On the development of a stochastic optimization algorithm with capabilities for distributed computing

A Thesis Submitted to the

University of Surrey

for the degree of

Doctor of Philosophy

by

Siyu Yang

under the supervision of

Prof. Antonis Kokossis and Dr. Franjo Cecelja

The Centre for Process and Information Systems Engineering
School of Engineering, University of Surrey
Guildford GU2 7XH, United Kingdom

Abstract

In this thesis, we devise a new stochastic optimisation method (cascade optimisation algorithm) by incorporating the concepts from Markov process whilst eliminating the inherent sequential nature that is the major deficit preventing the exploitation of advances in distributed computing infrastructures. This method introduces partitions and pools to store intermediate solution and corresponding objectives. A Markov process increases the population of partitions and pools. The population is distributed periodically following an external certain. With the use of partitions and pools, multiple Markov processes can be launched simultaneously for different partitions and pools. The cascade optimisation algorithm is suitable for parallel and distributed computing environments. In addition, this method has the potential to integrate knowledge acquisition techniques (e.g. data mining and ontology) to achieve effective knowledge-based decision making.

Several features are extracted and studied in this thesis. The application problems involve both the small-scale and the large-scale optimisation problems. Comparisons with the stochastic optimisation methods are made and results show that the cascade optimisation algorithm can converge to the optimal solutions in agreement with other methods more quickly. The cascade optimisation algorithm is also studied on parallel and distributed computing environments in terms of the reduction in computation time.

Acknowledgements

Firstly, I want to say a special thank you to my current supervisor, Prof Antonis Kokossis, for sharing his idea and expertise, understanding, patience, and valued comments. Secondly, I would like to acknowledge my previous supervisor, Dr Patrick Linke, for sharing his ideas and expertise in the first half of this project. Of course, I would like to appreciate my co-supervisor Dr Franjo Cecelja for valued comments and suggestions. I would also like to thank Dr Andy Tate and Dr Aidong Yang for never-ending support, patience as well as help with my work.

I would like to thank Du Du for his valued assistances with preparation and execution of experiments on Grid computing environments. And I would also like to thank Daniel Montolio-Rodriguez and Claudia Labrador-Darder for helping me to understand the basic principles of process design.

Thanks to all my friends I have made during the past years, especially Dr Ying Gao, Dr Athanasios I. Papadopoulos, Suresh, Alexandra, for making postgraduate life more enjoyable.

Finally, many thanks have to go to my family, my Mum and Dad, for their never-ending support and encouragement. I am especially grateful to Mingyao Zhu for taking care of me during the last period of the project.

Table of Contents

List of Figures

Chapter 2 Optimisation and Techniques

Chapter 4 The Cascade Optimisation Algorithm

Chapter 5 Algorithm implementation as a master-worker paradigm

Chapter 6 Evaluation and validation

Chapter 7 Large-scale applications

Chapter 8 Parallel and Distributed computing application

List of Tables

Chapter 1 Introduction

Optimisation has evolved from the academic interest to technology that is broadly applied to many fields. Classification of optimisation problems depends on different characteristics, e.g. continuity, linearity and convexity. According to the linearity, optimisation problems are classified into continuous problems or discrete problems. Linear problems or non-linear problems are two subclasses in terms of the linearity of problems. Optimisation problems can also be classified into convex problems that have no local optimal solutions and non-convex problems that might involve local optimal solutions according to their convexity properties. Depending upon the classification of optimisation problems, different optimisation techniques are proposed. Deterministic optimisation and stochastic optimisation are two major techniques differing in the way of searching for the optimal solutions.

Deterministic methods involve linear programming, non-linear programming, mixed integer linear programming, and mixed integer non-linear programming. Each of these programming models is associated with a specific class of problems. The methods are developed based on the properties of the problems and take advantage of different mathematical and geometric theories. Simplex method and ellipsoid method are two popular linear programming methods. Branch and Bound is often used to solve non-linear problems. Traditional methods for discrete problems rewrite the problem to linear or non-linear problems and involve Generalized Benders Decomposition and Outer Approximation. Global optimisation methods have been recently used in the mixed integer problems.

Stochastic optimisation methods apply a heuristic-based search system to make the algorithm less sensitive to modelling errors. These methods are more suitable for

highly dimensional problems with inherent system noise than the classical deterministic method. Large numbers of runs under varying initial conditions are utilized to statistically guarantee the convergence of these methods. If these runs all converge to the solutions that have identical or similar qualities, these solutions are considered to be the global optimal solutions. In this case, stochastic methods can converge to global optimal solutions even starting from different initial conditions. Numerous stochastic methods are developed but four popular ones will be reviewed in Chapter 2.

Application problems are often high dimensional and have large search spaces. The heuristic-based search system takes longer time to converge than the deterministic optimisation methods. Current research on stochastic optimisation techniques focuses on this problem. Parallel computing techniques have been proved to be an effective way forward. With the developments of new distributed computing techniques, a large-scale distributed computing environment can be implemented. Computers in this environment are remotely connected via the Internet. However, the sequential nature within the heuristic-search system prevents the stochastic methods from large-scale distributed applications. In addition, some parallel applications have the computers running synchronously so that the faster computers have to wait for the slower ones. Then the computing resources are not fully exploited and the benefit in speeding up the convergence is limited. A new stochastic optimisation method is proposed, which incorporates the concept from the heuristic-search system but meanwhile minimizes the limitations.

In Chapter 2, optimisation is described based on principle and classifications. The optimisation techniques and corresponding classifications are also reviewed in this chapter. The principