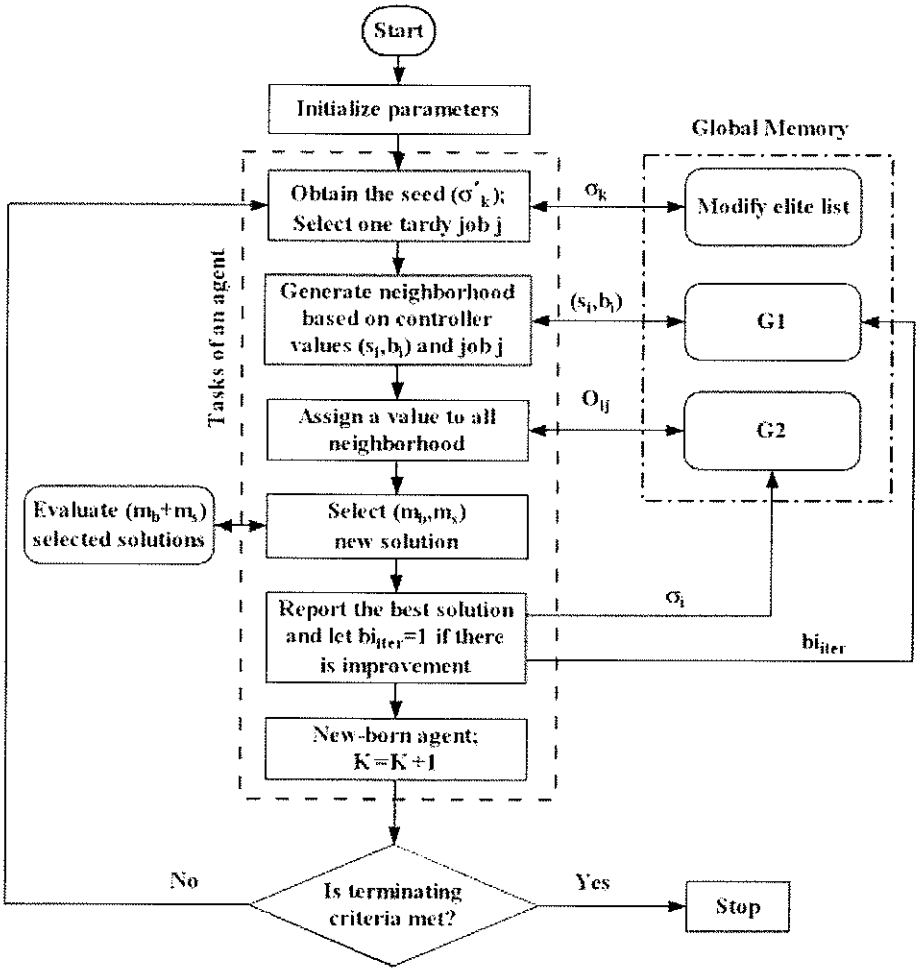


Figure 4-1. Flowchart of the algorithm for one agent

The functions of each agent, global memory, and the interactions between agents will be explained with an example in the following sections.

4.4.3 Definition of Agents in Co-MPS

Co-MPS is basically composed of "*K*" agents inside the solution space. Each agent includes the following:

1. Seed (σ_u^*): Which is a job sequence that is used to generate neighborhood solutions.
2. Local memory: Which is composed of two matrices having neighborhoods of the seed with regard to the controller S, B: $LS^{A_u}_{iter,\sigma_u^*}, LB^{A_u}_{iter,\sigma_u^*}$. Where A_u denotes the *u*-th agent with $1 \le u \le K$, "*iter*" denotes the current iteration number and σ_u^* denotes the seed or the local optima of the *u*-th agent. The second part of the local memory includes two matrices for assigning weights to the neighborhood solutions: $VS^{A_u}_{iter,\sigma_u^*}$, $VB^{A_u}_{iter,\sigma_u^*}$. The method of constructing these matrices with regards to the global memory will be discussed in Section 4.4.5.

Initially, each agent starts from one initial random seed and then uses this seed to generate the neighbourhood solutions around it based on "insert" and "exchange" moves. It is worth mentioning that these initial solutions can be also generated using a dispatching rule or heuristic. The process of generating new neighbourhood solutions is restricted by two dynamic controllers S and B that will be discussed in Section 4.4.4.

The dimensions of *LS*, *LB* matrices are $2*m_c*n$, where the value two denotes the two methods of generating neighbourhoods, "insert" and "exchange" moves.

4.4.4 Neighborhood generation mechanism inside Local Memory

As it has been discussed in section 3.3, the local memory of each agent is composed of four matrices: $LS^{A_u}_{iter,\sigma_u}$, $VS^{A_u}_{iter,\sigma_u}$, $LB^{A_u}_{iter,\sigma_u}$, and $VS^{A_u}_{iter,\sigma_u}$. *"S"* and *"B"* are the controllers that define the domain of the new neighborhood solutions. At the moment, it is assumed that they provide (s_{ij}, b_{ij}) values at the beginning of each iteration. Where (s_{ij}, b_{ij}) represents the distance of insert and exchange moves in *LS* and *LB*, respectively. In fact, G_1 has "$n*n$" elements which all of them are composed of the tupelos (s_{ij}, b_{ij}). . *"S"* and *"B"* will be discussed in Section 4.4.5, since they are part of the global memory.

As it has been discussed in Section 4.4.3, the first set of rows of *LS* and *LB* is constructed based on "insert" moves while the second set of rows of *LS* and *LB* is constructed by "exchange" moves. Now, suppose one has a 5-job SMTWT problem. Table 4-1 shows the data of 5 jobs that have to be scheduled.

Table 4-1. Data of the illustrative example

Jobs' parameters	Job number				
	1	**2**	**3**	**4**	**5**
Processing time	8	4	1	7	3
Due date	10	18	8	19	15
Weights	4	5	3	1	2

In the following, the method of constructing *LS* and *LB* by each agent will be explained with the above example.

At any iteration, all the tardy jobs of the current seed (σ_u^*) is identified. Where σ is a sequence with σ=*{(i, r(i)) : r(i) is the rank of job i}*. Next, for the first row of *LS* and *LB*, one of these tardy jobs is being randomly chosen $(j^*, r(j^*)) \in \sigma_u^*$.

- Suppose agent 1 has the seed: $\sigma_1^* = [4,3,1,2,5]$, which shows the order of processing jobs, i.e., first job 4 is processed, followed by 3, etc. Therefore, the tardy jobs will be identified as a completion vector of σ_1^* : [16, 20, 8, 7, 23] for all 5 jobs, which shows that job 1 is finished at time 16, job 2 is finished at time 20, etc. So, the tardiness vector will be: [6, 2, 0, 0, 8].

Hence, f(σ_1^*)=6*4+2*5+2*8=50. Correspondingly, jobs {1, 2, 5} are tardy jobs. Next, one tardy job will be randomly chosen. Let j^*={2}. The randomly chosen tardy job will be inserted within the S^*=*rand([1, $s_{j^*,r(j^*)}$])* distance before j^*. Where *"rand[1, s]"* means a random integer between 1 and *s*. The value of $s_{j^*,r(j^*)}$ is being reported from the global memory (G_1). S^* is randomly selected from *[1, $s_{j^*,r(j^*)}$]* without replacement. If the number of new neighborhood points is less than "*m*", replacement will be allowed. If "*w*" denotes the row of *LS* and *LB*, for $1 \le u \le K$, which shows the number of agents, one will have:

$$LS^{A_u}_{iter,\sigma_u^*}(w,t) = \{\sigma_u^{w,t}, \quad 1 \le t \le m_c, 1 \le w \le 2\} \qquad \textbf{Equation 4-1}$$

$$LB^{A_u}_{iter,\sigma_u^*}(w,t) = \{\sigma'^{w,t}_u \quad 1 \le t \le m_c, 1 \le w \le 2\} \qquad \textbf{Equation 4-2}$$

Let *m*=3. Since the selected job j^*={2}, and job 2 is at the fourth position in σ_1^* , agent 1 will ask the global memory to send the values of (s_{24},b_{24}) from G_1. Now, assume

(s_{24},b_{24})=(2,3). As I will describe later, these two numbers control the neighborhood generation. Now, since $s_{24}=2$, for constructing the first row of *LS*, one is restricted to insert job 2, one or two places from its current position. Therefore, the only neighborhood which can be generated in the first row of *LS* would be: (4, 2, 3, 1, 5) and (4, 3, 2, 1, 5). Since, there are two possible neighbors and $m_c=3$, one of them will be repeated in the first row of *LS*. In the second row of *LS* which is constructed by exchange moves, one is restricted to exchange job 2 with job 1 and job 3, because one can move job 2 with a maximum of 2 places before its position. As it can be seen from elements of *LS*, two of the elements of each row is the same, which is because of the fact that controller $S(s_{24}=2)$ prevents the agent to generate more diversified neighborhood, and the number of possible neighbors are less than "m_c".

$$LS = \begin{bmatrix} 4,2,3,1,5 & 4,3,2,1,5 & 4,3,2,1,5 \\ 4,2,1,3,5 & 4,3,2,1,5 & 4,2,1,3,5 \end{bmatrix}$$

The same task will be done for constructing *LB*, with restriction $b_{24}=3$. As it can be seen in the following matrix, *LB* is more diversified compared to *LS*.

$$LB = \begin{bmatrix} 4,3,2,1,5 & 4,2,3,1,5 & 2,4,3,1,5 \\ 4,3,2,1,5 & 4,2,1,3,5 & 2,3,1,4,5 \end{bmatrix}$$

As discussed in the example, the first rows of both matrices are from the "insert" move. So, in Eq. (4-1), $w=1$, and therefore for any agent "u" and for any element of the first row one will have:

$$\sigma_k^{1,j} = (i, r(i)) = \begin{cases} (j^*, r(j^*) - s^*) & for\ i = j^* \\ (i, r(i)+1) & for\ all\ r(j^*) - s^* \le r(i) \le r(j^*) \\ (i, \sigma_k^*(i)) & for\ rest\ of\ i's \end{cases} \quad \textbf{Equation4-3}$$

This equation means job *"j*"* at position *r(j*)* will be inserted *"s*"* places before its current position*(r(j*)-s*)*. Therefore all the jobs *"i"* which are in the position between the inserted job and its previous position *(r(j*))* will be shifted one place ahead *(r(i)+1)*. Jobs which are after the previous position of job *"j*"*, and jobs which are before the inserted position of job *"j*"* will remain in their positions.

For the second row of *LS*, random tardy job $(j^*, r(j^*)) \in \sigma_u^*$ is exchanged with the job of s^*=rand([1, $s_{(j^*,r(j^*))}$] distance before j*. Again, $s_{j^*,r(j^*)}$ is reported from the global memory. For exchange moves, *w=2*:

$$\sigma_u^{2,j} = (i, r(i)) = \begin{cases} (j^*, r(j^*) - s^*) & for\ i = j^* \\ (i, r(i)+s^*) & for\ i, where\ r(i) = r(j^*) \\ (i, \sigma_u^*(i)) & for\ rest\ of\ i's \end{cases} \quad \textbf{Equation 4-4}$$

The same structure is correct by replacing *LB* with *LS* and b^*=*rand([1,* $b_{j^*,r(j^*)}$*]* with s^*=*rand([1,*$s_{j^*,r(j^*)}$*]* in the Equations 4-3 and 4-4. It should be noted that from the equations, it can be deducted the fact that if s^* or b^* becomes small, the new neighborhood solutions will be close to the seed and close to each other, which makes the search more toward exploiting the region around σ_u^*. Having a new neighborhood close to the seed can be interpreted differently based on the way of the move. For example, for insertion moves in *LS* , one will have:

$$\forall \sigma \in LS^{A_u}_{iter,\sigma^*_u}(1,t) : \left|\sigma - \sigma^*_u\right| \leq s_{j^*,r(j^*)} \qquad \textbf{Equation 4-5}$$

If $s_{(j^*,r(j^*))} \to 1$, one will have neighbourhood solution all close to the seed in terms of definition of $\left|\sigma_1 - \sigma_2\right| = d$ where d denotes the number of jobs which are in different positions in two sequences.

4.4.5 Structure of global memory

At any iteration, all agents communicate with the global memory for the following reasons:

1. Generating neighborhood solutions or constructing *LS*, *LB*, agents communicate with the global memory for receiving $s(j^*,r(j^*))$ and $b(j^*,r(j^*)$ from the controllers.
2. Assigning a weight to their new generated neighborhood solutions.
3. Updating the global matrices based on the local improvements.

Global memory of Co-MPS consists of two matrices, G_1 and G_2, that are explained below.

- Domain controller matrix (G_1): where $G_1 = \{(s_{ij}, b_{ij}), i = 1,...,n,\ j = 1,...,n\}$, (s_{ij},b_{ij}) shows the domain of neighborhood for each job at each place. These values are the controllers which set restriction for each agent when generating their neighborhoods. s_{ij} restricts *LS*, and b_{ij} restricts *LB*. New neighbors are generated based on these restrictions. s_{ij} is defined by the behavior of the controller S and b_{ij} is defined by the behavior of the controller *B*

"*S*" and "*B*" are domain controllers which are inside the global memory. "*S*" is a controller that shrinks the domain of neighborhood generation if there are no-

improvements in consecutive steps reported from each agent. At the same time, it grows if there is any improvement reported by any agent. On the other hand, "*B*" is the controller which shrinks if there are consecutive improvements reported from every agent. In fact, when relatively flat areas of the objective function are encountered, "*S*" controls the exploitation while "*B*" controls the exploration. On the other hand, when climbing the hill, "*S*" controls the exploration while "*B*" controls the exploitation. The updating procedure of G_1 will be discussed in Section 4.4.7

- Weight matrix (G_2): At any iteration, the elements of the local memory of each agent are assigned an approximation value. This approximation value is calculated by the values inside the first part of the global memory. As it is shown in Figure 4-2, the global memory consists of weights for each job and each place. The method of updating G_2 will be explained in Section 4.4.7. O_{ij} denotes the weight of job "*i*" at position "*j*" in the sequence. In other words, O_{ij} will keep the information about the frequency and goodness of job "*i*" at position "*j*" in the past iteration of the algorithm.

$$\begin{bmatrix} O_{11} & O_{12} & O_{13} \\ O_{21} & O_{22} & O_{23} \\ O_{31} & O_{32} & O_{33} \end{bmatrix}$$

Figure 4-2. Weights inside the global memory

In the initial step all O_{ij}'s are equal to one. After first iteration, when each agent constructs the new neighbors of its initial seed, it will use the summation of all the O_{ij} related to its neighbors. For example, suppose if agent 1 has the seed 1-2-3-4 and generates 3-2-1-4 as a result of the exchange move. Then, it will add the values of

$O_{31}+O_{22}+O_{13}+O_{44}$ to get an approximated value of the objective function for the seed *(1-2-3-4)*. This value is the approximation of objective function since it has been gained through the previous information of optimization procedure. Similar to (s_{ij},b_{ij}) in G_1, G_2 has the following structure:

$$G_2 = \{O_{ij},\ i = 1,...,n,\ j = 1,...,n\} \qquad \textbf{Equation 4-6}$$

Constructing the weights for neighborhoods is being done in *VS*, and *VB*. They follow the following equations:

$$VB^{A_u}_{iter,\sigma^t_u}(w,t) = \sum_{i=1}^{n} O_{i,\sigma^t_{(w,t)}} \qquad \textbf{Equation 4-7}$$

$$VS^{A_u}_{iter,\sigma^t_u}(w,t) = \sum_{i=1}^{n} O_{i,\sigma^t_{(w,t)}} \qquad \textbf{Equation 4-8}$$

In Equations 4-7 and 4-8, $1 \le t \le m$ (the number of neighborhood solutions), $1 \le u \le K$ (number of agents) and $1 \le w \le 2$ (two rows for insertion and exchange moves).

For the example described in Section 4.4.4, elements of *VS* and *VB* are calculated from Equation 4-7 and 4-8 as shown below.

- A value has to be assigned to every solution in *LS* and *LB* based on the global memory G_2. Assume it is at iteration 10, the global memory G_2 is as the following:

$$G_2 = \begin{bmatrix} 5.2 & 7.8 & 6.1 & 1.2 & 3.2 \\ 10.3 & 2.1 & 1.5 & 8.9 & 6.1 \\ 2.3 & 1.7 & 10.2 & 5.1 & 3.4 \\ 9.1 & 4.5 & 3.1 & 3.2 & 9.5 \\ 1.6 & 0.7 & 5.4 & 7.2 & 9.5 \end{bmatrix}$$

As described in Section 4.4.4, *LS* and *LB* are:

As described in Section 4.4.4, *LS* and *LB* are:

$$LS = \begin{bmatrix} 4,2,3,1,5 & 4,3,2,1,5 & 4,3,2,1,5 \\ 4,2,1,3,5 & 4,3,2,1,5 & 4,2,1,3,5 \end{bmatrix}$$

$$LB = \begin{bmatrix} 4,3,2,1,5 & 4,2,3,1,5 & 2,4,3,1,5 \\ 4,3,2,1,5 & 4,2,1,3,5 & 2,3,1,4,5 \end{bmatrix}$$

For assigning a weight to the first element of *LS*, *LS*(1,1)=(4, 2, 3, 1, 5), the following has to be done:

$VS(1,1)=G_2(4,1)+G_2(2,2)+G_2(3,3)+G_2(1,4)+G_2(5,5)= 9.1+2.1+10.2+1.2+7.2=29.8$

All other elements of *LS* and *LB* will be given weights in the same manner as described above:

$$VS = \begin{bmatrix} 29.8 & 23.4 & 23.4 \\ 31.9 & 23.4 & 31.9 \end{bmatrix}$$

$$VB = \begin{bmatrix} 36.5 & 32.1 & 35.7 \\ 36.5 & 31.9 & 30.8 \end{bmatrix}$$

Having all the values for all the neighbors (*VB*,*VS*), each agent starts to perform a selection based on these values. Selection or discriminative sampling will be discussed in Section 4.4.6.

4.4.6 Discriminative sampling inside each agent

As it has been discussed in Section 4.4.5, after any iteration, agents construct the *VS* and *VB* matrix, which denote the weights of all new neighbors' solutions of their current seed. *VS* and *VB* are $2*m_c$ matrices for each agent at any iteration which consist of $E_{(w,t)}$. In the next step, a probability $p_{(w,t)}$ is assigned to each element of the *VS* and *VB* matrix. This is

done by assigning a probability to each element which is $pr_i = \frac{E_{(w,t)}}{\sum_{b=1}^{2}\sum_{i=1}^{m_s} E_{(w,t)}}$. The cumulative density function (CDF) will be constructed over pr_i. Then, the inverse sampling method is performed by generating a random number in [0, 1] and then finding its related element or (w, t). "m_s" elements are being selected in this manner for *VS* and "m_b" for *VB*.

In the next step, all these "m_s+m_b" solutions are being evaluated by the objective function, and the best solution is selected. If the new selected solution is better than the previous seed, it will replace the current seed for agent $u(\sigma_u^*)$, and the improvement is reported to the global memory for updating G_I.

For the example described in Section 4.4.4 with

$$VS = \begin{bmatrix} 29.8 & 23.4 & 23.4 \\ 31.9 & 23.4 & 31.9 \end{bmatrix}$$

$$VB = \begin{bmatrix} 36.5 & 32.1 & 35.7 \\ 36.5 & 31.9 & 30.8 \end{bmatrix}$$

In the next step, a "*pdf*" function is built for all the elements of *VS* and *VB*. Having the summation of all the elements in *VS*=108.1:

$$\text{PDF}(VS) = Pdf(VS) = [\frac{29.8}{163.8}, \frac{23.4}{163.8}, \frac{23.4}{163.8}, \frac{31.9}{163.8}, \frac{23.4}{163.8}, \frac{31.9}{163.8}].$$

Now, a CDF function is constructed by sorting the elements of PDF(*VS*) and adding the elements together:

CDF(*VS*)=[0.14,0.28,0.42,0.6,0.79,1], with vector(2, 3, 5, 1, 4, 6) that shows how they are sorted. Next, a random number within (0,1) is generated and the closest value from CDF(*VS*) is matched with that value. For example, if random number is 0.73, the fifth element of CDF(*VS*) is selected. Then, its corresponding rank, 4, is specified. Respectively, *LS* (4)=[4,2,1,3,5] is selected, which has the corresponding rank in *VS*.

It should be noted that the above procedure will be repeated "m_s" times for *LS* (*VS*) and "m_b" times for *LB* (*VB*). Then, all the selected solutions will be evaluated by the objective function.

4.4.7 Updating the global memory

As discussed in Section 4.4.5, global memory consists of G_1 and G_2. After, generating neighborhood around their current seed in LS and LB, assigning weights to these neighbors based on previous information about jobs and their position in *VS* and *VB* and performing a selection for new neighborhood, "m_s" new selected neighborhood from *LS* and "m_b" new neighborhood from LB will be evaluated by the objective function. Based on the new objective function values and the fact that these new neighborhood solutions improve the past optima of the agent, G_1 and G_2 are being updated as follows:

1. Updating G_1:

As it has been described in Section 4.4.4, for generating neighborhood around the current seed of any agent, one random tardy job is chosen and inserted or exchanged with its previous jobs based on $s^*=rand[1,s_{j^*,r(j^*)}]$. Where j^* is the chosen tardy job, and s^* is the distance for insertion or exchange). If inserting j^* in its new position($r(j^*)$-s^*) improves the local optima of the agent, $s_{(j^*,r(j^*)-s^*)}$ will be increased by α, where α is a parameter of

Co-MPS. Moreover, all the value of "s" for all positions around j^* should be increased by α plus their distance from j^*. In fact, controller S has an effect on all of the s values around j^*. On the other hand, all the jobs in the sequence should affect the values of s in G_1.

The following formula show the detailed change of S and B when job j^* is moved from its original position $r_1(j^*)$ to $r_2(j^*)=r_1(j^*)-s^*$ and the improvement is being reported on this move such that all other jobs change their positions from $r_1(j)$ to $r_2(j)$ based on Equations 4-3 and 4-4.

$$s_{j,i}^{new} = \begin{cases} s_{j,r_2(j)}^{old} + \alpha & for\ i = r_2(j) \\ s_{j,r_2(j)}^{old} + (\alpha) + \left|(i - (r_2(j))\right| & for\ rest\ of\ i's \end{cases} \quad \textbf{Equation 4-9}$$

$$b_{j,i}^{new} = \begin{cases} b_{j,r_2(j)}^{old} - \alpha & for\ i = r_2(j) \\ b_{j,r_2(j)}^{old} - (\alpha) + \left|(i - (r_2(j))\right| & for\ rest\ of\ i's \end{cases} \quad \textbf{Equation 4-10}$$

If there is no-improvement after n_α iterations inside each agent, S and B will act opposite to the above case. n_α is another parameter of the Co-MPS algorithm. Therefore, $s_{j,i}^{new}, b_{j,i}^{new}$ will be updated in G_1 based on the following formula:

$$b_{j,i}^{new} = \begin{cases} b_{j,r_2(j)}^{old} + \alpha & for\ i = r_2(j) \\ b_{j,r_2(j)}^{old} + (\alpha) + \left|(i - (r_2(j))\right| & for\ rest\ of\ i's \end{cases} \quad \textbf{Equation 4-11}$$

$$s_{j,i}^{new} = \begin{cases} s_{j,r_2(j)}^{old} - \alpha & for\ i = r_2(j) \\ s_{j,r_2(j)}^{old} - (\alpha) + \left|(i - (r_2(j))\right| & for\ rest\ of\ i's \end{cases} \quad \textbf{Equation 4-12}$$

For the example defined in Section 4.4.4, assume that agent 1 has the seed $\sigma_1^* = [4,3,1,2,5]$.

Then, it started to generate neighbors in *LS* and *LB*, based on the restriction imposed by and *B*:

$$LS = \begin{bmatrix} 4,2,3,1,5 & 4,3,2,1,5 & 4,3,2,1,5 \\ 4,2,1,3,5 & 4,3,2,1,5 & 4,2,1,3,5 \end{bmatrix}$$

$$LB = \begin{bmatrix} 4,3,2,1,5 & 4,2,3,1,5 & 2,4,3,1,5 \\ 4,3,2,1,5 & 4,2,1,3,5 & 2,3,1,4,5 \end{bmatrix}$$

Then, it assigned a weight to each neighbor and construct *VS* and *VB*:

$$VS = \begin{bmatrix} 29.8 & 23.4 & 23.4 \\ 31.9 & 23.4 & 31.9 \end{bmatrix}$$

$$VB = \begin{bmatrix} 36.5 & 32.1 & 35.7 \\ 36.5 & 31.9 & 30.8 \end{bmatrix}$$

In the next step, an agent will perform a selection to select "m_s" new solution form *LS* and "m_b" new solution from *LB*. The selection procedure or discriminative sampling will be discussed in Section 4.4.6. After the selection, all "m_s+m_b" will be evaluated by the objective function. Based on the improvement or non-improvement reported from the objective function, G_1 will update its elements (Controllers *S* & *B* will be enabled). Suppose after the selection, the best selected solution is *LS*(1,3)=(4,3,2,1,5) with an objective value of 56. Since 56<f(σ_1^*)=50, non-improvement will be reported from the objective function to G_1. Assume that n_α=1, which shows the number of steps of non-improvement to activate *S* and *B*. Therefore, *S* will decrease to allow for more exploitation around the current seed, which is a good candidate for local optima (since it

hasn't been changed in the last iteration), and also B will increase for exploration. Assuming $\alpha_c = 1$, G_1 will be updated by changing the following elements:

1. Changing in S_{ij}:

   a. $s_{4,1}$-1, $s_{3,2}$-1, $s_{2,4}$-1, $s_{1,3}$-1,$s_{5,5}$-1

   b. Since, $s_{4,1}$,$s_{3,2}$ and $s_{2,4}$ are changed, their neighbors will also change:

      - $s_{4,2}$-1+(2-1), $s_{4,3}$-1+(3-1), $s_{4,4}$-1+ (4-1), $s_{4,5}$-1+ (5-1)

      -$s_{3,1}$-1+|1-2|, $s_{3,3}$-1+(3-2), $s_{3,4}$-1+(4-2), $s_{3,5}$-1+(5-2)

      -- ...

2. Changing in B_{ij}:

   a. $b_{4,1}$+1, $b_{3,2}$+1, $b_{1,3}$+1, $b_{2,4}$+1, $b_{5,5}$+1

   b. Since, $b_{4,1}$, $b_{3,2}$, $b_{1,3}$, $b_{2,4}$, $b_{5,5}$ is changed, their neighbors will also change:

   - $b_{4,2}$+1+(2-1),$b_{4,3}$+1+(3-1),$b_{4,4}$+1+(4-1), $b_{4,5}$+1+(5-1)

   - $b_{3,1}$+1+|1-2|, $b_{3,3}$+1+(3-2),$b_{3,4}$+1+(4-2), $b_{3,5}$+1+(5-2),

   - ...

It should be noted that other agents will also affect G_1 in the same way as discussed above.

3. Updating G_2:

After all "m_s+m_b" selected solutions have been evaluated by the objective function, the best solution will be reported as the new seed of the agent. Next, the global memory (G_2) is updated by the function $f(\sigma\ new_u^*)$, which is an objective value of the new seed of an agent "u". Moreover, the new seed of agent "u" ($\sigma\ new_u^*$) is compared to other

$\sigma\ new_u^*$ of other agents to check if there is any improvement in the global best solution. If new global optimum is found by this agent, another function of $f(\sigma\ new_u^*)$ is used for updating the global memory.

It was mentioned in Section 4.4.4 that after any iteration, a value is added to the G_2 based on the reported new-optimum for every agent. The following formula show how this value is constructed and how it is added to G_2: (W_i is the given weighting factor of the SMTWT problem, and T_i is its tardiness).

$$f_{iter}^{u,i} = (1 - \frac{f(\sigma_u^*)}{\sum_{p=1}^{P} f(e_p)})(1 - \frac{f(i)}{f(\sigma_u^*)}) \quad , f(i) = W_i * T_i \; ; O_{i,\sigma_u^*(i)}^{new} = O_{i,\sigma_u^*(i)}^{old} + f_{iter}^{u,i} \quad \textbf{Equation 4-13}$$

In Equation 4-13, "e_p" denotes the elements in Elite lists. Elite list consists of "P" *best* solutions which have been reached so far. If the reported local optimum from any agent improves the global optimum, an extra value (F_{iter}) is used for updating G_2:

$$F_{iter} = (1 - (\frac{f(\sigma_u^*)}{\sum_{p=1}^{P} f(e_p)})^2)(1 - (\frac{f(i)}{f(\sigma_u^*)})^2) \quad , O_{i,\sigma_u^*(i)}^{new} = O_{i,\sigma_u^*(i)}^{old} + f_{iter}^{u,i} + F_{iter} \quad \textbf{Equation 4-14}$$

For the example from Section 3.4, let's see how G_2 is updated. In this step, all the agents report their best found solution, and the global memory will be updated based on these values. Let $P=2$, and suppose that in previous iterations, the best solutions found are: [(3,2,5,1,4), (2,3,5,1,4)], with objective values: [28, 28]. As discussed earlier, suppose the best selected solution reported by agent 1 is (4,2,1,3,5). As the objective value of the new seed is worse than the agents' current seed (56<50), so no-improvement will be

reported from this agent, and agent 1 will keep the seed of (4,3,1,2,5) and the following values will be added to the corresponding elements of G_2 by using Equation 4-9:

- (1-50/(28+28))*(1-0) will be added to (4,1) since job 4 is not a tardy job,
- (1-50/(28+28))*(1-0) will be added to (3,2) since job 3 is not a tardy job.
- (1-50/(28+28))*(1-24/50) will be added to (1,3) since 1 is a tardy job and it adds the value 24 to the objective function (its related cost).
- (1-50/(28+28))*(1-10/50) will be added to (2,4) since job 2 is a tardy job and it adds the value 10 to the objective function.
- (1-50/28+28))*(1-16/50) will be added to (5,5) since job 5 is a tardy job and it adds the value 16 to the objective function.

It should be noted that the current seed of the agent does not change the global optimum, so only the above calculation is needed for updating G_2 for agent 1.

4.4.8 Concept of new-born agents

At any iteration of the Co-MPS algorithm, a new-born agent will be created based on the following procedures. It should also be noted that at any iteration, only one new-born agent is being created. In the first half of iterations (1 to Maxiter/2), the first procedure is used. In the second half (from Maxiter/2 till the end), the second procedure is employed:

1. By comparing all the current local optima and elite list, if the probability of having job "*i*" in position *j* is more than a certain number (c_{iter}), job "*i*" will be fixed at position "*j*" for a new-born agent. Otherwise, a random job will be chosen for position "*j*". With the seed of fixed jobs *i* in position *j*, and having random selection for other position, the

following formula show how the probability can be calculated and compared to c_{iter} so that jobs with fixed position will be known for new-born agents:

$$\forall i = 1,..,n \ \ if \ \frac{n_{ij}}{P} \geq c_{iter} \Rightarrow \sigma^{*}_{new-born}(j) = i \qquad \textbf{Equation 4-15}$$

In the above formula, n_{ij} shows the number of occurrences of jobs i being at position j in all the elite list solutions. Similar to the concept of Simulated Annealing (SA), c_{iter} follows the following exponential function:

$$c_{iter} = e^{-(1-\frac{iter}{Maxiter})} \qquad \textbf{Equation 4-16}$$

, where "*Maxiter*" shows the maximum number of iterations for the algorithm.

As it can be seen from Equations 4-15 and 4-16, in the initial steps of the algorithm all the new-born agents are created randomly, since the n_{ij}'s are not enough for approximating a good new-born. As it goes on, the c_{iter} becomes bigger which allow more fixed positions for jobs, which leads to convergence. In last iterations, c_{iter} becomes close to 1, which will again lead to random generated sequence since n_{ij}'s would not be enough to approximate the exact global optima. Hence, this type of new-born strategy is used till "*Maxiter/2*" iteration. In the second half of iterations (form *Maxiter/2* till the end) the other type of new-born strategy is used, which will be discussed in the next section.

For the example problem, assume that from the previous iteration one has an elite list of [(3,2,5,1,4),(2,3,5,1,4)]. Moreover, [(4,3,1,2,5);(3,2,4,1,5);(3,5,2,1,4)] are the current seeds of all the agents in this iteration. Therefore, matrix N_e will be formed as:

$$N_c = \begin{bmatrix} 0 & 0 & 1/5 & 4/5 & 0 \\ 1/5 & 2/5 & 1/5 & 1/5 & 0 \\ 1/5 & 2/5 & 0 & 0 & 0 \\ 1/5 & 0 & 1/5 & 0 & 3/5 \\ 0 & 0 & 2/5 & 0 & 2/5 \end{bmatrix}$$

Where n_{ij} shows the probability of having Job "i" in the" j-th" position among the elite list and current local optima. For example, n_{14}=4/5 shows in four out of five sequences, Job 1 is in the 4th position. Now, c_{iter} has to be calculated, $c_{iter} = e^{-(1-\frac{iter}{Maxiter})}$, assuming maximum iteration of 100, if it is at the 10th iteration, c_{iter}=0.406. For those elements in the N matrix which have a probability more than 0.406, the job will be fixed. This is true for job 1 being in the 4th position and job 4 being in the 5th position. Consequently, a new-born agent has two fixed positions and other positions will be filled randomly: for example, New-born=(2,3,5,1,4).

2. Assignment problem for new-born agents: An assignment problem can be defined based on the n_{ij}'s, which shows how many times job "i" is in position "j" for all the elite list and current local optima. In fact with the objective function of maximizing the summation of "n_{ij}'s", one wants to assign every position of the sequence (from 1 to n), a particular job to make an initial seed for a new born agent. These types of new-born agents will be created in the last "Maxiter/2" iterations. In the first half of the iterations, there have not been enough solutions for matrix "N" to contain a good approximation of the global optima. Hence, second type of new-born generation will be used in the second-half of the iterations. Although the well-known Hungrain method or Munkres algorithm for solving assignment problem is a polynomial cost algorithm ($O(n^3)$), but for large

problems still it takes a large amount of time for solving to optimality. Therefore, a sub-optimal assignment algorithm of Markus Buehren (*www.Lss.uni-stuttgart.de*) is used for this type of new-born agents. The sub-optimal assignment algorithm is for minimization. Therefore, a large value is subtracted from the n_{ij}'s matrix and then the "Buehren" algorithm is called. The algorithm searches the matrix for the minimum element and makes the corresponding row-column assignment. After setting all elements in the given row and column to infinity (i.e. forbidden assignment), the search procedure is repeated until all assignments are done or only infinite values are found.

For the example problem, if the search is in the last "half" part of the iterations, the assignment problem will be solved for the following matrix:

$$N_c' = \begin{bmatrix} 0 & 0 & 1 & 4 & 0 \\ 1 & 2 & 1 & 1 & 0 \\ 1 & 2 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 3 \\ 0 & 0 & 2 & 0 & 2 \end{bmatrix}$$, and since the objective function is maximization, a function is subtracted from the matrix and then the assignment problem sub-optimal solver is called (which can be solved in polynomial time). This is due to the fact that the regular objective function of the assignment problem is minimization, and for solving a maximization problem one has to change the objective function to minimization by subtracting it by a function. A new-born agent will be created based on the solution of the assignment problem: New-born= (3, 2, 5, 1, 4).

4.4.9 Local search

Co-MPS algorithm produces high quality results in few steps, but on average it will not find the exact global optima in relatively few steps. In order to overcome this problem and improve the performance of the Co-MPS algorithm, a local search is introduced which will be performed when an agent reports the same seed for "2* n_α" iterations. This number is chosen based on the numerous runs of Co-MPS algorithm without the local search. The local search procedure is basically composed of multiple switch operations of tardy jobs and jobs with high job allowances. Job allowance is the difference between the completion time of a job and its due date. The steps of the local search are as follows:

Step 1. Find the tardy jobs of the current seed(σ^*), *tardy*=$\{t_1,..t_m\}$

Step 2. Calculate the job allowance of all the previous non-tardy jobs of t_i; Sort them in a descending order in E_i (E_i is the list of jobs before job t_i in σ^*);$i=1$.

Step 3.j=1; . While $j<length\ (E_i)$: Do the following operations

Step 3.1. Switch t_i with the j-th element in E_i

Step3.2. Calculate the objective function of the new sequence; Store the sequence and its objective function in the set *local_obj* ={(sig,f(sig)); $j=j+1$; If $f(sig)<f(\sigma^*)$, $s=\sigma^*$, go to Step 4; else Go to Step 3.1

Step 4. $i=i+1$; Update the rest of E_i's for new σ^*; If $i<Length(\ tardy)$ Go to Step 3; Else go to Step 5

Step 5. Calculate the minimum value of *f(sig)* in *local_obj*; report the minimum value and its corresponding sequence.

The local search algorithm is basically using the sequence of switch operations for every tardy job in the initial sequence. Series of switch operations have been used in other works such as iterated local search or dynasearch in Crauwels et al. (2002) and Maheswaran (2005), respectively.

For the example problem, suppose that agent 1 has the seed $\sigma_1' = [4,3,1,2,5]$ with an objective value f(σ_1^*)=50.

Step1. The tardy jobs are: (1,2,5). So, the job allowance for all non-tardy jobs will be:

Job-allowance (4)=19-7=12; Job-allowance(3)=8-8=0;

Step2. E_i's ($i=1,2,3$) will be formed as : $E_1=\{3,4\}$, $E_2=\{4,3,1\}$, $E_3=\{4,3,1,2\}$. The reason of sorting all $_i$'s in descending order is that the chance of having a better objective value by exchanging tardy jobs with jobs with high job-allowance is more than the jobs with low job-allowance.

Step3. In this step, first the tardy job in the σ^* will be exchanged with the first element in e_1, which result in sig=(1,3,4,2,5). Then, the objective value of this new sequence will be evaluated, f(sig)=29. Since, f(s)<f(σ^*), it will go to Step 4.

Step 4. E_i's (i=2,3) will be updated for the new sequence. E_2={4,1,3}, E_3={4,1,3,2}. Since job-allowance of job 4 is:19-16=3; job-allowance of job 1 is: 10-8=2; Now one will go back to step 3 to do the switch operation for the new σ^*.

Step5. Go back to Step3 in the algorithm; It will exchange job 2 with job 4 in the (1,3,4,2,5); So, s=(1,3,2,4,5), f(s)=20. Since, f(sig)<f(σ^*)=29. Save *"sig"* and *f(sig)* in local_obj ;Stop the switch operation, σ^*=(1,3,2,4,5) and go to Step4.

Step 6. Go to Step 4 in the algorithm; E_i (i=3) will be updated for the new seed; E_3={2,1,3,4}. Since, the job-allowance for job 2 is 18-13=5 and job-allowance for job 1 is 10-8=2. Then, one will go back to Step 3 of the algorithm to perform the switch operations

Step 7. It will exchange job 2 with job 5 in the (1, 3,2,4,5) sequence. So, *sig = (1,3,5,4,2)* and *f(sig)=26*. Since *f (sig)>f(σ*)=20*. It will continue with another exchange operation of job 5 and job 1 (second element in e_3). So, *sig* = (5,3,2,4,1)=52. Again no improvement was found, so *sig* = (1,5,2,4,3) and *f(s)=48*. Since *i=4* and *i>length (tardy)*, it will go to the step 5 of the algorithm.

Step 8. Since no-improvement has happened by the last set of exchanges, the best result which was saved in the set *"local_obj"* will be reported as sig=(1,3,2,4,5) and *f(sig)=20* and It will exit the algorithm.

As it was shown in the example, the local search breaks the whole sequence into smaller sequences based on the tardy jobs. Then, by switching the first tardy job with its previous jobs with a higher job-allowance, the algorithm tries to find a better sequence. In the next steps, the candidate list for switch operations will be updated for the second tardy job. Thus, the local search composed of multiple-switch operations of the tardy jobs and their previous non-tardy jobs.

As discussed earlier, at the beginning of any iteration in the Co-MPS algorithm, every agent start to generate new neighborhoods and following a certain procedures which I

discussed in Sections 4.4.4, 4.4.5 and 4.4.6, they will move to a better found neighbor solution or if there wasn't any improvement they will stay at their current seed. Next, the global memories G_1 and G_2 are being updated as discussed in Section 4.4.7. Next, a new-born agent will be created based on the comparison between the elite solutions, as discussed in Section 4.4.8. If there have been consecutive no-improvements in any agent, they will go through a local search as discussed in Section 4.4.9.

In the next step, if the stopping criteria haven't been met, all the agents will go back to the step one of the algorithm, which is to generate new neighbors around their current seed. In the example, which I have been using from Section 4.4.4, after agent 1 went through a local search and found a new improved seed of (1,3,2,4,5), it will go back and start to generate neighborhoods around this new seed in its local memory(*LS* and *LB*) as discussed in Section 4.4.4. Also, the new-born agents which have been discussed in Section 4.4.8, will go back to the step 1 of the algorithm and go through the same procedures as other agents.

In Section 4.4.10, I will bring the complete description of the steps of the Co-MPS algorithm.

4.4.10 Steps of the Co-MPS algorithm

After describing the components of the Co-MPS algorithm, the main steps of the Co-MPS algorithm are as follows:

Step1. Initialization: Set the parameters (K, m_c, α_c, n_α, m_b, m_s, P); "*iter*"=1; Generate K random sequence or based on dispatching rules, σ_u^* for u=1,..,K, as seeds for every agents; Set G_2 equal to $n*n$ matrix of 1's; Set G_1 equal to $n*n$ matrix of tupelos (n,n);

Step2. Each agent will perform the following:

Step 2.1. Select a tardy job in σ_u^*; Generate neighborhoods (*LS, LB*) based on the (s_{ij}, b_{ij}) from the global memory G_1.

Step 2.2. Construct *VS* and *VB* for weights of *LS*, and *LB* based on the values from G_2.

Step 2.3. Perform discriminative sampling to find m_s+m_b new solutions from *LS*, and *LB* respectfully.

Step 2.4. Evaluate the objective function of these m_s+m_b new points; Update the elite list; Set σ_u^* = the best local optima;

Step 2.5. Report any improvement and the current local optima to the global memory

Step 3. Find the best solution reported by all the agents; Update G_1 based on improvements reported from agents; Update the elite list; Update G_2 based on f_{iter}^u and F_{iter}.

Step 4. Define a new-born agent based on the "c_{iter}" and list all local optima plus elite list.

Step 5. For any agents, if there consecutive $2*n_\alpha$ non-improvements, apply the local search; Replace σ^* with the new local optima founded by the local search procedure.

Step 6. If finishing criteria are reached, stop (*iter=Maxiter* or global optima reached);

Else *iter=iter+1;K=K+1;* go to Step 2.

4.5 Results of Co-MPS on SMTWT problem

The algorithm is implemented using Matlab toolbox on Pentium 4, 3.0 GHz machine. Co-MPS has been used to solve the instances of SMTWT problem in OR-lib. Three set of test cases is presented in OR-lib with n=40, 50 and 100. Each problem size is composed of 125 instances. Each instance has been tested 25 times by the Co-MPS algorithm. Initial solutions of agents have been found using three dispatching rules:

1.WSPT: In this dispatching rule, all jobs are sorted such that $(w_1/p_1)>=(w_2/p_2)>=\ldots>=(w_n/p_n)$.

2. EDD: All the jobs are sorted by their earliest due dates.

3. R&M: This heuristic developed by Rachamadugu and Morton (1982) (also known as apparent urgency rule) sorts the jobs based on their priorities (π_i), where:

$\pi_i = (w_i / p_i)[\exp\{-(S_i)^+ / ap_{av}\}]$, where $S_i=d_i-p_i-t$, "w_i" is the weight of job i, "p_i " is the processing time of job i and d_i is the due date of job "i" If S_i is negative, the job is sure to be tardy and receives full WSPT priority. Typical value of the factor "a" is suggested as a=2, and p_{av} is the average processing time of all the jobs.

The used parameters of the Co-MPS are shown in Table 4-2. The parameters' values are tuned based on numerous runs of different problem sizes.

Table 4-2. Parameter setting for different problem sizes of SMTWT

Problem size (n)	K	m_c	α_c	n_α	m_b	m_s	P	Max-iteration
40	7	30	2	5	3	4	15	40
50	10	30	2	5	3	4	15	120
100	20	40	3	12	8	18	30	250

Measure of performance for SMTWT is as the followings:

1. ARPD: The average relative percent deviation from the optimal (or best known) value over all the 125 instances times 25 (number of runs for each instance).

ARPD=$\frac{x - x^*}{x^*} \times 100$, where x is the global optima found by the algorithm and x^* is the optimal solution (or best known) solution reported by OR-lib.

n_{hit}: Percentage of finding the optimal value reported by OR-lib in the 25 runs for 125 instances.

2. Average CPU time: Average run time in seconds over all 25 runs for 125 instances.

The algorithm stops when the optimum or maximum iteration is reached.

Table 4-3 shows the result of the Co-MPS algorithm for different problem sizes. As it can be seen from Table 4-3, R&M heuristic generates better results compared to other heuristics in terms of ARPD. "WSPT" rule outperforms "EDD" for 40 and 50 job problem, but not for the 100-job problem. For problem sizes of 40, and 50, the Co-MPS algorithm produces the results with high quality and hits the optimum solution more than

90% percent of time with the R&M heuristic. For the 100-job problem, although the Co-MPS can not maintains its quality for lower dimensions but still can generate very good solutions in a reasonable time. For future research, the authors believe that combining the Co-MPS with a local search procedure at the end of its iterations will generate very high quality results for the 100 job-SMTWT problem.

Table 4-3. Results of Co-MPS for different problem sizes of SMTWT

Problem size (n)	Method of generating initial solution	ARPD	n_{hit}	Ave. CPU time (sec)
40	EDD	1.19	0.70	7.01
	WSPT	0.98	0.87	7.14
	R&M	0.19	0.93	7.58
50	EDD	1.03	0.75	13.84
	WSPT	0.99	0.85	13.95
	R&M	0.29	0.93	14.89
100	EDD	2.01	0.36	79.57
	WSPT	2.11	0.34	81.19
	R&M	1.56	0.48	81.49

Table 4-4 shows the comparison of Co-MPS (with R&M heuristic for initial solution) with PSO algorithm and DE algorithm of Tasgetrin et al. (2006). Tasgetrin et al.(2006) did not report the computing machine specifications (speed and RAM capacity).

Table 4-4. Comparison between Co-MPS and Tasgetrin et al. (2006)

Method	Problem size (n)								
	40			50			100		
	ARPD	n_{hit}	CPU time	ARPD	n_{hit}	CPU time	ARPD	n_{hit}	CPU time
PSO	1.31	0.63	6.02	1.10	0.48	13.72	2.35	0.26	85.37
DE	1.61	0.76	3.04	1.24	0.61	8.53	2.31	0.24	66.63
Co-MPS	0.19	0.93	7.58	0.29	0.93	14.89	1.56	0.48	81.49

As it can be seen from Table 4-5, our results are superior to PSO in terms of ARPD.

Table 4-5 shows the comparison between Co-MPS and taboo search results of Bilge et al. (2007). The termination criterion of Co-MPS is set as the CPU-time reported by Bilge et al. (2007).

Table 4-5. Comparison between Co-MPS and Bilge et al. (2007)

Method	Problem size (n)					
	40		50		100	
	ARPD	CPU time	ARPD	CPU-time	ARPD	CPU-time
[19]	0.034	27.1	0.003	41.5	0.007	283.5
Co-MPS	0.026	27.1	0.095	41.5	0.251	283.5

As it can be seen from the table, our results are superior for the 40 jobs problem in terms of ARPD. The initial solution of the Co-MPS algorithm has been generated based on the R&M heuristic.

4.6 Discussion

As it has been discussed in Section 4.4.2 for the general description of the Co-MPS algorithm, the proposed method composed of *"K"* agents which generates neighborhood around their current local optima. At the same time, the first component of the global memory (G_1) which is composed of two values for controllers B and S restricts the neighborhood generation for every sequence. G_1 keeps the information of the allowable range for each job *"i"* at position *"j"* based on the improvements from past iterations. After generating neighborhood solutions for every sequence, these neighborhood solutions will be evaluated based on the global memory (G_2). This evaluation is basically an approximation of the objective function since G_2 is composed of the measure of "goodness "of every job *"i"* at position *"j"* based on the past information form the algorithm. After all the neighborhood solutions around the current sequence of every agent is being approximated, they will go through a selection based on their

approximated values and "m_s+m_b" new solutions will be selected. These new selected solutions will be evaluated by the objective function. Based on the objective function of these values, G_1 and G_2 will be updated, and agents will move their current seed to the better local solution.

Co-MPS algorithm is a meta-heuristic algorithm which has been proposed for combinatorial optimization in complex systems. It has similarities to Ant algorithms, Tabu search, Simulated annealing and Genetic algorithms. I will discuss the similarities and differences between Co-MPS and other major meta-heuristics in the following sections:

Co-MPS and Ant algorithms

- Co-MPS came from a meta-model concept of design optimizations, which tries to have a control over all the solution space. In fact, these algorithms are iteratively improving heuristics and not constructive methods. (Such as Ant systems or any ACO algorithm)
- The global memory of Co-MPS has almost the same structure as pheromone trails of connections in Ants systems. The main difference is that τ_{ij} shows the weight for having j^{th} component after i^{th} component, without considering the position of any of i and j^{th} component in the main solution. In Co-MPS, Q_{ij} in the global memory denotes the weight of having "i^{th}" component at the "j^{th}" position. Also, the way that the global memory is being updated is dependent of the relative

measure of objective function of every agent and the summation of elite solutions objective function, in addition to the relative measure of heuristic information.

- In Co-MPS, moving to a new neighborhood solution is done in the local memory of each agent, which assigns the value from the global memory to every neighborhood of the current seed of any agent and then performs a selection to find the new solution. In fact, Co-MPS makes an approximation over the neighborhoods of the current solution. This approximation evolves over the optimization run-time and becomes more exact at the end of the algorithm. In Ants system, the probability distribution is about selecting a new sequence to enter the agents' current solution, not changing the whole neighborhood. This probability also becomes more exact in the last iteration of the Ant algorithm, given pheromone trails of all the agents in previous solutions.
- Generating neighborhoods in Co-MPS is restricted by G_l based on behavior of S and B, which is a complex direct search methods or a generalized fuzzy tabu-move.

Co-MPS and Tabu search

G_l which restricts the neighborhood search is a generalized fuzzy tabu move, in the sense that it will gradually decrease or increase the degree of freedom of every job at their positions based on the improvement or consecutive non-improvements in the past iterations of the optimization process. In tabu search, depending on the definition of a tabu move, the neighborhood generation will be restricted if a tabu move has occurred in past iterations. For example, after visiting the neighborhood which exchanges the

second job with the third job, this particular "exchange" move can be saved as a tabu move. In Co-MSP algorithm, after having consecutive improvements from exchange or insert moves which results in job "2" at second position, the degree of freedom of job "2" will be small enough around the second position, as controlled by the controller "*S*" to prevent it from any further exchanges or insertion moves in *LS*. (At the same time, controller "*B*" will make this distance big in *LB* to prevent the search being trapped in a local optimum).

Co-MPS and Simulated annealing

In Co-MPS, we used the concept of cooling mechanism for generating new-born solutions. "Cooling mechanism" is the same as the procedure which have been used in Simulated annealing algorithm. After any iteration in Co-MPS, the best known solutions are compared with each other. If the probability of having a job at a certain position is higher than a dynamic probability function, I accept and freeze that job in that position. As the number of iterations reaches the maximum, accepting a job at specific position is done if many occurrences of this job at the specific position have been seen in elite solutions.

As described by (Glover (1997) and Glover and Laguna (1997)), "fragments" (elements) of excellent solutions are often identified quite early in the searching process, but that the challenge is to complete these fragments or to recombine them. The concept of "new-born" agents is a novel concept which is used to identify these fragments and combine them in a unique way.

Co-MPS and Genetic algorithm

The concept of "selection" or "discriminative sampling" which is done based on the inverse CDF of the particular samples has been used in Co-MPS method. The same concept has been used in "Genetic algorithms". Selection is used to select new neighborhood solutions to be evaluated by the objective function. Corresponding CDF is constructed based on the approximation values on the global memory G_2 which shows how good the current neighborhood solution is.

Comparing to all other meta-heuristics, Co-MPS has the main advantage of significantly reducing the number of objective function evaluations, by approximating the neighborhood solutions of every seed, instead of directly evaluating the objective function. Although in many combinatorial optimization problems, the objective function is a linear function, for complex systems the objective function is usually hard to be mathematically formulated, and hence a discrete-event simulation instead of the explicit objective function is usually used. Co-MPS is best suited for these types of simulation-based optimization problems. In this Chapter, I used a simple case of single machine total tardiness problem to test the potentials of the Co-MPS algorithm. The more complex single machine total tardiness problems, i.e. dependent set-up times, having labors, shifts for labors, etc., can also be addressed simply by substituting the objective function with the simulation program. Having G_2 as an approximation matrix which saves the information based on the frequency and "goodness" of every job at each position will give an overall view of the whole problem. In addition to significantly reducing the number of black-box call or objective function evaluations (in Co-MPS, G_2 will result in

"$4*m_c-m_s-m_b$" reduction of objective function evaluations in every iteration compare to the case which there is no G_2), it is helpful in identifying near-optimal solutions in early iterations of the optimization algorithm. Also, it has the information which helps the designer of any complex combinatorial system, to perform the sensitivity analysis and know the "good" positions of every job which have higher weights than the others. At the same time, G_1 will also show which jobs have higher degree of freedom to move to the other positions. All in

all, the global memory of Co-MPS can be used in post-analysis discussion of every complex systems design or scheduling. These characteristics of the Co-MPS differentiate it form all other meta-heuristics.

4.7 Summary

Based on the previously developed discrete mode pursuing sampling (DMPS) approach, a variation of DMPS for combinatorial optimization problems has been developed in this chapter. Combinatorial mode pursuing sampling (Co-MPS) can be categorized as a meta-heuristic approach for simulation-based optimization problems in the sense that it starts from a few samples of the solution space and intelligently moves to the better neighborhoods of the initial samples. Co-MPS is composed of number of agents with local memories, which generate neighborhood around their current optima. At any iteration, every agent uses a global memory to assign weights to the neighborhood solutions. This due to the fact that Co-MPS is developed for simulation-based optimization functions, and assigning weights or approximating neighborhood solutions significantly reduce the number of function evaluations or black-box calls. Based on

these weights, a selection or inverse CDF sampling is performed inside each agent for finding new solutions. The current global memory is then updated by a function of values reported from all agents. On the other hand, generating neighborhoods with "insert" and "exchange" moves are restricted by two domain controllers *"S"* and *"B"*. These two controllers control the domain of new neighborhood solutions based on the past improvements during the search procedure. They dynamically control the exploration and exploitation, although none of them is directly assigned to these tasks. In the early steps of the algorithm, *"B"* is in charge of the exploitation, while in the latest steps *"S"* will be in charge for exploiting around the current optima. To our knowledge, none of the metaheuristic approaches has used an intelligent domain controller for generating the neighborhoods. Results of Co-MPS, on benchmark problem from both OR-library and reported results in the literature show that the proposed method is a promising meta-heuristic global optimization algorithm for different sizes of SMTWT problem. In addition to the potential of Co-MPS to find the global optima for different instances of SMTWT problem, the number of function evaluations is significantly less than other methods, since there would be no need for evaluating all the neighborhood solutions. In fact, one component of the global memory of Co-MPS (G_2) will help to approximate all the neighbors of current optima of any agents and only the selected neighbors will be evaluate by the objective function or black-box. Applying Co-MPS to other combinatorial problems and especially complex systems optimization with discrete-event simulation will be further studied.

Chapter 5

A MPS-based Meta-heuristic for Traveling Salesman Problem

5.1 Overview of the proposed algorithm

A novel meta-heuristic approach for TSP has been developed (TSP-MPS) based on Combinatorial Mode Pursuing Sampling method (Co-MPS). As discussed in Chapter 4, Co-MPS was successfully applied to the single machine problem with total weighted tardiness. Similar to the Co-MPS, the proposed TSP-MPS method employs a number of agents inside the solution space and each of them generates a neighborhood around its current seed or optimum. The double-sphere method previously developed for Co-MPS dynamically controls the convergence behavior by varying the domain of generating new neighborhood tours of TSP. This novel technique features two controllers whose radii are dynamically enlarged or shrunk in order to control the degree of "exploration" and "exploitation. The new neighborhood tours of each agent enter the selection procedure, which assigns a value to each tour based on the information from the global memory and then performs the selection or CDF inverse sampling, the same concept used in D-MPS (Chapter 3). The global memory is being updated base on the improvement reported from each agent. In order to create a new-born agent at the beginning of any iteration, all agents communicate with the global memory and also communicate with each other. The double-sphere method also changes the radii of both controllers based on the improvement or consequent non-improvement reported from all the agents. TSP-MPS differs from Co-MPS on generating neighborhood and local search procedures.

Experimental results of TSP-MPS on benchmarks of three medium size instances of TSP are promising compared to the reported results in the literature.

5.2 Introduction

Traveling Salesman Problem is determining the shortest path in a given collection of cities and the distance matrix between every pair, so that the route visits each city precisely once and then returns to its starting point. More mathematically I may define the TSP as follows: Given an integer $n_t \geq 3$ and $n_t * n_t$ matrix $C = (c_{ij})$, where each c_{ij} is a nonnegative integer, the question to be answered is "Which cyclic permutation (π) of the integers from 1 to n_t minimizes the sum $\sum_{i=1}^{n_t} c_{i\pi(i)}$?"

It should be noted that this work deals with the symmetric TSP, so for any i ,j , $c_{ij}=c_{ji}$. TSP can also be modeled as a graph problem by considering a complete graph $G=(V,E)$, and assigning each edge $uv \in E$ the cost c_{uv}. A tour is then a circuit in G that meets every node. In this context, tours are sometimes called *Hamiltonian circuits.*

The traveling salesman problem is a relatively old problem; it was documented as early as 1759 by Euler, whose interest was in solving the knights' tour problem. A correct solution would have a knight visit each of the 64 squares of a chessboard exactly once on its tour. The term 'traveling salesman' was first introduced n 1932, in a German book written by a veteran traveling salesman. The TSP was introduced by the RAND

Corporation in 1948. TSP has occupied thoughts of numerous researchers for the following reasons:

1. Although very easy to describe, it is very difficult to solve. In fact, no polynomial time algorithm is known which can solve any TSP instances to optimality. TSP is a classical example of NP-hard problem.

2. TSP is applicable to a broad range of scheduling problems.

3. TSP has become a test problem for many meta-heuristic algorithms.

TSP arises in many practical problems, e.g. finding the minimum tour of the cities, minimum tour of drilling holes in a circuit, vehicle routing problem, etc. Recently, large TSP problems can be solved efficiently to optimality. As of 1994, the record is 7397 nodes (Applegate et al. (1995)). The node coordinates for benchmark instances are contained in the "TSPLIB" in Reinelt (1991) (http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsplib.html).

TSP has been tackled with several different approaches archived in the literature: heuristics, linear programming (LP) models, integer programming (IP) models, and meta-heuristics. The literature review on TSP will be discussed in Section 5.3. In this chapter, a novel meta-heuristic algorithm has been proposed. This algorithm will be noted as 'TSP-MPS' (TSP-Mode Pursing Sampling algorithm). The TSP-MPS algorithm has a certain characteristics which differentiate it from other meta-heuristic approaches in the literature. Although a global memory or local memory for a search procedure has been used in other meta-heuristics, the structure of these memories, the way they interact with each other and being updated in TSP-MPS are different from those of Ant algorithms,

Tabu search, etc. Moreover, a new-domain controller has been proposed which has control over the domain of generating neighborhood solutions based on the previous results from the search algorithm. TSP-MPS differentiates from Co-MPS in generating neighborhood solutions and also local search procedure. The results of the TSP-MPS algorithm show high quality solutions for three instances of TSP, comparing to those currently in the literature.

This chapter is organized as follows. In next section, the literature review on Traveling Salesman Problem (TSP) will be discussed. In Section 5.4, I will describe the TSP-MPS algorithm and its characteristics. Results of the TSP-MPS algorithm for three benchmark instances will be presented in Section 5.6.

5.3 Literature review of TSP

TSP has a very broad literature in terms of solution methodology. Here, I will briefly divide the literature into three categories, which are related to our approach to TSP:

1. Heuristic approaches which are related in neighborhood search and initial solution generation of TSP-MPS algorithm.
2. LP and IP programming methods
3. Meta-heuristic methods

5.3.1 Heuristics for the TSP

Heuristics are methods that cannot be guaranteed to produce global optima, but will produce fairly good solutions in most of the times. For the TSP, there are two types of

heuristics. The first category of heuristics tries to construct a "good" tour from scratch. The second category starts from an initial tour and improves the existing tour by means of local improvements. Although it is difficult to get a really good tour by the construction method, I will briefly describe two methods here. I use constructive heuristics as the initial tour generator of the TSP-MPS method. I will describe the second category more deeply since they will be used in the TSP-MPS method.

5.3.1.1 Constructive heuristics

Here, I describe two of the most studied constructive heuristics for the TSP: the Nearest Neighbor Algorithm, and the Insertion Method.

1. Nearest Neighbor Algorithm: This constructive algorithm is a simple greedy algorithm. It starts at any node; visits the nearest node not yet visited, then returns to the start node when all other nods are visited. Johnson et al. (1997) reported that on problems in TSPLIB, the average costs of tours found by the Nearest Neighbor Algorithm are about 1.2 times the cost of the corresponding optimal tours. Finding the guaranteed bound will be possible if the edge costs are nonnegative and satisfy the triangle inequality (which happens in tours of cities or any Euclidean TSP problem): $c_{uv}+c_{vw}>=c_{uw}$, for all $u,v,w \in V$. In this case, Rosenkrantz et al. (1977) showed that a Nearest Neighbor tour was never more than ½*[log2*r]+1/2 times the optimum, where r is the cardinality of V.

2. Insertion Method: Insertion methods provide a different set of tour construction heuristics. They start with a tour joining two of the nodes, and then add the remaining nodes one by one, in such a way that the tour cost is increased by a minimum amount.

There are several variations, depending on which two nodes are chosen to start, and more importantly, which node is chosen to be inserted at each stage.

As described in Cook et al. (1998), usually the best insertion method is the "Farthest Insertion". This heuristic works as follows:

Step1. Start with an initial tour passing through two nodes that are the ends of some high-cost edge.

Step 2. For each un-inserted node v, compute the minimum cost between v and any node in the tour constructed thus far.

Step 3. Next node to be inserted in the tour is the node for which the cost computed in step 2 is the maximum; Go back to Step 2.

"Nearest Insertion" is another variant of insertion heuristics which, at each stage, chooses the next node to insert the one for which the cost to any node in the tour is minimum. Another variant is the "Cheapest Insertion". In this case, the next node for insertion is the one that increases the tour cost the least. Usually the solutions produced by Nearest Insertion and Cheapest Insertion is inferior to those produced by Farthest Insertion. On the TSPLIB problems, Johnson et al. (1997) reported that, on average, Farthest Insertion found tours of length about 1.16 times that of the optimal tours.

5.3.1.2 Tour improvement heuristics

There are several standard methods to improve an existing tour T. Some of them are: *2-opt*, *3-opt*, and *k-opt*.

- *2-opt*, *3-opt* and *k-opt* methods: The simplest one is *2-opt*, or *2-optimal*. It starts by selecting two non-adjacent pair of edges of tour *T*. If these edges are deleted, then *T* breaks up into two paths T_1, T_2. As it can be seen in Figure 5-1, there is a unique way that these two paths can be recombined to form a new tour *T'*. If $c(T')<c(T)$, then *T* is replaced by *T'* and the process repeats. For a general cost function, in order to check whether a tour is *2-optimal*, *O* (|*V*^2|) pairs of edge have to be checked. But, this does not mean that one can transform a tour into a *2-optimal* tour in polynomial time. Papdimitriou and Steiglitz (1977) showed that if one makes unfortunate choices, one may perform an exponential number of interchanges, before a *2-optimal* tour is found. This is due to the fact that after any exchange, a new set of edges will need to be checked.

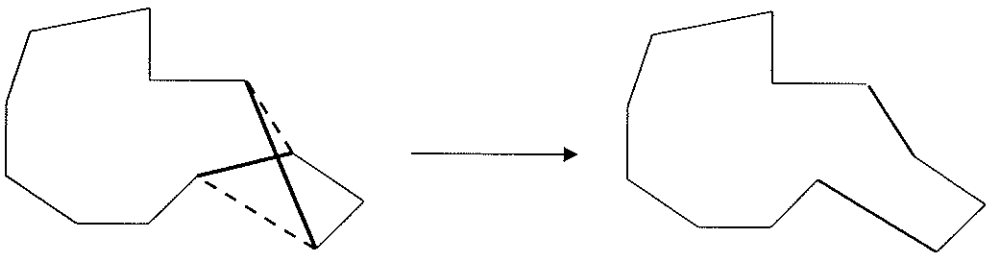


Figure 5-1. 2-Interchange (2-opt move)

2-opt algorithm can be generalized to *k-opt* algorithm, simply by considering all subsets of the edge-set of a tour of size *"k"*, or size at most *"k"*, removing each subset in turn, and then checking to see if the resulting paths can be recombined to form a tour of lower cost. The problem with *k-opt* moves is that the number of subsets grows exponentially with *"k"*. For this reason, as reported by Cook et al. (1998), *k-opt* moves for k>3 is

seldom used. The *2-opt* and *3-opt* moves are used as a neighborhood generator mechanism for all the agents in TSP-MPS algorithm.

5.3.2 LP and IP programming methods

Dantzig et al. (1954) proposed to attack the TSP with linear-programming methods. As mentioned by Cook et al. (1998), the approach they outlined is still the most effective method known for computing good lower bounds for the TSP. Their work also plays a very important role in the history of combinatorial optimization, since it was the first time cutting-plane methods were used to solve a combinatorial problem. Here I will mention their integer model.

Let "x" be the characteristic vector of a tour in a graph $G= (V,E)$. Then "x" satisfies:

$$Minimize \sum c_e x_e : e \in E)$$

$$\begin{aligned} & x(\delta(v)) = 2 \ for\ all\ v \in V \\ & x(\delta(S)) \geq 2,\ for\ all\ \varnothing \neq S \neq V \\ & x_e = \{0,1\},\ for\ all\ e \in E \end{aligned} \qquad \textbf{Equation 5-1}$$

In Eq. (5-1) $\delta(v)$ denotes the degree of node v, which means the number of edges connected to node v. The second constraint confirms that there should be a tour which passes through every node. In fact, it guarantees the connectivity of the tour, ensuring that the summation of degrees of every subset of set V is more than two. In the following example, we show how the second constraints guarantee the connectivity of tour "x".

Suppose $G=(V,E)$, $V=\{1,2,3,4,5,6\}$, and suppose they are connected with the following characteristic vector x:

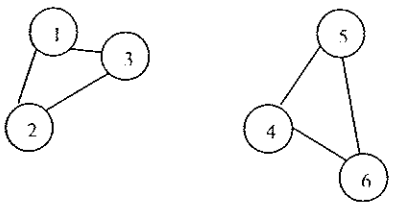


Figure 5-2. Example for the Eq.(5-1)

As it can be seen in Figure 5-2, the first constraint of Eq. (5-1) is satisfied since the degree of all the nodes are equal to two. The second constraint is not satisfied, since we can find the set $S= \{1, 4\}$, which has two nodes and the summation of degrees of the subset graph is zero.

The solution space of the Eq. (5-1) is a subset of E. At each iteration of solving the optimization problem of Eq. (5-1), it would check if the subset of E (set of x_e's) satisfy the constraints. If so, the corresponding objective function will be evaluated. As we mentioned earlier, Dantzig et al. (1954) approached the integer program (Eq. (5-1)) with cutting-plane method.

5.3.3 Meta-Heuristic methods

Huge amount of research has focused on meta-heuristic approaches for the TSP. General purpose meta-heuristics methods like simulated annealing (Johnson et al. (1995)) & Pepper et al. (2002)), tabu search (Knox et al. (1994)), iterated tabu search (Misevičius et al. (2005)) genetic algorithms (Freisleben et al. (1996)) have been applied for solving the TSP.

Johnson et al. (1995) used the simulated annealing algorithm with the *2-opt* move as their neighborhood generating mechanism. As they mentioned that they used some heuristics for speeding up the neighborhood generation. Due to the *2-opt* neighborhood, their particular implementation takes O (n^2) with a large constant of proportionality. Their results are comparable to "Lin Kernighan algorithm," (Cook et al.(1998))(which is one of the best heuristics that exist in the TSP literature. Pepper et al. (2002) used hybrid simulated annealing with other heuristics; their results are of a high quality in the TSP-literature. As mentioned Knox et al. (1994), there are several ways of implementing the tabu search for TSP. One involves adding the two edges being removed by a *2-opt* move to the list. A move will then be considered tabu if it tries to add the same pair of edges again. Another way is to add the shortest edge removed by a *2-opt* move, and then making tabu any move involving this edge. They have used both of the methods and compared the results. Although the basic idea of tabu search is to allow for even non-improving neighbors and at the same time having a tabu list for avoiding a cycle, Misevičius et al. (2005) discussed that using iterated tabu search (ITS) would be more suitable for large instances of TSP. ITS is composed of basic features of tabu search plus the intensification and diversification schemes. Their results are one of the best currently available in the literature. The performance of TSP-MPS will be compared to their result in Section 5.6.

There has been intensive research on genetic algorithms and the TSP. Freisleben et al. (1996), Jog et al. (1991), Homaifar et al. (1992)) have studied several crossover techniques and mutations. Jog et al. (1991) proposed a partially modified crossover

(PMX) with the tour notation and no mutation operator, and found that this algorithm gave a tour whose length was ten percent larger than the known optimum for the 33 city. Homaifar et al. (1992) also tested an algorithm that used only the *2-opt* mutation operator and no crossover. This also performed decently, however not as well as the previous case where he used matrix crossover. Recently, parallel genetic algorithms with more complex crossovers and *2-opt* mutation have been used in Freisleben et al. (1996), which produced fairly high quality results.

5.4 Description of TSP-MPS Algorithm

5.4.1 Background of TSP-MPS

TSP-MPS is basically another version of Co-MPS, which was discussed in Chapter 4. TSP-MPS can also be considered as a population-based metaheuristic algorithm. In this approach, there are "*K*" agents in the solution space; each of them has a point-to-point approach of finding better solution and moving to a new solution. At any iteration, every agent starts to generate a neighborhood around its current seed with respect to values from controller *B* and *S*. Then a value is assigned to these neighbors based on the global memory. Selection is being performed for finding a new seed of every agent.

TSP-MPS is based on the same concepts as Co-MPS, except for some minor modifications. As discussed in Chapter 4, Co-MPS is composed of ideas from other metaheuristic algorithms and a novel domain controller which adaptively determines the domain for neighborhood generation. The concept of selection originates from genetic algorithm, while having several agents interacting with each other is inspired by the "Ant

algorithm." At the end of any iteration, there is a new-born agent which has the intrinsic characteristics of other agents. The method of generating new-born agents comes form "Simulated Annealing." In the last half of iterations, an assignment problem will be solved for assigning every position a city. The assignment matrix will be constructed based on the elite list solutions.

TSP-MPS differentiates form Co-MPS in generating neighborhood. As discussed in Chapter 4, generating neighborhood for every agent was based on the "insert" and "exchange" moves of a random tardy job. In TSP-MPS, neighborhoods are generated based on *2-opt* and *3-opt* moves, which I will discuss in Section 5.4.2.1. Another main difference of TSP-MPS and Co-MPS algorithm is about generating initial solutions, which I use constructive methods (Farthest Insertion, described in 5.3.3.1) in TSP-MPS instead of dispatching rules in Co-MPS. In section 5.4.2, I will describe the main algorithm of TSP-MPS. In Section 5.5, features of TSP-MPS will be described with an illustrative example. In Section 5.6, results of TSP-MPS on three instances of TSP-LIB will be described and compared with other results in the literature.

5.4.2 TSP-MPS algorithm

TSP-MPS algorithm is composed of "K" agents ($u=1,...,K$) and a global memory which sends the appropriate values to the agents during the run time. The flowchart of the TSP-MPS and steps of the algorithm are similar to Co-MPS. In fact, instead of jobs in Co-MPS, cities should be sequenced in TSP-MPS and the objective function is the

summation of cost for the tour of the cities which is different from the weighted tardiness in Co-MPS.

In TSP-MPS every agent starts to generate neighborhood around its current seed in two matrixes *LS* and *LB*. *LS* and *LB* are $2 \times m_c$ matrices, where "m_c" is the parameter of TSP-MPS algorithm. Neighborhood solutions in *LS* are restricted by the values of s_{ij} and neighborhood solutions in *LB* are restricted by b_{ij}. The tupelos (s_{ij}, b_{ij}) comes form the global memory G_1. At any iteration, elements of G_1 will be updated based on the improvement or consecutive non-improvement to guide the neighborhood generation mechanism more toward exploitation around the seed of every agent or exploration of the solution space.

In the next step, all the elements of *LS* and *LB* will be assigned a value from G_2, which is the second component of the global memory. Next, the selection will be performed on the elements of *LS* and *LB* of every agent and "m_s", "m_b" solutions will be selected respectively. At the end, these new selected solutions will be evaluated by the objective function and global memory (G_1 and G_2) will be updated based on the relative measure of the objective value of improved solutions and the summation of the elite list solutions. Also, the best solutions will be gathered and compared to each other to build a new-born solution. The new-born generation mechanism is the same as Co-MPS.

Here, I will focus on the differences of TSP-MPS which are: neighborhood generation mechanism, updating G_2, and initial solution generation. Initial solution generation was discussed in Section 5.3.1.1. Neighborhood generation mechanism inside the local memory of each agent will be discussed in Section 5.4.2.1. Section 5.4.2.2 will describe the updating procedure for G_2.

5.4.2.1 Neighborhood generation mechanism inside local memory

As it has been discussed in Chapter 4, the local memory of each agent is composed of four matrices: $LS^{A_u}_{iter,\sigma_u^*}$, $VS^{A_u}_{iter,\sigma_u^*}$, $LB^{A_u}_{iter,\sigma_u^*}$, and $VS^{A_u}_{iter,\sigma_u^*}$. "*S*" and "*B*" are the controllers that define the domain of the new neighborhood solutions. These controllers provide (s_{ij},b_{ij}) values at the beginning of each iteration. Where (s_{ij},b_{ij}) represents the distance of *2-opt* and *3-opt* moves in *LS* and *LB*. "*S*" and "*B*" have dual behavior with respect to each other in case of improvement or consecutive non-improvement.

The first set of rows of *LS* and *LB* is constructed based on "*2-opt*" moves while the second set of rows of *LS*, *LB* is constructed by "*3-opt*" moves. In the following, the method of constructing *LS* and *LB* by each agent will be explained.

At each iteration, one random node of the current seed (σ_u^*) of agent "*u*" is identified. Where σ is a sequence with $\sigma = \{(i, r(i)) : r(i)$ *is the rank of node "i" in the tour*$\}$. Next, for the first row, three more nodes will be chosen $(j^*, r(j^*)) \in \sigma_u^*$ in the range of s_{ij}. Then, the *2-opt* move is performed on the selected three nodes plus the original node.

For the second row, again a random node $(j^*,r(j^*)) \in \sigma_u^*$ is selected . Then, five other random nodes are being selected in the range of s_{ij}. Next, the *3-opt* move is performed on the five selected nodes plus the original node.

The same structure is performed by replacing *LB* with *LS* and b_{ij} instead of s_{ij} in the above descriptions. It should be noted that if s_{ij} or b_{ij} becomes small, the new neighborhood solutions in *LS* or *LB* will be close to the seed and each other, which makes the search move toward exploiting the region around σ_u^*.

In the next step, matrixes *VS* and *VB* are constructed based on the weights which are assigned to *LS* and *LB* from G_2. In the next step, the selection will be performed on *VS* and *VB* of every agent for selecting "m_s+m_b" new solutions from *LS* and *LB*. These new solutions will be evaluated by the objective function and will be used for updating G_1 and G_2.

5.4.2.2 Structure of Global Memory

At any iteration, all agents communicate with the global memory for the following reasons:

4. Generating neighborhood solutions or constructing *LS*, *LB*; Agents communicate with Global Memory for $s(j^*,r(j^*))$ and $b(j^*,r(j^*)$ from the controllers;
5. Assigning a weight to their new generated neighborhood solutions;
6. Updating the global matrices based on the local improvements.

Global memory of Co-MPS consists of two matrices; G_1 and G_2, which have the same characteristics as the global memory in Co-MPS.

As it was described in Section 5.4.2.1, "m_s+m_b" selected solutions from *LS* and *LB* are evaluated by the objective function, and the best solution is selected. If the new selected solution is better than the previous seed, it will replace the current seed for agent "u" (σ_u^*), and the improvement is reported to the global memory for updating G_1. It should be noted that all "m_b+m_s" are evaluated and therefore they are compared to the elite list *(E)* to check if there is any improvement by these points. Next, the global memory is updated by the function $f(\sigma_{new,u}^*)$. Moreover, $\sigma_{new,u}^*$ is compared to other $\sigma_{new,u}^*$ of other agents to check if there is any improvement in the global best solution. If a new global optimum is found by this agent, another function of $f(\sigma_{new,u}^*)$ is used for updating the global memory.

I assume that G_2 is composed of O_{ij}'s which shows the weight of city "i" being at position "j", the following formula show how this value is constructed and how it is added to G_2:

$$f_{iter}^{u} = (1 - \frac{f(\sigma_{new,u}^{*})}{\sum_{p=1}^{P} f(e_p)}),\ O_{ij}^{new} = O_{ij}^{old} + f_{iter}^{u},\ for\ all\ (i,j) \in \sigma_{new,u}^{*} \quad \textbf{Equation 5-2}$$

"e_p" denotes the elements in elite list. Elite list consists of "P" best solutions which have been reached so far. If the reported local optimum of an agent improves the global optimum, an extra value (F_{iter}) is used for updating G_2:

$$F_{iter} = (1 - \frac{f(\sigma_u^*)}{\sum_{p=1}^{P} f(e_p)})^2,$$

$$O_{ij}^{new} = O_{ij}^{old} + f_{iter}^{u} + F_{iter}, for\ all\ (i,j) \in \sigma_{new,u}^{*} \qquad \textbf{Equation 5-3}$$

5.5 Illustrative example

In this section, an example will be used to illustrate the steps of the TSP-MPS algorithm. Table 5-1 shows the cost matrix of 5 nodes of the TSP.

Table 5-9. Cost Data for TSP example

City	1	2	3	4	5
1	0	4	7	10	3
2	4	0	9	8	2
3	7	9	0	6	7
4	10	8	6	0	9
5	3	2	7	9	0

Step 1. Initialization: Initial tour will be constructed based on the "Farthest insertion" method. First two highest cost nodes will be inserted: *1-4*. In the next step, the minimum cost of all other nodes connecting to 1 and 4 will be computed:

Node 2: 2-1, cost: 4. Node 3: 3-4, cost: 6. Node 5:5-1, cost: 3.

In the next step, the maximum of the above node will be inserted as the new node: *Node 3:3-4, cost: 6*. The new sub-tour will be: *1-4-3*. The above steps will be continued till the complete tour is being generated: *1-4-3-2-5-1* with the cost of *36*.

Step 2. Starting from the constructed tour, all agents start to generate neighborhood, assign weights to them, and then select a new neighborhood as the following:

Step 2.1:

- **Selecting a range for generating neighborhood:** Assume a random node *"3"* is selected. Since it is at the first iteration, $s_{33}=5$. Then, three other nodes are selected which are in the range s_{33}. Assume that *2,5,* and *1* are selected.
- **Generating neighborhood:** The first row of *LS* is dedicated to *2-opt* moves, so the first element of the *LS* matrix will be constructed based on the *2-opt* move of *3-2, 5-1 to 3-5, 2-1*. So the new neighbor will be: *1-4-3-5-2-1*. The same task will be done for constructing *LB*, with a restriction of $b_{33}=1$. The second row of *LS* and *LB* will be based on the *3-opt* movement after selecting six nodes, which does not apply to this example since the total number of cities is only five.

Step 2.2: Constructing VS, and VB: A value has to be assigned to every solution in *LS* and *LB* based on Global Memory G_2. For assigning a weight to the first element of *LS*, *LS(1,1)=(1,4,3,5,2),* the following has to be done:

$VS(1,1)=G_2(1,4)+G_2(4,2)+G_2(3,3)+G_2(5,4)+G_2(2,5)$. All other elements of *LS* and *LB* will be given weights in the same manner as described above:

Step 2.3: Discriminative sampling: In this step, a "PDF" function is built for all the elements of *VS* and *VB*. Having the summation of all the elements in *VS*. Now, a CDF function is constructed by sorting the elements of PDF(*VS*) and adding the elements together. Then, "m_s" new solutions will be selected from *LS*, and "m_b" from *LB*.

Step 2.4: The two selected neighborhood solutions will be evaluated by the objective function. If the new solution has a lower objective value than *(1,4,3,2,5),* then it would be selected as the best neighborhood.

Step 3: In this step, all the agents report their best found solution, and the global memory will be updated based on these values.

- **Updating G_2:** Suppose the route *1-4-3-2-5* is selected as the new seed with a cost of 36. And suppose the summation of the seed in the elite list and the current optima are 50. For updating G_2, following values will be added to the corresponding elements:

(1-36/50) will be added to *(1,1), (4,2), (3,3), (2,4)* and *(5,4)*

-**Updating G_1:** If the new seed improves the previous optima, the s_{ij}'s will be decreased by α_c , while b_{ij}'s will be increased. Otherwise, s_{ij}'s will be increased and b_{ij}'s will be decreased. Considering $\alpha_c = 1$, updating 1 will be as the following:

- $s_{11(new)}=s_{11(old)}+1; s_{42(new)}=s_{42(old)}+1; s_{33(new)}=s_{33(old)}+1; s_{24(new)}=s_{24(old)}+1$ $; s_{55(new)}=s_{55(old)}+1;$
- $b_{11(new)}=b_{11(old)}-1; b_{42(new)}=b_{42(old)}-1; b_{33(new)}=b_{33(old)}-1; b_{24(new)}=b_{24(old)}-1;$ $b_{55(new)}=b_{55(old)}-1;$

Step 4: New-born agents: Assume that from previous iterations, one has an elite list of *[(3,2,5,1,4),(2,3,5,1,4)]*. Moreover, one has *[(4,3,1,2,5);(3,2,4,1,5);(3,5,2,1,4)]* as the current seed of all the agents in this iteration. Therefore, matrix "N_e" will be formed as:

$$N_e = \begin{bmatrix} 0 & 0 & 1/5 & 4/5 & 0 \\ 1/5 & 2/5 & 1/5 & 1/5 & 0 \\ 1/5 & 2/5 & 0 & 0 & 0 \\ 1/5 & 0 & 1/5 & 0 & 3/5 \\ 0 & 0 & 2/5 & 0 & 2/5 \end{bmatrix}$$

N_{ij} shows the probability of having node "i" in the "j^{th}" position among the elite list and current local optima. For example, N_{14}=4/5 shows that in four out of five sequences, node 1 is in the 4th position. Now, c_{iter} has to be calculated, $c_{iter} = e^{-(1-\frac{iter}{Maxiter})}$. Assuming a

maximum iteration of 100, if one is at the 10^{th} iteration, $c_{iter}=0.406$. For those elements in the N matrix which have a probability more than 0.406, the corresponding cities will be fixed at their positions. This is true for node one being in the 4^{th} position and node four being in the 5^{th} position. Consequently, the new-born agent has two fixed positions and other positions will be filled randomly: for example, *New-born=(2,3,5,1,4).*

If one is in the last "half" iteration of the algorithm, the assignment problem will be solved for the following matrix:

$$N_c' = \begin{bmatrix} 0 & 0 & 1 & 4 & 0 \\ 1 & 2 & 1 & 1 & 0 \\ 1 & 2 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 3 \\ 0 & 0 & 2 & 0 & 2 \end{bmatrix}$$, and since the objective function is maximization, one will subtract the whole matrix by a large number and then call the assignment problem sub-optimal solver (which can be solved in polynomial time). A new-born agent will be created based on the solution of the assignment problem: *New-born= (3, 2, 5 ,1,4).*

Step 5: If the finishing criterion has not been met (the maximum number of iterations reached), one will go back to Step 2 and all the agents will follow the same procedure.

5.6 Results of TSP-MPS algorithm

The algorithm is implemented using Matlab toolbox on a Pentium 4, 3.00 GHz machine. TSP-MPS has been used to solve the instances of TSP problem in TSP-lib (http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsplib.html), namely: Burma14 (14 cities), Gr21 (21 cities), and Gr96 (96 cities).

The used parameters of the TSP-MPS are shown in Table 5-2. The parameters' values are tuned based on numerous runs of different problem sizes.

Table 5-2. Parameter setting for different problem sizes of TSP

TSP-Instance	K	m	α	n_α	m_b	m_s	P
burma14	4	10	1	2	3	3	6
gr21	8	15	2	3	5	5	9
gr96	25	40	3	8	10	20	28

1. Burma 14: The problem of Burma14 is to find the shortest tour between 14 cities, specified by their latitude and longitude in the data set, with each city being visited exactly once. There are fourteen possible starting points for each closed tour and each tour can be traveled in two different directions. Therefore, the total number of possible solutions will be 13!/2. The data for the burma14 is shown in Appendix I. The length distance (*ij*) of path between any two cities *i* and *j* (latitudes la_i, la_j and longitudes lo_i , lo_j respectively) is given by the following calculation:

$Distance(ij)=[6378.388*acos(0.5*((1+q1)*q2)-((1-q1)*q3)+1),$

where $q1=cos(lo_i-lo_j)$, $q2=cos(lai-laj)$, $q3=cos(la_i+la_j)$ **Equation 5-4**

The TSP-MPS algorithm is executed 25 times for "burma 14." The results are in Table 5-3.

Table 5-3. Results of TSP-MPS on burma14

Problem	Nopt(/25)	ARPD	Av.CPU-time(sec.)
Burma14	23	0.096	4.31

"ARPD" is the average relative percent deviation from the optimal (or best known) value over all 25 runs.

ARPD= $\frac{x - x^*}{x^*} \times 100$, where x is the global optima found by the algorithm and x* is the optimal solution or best known solution reported by TSP-lib.

The algorithm stops when it hit the optimum or the maximum number of iterations is reached. "Nopt" is the number of times the algorithm reaches the optimal value.

2. Gr21: 21-city problem

Data given for gr21 is in the explicit format of the distance matrix, which is different from burma14 (Geographic distance). Data for "gr21" is in Appendix II. Table 5-4 shows the result of TSP-MPS for "gr21".

Table 5-4. Results of TSP-MPS on "Gr21"

Problem	Nopt(/25)	ARPD	Av.CPU-time(sec.)
Gr21	21	0.121	9.31

3. Gr96: Same as burma14, the data of "Gr 96" are geographical longitudes and altitudes. Data for "Gr96" are in Appendix 3. Table 5-5 shows the results of TSP-MPS on "Gr96."

Table 5-5. Results of TSP-MPS on "Gr96"

Problem	Nopt(/25)	ARPD	Av.CPU(sec.)
Gr96	16	0.623	35.7

Table 5-6 gives a comparison of the TSP-MPS algorithm and two other methods that performs excellent on TSP instances: SA (Simulated annealing) and ITS (Iterated tabu search)(Misevičius et al. (2005)).

Table 5-6. Comparison of TSP-MPS with two of the best meta-heuristic methods for TSP

		ARPD			N_{opt}		
	Z_{opt}	SA	ITS	TSP-MPS	SA	ITS	TSP-MPS
Burma14	3323	0	0	0.09	10/10	10/10	22/25
Gr21	2707	0	0	0.121	10/10	10/10	21/25
Gr96	55209	0.59	0.013	0.62	2/10	9/10	16/25

As it can be seen from Table 5-6, SA and ITS perform better than TSP-MPS, except for "Gr96," for which TSP-MPS has advantage on SA, in terms of the number of optima reached.

5.7 Summary

Based on the previously developed combinatorial mode pursuing sampling (Co-MPS) approach, a variation of Co-MPS for Traveling Salesman Problem has been developed in this paper. TSP-Mode Pursuing Sampling (TSP-MPS) can be categorized as a meta-heuristic approach in the sense that it starts from a few samples of the solution space and intelligently moves to the better neighborhoods of the initial samples. TSP-MPS is composed of a number of agents with local memories, which generate neighborhoods around their current optima. At any iteration, every agent uses the global memory to assign weights to the neighborhood solutions. Based on these weights, a selection or inverse CDF sampling is performed inside each agent for finding new solutions. Current global optimal solution is updated by a function of values reported from all agents. On the other hand, generating neighborhoods with *2-opt*-moves and *3-opt*-moves are restricted by two domain controllers "S" and "B". These two controllers control the

domain of new neighborhood solutions based on the past improvements during the search procedure. They dynamically control the exploration and exploitation, although none of them are directly assigned to these tasks. To our knowledge, none of the existing meta-heuristic approaches has used an intelligent domain controller for generating the neighborhoods. Results of TSP-MPS on the benchmark problem from both TSP-lib and reported results in the literature show that the proposed method is a promising meta-heuristic global optimization algorithm for medium size TSP instances.

Chapter 6

Mode Pursuing Sampling Method for Patient Appointment Scheduling

6.1 Introduction

The present chapter is the culmination of a novel simulation-based optimization algorithm for patient appointment scheduling, namely AS-MPS. AS-MPS is a version of CO-MPS previously described in Chapter 4. AS-MPS uses a simulation model of Pre-Admission clinic at HSC hospital in Winnipeg as a black-box function. The optimization algorithm of AS-MPS is composed of several agents. Each agent searches the solution space to find the best schedule based on the values reported form the simulation model. These agents interact with each other and the global memory to move toward the global optima.

Healthcare is one of the largest domestic industries in most developed countries. According to the Canadian Institute for Health Information (Canadian Institute for health information, http://www.cihi.ca), Canadian health expenditures totaled $56 billion in 1999, which accounted for 9.2 percent of the GDP or $1,828 per capita. Similar statistics are provided in Carter (2002) for most of the developed countries.

Healthcare's current problems are discussed in detail in (IOM report (2000, 2001), Carter (2002), Reid et al. (2005)). In IOM report (2001) the following points have been

mentioned as the causes of healthcare problems: Advent of new technologies, Several actors with different objectives, and Baby-booming patients arrival.

Many studies of healthcare optimization are focused on patient-flow inside the hospitals. Most of them studied the ways for reducing the harmful delays/diversions. Blake et al. (2002) mentioned that most of the delays are the result of resources' loadings not being leveled enough. Dealing with smoothing the patient flow or making the capacity leveled is another form of dealing with variability and trying to minimize the unwanted variability. Litvak et al. (2000) and Khandelval et al. (1999) have studied the census variability.

Khandelval et al. (1999) argues that there are two different types of variability in the system: Natural and Artificial. Natural variability can be found in any stochastic system. As an example, natural variability can be seen in arrival time of patients (flow variability), different length of stay for two patients which had the same surgery by the same surgeon (clinical variability) or different operation duration for two surgeons doing a same surgery (professional variability). On the other hand, artificial variability is the result of having a non-optimized system. The causes for artificial variability are the same causes which introduce delay/diversion in patient flow. Resources in different departments or inside each department should be leveled in the sense that they could handle the natural variability. If they are not leveled, it means there is an artificial variability, which will incur a peak load in the census.

In Section two of this chapter, the appointment scheduling problem will be discussed and a summary of literature review will be described. Furthermore, the gaps in the current literature and research motivative are discussed. In Section three, the pre-admission clinic (PAC) at HSC hospital will be introduced and the simulation model of this clinic will be discussed. Section four describes the AS-MPS algorithm in details. The result of AS-MPS algorithm for one-day operation of PAC clinic will be given at the end of Section five. Finally, Section six will conclude this chapter and the advantageous of the AS-MPS algorithm will be discussed.

6.2 Appointment Scheduling

As it is discussed in Carter (2002), hospitals are major subsystems of the healthcare system. As it was described in Section 6.1, smoothing the patient flow inside the hospital needs a major consideration to reduce the artificial variability which can be the appropriate scheduling. One of the major scheduling activities within any hospital is appointment scheduling or outpatient scheduling. This chapter will focus on appointment scheduling.

In Section 6.2.1, Appointment Scheduling (AS) problem will be introduced and the mathematical model for a simple version of AS will be discussed. In Section 6.2.2, a brief summary of literature review will be discussed and the gaps in the literature will be introduced.

6.2.1. Appointment Scheduling Problem

The objective of outpatient scheduling is to find an appointment system for which a particular measure of performance is optimized in a clinical environment. The underlying problem applies to a wide variety of environments, such as general practice patient scheduling, scheduling patients for hemodialysis, radiology scheduling, etc.

Outpatient clinics can be regarded as queuing systems, which represent a unique set of conditions that must be considered when designing AS. The simplest case is when all scheduled patients arrive punctually at their appointment times and a single doctor serves them with stochastic processing times. The formulation becomes more complicated as multiple doctors and multiple services are considered. Presence of unpunctual patients, no-shows, walk-ins, and/or emergencies may intervene to upset the schedule. Furthermore, doctors may be late to start a clinic session or they may be interrupted during the course of the day due to activities not directly related to consultation.

The most related objective function which can incorporate multiple objectives is the cost function. For the simplest case (single-server, single-type patients) with cost objective function, a mathematical formulation can be discussed as the followings:

Assume that the arrival time of patient i is normally distributed with mean A_i and standard deviation of 1. Let T_i be the length of service time, b_i be start of service time, e_i be end of service time. Moreover, assume that $A_1=b_1=0$. Therefore, $b_i = \max(A_i, e_{i-1})$ where $e_i=b_i+t_i$. Consequently, patients' idle time will be equal to $P_i = \max(0, b_i - A_i)$.

On the other hand, physician idle time will be: $M_i = max\ (0, A_i - e_{i-1})$. The objective is to minimize the total cost of both patients' idle time and physician's idle time: In other words; minimizing $E(C) = r \times E(P) + s \times E(M)$, where r is the unit idle cost ($/Unit time) for patients and "s" is for Physician. The constraints for this problem will include the clinic opening hours and the other logical constraints.

In the general form of the outpatient (or appointment) scheduling, the above simple assumptions will be changed to following complex ones:

- Patients waiting time=Waiting time of the patients in the line+ Waiting time between services (another term in Cost objective function)
- Patients have due date for the next stage of their flow through the system. (another term in Cost objective function)
- Each patient go through different services, depending on his/her type.
- Staff are not available all the time (Shifts are assigned to staff)
- Service time of every staff member is dependent upon patients' type, staff skill, and patients' age.
- Presence of "No shows" of the patients with a predefined distribution.
- Considering several resources, such as number of rooms, beds, instruments, etc.
- Presence of "Walk-ins"

In the general form of appointment scheduling, having the mathematical formula for the linear programming or queuing system will be almost impossible to achieve. Therefore, several rules of scheduling have been proposed in the literature and they have been

compared to each other using the simulation model for any specific clinic. Literature review related to this problem and several rules of patient scheduling will be discussed in the next section.

6.2.2. Literature review on Appointment Scheduling

As it was discussed in the previous section, several rules have been pointed out as scheduling rules for appointment scheduling. Most of the studies compare these rules for specific clinics based on the simulation models. Appointment scheduling rules can be categorized with three features as the following:

- Block size (n_i): The number of patients scheduled to the i^{th} block
- Begin-block (n_1), also called the initial block, is the number of patients given an identical appointment time at the start of a session.
- Appointment interval (a_i) is the interval between two successive appointment times, also called "job allowance".

Every combination of above three characteristics makes an appointment rule. In the sequel, the dominant rules found in the literature will be mentioned:

1. Single-block rule assigns all patients to arrive as a block at the beginning of the clinic session. For example, all morning patients are scheduled for 9:00 a.m. and they are seen on a first-come, first-served basis. As one can see, most of the earlier studies praise the advantages of individual appointments, pioneering the shift from single-block to individual-block systems (Lindley (1952), Bailey

(1952)). Single-block systems are still used, mostly in public clinics, probably because they require the least administrative effort. Babes et al. (1991) investigated a public clinic in Algeria that uses a single-block AS.

2. Individual-block/Fixed-interval rule assigns each patient unique appointment times that are equally spaced throughout the clinic session. A number of studies investigate this type of an appointment rule (Klassen et al. (1996), Rohleder et al. (2000)).

3. Individual-block/Fixed-interval rule with an initial block is a combination of the previous rule with an initial group of "n_I" patients ($n_I > 1$) called at the start of the clinic session. The goal is to keep an inventory of patients so that the doctor's risk of staying idle is minimized if the first patient arrives late or fails to show up. Bailey (1952) is the first to suggest an individual-block system with two patients assigned at the beginning of the session and the rest scheduled at intervals equal to the mean consultation time ($n_I = 2,\ n_i = 1,\ a_i = [\mu]$, μ is the mean consultation time.)

4. Multiple-block/Fixed-interval rule is one in which groups of m patients are assigned to each appointment slot with appointment intervals kept constant. Soriano (1966) studies an appointment system where patients are called two-at-a-time with intervals set equal to twice the mean consultation time ($n_i = 2,\ a_i =$

$2[\mu])$. Cox et al. (1985) find that multiple-block rules perform the best in their particular environments.

5. Multiple-block/Fixed interval rule with an initial block is simply a variation of the above system with an initial block $(n_I > m)$. Cox et al. (1985) is one of the studies that investigated this particular type of rule.

6. Variable-block/Fixed-interval rule allows different block sizes during the clinic session, while keeping appointment intervals constant. Liu et al. (1998) and Bosch et al. (1999) investigated this rule in their studies.

7. Individual-block/Variable-interval rule is one in which customers are scheduled individually at varying appointment intervals. Ho et al. (1992) introduced a number of variable-interval rules and test their performance against traditional ones using simulation. They found that, among the rules they tested, increasing appointment intervals toward the latter part of the session improves performance the most. Some recent analytical studies show that, for i.i.d. (identically distributed) service times and uniform waiting costs for all patients, optimal appointment intervals exhibit a common pattern where they initially increase toward the middle of the session and then decrease. This is referred to as the "dome" shape, studied by Denton et al. (2001) . In addition, Bosch et al. (2000) analyzed individual-block/variable-interval rules.

In the majority of the studies, patients are assumed to be homogeneous and they are scheduled on a first-call, first-appointment (FCFA) basis. When there are patient groups (classes) that are known to be distinct in terms of various attributes (e.g., service time characteristics, arrival patterns, costs of waiting, etc.), then this raises the issue whether an AS can be improved by recognizing such differences.

In outpatient scheduling, patient classification can be used for two purposes: to sequence patients at the time of booking; and/or to adjust the appointment intervals based on the distinct service time characteristics of different patient classes. Since the schedule has to be ready in advance and the arriving requests are handled dynamically, the use of patient classification in outpatient settings is somewhat limited. A realistic application requires that the patients are classified into a manageable number of groups and that they are assigned to pre-marked slots when they call for appointments. In the literature, some of the classification schemes used for scheduling purposes include new/return, variability of service times (i.e., low/high-variance patients), and type of procedure. These factors are discussed in studies by Cox et al. (1985) and Bosch et al. (2000). In an application to a radiology department, Walters (1973) investigated the possibility of improving the AS by dividing patients with similar exam times into different sessions. It is found that examination times depend on factors such as patient's age, physical mobility, and type of service. For example, older patients with limited mobility (trolley, wheelchair) require, on average, considerably more time than the younger and walking patients.

As it was discussed above, in order to find the best rule for a general outpatient scheduling, several researchers claimed that there is no specific rule that determines the

optimal or even near optimal solution. In fact, there is no algorithm that can be used for any outpatient clinic. Most of the queuing models studied in the literature dominantly represent single-server, single-phase systems and most studies analyze the environment of a specific clinic. Thus, their findings lack generalized applicability, i.e. Multi-stage, Multiple-server considering all the real-life situations. In this thesis, a meta-heuristic algorithm is proposed to solve this problem. This algorithm uses several AS-rules and incorporates a simulation model which can be implemented in complex real life scenarios of a general outpatient clinic.

In the next Section, the operations and characteristics of Pre-admission clinic of HSC hospital will be discussed. The pre-Admission clinic (PAC) can be classified as a general clinic since it has multiple-server and several patients who go through different routes based on their types. In Section 6.5, the AS-MPS algorithm will be discussed which has been used for finding the optimal scheduling of PAC.

6.3 Pre-admission clinic (PAC)

The Pre-Admission Clinic (PAC) is a clinic that elective surgery patients attend prior to surgery. The purpose of PAC is to assess a patient's medical fitness for surgery and anesthesia. It is also an opportunity for the patient, and their family, to be provided with information about what to expect before and after surgery. All this ensures that patients are well prepared for their operation. Also, this clinic allows the patient to stay at home until the surgery day, instead of being admitted one or two days ahead.

At PAC, patients will be asked about their health and medical history, as well as having the necessary investigations needed before surgery can be carried out. Nurses evaluate and educate the patient, while an anesthetist assesses the patient's medical condition, determines fitness for surgery, and discusses anesthesia options. Old charts are reviewed and discharge planning is confirmed. Other professionals may also need to see the patient, such as a physiotherapist. As it is shown in Figure 6-1, the main steps (physical and informational) which every elective surgery patient should follow are as the followings:

Step 1- Receiving Inputs: Data for the patients comes from physicians' offices to the PAC clinic in two ways, patients Booking card and phone.

Step 2 - Patient Scheduling: In the second step, another record is added to the patient database which is his/her PAC appointment.

Step 3- Clinical Processes: When patients come for PAC appointment, based on his/her type of disease, it could go through different processes. Service time for these processes is related to different factors, such as category and patients' age.

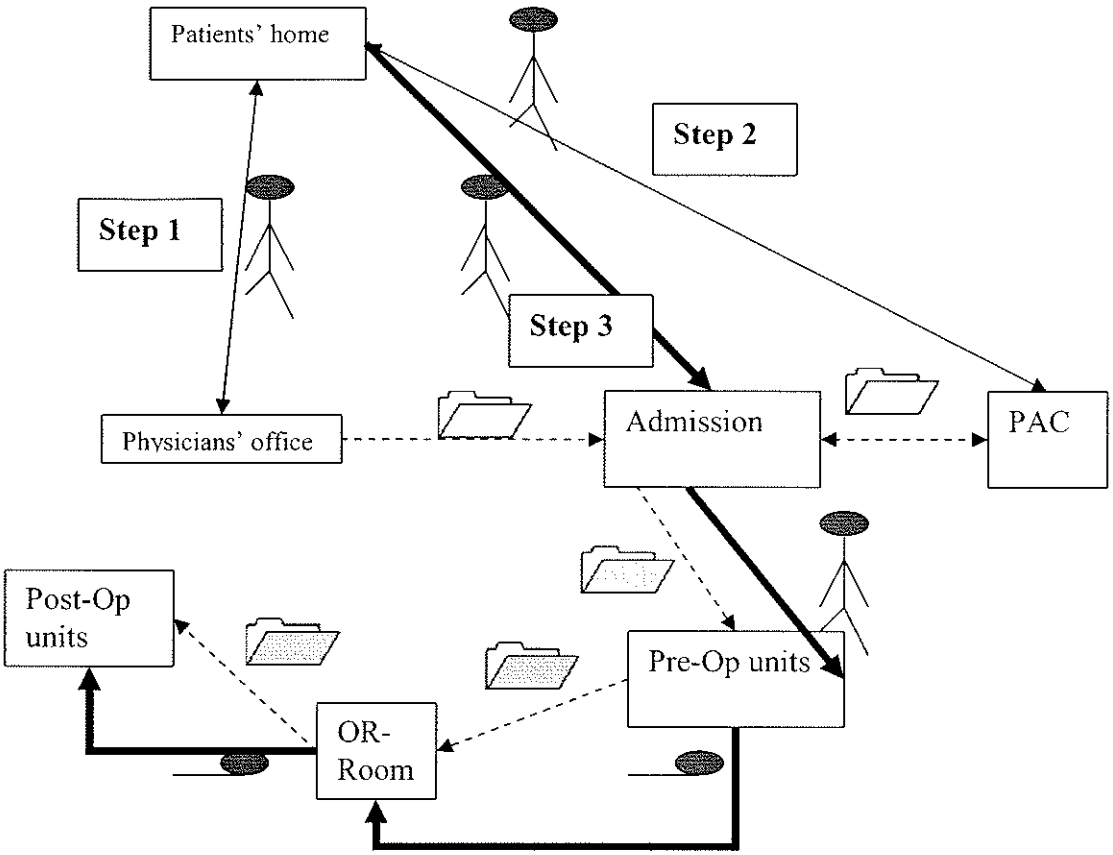


Figure 6-1. Information and physical flow of patients in PAC

In Figure 6-1, bold arrows show the flow of patient on the day of the surgery. Dashed arrows show the flow of information and the folder shows the booking card of the patient. As it is shown in Figure 6-1, step 2 is the step which patient travels from his/her home to PAC department, days before his/her surgery. As it is shown in Figure 6-2, folders are the booking card of the patient, which is being completed during the patient's journey in the hospital. Darker folder shows more complete booking card of the patient. In Section 6.3.1, I will discuss the characteristics of the PAC department.

6.3.1 Characteristics of PAC

Hours of operation of PAC are 9:00am-5:00pm everyday with one hour lunch time from 12:00-1:00 pm, except on Tuesdays and Thursdays the hours are 8:30 am-5:00pm and are allocated to "cardiac" patients. In this study, one regular work day of PAC (9:00 am-5:00pm) will be studied. Staffs of PAC department are as follows:

- Admission desk clerk
- Nurse: 2 full time nurse, 1 part time
- Full time physician (anesthetic)
- Physiotherapist
- Blood Nurse
- Women's nurse (Nurse form the women department)
- Surgeon's nurse

Patients who come to PAC will be seen by a variety of hospital staff, depending on the nature of their surgery. Table 6-1 shows the different categories of patients and the people they must see. Figure 6-2 shows the main steps of PAC with the flow of patients inside PAC.

Table 6-1. Different categories of patients and their required resources of PAC

Patient Type	Nurse		Anaesthetist	Physiotherapist	Cardiac Surgery Nurse
	PAC	Women			
Cardiac Surgery	×		×	×	×
Thoracic	×		×	×	
Vascular	×		×	×	
Women		×	×		
Other	×		×		

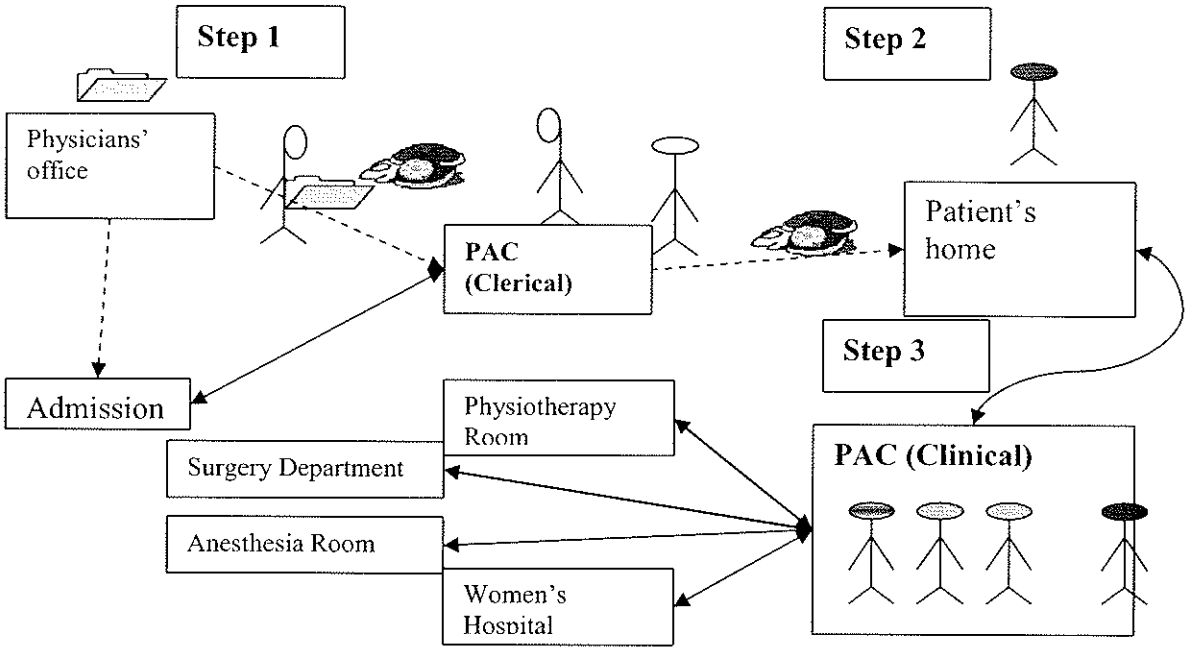


Figure 6-2. Main steps in PAC

Figure 6-2 shows the informational and physical process related to PAC in HSC hospital. Step 1 of this figure shows the booking of a patient in PAC which is done through the phone from the Physicians' office. Step 2 denotes that clerks of PAC informthe patient of his/her appointment. Step 3 shows the physical travel of patients form his/her home to the PAC department which is followed by visiting several staff in the PAC department depending on patients' type. Figure 6-2 also shows how other departments in the hospital are related to the PAC department.

6.3.2 Simulation of PAC department

As it was discussed in Section 6.1, the clinical procedure of PAC has been simulated with Witness 2004. The inputs and outputs of the simulation model will be discussed in the following section. It should be discussed that the distributions of service times of several staff and their parameter values have been approximated based on the data gathered during the visits to the PAC department of HSC hospital in fall 2006.

-Inputs of the simulation model:

- Arrival of patients: Normal distribution with the mea of their scheduled time and standard deviation of 1.
- Patients type: (C: Cardiac Surgery, T: Thoracic, V: Vascular, W: Women, O: Others).
- Service time of staff: Depending on patients' type and staff, it has an "*Erlang*" distribution with parameter "*2*" and the mean of the service which has been examined during the hospital visits.

Table 6-2 Service time distribution matrix for each patient type and the specific clinicians

	Nurse(1)	Doctor(2)	Physio(3)	Blood_N(4)	Surgery_N(5)
C(1)	*Erl(10,2)*	*Erl(12,2)*	*Erl(11,2)*	*Erl(7,2)*	*Erl(8,2)*
T(2)	*Erl(6,2)*	*Erl(10,2)*	*Erl(11,2)*	-	-
V(3)	*Erl(6,2)*	*Erl(10,2)*	*Erl(11,2)*	-	-
W(4)	*Erl(6,2)*	*Erl(7,2)*	-	-	-
O(5)	*Erl(5,2)*	*Erl(6,2)*	-	-	-

Table 6-2 shows the service time distribution matrix.

Table 6-2 shows the service time distribution matrix.

- Session starts at 9am and finishes at 12 p.m. Then, there is an hour break for the lunch and the staff will work from 1pm-5pm.
- Assigning shift to staff
  - Nurse (1): There are 2 full time nurses, one part time.
  - Doctor (2): There is one full time doctor.
  - Physiotherapist (3): There is one full time physiotherapist.
  - Blood Nurse (4): There is one full time blood nurse.
  - Surgery Nurse (5): There is one full time surgery nurse.

It should be noted that having a physiotherapist, blood nurse and surgery nurse always available is not realistic, but for this study I assume they are available. In reality there are specific days when these clinicians dedicate their time for PAC.

- Hourly cost of patients' waiting time in the line: I assume a 1$ cost for every patient waiting in the line. Also, 2$ cost for patients waiting in the room for the next clinician.
- Cost of each staff's idle time per minute: Table 6-3 shows the cost of service for each clinician per minute.

Table 6-3. Cost of clinicians per minute of service

	Nurse(1)	Doctor(2)	Physio (3)	Blood Nurse	Surgery Nurse
Cost of service per minute	4$	8$	7$	4$	5$

Table 6-4. Number of patients and their types to be scheduled for one day of PAC department

Patients' type	Number of patients to be scheduled
C(1)	3
T(2)	1
V(3)	2
W(4)	2
O(5)	2

- Outputs of the simulation model:

- Patients waiting time in the line
- Patients idle time between the services (waiting for next clinicians)
- Idle time of each staff

It should be noted that the objective function is the summation of above idle times multiplied by the appropriate cost which is defined as input.

- Idle time of all the patients: After every run of the simulation, all the patients' idle time which is needed for calculating the objective function is reported by the simulation model. The patient with the highest idle time will be used by the optimization function as the bottleneck and will be changed its position in the sequence for the neighborhood generation of the current optima of every agent. Other patients' idle time will be used in updating the global memory G_2.

6.4 AS-MPS algorithm

AS-MPS algorithm is a meta-heuristic simulation-based optimization algorithm. It is a version of Co-MPS algorithm which has been previously discussed in Chapter 4. The main scheme of the algorithm is depicted in Figure 6-3.

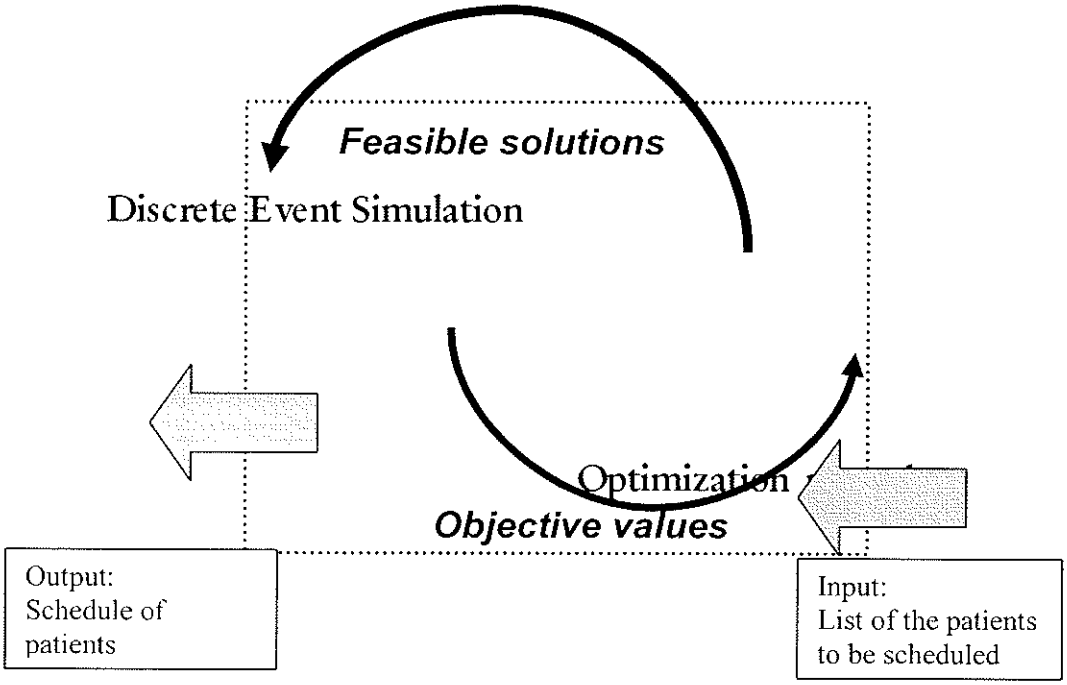


Figure 6-3. Main structure of AS-MPS algorithm

The flowchart of the optimization algorithm is the same as one which was discussed in Chapter 4 for Co-MPS algorithm. The objective value will be evaluated by the simulation model in AS-MPS. Main differences of AS-MPS and Co-MPS include the different classes of agents, different neighborhood generating mechanism (Section 6.5.1), and different ways of updating global memory which will be discussed in Section 6.5.2.

As described in Section 6.4, the simulation part of the algorithm has been done in WITNESS environment. The optimization procedure has been implemented in MATLAB. Here, the optimization procedure of AS-MPS will be discussed. AS-MPS is composed of

"K" agents who start from one seed as an initial schedule of patients and then move to better schedules during the optimization run-time.

Agents are categorized in "C_a" classes, where "C_a" is the parameter of AS-MPS. Each class of agents follows one specific appointment scheduling rule. Figure 6-4 shows a typical flowchart of steps of any agent at any iteration.

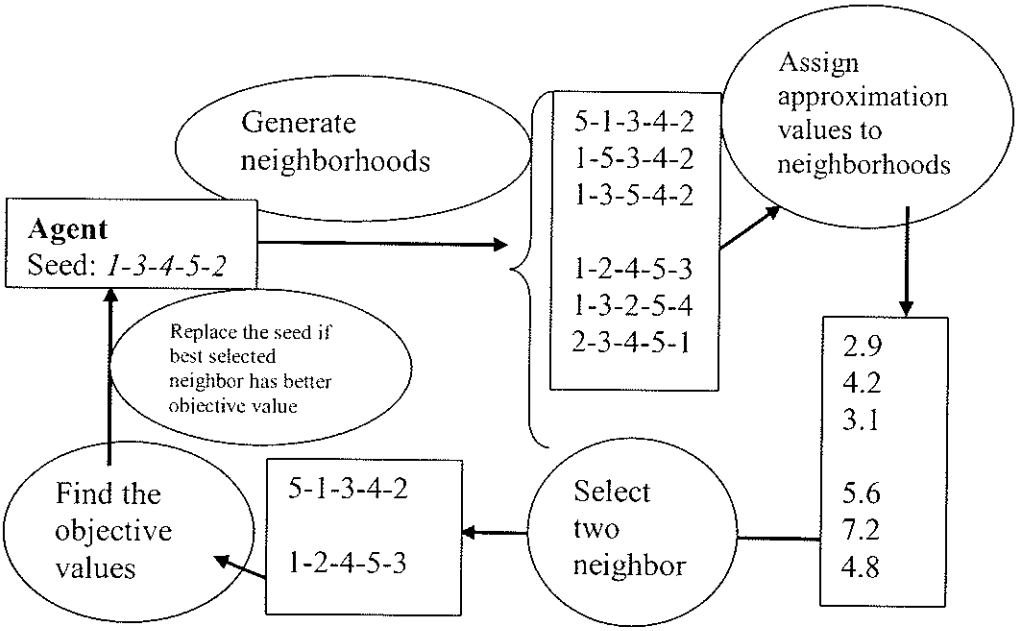


Figure 6-4. Typical flowchart of tasks of an agent in AS-MPS algorithm

6.4.1 Tasks of an agent

Every agent in the AS-MPS follows a specific appointment scheduling rule. Therefore, any agent has a specific structure for the block-size of the schedule and a timeline which shows the allowable interval of the schedule. In the implementation of AS-MPS for PAC, three classes of agents have been used:

1. Single block, 2 initials, fixed interval: every 30 minutes. (First four agents)

So the structure of the schedule is 1*17 matrixes, with the first element can have up to two patients. The reason of having maximum of two initial patients is because of the scheduling rule which is followed by this class of agents. An example of structure and timeline of first four agents is shown below:

- Structure : [1 2] [3] [4] [ ] [5] [ ] [ ] [ ] [6] [7] [8] [9] [ ] [10] [ ] [ ] [ ]
- Time line: 0 30 60 90 120 150 180 210 240 270 300 330 360 390 420 450 480

2. Multiple block-fixed interval: 2 patients, every hour (agents 4 to 8).

The structure of agents 4 to agent 8 is a 1*9 matrix, which every element can get up to two patients. An example of the mentioned structure and timeline is as follows:

- Structure: [1 2] [3 4] [5 6] [ ] [7 8] [9 10] [ ] [ ] [ ]
- Time line: 0 60 90 120 180 240 300 360 420 480

3. Multiple block-variable intervals: every 15 minutes, 4 patients (for all the new-born agents). The structure of all the new-born agents is a 1*33 matrix, which every element can get up to four patients. An example of the structure and timeline of a new-born agent is as follows:

- Structure: [1] [2 3] [4 5 6] [ ] [ ] [7 8 9 10] ....[ ]
- Time line:0 15 30 45 60 75 ...480

It should be noted that the simulation won't allow any patient during the lunch time which is between the "180-240" period of every day schedule.

As it is depicted in Figure 6-4, at any iteration, all the agents start to generate neighborhood around their current seed, which is a current optimum of a specific agent. Neighborhood generation is done by "insert" and "exchange" moves of a selected patient. As it was described in Section 6.4.2, one of the outputs of the simulation model is the idle time for every patient. Patient with the highest idle time will be used in generating neighborhood, other patients idle time will be used in updating G_2. In fact, patient *"j"* with the highest idle time is chosen to be exchanged or inserted in previous positions of the seed for generating new schedules.

It should be noted that, generating neighborhood solutions is restricted by (s_i,b_i) which shows the maximum distance which the chosen patient *"j"* can be exchanged or inserted with other patients. In fact, (s_i,b_i) restricts the neighborhood generation. (s_i,b_i) are denoted by the global memory G_1, which shows the degree of freedom with which any patients can be "inserted" or "exchanged" for generating new-neighborhood. At any iteration, two neighborhood *LS* and *LB* are constructed based on two controllers *S and B*, respectively.

LS and *LB* are *2*m* double matrixes in which every element has the same structure of the agent' seed. If *LS* and *LB* are being constructed by the agent of a specific class, the timeline and structure of that class will be inherited by *LS* and *LB*. The first rows of *LS* and LB are dedicated to insert moves for generating neighborhood from the current seed of the agent, and the second row is for exchange moves.

Here, generating neighborhoods will be illustrated by an example:

Assuming that for the current iteration, Agent 1 has the following seed:

Agent1, Seed:{ [1 2] [3] [4] [ ] [5] [ ] [ ] [ ] [6] [7] [8] [9] [ ] [10] [ ] [ ] [ ]}.

The timeline of agent 1 is: [0 30 60 90 120 150 180 210 240 270 300 330 360 390 420 450 480].

The timeline is every half hour, which means this agent will schedule patients every half hour, with two patients at the beginning of the clinic session.

Figure 6-5 shows the data for G_1 in the current iteration of the ASMPS algorithm.

Patient\Time	0	15	30	...	120	...	480
1	(20,30)	(23,31)	(25,21)	...	(15,21)	...	(32,11)
2	(17,15)	(19,22)	(30,19)	...	(17,25)	...	(18,32)
3	(20,31)	(12,14)	(18,31)	...	(19,11)	...	(17,20)
4	(18,32)	(12,14)	(11,10)	...	(19,22)	...	(23,29)
5	(12,18)	(19,29)	(14,17)	...	(6,7)	...	(19,5)
6	(8,12)	(12,30)	(23,12)	...	(12,20)	...	(19,23)
7	(12,11)	(19,20)	(13,16)	...	(13,4)	...	(10,20)
8	(13,18)	(18,10)	(13,19)	...	(13,12)	...	(19,18)
9	(15,18)	(19,17)	(20,21)	...	(10,11)	...	(20,23)
10	(19,29)	(30,31)	(12,20)	...	(20,23)	...	(22,31)

Figure 6-5. Data for G_1 in ASMPS algorithm

As it can be seen for every patient, there is vector of length 33. This is due to the fact that the resolution of scheduling in our study is 15 minutes. In fact the sequencing of patients is done based on the 15 minutes resolution.

Assuming that patient 5 which is in the "120" position has been reported as the highest idle time patient from the simulation model. This means that patient "5" which is scheduled at time 11 am, is the bottleneck in the schedule of agent 1.Now, for generating new neighborhood of agent 1, patient "5" will be inserted or exchanged with previous patients in the range which is allowable by the *(5,120)* element in G_1. As can be seen in the matrix for G_1, element *(5,120)= (6,7)*. The first value in every element of G_1 is controlled by the behavior of controller *S* and will be used for generating neighborhood from the seed of agent 1 in *LS*. The second value of G_2 is controlled by controller *B* and will be used to generate neighborhood in *LB*.

In the above example, the first value of element *(5,120)* is 6. So, the neighborhoods which are generated in *LS* can have a maximum of "6" distance form the current seed of agent 1. By "6" distance, I mean 6*15 minutes form its current position, which is at the 120 minute position.

The first row of *LS* is for insert moves, so one of the neighborhoods of agent 1's seed in the first row of *LS* could be:

{[1 2] [5] [3] [4] [ ] [ ] [ ] [ ] [6] [7] [8] [9] [ ] [10] [ ] [ ] [ ]}

As it can be seen from the new neighbor of the agent 1's seed, the patient "5" is inserted in the second place which is 90 minutes before the 120 minute position. (i.e. 6*15 minutes movement).

The same procedure of inserting patient "5" within the maximum distance of 5 will be repeated *"m"* times. Also, the second row of LS is dedicated to the neighborhood constructed from the exchange moves of patient "5" within the maximum distance of 5. One of these exchange neighborhoods can be:

{[1 2] [3] [5] [ ] [4] [ ] [ ] [ ] [6] [7] [8] [9] [ ] [10] [ ] [ ] [ ]}

As it can be seen from the new neighbor of agent 1's seed in LS, patient "5" is moved 60 minutes before its current position, which is allowed by 6*15 minutes reported from the first value of the *(5,120)* element of G_1.

"m_a" neighbors will be created based on the exchange moves in the second row of LS. The same procedure which was described for *LS*, will be repeated for *LB*, based on the value "7" which is the second element of (5, 120) from the patients "5" in the 120 minute position in G_1.

After generating neighborhoods around their current seeds, every agent assigns an approximation value to all the neighborhood solutions based on the global memory (G_2).

Assume that G_2 has the following structure in the current iteration of the algorithm:

Patient\Time	0	15	30	...	120	...	480
1	4.1	5.6	8.1	...	1.4	...	3.2
2	4.5	3.7	8.1	...	9.1	...	5.6
3	9.2	9.4	5.4	...	7.5	...	6.5
4	8.4	7.5	9.5	...	4.3	...	8.5
5	7.4	3.5	6.4	...	8.5	...	7.5
6	7.5	6.5	4.3	...	6.2	...	4.3
7	7.3	4.6	7.1	...	9.5	...	5.7
8	6.5	5.4	7.5	...	4.3	...	8.4
9	6.5	6.8	4.3	...	7.5	...	5.6
10	3.4	4.3	2.5	...	7.5	...	5.4

Figure 6-6. Data for G_2 in ASMPS algorithm

If one of the neighborhoods of the seed of agent 1 in the first row of *LS*, is as the following:

{[1 2] [5] [3] [4] [ ] [ ] [ ] [ ] [6] [7] [8] [9] [ ] [10] [ ] [ ] [ ]}

With the assumed timeline of this class:

[0 30 60 90 120 150 180 210 240 270 300 330 360 390 420 450 480]

The following calculation will assign an approximation value to this neighbor based on G_2:

$\{G_2(1,0)+G_2(1,15)\}+\{G_2(2,0)+G_2(2,15)\}+\{G_2(5,30)+G_2(5,45)+G_2(5,15)\}+\{G_2(3,60)+G_2(3,45)+G_2(3,75)\}+\{G_2(4,90)+G_2(4,105)+G_2(4,75)\}+\{G_2(6,240)+G_2(6,255)+G_2(6,225)\}+\{G_2(7,270)+G_2(7,285)+G_2(7,255)\}+\{G_2(8,300)+G_2(8,315)+G_2(8,285)\}+\{G_2(9,330)+G_2(9,345)+G_2(9,315)\}+\{G_2(10,390)+G_2(10,405)+G_2(10,375)\}=$

{4.1+5.6}+{4.5+3.7}+{6.4+...+3.5}+...

As the calculation shows, for every patient at time *"t"* in the neighborhood matrix of *LS* or LB, the weight which is assigned is composed of three elements. The first element is the related (patient, scheduled time) in G_2 matrix. The second and third element are the scheduled time less and plus 15 minutes in G_2 matrix. The reason that these two elements are being added is because of the time resolution of 15 minutes and the fact that different agents have different time line, but G_2 is the global memory for all the agents.

Adding two neighbor elements of G_2 in addition to the related element has not been used in Co-MPS or TSP-MPS, since they were sequencing problems without a timeline.

After assigning an approximation value to all their neighbors, every agent performs the selection to choose *"m_s"* new-solution from *LS* and *"m_b"* new solutions from *LB*. The higher the approximation weight, the more chances the neighbors will be selected. The selection procedure is the same as Co-MPS algorithm which was discussed in Chapter 4.

6.4.2 Updating the global memory

Global memory of AS-MPS algorithm consists of two matrices: G_1 and G_2. Both of them are *n- by-* t_a, where *n* denotes the number of patients and t_a is the time resolution. t_a was fixed to 15 minutes for implementation of AS-MPS for PAC.

Every element of G_1 consists of two value (s_i, b_i), where each of them denotes the restriction for generating neighborhood. The value of (s_i, b_i) is changed based on the DSM (Double sphere method) which has also been used in Co-MPS algorithm.

- *First component of global memory: G_1*

In each iteration, If the new selected neighbor (σ^*) improves the current optima of any agent, (s_{ij}, b_{ij}) for all patients "i" in position "j" in the new selected neighbor will be updated. "s_{ij}" which is controlled by controller S will be increased in the case of improvement, to allow for more exploration. On the other hand, "b_{ij}" for all patients "i" in the j^{th} position will be decreased for allowing more exploitation. Hence, if there are consecutive improvements reported from the simulation model which has patient "i" at position "j", generating neighbors in LB will have a small degree of freedom around patient "i" at position "j". In this case, the neighborhood search will have the same task as a local search.

In addition to the increase for patients "i" at position "j" in the sequence of new selected neighbor, close positions around patients "i" will be also updated. Following formulas show the update procedure for G_1, in the case of improvement:

$$s_{i,j}^{new} = \begin{cases} s_{i,j^*}^{old} + \alpha_a & for (i,j^*) \in \sigma^* \\ s_{i,j}^{old} + (\alpha_a) + |(j - j^*)| & for\ rest\ of\ j's, till\ j - j^* = T \end{cases} \quad \textbf{Equation 6-1}$$

$$b_{i,j}^{new} = \begin{cases} b_{i,j^*}^{old} - \alpha_a & for (i,j^*) \in \sigma^* \\ b_{i,j}^{old} - (\alpha_a) + |(j - j^*)| & for\ rest\ of\ j's, till\ j - j^* = T \end{cases} \quad \textbf{Equation 6-2}$$

"α_a" is the parameter of the AS-MPS algorithm which denotes the degree of aggressiveness in G_1. "T" is another parameter of AS-MPS algorithm which shows the distance which the neighborhood will be affected by G_1.

If any agent reports n_α iterations without improvement, G_1 will be updated as follows:

$$s_{i,j}^{new} = \begin{cases} s_{i,j^*}^{old} - \alpha_a & for (i,j^*) \in \sigma^* \\ s_{i,j}^{old} - (\alpha_a) + |(j - j^*)| & for\ rest\ of\ j's, till\ j - j^* = T \end{cases} \quad \textbf{Equation 6-3}$$

$$b_{i,j}^{new} = \begin{cases} b_{i,j^*}^{old} - \alpha_a & for (i,j^*) \in \sigma^* \\ b_{i,j}^{old} - (\alpha_a) + |(j - j^*)| & for\ rest\ of\ j's, till\ j - j^* = T \end{cases} \quad \textbf{Equation 6-4}$$

In fact, controller "S" will exploit around the current solution which has not been improved after n_α iterations, while controller "B" will explore the solution space.

Here the updating procedure of G_1 will be described by an example:

Assume that the new seed of agent 1 (σ^*) which improved the objective function of this agent is as the following:

{[1 2] [3] [5] [ ] [4] [ ] [ ] [ ] [6] [7] [8] [9] [ ] [10] [ ] [ ] [ ]}

Since agent 1 belongs to the first class of agents, its timeline would be:

[0 30 60 90 120 150 180 210 240 270 300 330 360 390 420 450 480]

Assume that α_a=1 and T=1. As it was described in Equation 6-1, s_{ij}'s of the entire patients "i" in position "j" which is in the σ^* will be changed as the followings:

- $\{s_{1,0}=s_{1,0}+1,\ s_{2,0}=s_{2,0}+1, s_{3,30}=s_{3,30}+1,\ s_{5,60}=s_{5,60}+1,\ s_{4,120}=s_{4,120}+1, s_{6,240}=s_{6,240}+1,$ $s_{7,270}=s_{7,270}+1,\ s_{8,300}=s_{8,300}+1, s_{9,330}=s_{9,330}+1,\ s_{10,390}=s_{10,390}+1\}$
- $\{b_{1,0}=b_{1,0}-1,\ b_{2,0}=b_{2,0}-1,\ b_{3,30}=b_{3,30}-1,\ b_{5,60}=b_{5,60}-1,\ b_{4,120}=b_{4,120}-1,\ b_{6,240}=b_{6,240}-1,$ $b_{7,270}=b_{7,270}-1,\ b_{8,300}=b_{8,300}-1,\ b_{9,330}=b_{9,330}-1,\ b_{10,390}=b_{10,390}-1\}$

Also, patient "i" which is in other positions will be changed as the followings:

- $\{s_{1,15}=s_{1,15}+1+1; s_{2,15}=s_{2,15}+1+1; s_{3,45}=s_{3,45}+1+1, s_{3,15}=s_{3,15}+1+1; s_{5,75}=s_{5,75}+1+1,$ $s_{5,45}=s_{5,45}+1+1; s_{4,135}=s_{4,135}+1+1, s_{4,105}=s_{4,105}+1+1; s_{6,255}=s_{6,255}+1+1, s_{6,225}=s_{6,225}+1+1;$ $s_{7,285}=s_{7,285}+1+1, s_{7,255}=s_{7,255}+1+1; s_{8,315}=s_{8,315}+1+1,$ $s_{8,285}=s_{8,285}+1+1; s_{9,315}=s_{9,315}+1+1, s_{9,285}=s_{9,285}+1+1; s_{9,345}=s_{9,345}+1+1, s_{9,315}=s_{9,315}+1+1; s_{10,405}=s_{10,405}+1+1, s_{10,375}=s_{10,375}+1+1\}$
- $\{b_{1,15}=b_{1,15}-1+1; b_{2,15}=b_{2,15}-1+1; b_{3,45}=b_{3,45}-1+1, b_{3,15}=b_{3,15}-1+1; b_{5,75}=b_{5,75}-1+1,$ $b_{5,45}=b_{5,45}-1+1; b_{4,135}=b_{4,135}-1+1, b_{4,10}=b_{4,105}-1+1; b_{6,255}=b_{6,255}-1+1, b_{6,225}=b_{6,225}-1+1;$ $b_{7,285}=b_{7,285}-1+1, b_{7,255}=b_{7,255}-1+1; b_{8,315}=b_{8,315}-1+1, b_{8,285}=b_{8,285}-1+1;$ $b_{9,285}=b_{9,285}; b_{9,345}=b_{9,345}+1+1, b_{9,315}=b_{9,315}+1+1; b_{10,405}=b_{10,405}+1+1, b_{10,375}=b_{10,375}+1+1\}$
- *Second component of global memory: G_2*

G_2 is another part of global memory which consists of $n*t$ elements, same as G_1. Every element *(i,j)* of G_1 denotes the approximation value of job "i" being in the time slot "j". The value of elements on G_1 is being updated based on the relative value of the current

optima of every agent and the summation objective values of the elite list. Elite list consists of the *"P"* best solutions *(e_i, i=1,..P)*. (*"P"* is the parameter of the AS-MPS algorithm).

As it was discussed in section 6.4.1.1, the simulation model will return all the patients' idle time. For updating G_2, the idle time of all the patients in the current seed of the agent will be used. Suppose O_{ij} denotes the elements of the G_2 matrix in row *"i"* and column *"j"*, after each iteration if the objective function of the current seed of the agent *"u"*(σ_u^*) does not improve the global optima, the value of $f_{iter}^{u,i}$ which shows the updating value for agent *"u"* at iteration "*iter*" and for patient *"i"*. Then $f_{iter}^{u,i}$ will be added to the O^{old}_{ij} which result in O^{new}_{ij}:

$$f_{iter}^{u,i}=(1-\frac{f(\sigma_u^*)}{\sum_{p=1}^{P} f(e_p)})(1-\frac{f(i)}{f(\sigma_u^*)}) \ , f(i)=Idle\ time\ of\ patient\ "i";O^{new}_{i,\sigma_u^*(i)}=O^{old}_{i,\sigma_u^*(i)}+f_{iter}^{u,i} \quad \textbf{Equation 6-5}$$

If the objective value of σ_k^* improves the global optima, the following value of "F_{iter}" will be calculated and then it will be added to the corresponding element in G2.

$$F_{iter}=(1-(\frac{f(\sigma_u^*)}{\sum_{p=1}^{P} f(e_p)})^2)(1-(\frac{f(i)}{f(\sigma_u^*)})^2) \ , O^{new}_{i,\sigma_u^*(i)}=O^{old}_{i,\sigma_u^*(i)}+f_{iter}^{u,i}+F_{iter} \quad \textbf{Equation 6-6}$$

6.4.3 New-born agents

In every iteration of AS-MPS, a new-born agent is being generated based on the information received from all the agents. The way of generating a new-born agent is the same as the Co-MPS algorithm, with considering the fact that the time-line of every new-born agent is the same as the time-line of the global memory. For this reason, constructing "N_a" matrix is different from the Co-MPS. Here, the difference will be discussed by an example:

Assume, one of the elite list solutions is as the followings:

Structure: {[1 2] [3 4] [5 6] [ ] [7 8] [9 10] [ ] [ ] [ ]}

Time line: 0 60 90 120 180 240 300 360 420 480

Figure 6-7 shows the structure of the N_a matrix.

	0	15	30	...	120	...	480
1	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0

Figure 6-7. Structure of N_a matrix for ASMPS algorithm

For the above example, following changes will be done for "N_a" matrix:

2. $\{N_a(1,15)\},\{N_a(2,15)\},\{N_a(3,45),N_a(3,75)\},\{N_a(4,45),N_a(4,75)\},\{N_a(5,75),N_a(5,105)\},\{N_a(6,105),N_a(6,75)\},\{N_a(7,165),N_a(7,195)\},\{N_a(9,255),N_a(9,225)\},\{N_a(10,255),N_a(10,225)\}$ will be added by one.

It should be noted that second type of new-born agent generation method in Co-MPS won't be used in the AS-MPS algorithm, since several classes of agents are being used in AS-MS algorithm compared to one type of agents in Co-MPS.

6.5 Results of AS-MPS algorithm for PAC department

As it was discussed in Section 6.4, at any iteration of the AS-MPS algorithm, one ".bat" file is generated by the optimization algorithm which calls the simulation model. Also, another ".txt" file is generated for the input to the simulation. After the simulation model is being performed, the output will be written in another ".txt" file. Next, the optimization algorithm will read that text file for the next iteration. Figure 6-8 shows the relation of optimization procedure, simulation procedure, and the files as the interface between them.

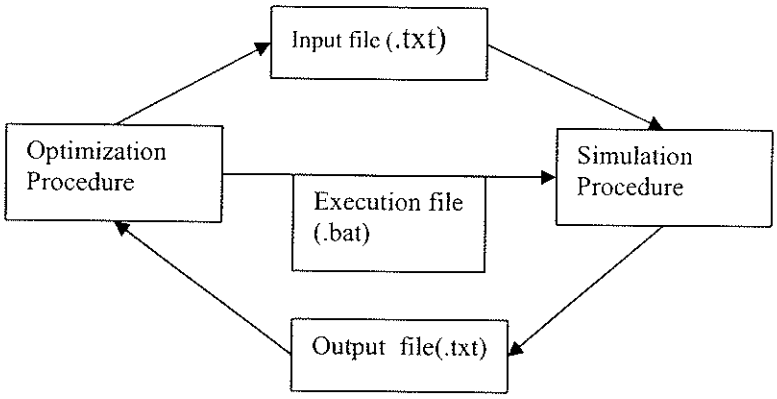


Figure 6-8. Relationship of files, optimization procedure and simulation procedure

The hypothetical data which have been used in this implementation are given in Section 6.3.2. The input text file which has been used in the implementation is shown in Table 6-5. Table 6-6 shows a typical output file from the simulation model.

Table 6-5. Input text file for the implementation

Patient's name	Scheduled arrival time	Patient's type	Flag for the last patient
Patient 1	0	3	e=0
Patient 2	0	3	e=0
Patient 3	30	1	e=0
Patient 4	60	4	e=0
Patient 5	120	1	e=0
Patient 6	240	5	e=0
Patient 7	270	1	e=0
Patient 8	300	5	e=0
Patient 9	330	4	e=0
Patient 10	390	2	e=1

Flag for the last patient in Table 6-5 send the information to simulation model so that it will start generating the output after the last patient leave the model.

As it was discussed in Section 6.5.1, the AS-MPS algorithm has been run with 8 initial agents, first four from class 1 and the second from class 2. All the new-born agents follow the class three's structure. The initial parameters of AS-MPS set for all the 10 runs are: (K, m, α, n_α, $m_{b,}$, m_s, P,T, Maxiter)=(8, 10, 1, 2, 3,3,10,2,8). Table 6-7 shows the result of the best run for AS-MPS.

Table 6-6. Output text file for the implementation

Summation of idle time of the patients in waiting line.	3592
Summation of idle time of patients between services	2898
Summation of idle time of nurses	485
Doctor's idle time (2)	212
Physio idle time (3)	103
Blood nurse idle time(4)	149
Surgery-nurse idle time	81
Patient 1 idle time	260
...	...
Patient 10 idle time	206
Doctor's idle time (2)	212

Table 6-7. Results of AS-MPS algorithm for one-day operation of PAC (Cost)

	Iter 1	Iter 2	Iter 3	Iter 4	Iter 5	Iter 6	Iter 7	Iter 8
A1,c1	14728	12718	12710	11989	11785	10561	10110	9560
A2,c1	14955	12845	12843	12844	11707	11693	10720	10484
A3,c1	13846	11973	10757	10406	9565	9564	9560	9532
A4,c1	14257	11972	11876	11742	11732	11622	11620	11322
A5,c2	53205	46088	46088	24425	17280	12841	11620	10401
A6,c2	51900	48905	46086	44764	20760	16225	16220	16125
A7,c2	52405	46083	20762	20757	12845	12832	11860	10641
A8,c2	53167	42106	42104	41375	33108	33101	29860	16703
A9,c3	0	52204	40102	35765	30302	26122	9860	9266
A10,c3	0	0	38901	11707	9911	9793	9640	9123
A11,c3	0	0	0	22215	22110	11120	9860	9320
A12,c3	0	0	0	0	11720	9717	9706	9701

Each row shows the current optima or seed of every agent and every column shows the results of an iteration. As it can be seen from the table, the algorithm starts with eight agents and at any iteration a new-born agent is added to the current number of agents. The global optima found by the AS-MPS algorithm is 9123 which means 9123 $. This

idle time. It should be noted that a maximum of twelve agents was allowed to be generated, because of the computation cost. The optimal solution which has been found by agent 10 is as follows:

{[7 3] [ ] [ ] [1] [ ] [ ] [ ] [10] [ ] [2] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [4 8] [ ] [ ] [ ] [6] [ ] [ ] [ ] [5] [ ] [ ] [ ] [9] [ ] [ ] [ ]}. With the time line of all class 3 agents:

{0,15,30, 45, 60, 75, 90, 105, 120, 135, 150, 165, 180, 195, 210, 225, 240, 255, 270, 285, 300, 315, 330, 345, 360, 375, 390, 405, 420, 435, 450, 465, 480}.

As it can be seen from the optimal solution, it does not follow a specific rule of scheduling, since it is a new-born agent. In fact, it is a combination of two other rules in the way that makes the solution near-optimal for PAC. The optimal schedule which has been generated by agent 10 is summarized in Table 6-8.

It should be noted that the results are for a regular day, without considering special days for cardiac surgery. The input data for patients is based on the assumption and it is not from the PAC department. In future studies, the real data can be gathered from the PAC department for making the model close to the reality.

Table 6-8. Optimal result of AS-MPS algorithm for one day simulation of PAC department (Timetable)

Patients' name	Scheduled time
Patient 1	9:45 am
Patient 2	11:15 am
Patient 3	9:00 am
Patient 4	1:30 pm
Patient 5	3:30 pm
Patient 6	2:30 pm
Patient 7	9:00 am
Patient 8	1:30 pm
Patient 9	4:15 pm
Patient 10	10:45 am

Table 6-9 shows the result of 10 runs of the AS-MPS algorithm for the PAC-department at HSC hospital in Winnipeg.

Table 6-9. Results of AS-MPS algorithm for one day of PAC department (Cost)

Run No.	Objective value	Number of function eval.	CPU-time(second)
1	9354	516	612.3
2	9123	516	628.8
3	9710	516	640.5
4	9650	516	612.2
5	1012	516	640.7
6	9614	516	610.1
7	9196	516	619.6
8	9218	516	625.3
9	9808	516	618.2
10	9621	516	614.9

6.6 Summary

A meta-heuristic simulation-based optimization algorithm has been proposed in this report which has been implemented for Pre-admission department at HSC hospital in Winnipeg. The proposed algorithm composed of several agents which interact with each other for finding the global optima of patients' schedule. In AS-MPS algorithm, every agent belongs to a class. Each class of agents follows one specified appointment scheduling rule. Also, every selected sequence of patients by agents is being evaluated by the simulation model which is used as a black-box function. In this sense, the AS-MPS algorithm can be used for any outpatient clinic scheduling problem, without dependency on any complex assumption.

The new-born agents of the AS-MPS algorithm are being constructed based on the interaction of previous agents and they have a free structure in the sense that no specific rule is used for their schedule. Hence, they incorporate the combination of other rules for constructing their schedule.

Results of AS-MPS algorithm have been reported for one-day operation of Pre-admission clinic at HSC centre in Winnipeg. Future research will be focused on the schedule of one-week and the comparison between the current schedule used in PAC and the output of the AS-MPS algorithm.

Chapter 7

Conclusions

A major challenge in modern optimization is to develop methodologies for problems with (often expensive) black-box objective functions. In mechanical design optimization problems, one of the main challenges is to develop methodologies that minimize the number of calls to expensive simulation programs. In combinatorial optimization problems and specially production scheduling, search procedures which intelligently move toward the interesting solutions have been studied recently due to the huge solution space of these problems. In the real situations of any complex scheduling system, having an explicit objective function is almost impossible, so optimization methodologies that are suitable for simulation models are preferable for practical situations. In healthcare scheduling problems, the main challenge is to develop algorithms which can be used in any complex system. Discrete-event simulation can be used to simulate these types of complex systems, and it is of great interest to develop an optimization methodology that uses the simulation as an objective function and optimizes the performance of the system.

This research aimed at developing practical optimization algorithms and their software tools to help design and system engineers solve their related optimization algorithms. Four separate algorithms have been developed with a common origin from the MPS (Mode Pursuing Sampling) proposed by Wang et al. (2004a). MPS is basically an optimization concept which tries to evolve the approximation of the objective function

around its mode. MPS can be categorized as a direct, non-gradient, and simulation-based optimization concept.

7.1 Contributions

The main contributions of this thesis can be categorized from both theoretical and practical perspectives.

1.1.1 Theoretical contributions

1. Generalization of the Mode Pursuing Sampling (MPS) concept into the discrete domain and combinatorial optimization. The core of the MPS concept is the integration of metamodeling and discriminative sampling. MPS sample points in the solution space and fits an approximation curve to the sample points. Then, at any iteration, it discriminatively selects points based on the goodness of approximated points and therefore samples more points on the mode of the function and less in other areas. The accuracy of approximation progressively improves around those interesting regions. In their original work, Wang et al. (2004a) proposed the MPS concept for continuous black-box optimization problems. In this thesis, the C-MPS method is generalized for discrete domain. Later, the concept has been generalized to combinatorial optimization domains, which instead of having a linear function to fit the samples, a global memory ranks all the activities and their positions based on the information received from the objective function. In fact, instead of having the approximation or linear spline

function evolves around the mode of the function, the moves which generate better sequences will be more focused in the combinatorial version of MPS.

2. DSM (Double Sphere Method): One of the main contributions of this thesis is introducing the DSM method. Applying the pure MPS concept to discrete domain would not generate good results without the DSM method. DSM is composed of double spheres whose radii enlarge or shrink with regard to information from previous iterations and provide the domain for generating samples. Depending on the improvement or non-improvement of the objective function reported from previous iterations, one of the spheres shrinks to provide more exploitation around the current local optima while the other expands to avoid being trapped into local optima. In the combinatorial version of MPS, DSM again controls the domain of generating neighborhoods for next iterations. In the combinatorial version, there are no spheres, since one is dealing with the combinatorial solution space. Instead two controllers B and S control the degree of freedom for generating next neighborhood of current local optima. It is believed that the DSM could be extended and incorporated into other direct search methods for better efficiency and effectiveness.

1.1.2 Practical contributions

In this thesis, a number of optimization methodologies and software tools have been provided that can be used to solve problems in mechanical design, combinatorial job sequencing, and healthcare scheduling optimization.

In specific, four versions of MPS-based optimization algorithms have been developed in this research:

1. D-MPS (Discrete Variable Mode Pursuing Sampling): D-MPS is a novel approach for discrete domain design problems. Approximation evolves while the DSM (Double Sphere Method) dynamically moves the search toward the mode of the objective function when it is necessary. D-MPS has been described in Chapter 3.
2. Co-MPS (Combinatorial Mode Pursuing Sampling): Co-MPS is a variation of D-MPS for combinatorial problems, with application to the single machine total tardiness problem. Several agents search for an optimal solution based on the information from global memory. Global memory (approximation) which is a memory of jobs being in specific positions evolves through the run-time. Moreover, two controllers dynamically control the neighborhood generation. Co-MPS have been described in Chapter 4.
3. TSP-MPS (Traveling Salesman Problem-Mode pursuing Sampling): TSP-MPS is a version of Co-MPS for TSP instances. The main difference of TSP-MPS and Co-MPS is on generating neighborhood and initial solutions. The results on small and medium size instances of TSP-MPS are comparable to current best methods of solving TSP. TSP-MPS has been described in Chapter 5.
4. AS-MPS(Appointment Scheduling-Mode Pursuing Sampling): AS-MPS is a variation of Co-MPS for appointment scheduling in healthcare, where several classes of agents search for the optimal appointment scheduling of patients in a clinic, and each of them follows a distinct rule of appointment scheduling. "New-

born" agents, created at each iteration, combine the characteristics of other classes of agents. Discrete-event simulation models are used as a black-box function of the optimization in AS-MPS. AS-MPS has been described in Chapter 6.

All of the four versions of the MPS-based optimization algorithms can be used in real-world applications and they all use a black-box function as an objective function. In D-MPS algorithm, black-box function can be finite-element analysis software, e.g. crash-simulation software. In Co-MPS, black-box functions can be a discrete event simulation if the processing time and arrival time of the jobs are stochastic. In TSP-MPS, black-box function can be a simulation of TSP type of problems such as the VRP (Vehicle Routing Problem). In AS-MPS, the objective function can be a discrete-event simulation for a clinic inside a hospital.

All the four versions of MPS-based optimization algorithms are promising in terms of the total number of function evaluations and finding good solutions in a relatively small number of iterations. One useful feature of the MPS-based algorithms is their plug-and-play characteristic. Integrating the MPS-base algorithms with real-world application systems should be able to significantly enhance the performance of the systems.

7.2 Future Work

Future work for the D-MPS will be focused on using the algorithm along with a real world application of a discrete domain mechanical design problem. For combinatorial mode pursuing sampling, the future work will be on generalizing the algorithm for job-

shop scheduling problems and implementing the algorithm with more complex discrete event simulation programs. In the healthcare domain, the AS-MPS algorithm will be generalized for surgery scheduling with historical data. Combining the software tools with more user friendly interfaces will be another area of future research.

References

Abdollah Homaifar, Shanguchuan Guan, and Gunar E. Liepins., 1992. *Schema analysis of the traveling salesman problem using genetic algorithms.* Complex Systems, 6(2), pp.183–217.

Abdul-Razaq, T.S., Potts, C.N., and Van Wasswnhove, L.N., 1990, *A survey of algorithms for the single machine total weighted tardiness scheduling problems*, Discr. Appl. Math., 26, pp.235-253.

Akbarzadeh, M. R. and Khorsand, R., 2005, *Evolutionary Quantum Algorithms for Structural Design*, in: Proc. of IEEE Int. Conf. on Systems, Men, and Cybernetics, Hawaii, USA, October 10-12, pp.3077-3082.

Applegate, D., R.Bixby, V.Chavatal, and W.cook, 1995 ,*Finding cuts in the TSP*, DIMACS Technical Report 95-05.

Arora, J. S., 1989, *Introduction to Optimum Design*, McGraw-Hill Higher Education, New York.

Azadivar ,Farhad, 1999, *Simulation optimization methodologies*, Proceedings of the 1999 Winter Simulation Conference.

Azadivar, F. and Y. H. Lee. 1988. *Optimization of discrete variable stochastic systems by computer simulation*. Mathematics and Computers in Simulation, 30, pp.331-345.

Babes, M. and G.V. Sarma (1991), *Out-Patient Queues at the Ibn-Rochd Health Center*, Journal of the Operational Research Society, 42, 10, pp.845-855.

Babu, P., Peridy, L., and Pinson, E., 2004, *A branch and bound algorithm to minimize total weighted tardiness on a single processor*, Annals of Operations Research, 129, pp.33-46.

Bailey, N. (1952), *A Study of Queues and Appointment Systems in Hospital Outpatient Departments with Special Reference to Waiting Times*, Journal of the Royal Statistical Society 14, pp.185-199.

Baker, K.R., and Scharge, L.E., 1978, *Finding an optimal permutation by dynamic programming: an extension to precedence-related tasks*, Oper. Res., 26, pp.111-120.

Biles, W. E. 1974. *A gradient regression search procedure for simulation experimentation.* Proceedings of 1974 Winter Simulation Conference, pp.491-497.

Bilge, U., Kurtulan, F., and Pekgun, M. P., 2003, *A tabu search algorithm for parallel machine total tardiness problem*, Computers and Operations Research, 31, pp.397-414.

Bilge, U., Kurtulan, M., and Furkan, K., 2007, *A tabu search algorithm for the single machine total weighted tardiness problem*, European Journal of Operational Research, 176, pp.1423-1435.

Bischof , C. H., A. Carle, G. Corliss, A. O. Griewank, and P. Hovland, 1992 *ADIFOR: Generating derivative codes from Fortran programs*, Sci. Programming, 1, pp. 11–29.

Bischof, C., A. Carle, P. Khademi, and A. Mauer, 1996, *The ADIFOR* 2.0 *system for the automatic differentiation of Fortran* 77 *programs*, IEEE Comput. Sci. Engrg., 3, pp.10-27.

Blake , John T. , Michael W. Carter, 2002, *A goal programming approach to strategic resource allocation in acute care hospitals*, European Journal of Operational Research 140 pp.541–561.

Bosch Vanden, P. M., C. D. Dietz, and J. R. Simeoni (1999), *Scheduling Customer Arrivals to a Stochastic Service System*, Naval Research Logistics, 46, pp.549-559.

Bosch Vanden, P. M., C. D. Dietz, and J. R. Simeoni (2000), Minimizing Expected Waiting in a Medical Appointment System, IIE Transactions, 32, 9, pp.841-848. Box ,G. E. P., 1957, Evolutionary operation: A method for increasing industrial productivity, Appl.Statist., 6 , pp. 81–101.

Box, G. and N. R.Draper, 1969, *Evolutionary Operation: A Statistical Method for Process Management*, John Wiley & Sons, Inc., New York.

Branin, F. H. and Hoo, S. K., 1972, *A Method for Finding Multiple Extrema of a Function of N Variables*, In: Lootsma, F. (Editor), *Numerical methods for non-linear optimization*, Academic Press, New York, pp.231-237.

Bussieck, R., Drud, S. and Meeraus, A., 2003, *MINLPLib-A Collection of Test Models for Mixed-Integer Nonlinear Programming*, INFORMS Journal on Computing, 15(1), pp.114-119.

Canadian Institute for health information, http://www.cihi.ca

Cao, Y. j. and Wu, Q. H., 1997, *Mechanical Design Optimization by Mixed-Variable Evolutionary Programming*, in: *Proceedings of IEEE Conference on Evolutionary Computation*, ICEC, Indianapolis, Indiana, USA, April 13-16.

Capriles, P. V. S. Z., da Fonseca, L. G., Barbosa, H. J. C. and Lemonge, A. C. C., 2005, *Ant Colony Algorithms Applied to Discrete Optimization Problems*, in: *XXVIII Congress*

Nacional de Matemátical Aplicada e Computacional (CNMAC), Sao Paulo, Brazil, Sept. pp.12-15.

Carter, Michael , 2002, *Diagnosis: Mismanagement of resource*, OR/MS Today, April, pp.26-30.

Chen, D. H., Z. Saleem, and D. W. Grace, 1986, *A new simplex procedure for function minimization*, Internat. J. Modelling & Simulation, 6, pp. 81–85.

Chen, W., 1995, *A Robust Concept Exploration Method for Configuring Complex System*, Ph.D. Dissertation Thesis, Mechanical Engineering, Georgia Institute of Technology, Atlanta, GA.

Coffman, E.G., Yannakakis, M., Magazine, M.J., and Santos, C., 1991, *Batch sizing and job sequencing on a single machine*, Annals of Operations Research, 26, pp.135-147.

Congeram, R.K, Potts, C.N., Van de Velde, S.L., 2002, *An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem*, Informs Journal on Computing, 14 (1), pp.52-67.

Conn,A. R., K. Scheinberg, and P. L. Toint, , 1997, *On the convergence of derivative-free methods for unconstrained optimization*, in Approximation Theory and Optimization: Tributes to M.J.D. Powell, M. D. Buhmann and A. Iserles, eds., Cambridge University Press, Cambridge, UK, pp. 83–108.

Coope, I. D. and C. J. Price, 2000, *A direct search conjugate directions algorithm for unconstrained minimization*, ANZIAM J., 42, pp. C478–498.

Coope, I. D. and C. J. Price, 2001, *On the convergence of grid-based methods for unconstrained optimization*, SIAM J. Optim., 11 pp. 859–869.

Cox, T. F., J. F. BIRCHALL, AND H. WONG (1985), *Optimizing the Queuing System for an Ear, Nose and Throat Outpatient Clinic*, Journal of Applied Statistics, 12, pp.113-126.

Crainic, T.G., Toulouse, M., and Gendreau, M., 1997, *Toward a taxonomy of parallel tabu search heuristics*. INFORMS Journal on Computing, 9,pp. 61–72.

Crauwels, H.A., Potts, C.N., and Van Wasswnhove, L.N., 2002, *Local search heuristics for the single machine total weighted tardiness scheduling problem*, INFORMS Journal on Computing, 14, pp.52-67.

Currin, C., Mitchell, T. J., Morris, M. D. and Ylvisaker, D., 1991, *Bayesian Prediction of Deterministic Functions, With Applications to the Design and Analysis of Computer Experiments*, Journal of American Statistical Association, 86(416), pp.953-963.

Dantzig , G., R.Fulkerson, S.Johnson, 1954, *Solution of a large-scale traveling-salesman problem*, Operations Research (2), pp.393-410.

Davidon ,W. C., 1959, *Variable Metric Method for Minimization*, Tech. Rep. 5990, Argonne National Laboratory, Argonne, IL.

Davis, T.E. 1991, *Toward an Extrapolation of the Simulated Annealing Convergence Theory onto the Simple Genetic Algorithm* (Doctoral Dissertation), University of Florida, Gainesville, Florida.

De Boor, C. and Ron, A., 1990, *On Multivariate Polynomial Interpolation*, Constructive Approximation, 6, pp.287-302.

Denton, B. AND D. GUPTA (2001), *A Sequential Bounding Approach for Optimal Appointment Scheduling,* Unpublished working paper, DeGroote School of Business, McMaster University, Hamilton, Ontario.

Droste, S., Jansen, T. and Wegener, I., 2001, *A New Framework for the Valuation of Algorithms for Black-Box Optimization*, CI-118/01, University of Dortmund, Dortmund, Germany.

Dyn, N., Levin, D. and Rippa, S., 1986, *Numerical Procedures for Surface Fitting of Scattered Data by Radial Basis Functions*, SIAM Journal of Scientific and Statistical Computing, 7(2), pp.639-659.

Eddy, J. and Lewis, K. E., 2002, *Visualization of Multi-dimensional Design and Optimization Data Using Cloud Visualization*, ASME 2002 Design Engineering Technical Conference and Computers and Information in Engineering Conference, Montreal, Canada, September 29 - October 2, DETC2002/DAC- 34130.

Fletcher, R. and Leyffer, S., 1994, *Solving Mixed Integer Programs by Outer Approximation*, Math. Program, 66, pp.327-349.

Friedman, J. H., 1991, *Multivariate Adaptive Regressive Splines*, The Annals of Statistics, 19(1), pp.1-67.

Freisleben , B., P. Merz., 1996, *A genetic local search algorithm for solving symmetric and a symmetric traveling salesman problems.* Proceedings of the 1996 IEEE International Conference on Evolutionary Computation, Nagoya, Japan, pp.616–621.

Fu, J. C. and Wang, L., 2002, *A Random-Discretization Based Monte Carlo Sampling Method and Its Applications*, Methodology and Computing in Applied Probability, 4, pp.5-25.

Fu, J. F., Fenton, R. G. and Cleghorn, W. L., 1991, *A Mixed Integer Discrete-Continuous Programming Method and its Application to Engineering Design Optimization*, Engineering Optimization, 17(3), pp.263-280.

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

Gendreau, M., Soriano, P. and Salvail, L., 1993, *Solving the maximum clique problem using a tabu search approach*. Annals of Operations Research, 41, pp.385–403.

Gill ,P. E., W. Murray, and M. H. Wright, 1981, *Practical Optimization*, Academic Press, London.

Giunta, A. A., Balabanov, V., Haim, D., Grossman, B., Mason, W. H., Watson, L. T. and Haftka, R. T., 1997, *Multidisciplinary Optimization of a Supersonic Transport Using Design of Experiments theory and Response Surface Modeling*, Aeronautical Journal, 101(1008), pp.347-356.

Glad ,T. and A. Goldstein, 1977, *Optimization of functions whose values are subject to small errors*, BIT, 17 , pp. 160–169.

Glover, F. (1986). *Future Paths for Integer Programming and Links to Artificial Intelligence*. Computers & Operations Research 13, pp.533–549.

Glover, F. , 1989, *Tabu Search—Part I*, ORSA Journal on Computing 1, pp.190–206.

Glover, F. 1997, *Tabu Search and Adaptive Memory Programming—Advances, Applications and Challenges*. In R.S. Barr, R.V. Helgason and J.L.Kennington (eds.), Advances in Metaheuristics, Optimization and Stochastic Modeling Technologies. Boston: Kluwer, pp. 1–75.

Glover, F. and G.A. Kochenberger. (eds.), 2003, *Handbook of Metaheuristics*, Boston: Kluwer.

Glover, F. and Laguna, M., 1997, *Tabu Search,* Kluwer Academic Publishers.

Glover, F., 1997, *Tabu Search and Adaptive Memory Programming—Advances, Applications and Challenges* In R.S. Barr, R.V. Helgason and J.L.Kennington (eds.), *Advances in Metaheuristics, Optimization and Stochastic Modeling Technologies.* Boston: Kluwer, pp. 1–75.

Griewank ,A. O., 1989, *On automatic differentiation*, in Mathematical Programming: Recent Developments and Applications, M. Iri and K. Tanabe, eds., Kluwer Academic, Dordrecht, The Netherlands, pp. 83–108.

Gu, L., 2001, *A Comparison of Polynomial Based Regression Models in Vehicle Safety Analysis*, in: Diaz, A. (Ed.), 2001 ASME Design Engineering Technical Conferences - Design Automation Conference, ASME, Pittsburgh, PA, September 9-12, DAC-21063.

Guikema ,Seth D., Rachel A. Davidson, Zehra Çağnan, 2004, *Efficient Simulation-based discrete optimization*, Proceedings of the 2004 Winter Simulation Conference.

Gupta, O. K. and Ravindran, A., 1985, *Branch and Bound Experiments in Convex Nonlinear Integer Programming*, Manage Sci., 31(12), pp.1533-1546.

Hedar, A.-R. and Fukushima, M., 2004, *Derivative-Free Filter Simulated Annealing Method for Constrained Continuous Global Optimization*, Journal of Global Optimization, 35(4), pp.521-549.

Ho, C. AND H. LAU (1992), *Minimizing Total Cost in Scheduling Outpatient Appointments*, Management Science, 38, 12, pp.1750-1764.

Ho, Y. C., and X. R. Cao. 1991. *Perturbation analysis of discrete event dynamic systems*, Kluwer Academic Publishers.

Ho, Y. C., X. R. Cao, and C. Cassandras. 1983. *Infinitesimal and finite perturbation analysis of queuing networks*, *Automatica*, 19, pp.439-445.

Hodgson, I. J., 1977, *A note on single machine sequencing with random processing times*, Management Science, 23, pp.1144-1146.

Hooke ,R. and T. A. Jeeves, 1961, *Direct search solution of numerical and statistical problems*, J.ACM, 8, pp. 212–229.

Horst, R. and Pardalos, P. M., 1995, *Handbook of Global Optimization*, Kluwer Academic Publishers, Dordrecht, Boston, London.

Huang, M. and Arora, J. S., 1997, *Optimal Design Discrete Variables: Some Numerical Experiments*, International Journal for Numerical Methods in Engineering, 40, pp.165-188.

Institute of Medicine (IOM), 2000, *To Err Is Human: Building a Safer Health System*, edited by L.T. Kohn, J.M. Corrigan, and M.S. Donaldson. Washington, D.C.: National Academy Press.

Institute of Medicine (IOM).,2001, *Crossing the Quality Chasm: A New Health System for the 21st Century*. Washington, D.C.: National Academy Press.

Johnson , D.S. and L.A. McGeoch, 1997, *The traveling salesman problem: A case study in local optimization*, in: Local Search in Combinatorial Optimization (E.H.L. Aarts and J.K.Lenstra, eds.), Wiley, New York, pp. 215-310.

Johnson, M. E., Moore, L. M. and Ylvisaker, D., 1990, *Minimax and Maximin Distance Designs*, Journal of Statistical Planning and Inferences, 26(2), pp.131-148.

Jones, D. R., Perttunen, C. D. and Stuckman, B. E., 1993, *Lipschitzian Optimization without the Lipschitz Constant*, J. Optim. Theory Appl., 79, pp.157-181.

Jones, D. R., Schonlau, M. and Welch, W. J., 1998, *Efficient Global Optimization of Expensive Black Box Functions*, Journal of Global Optimization, 13, pp.455-492.

Joshi , B., D. Morris, N. White and R. Unal, 1996, Optimization of Operations resources via discrete event simulation modeling, Presented at the 6th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization.

K. DeJong, M.D. Vose and L.D. Whitley (eds.), Evolutionary Algorithms: IMA Volumes in Mathematics and its Applications, Vol. 111. Springer-Verlag, New York, pp. 89–109.

Khompatraporn, C., Pinter, J. D. and Zabinsky, Z. B., 2005, *Comparative Assessment of Algorithms and Software for Global Optimization*, Journal of Global Optimization, 31, pp.613-633.

Kiatsupailbul, S., Smith, R. L. and Zabinsky, Z. B., 2001, *Discrete Hit-and-Run Algorithm for Generating Samples from General Discrete Multivariate Distribution*, Department of Industrial and Operations Engineering, University of Michigan, Ann Arbor, Michigan, USA.

Klassen, K. J. AND T. R. ROHLEDER (1996), *Scheduling Outpatient Appointments in a Dynamic Environment*, Journal of Operations Management, 14, 2, pp.83-101.

Knox , J.., 1994, *Tabu search performance on the symmetric traveling salesman problem.* Computers & Operations Research, 1994, Vol.21, pp. 867–876.

Koehler, J. R. and Owen, A., 1996, *Computer Experiments*, In: Ghosh, S. and Rao, C. R. (Editors), Handbook of Statistics, Elsevier Science, New York, pp. 261-308.

Laguna, M., Barnes, J.W., and Glover, F., 1991, *Tabu search methods for a single machine scheduling problem*, Journal of Intelligent Manufacturing, 2, pp.63-74.

Lampinen, J. and Zelinka, I., 1999, *Mixed Integer-Discrete-Continuous Optimization By Differential Evolution, Part 2: A Practical Example*, in: *Proceedings of MENDEL '99, 5th International Mendel Conference on Soft Computing*, Brno, Czech Republic, June 9-12.

Langley, P. and Simon, H. A., 1995, *Applications of Machine Learning and Rule Induction*, Communications of the ACM, 38(11), pp.55-64.

Lawler, E.L, 1973, *Optimal Sequencing of a single machine subject to precedence constraints*, Management Science, 19 (5), pp.544-546.

Lemonge, A. C. C. and Barbosa, H. J. C., 2004, *An Adaptive Penalty Scheme for Genetic Algorithms in Structural Optimization*, International Journal for Numerical Methods in Engineering, 59, pp.703-736.

Lenstra, J.K., and Rinnooy Kan, A.H.G, 1980, *Complexity results for scheduling chains on a single machine*, European J. Oper. Res., 4, pp.270-275.

Lewis ,R. M. and V. Torczon, 1996, *Rank Ordering and Positive Bases in Pattern Search Algorithms*, Tech. Rep. 96–71, Institute for Computer Applications in Science and Engineering, NASA Langley Research Center, Hampton, VA.

Lewis, R. M., V. Torczon, and M. W. Trosset, 2000, *Direct search methods: Then and now*, J. Comput. Appl. Math., 124, pp. 191–207.

Lin, Y. C., Wang, F. S. and Hwang, K., 1999, *A Hybrid Method of Evolutionary Algorithms for Mixed-Integer Nonlinear Optimization Problems*, in: *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99*, Vol. 3, IEEE, Washington DC, USA, July 6-9, Cat. No. 99TH8406.

Lin, Y., 2004, *An Efficient Robust Concept Exploration Method and Sequential Exploratory Experimental Design*, Ph.D. Dissertation Thesis, Mechanical Engineering, Georgia Institute of Technology, Atlanta, pp. 780-788.

Lindley, D. V. (1952), *The Theory of Queues with a Single Server*, Proceedings Cambridge Philosophy Society, 48, pp.277-289.

Litvak, Lung, 2000, *Cost and quality under managed care: irreconcible differences*, American J of mnaged care, (6), pp.305-312.

Liu, L. and X.Liu (1998a), *Block Appointment Systems for Outpatient Clinics with Multiple Doctors*, Journal of the Operational Research Society, 49, 12, pp.1254-1259.

Lou, X., and Chu, F., 2006, *A branch and bound algorithm of the single machine schedule with sequence dependent setup times for minimizing total tardiness*, Applied Mathematics and Computation, 183(1), pp.575-588.

Maheswaran, R., and Ponnambalam, S.G., 2005, *An intensive search evolutionary algorithm for single-machine total-weighted-tardiness scheduling problems*, International Journal of Advanced Manufacturing Technology, 26, pp.1150-1156.

Martin, J. D. and Simpson, T. W., 2005, *Use of Kriging Models to Approximate Deterministic Computer Models*, AIAA Journal, 43(4), pp.853-863.

McNaughton, R., 1959, *Scheduling with deadlines and loss functions*, Management Science, 6, pp.1-12.

Meckesheimer, M., Booker, A. J., Barton, R. R. and Simpson, T. W., 2002, *Computationally Inexpensive Metamodel Assessment Strategies*, AIAA Journal, 40(10), pp.2053-2060.

Merkle, D., and Middendorf, M., 2003, *An ant algorithm with global pheromone evaluation for scheduling and a single machine, Applied . Inteligence.*, 18, pp.105-111.
Michea Gendreau, Jean-Yves Potvin, 2005, *Metaheuristics in Combinatorial Optimization*, Annals of Operations Research 140, pp.189–213, 2005.

Misevičius , Alfonsas, Jonas Smolinskas, Arūnas Tomkevičius, 2005, *Iterated Tabu Search for the traveling salesman problem: New Results*, ISSN 1392-124 Information technology and control, Vol34, No.4.

Moss, Fiona, 2004, *The clinician, the patient and the organization: a crucial three sided Relationship*, Qual Saf Health Care; 13, pp.406–407.

Myers, R. H. and Montgomery, D., 1995, *Response Surface Methodology: Process and Product Optimization Using Designed Experiments*, John Wiley and Sons, Inc., Toronto.
Nocedal ,J. and S. Wright, 1999, *Numerical Optimization*, Springer Ser. Oper. Res., Springer- Verlag, New York., Norwell, MA.

OR-lib: http://people.brunel.ac.uk/~mastjjb/jeb/info.html

Osio, I. G. and Amon, C. H., 1996, *An Engineering Design Methodology with Multistage Bayesian Surrogates and Optimal Sampling*, Research in Engineering Design, 8(4), pp.189-206.

Osman I.H, Kelly J.P, 1996, *Meta-Heuristics-Theory and applications*, Kluwer Academic Publishers.

Osman, I.H. and G. Laporte., 1996, *Metaheuristics: A Bibliography,* Annals of Operations Research 63, pp.513–623.
Papadimitriou , C. and K.Steiglitz, 1977, *On the complexity of local search for the traveling salesman problem*, SIAM Journal on Computing-*6,* pp.76-83.

Pepper , J., B. Golden, E. Wasil. 2002, *Solving the traveling salesman problem with annealing-based heuristics: a computational study*. IEEE Transactions on Systems, Piscataway, NJ., pp. 18–32.

Prasanna Jog, Jung Y. Suh, and Dirk Van Gucht., 1991, *Parallel genetic algorithms applied to the traveling salesman problem.* SIAM Journal of Optimization, 1(4):515–529, *Prinzipen der biologischen Evolution,* Frommmann-Holzboog Verlag, Stuttgart

Proctor P. Reid, W. Dale Compton, Jerome H. Grossman, and Gary Fanjiang, 2005, *Building a Better Delivery System: A New Engineering/Health Care Partnership,* Editors, Committee on Engineering and the Health Care System, National Academy of engineering, Institute of Medicine.

Prudius , Andrei A., Sigrún Andradóttir, *Simulation Optimization Using Balanced explorative and exploitative search*, Proceedings of the 2004 Winter Simulation Conference.

Rachamadugu, R.V., and Morton, T.E., 1982, *Myopic Heuristics for the single Machine Tardiness Problem*, Working Paper, Carnegie Mellon University, Pittsburgh, PA, 30–82-83, GSIA.

Reaume, J., Romeijn, H. and Smith, R., 2001, *Implementing Pure Adaptive Search for Global Optimization Using Markov Chain Sampling*, Journal of Global Optimization, 20, pp.33-47.

Rechenberg, I., 1973, *Evolutions strategic: Optimierung technischer Systeme nach*
Reinelt ,G., 1991, *TSPLIB-A traveling salesman problem library*, ORSA Journal on Computing 3, pp. 376-384.

Rohleder, T. R. and K. J. Klassen. 2000. *Using client-variance information to improve dynamic appointment scheduling performance. Omega* 28: pp.293--302.

Rosenbrock, H. H., 1960, *An automatic method for finding the greatest or least value of a function*, Comput. J., 3 pp. 175–184.

Rosenkrantz ,D.J., R.E. Stearns, and P.M. Lewis II, 1997, *An analysis of several heuristics for the traveling salesman problem*, SIAM Journal on Computing 6 , pp.563-581.

Sasena, M., Parkinson, M., Goovaerts, P., Papalambros, P. and Reed, M., 2002, *Adaptive Experimental Design Applied to An Ergonomics Testing Procedure*, ASME 2002 Design Engineering Technical Conferences and Computer and Information in Engineering Conference, ASME, Montreal, Canada, September 29-October 2, DETC2002/DAC-34091.

Sergeyev, Y. and Kvasov, D., 2006, *Global Search Based on Efficient Diagonal Partitions and Set of Lipschiz Constants*, SIAM Journal of Optimization, 16(3), pp.910-937.

Shan, S. and Wang, G. G., 2005, *Failure Surface Frontier for Reliability Assessment on Expensive Performance Function*, Transactions of ASME, Journal of Mechanical Design, in production.

Sharif, B., Wang, G., and ElMekkawy, T. Y., 2007, *Mode Pursuing Sampling Method for Discrete Variable Optimization on Expensive Black-Box Functions*, Transactions of ASME, Journal of Mechanical Design, (In Press).

Schwimer, J., 1972, *On the N-job, one-machine, permutation-independent scheduling problem with tardiness penalties: a branch-bound solution*, Management Science, 18, pp.301-313.

Simpson, T. W., Lin, D. K. J. and Chen, W., 2001, *Sampling Strategies for Computer Experiments: Design and Analysis*, International Journal of Reliability and Application, 2(3), pp.209-240.

Soriano, A. (1966), *Comparison of Two Scheduling Systems*, Operations Research, 14, pp.388-397.

Starfield, B., 2000,. *Is U.S. health really the best in the world?*, Journal of the American Medical Association 284(4), pp. 483–485.

Swann, W. H., 1972, *Direct search methods*, in Numerical Methods for Unconstrained Optimization, Academic Press, London, New York, pp. 13–28.

Tasgetiren, M.f., Sevkli, M., Liang, Y.C., and Gencyilmaz, G., 2006, *Particle swarm optimization and differential evolution for the single machine total weighted tardiness problem*, Int. J. of Production Research, 44 (22), pp.4737-4754.

Torczon ,V., 1991, *On the convergence of the multidirectional search algorithm*, SIAM Journal of Optimization.,1, pp. 123–145.

Torczon ,V., 1997, *On the convergence of pattern search algorithms*, SIAM Journal of Optimization, 7, pp. 1–25.

Trosset , M. W., 1997, *I know it when I see it: Toward a definition of direct search methods*, SIAG/OPT Views-and-News: A Forum for the SIAM Activity Group on Optimization, 9, pp. 7–10.

Ullman, D. G., 2002, *Toward the Ideal Mechanical Engineering Design Support System*, Research in Engineering Design, 13, pp.55-64.

Vijay K. Khandelwal, Terry Lynch, *Reengineering of the Patient Flow Process at the Western Sydney Area Health Service*, Proceedings of the 32nd Hawaii International Conference on System Sciences – 1999.

Walter, S. D. (1973), *A Comparison of Appointment Schedules in a Hospital Radiology Department*, British Journal of Preventive and Social Medicine, 27, pp.160-167.

Wang, G. G. and Shan, S., 2006, *Review of Metamodeling Techniques in Support of Engineering Design Optimization*, Transactions of ASME, Journal of Mechanical Design, April 2007 (In press).

Wang, L., S. Shan, and G. G. Wang, 2004a, *Mode-Pursuing Sampling Method for Global Optimization on Expensive Black-box Functions*, Journal of Engineering Optimization, 36(4), pp. 419-438.

Wang, G. G. and Simpson, T. W., 2004, *Fuzzy Clustering Based Hierarchical Metamodeling for Space Reduction and Design Optimization*, Journal of Engineering Optimization, 36(3), pp.313-335.

Wang, G. G., 2003, *Adaptive Response Surface Method Using Inherited Latin Hypercube Design Points*, Transactions of ASME, Journal of Mechanical Design, 125, pp.210-220.

Welch, W. J., Buck, R. J., Sacks, J., Wynn, H. P., Mitchell, T. J. and Morris, M. D., 1992, *Screening, Predicting, and Computer Experiments*, Technometrics, 34(1), pp.15-25.

Winer, E. H. and Bloebaum, C. L., 2002, *Development of Visual Design Steering as an Aid in Large-scale Multidisciplinary Design Optimization*. Part II: Method Validation, Structural and Multidisciplinary Optimization, 23(6), pp.425 - 435.

Zabinsky, Z., 1998, *Stochastic Methods for Practical Global Optimization*, Journal of Global Optimization, 13, pp.433-444.

Zhang, C. and Wang, H. P., 1993, *Mixed-Discrete Nonlinear Optimization with Simulated Annealing*, Engineering Optimization, 21(44), pp.277-291.

Zhang, L. and Subbarayan, G., 2002, *An Evaluation of Back Propagation Neural Networks for the Optimal Design of Structural Systems: Part I. Numerical Evaluation*, Computation Methods in Applied Mechanic Engineering, 191, pp.2887-2904.

Appendix A
Data for TSP

Table A-1. Data for Burma14

City	Latitude	Longitude
0	16.47	96.10
1	16.47	94.44
2	20.09	92.54
3	22.39	93.37
4	25.23	97.24
5	22.00	96.05
6	20.47	97.02
7	17.20	96.29
8	16.30	97.38
9	14.05	98.12
10	16.53	97.38
11	21.52	95.59
12	19.41	97.13
13	20.09	94.55

Table A-2. Data for Gr21

	1	2	3	4	5	6	7	8	9	10
1	0									
2	510	0								
3	635	355	0							
4	91	415	605	0						
5	385	585	390	350	0					
6	155	475	495	120	240	0				
7	110	480	570	78	320	96	0			
8	130	500	540	97	285	36	29	0		
9	490	605	295	460	120	350	425	390	0	
10	370	320	700	280	590	365	350	370	370	0
11	155	380	640	63	430	200	160	175	535	240
12	68	440	575	27	320	91	48	67	430	300
13	610	360	705	520	835	605	590	610	865	250
14	655	235	585	555	750	615	625	645	775	285
15	480	81	435	380	575	440	455	465	600	245
16	265	480	420	235	125	125	200	165	230	475
17	255	440	755	235	650	370	320	350	680	150
18	450	270	625	345	660	430	420	440	690	77
19	170	445	750	160	495	265	220	240	600	235
20	240	290	590	140	480	255	205	220	515	150
21	380	140	495	280	480	340	350	370	505	185

Table A-2 (Continued)

	1	2	3	4	5	6	7	8	9	10
1	0									
2	510	0								
3	635	355	0							
4	91	415	605	0						
5	385	585	390	350	0					
6	155	475	495	120	240	0				
7	110	480	570	78	320	96	0			
8	130	500	540	97	285	36	29	0		
9	490	605	295	460	120	350	425	390	0	
10	370	320	700	280	590	365	350	370	370	0
11	155	380	640	63	430	200	160	175	535	240
12	68	440	575	27	320	91	48	67	430	300
13	610	360	705	520	835	605	590	610	865	250
14	655	235	585	555	750	615	625	645	775	285
15	480	81	435	380	575	440	455	465	600	245
16	265	480	420	235	125	125	200	165	230	475
17	255	440	755	235	650	370	320	350	680	150
18	450	270	625	345	660	430	420	440	690	77
19	170	445	750	160	495	265	220	240	600	235
20	240	290	590	140	480	255	205	220	515	150
21	380	140	495	280	480	340	350	370	505	185

Table A-3. Data for Gr96

City	Latitude	Longitude	City	Latitude	Longitude	City	Latitude	Longitude
1	23.31	1 14.55	23	15.36	32.32	44	12.22	1.31
2	28.06	-15.24	24	13.11	30.13	45	13.31	2.07
3	32.38	-16.54	25	13.38	25.21	46	12.00	8.30
4	31.38	-8.00	26	15.20	38.53	47	11.51	13.10
5	33.39	-7.35	27	9.00	38.50	48	12.07	15.03
6	34.02	-6.51	28	11.36	43.09	49	6.27	3.24
7	34.05	-4.57	29	18.06	15.57	50	6.27	7.27
8	35.48	-5.45	30	14.40	17.26	51	0.20	6.44
9	35.43	-0.43	31	13.28	16.39	52	3.45	8.47
10	36.47	3.03	32	11.51	15.35	53	3.52	11.31
11	22.56	5.30	33	16.46	3.01	54	4.22	18.35
12	36.22	6.37	34	12.39	8.00	55	0.23	9.27
13	36.48	10.11	35	10.23	9.18	56	-4.16	15.17
14	34.44	10.46	36	9.31	13.43	57	-4.18	15.18
15	32.54	13.11	37	8.30	13.15	58	0.04	18.16
16	32.07	20.04	38	6.18	10.47	59	-5.54	22.25
17	31.12	29.54	39	5.19	4.02	60	0.30	25.12
18	31.16	32.18	40	6.41	1.35	61	-3.23	29.22
19	29.58	32.33	41	5.33	0.13	62	-1.57	30.04
20	30.03	31.15	42	6.08	1.13	63	0.19	32.25
21	24.05	32.53	43	6.29	2.37	64	-1.17	36.49

Table A-3 (Continued)

City	Latitude	Longitude	City	Latitude	Longitude	City	Latitude	Longitude
65	2.01	45.20	86	-26.15	28.00	96	-4.38	55.27
66	-4.03	39.40	87	-29.12	26.07			
67	-6.10	39.11	88	-29.55	30.56			
68	-6.48	39.17	89	-33.00	27.55			
69	-8.48	13.14	90	-33.58	25.40			
70	-12.44	15.47	91	-33.55	18.22			
71	-11.40	27.28	92	-23.21	43.40			
72	-12.49	28.13	93	-18.55	47.31			
73	-15.25	28.17	94	-12.16	49.17			
74	-20.09	28.36	95	-20.10	57.30			
75	-17.50	31.03	96	-4.38	55.27			
76	-15.47	35.00	86	-26.15	28.00			
77	-19.49	34.52	87	-29.12	26.07			
78	-25.58	32.35	88	-29.55	30.56			
79	-15.57	-5.42	89	-33.00	27.55			
80	-37.15	-12.30	90	-33.58	25.40			
81	-22.59	14.31	91	-33.55	18.22			
82	-22.34	17.06	92	-23.21	43.40			
83	-26.38	15.10	93	-18.55	47.31			
84	-24.45	25.55	94	-12.16	49.17			
85	-25.45	28.10	95	-20.10	57.30			

Appendix B
CD Contents

Directory Filename	Descriptions
Readme CD	User readme file for CD instruction
D-MPS	
▪ Readme.txt	Instructions on how to use D-MPS
▪ dmps.m	Main file for D-MPS method
▪ Inputfile.m	Function for inputting the data
▪ fx1.m	Objective function to be optimized
▪ constgr.m	Constraints of the function
▪ approx.m	Function for approximation in D-MPS
▪ spheregr.m	Function for DSM method
▪ samplingnew4.m	Function for sampling new solutions
▪ divdz.m	Function for preventing division by zero
▪ rr.txt, gt.txt, pv.txt	Example input files for SC function, Gear train problem and Pressure vessel problem, respectively.
Co-MPS	
▪ Readme.txt	Instruction on how to use Co-MPS
▪ CompsRun.m	Main file for executing Co-MPS
▪ comps-Final.m	Function which contains the main code
▪ objavtardy.m	Function which computes the objective

- samplingc.m — Function for sampling new solutions
- swap2.m — Function for generating new neighborhood
- assignmentsuboptimal2.m — Function for solving the assignment problem
- edd.m, wspt.m, rm.m — Function for generating initial solutions
- myrandint.m — Function for generating random numbers
- Input.txt — Inut data file
- data.txt — Data file for the input file
- Output.txt — Output data file
- wt40.txt,wt50.txt,wt100.txt — Data file for Single machine problem
- wtopt40.txt,wtopt50.txt,wtopt100.txt — Data for the optimal solution of Single machine problem

TSP-MPS

- Readme.txt — Instruction on how to use TSP-MPS
- TSPRun.m — Main file for executing TSP-MPS
- TSP_Final.m — Function which contains the main code
- objtsp.m — Function which computes the distance matrix
- objtspmain.m — Function which computes the cost of every tour
- inserttsp.m — Function for generating initial solutions for TSP
- samplingc.m — Function for sampling new solutions

- swap4.m — Function for generating new neighborhood
- assignmentsuboptimal2.m — Function for solving the assignment problem
- myrandint.m — Function for generating random numbers
- Input.txt — Input data file
- data.txt — Data file for the input file
- Output.txt — Output data file
- Burma14.txt,gr21.txt,gr96.txt — Data file for three instances of TSP

AS-MPS

- Readme.txt — Instruction on how to use AS-MPS
- temppac4.avi — Video file of the simulation model
- pac4.mod — Simulation file of PAC in Witness
- ascompsn.m — Main file of AS-MPS algorithm
- pacw2.m — File for creating "probat.wcl"
- samplingc.m — Function for sampling new solutions
- swap3.m — Function for generating new neighborhood
- wrt.m — Function for generating "p.par"
- myrandint.m — Function for generating random numbers
- p.par — Input file for simulation model
- Rep.txt — Output file from simulation model
- output.txt — Output file of the AS-MPS algorithm
- probat.wcl — File for executing "witness.exe"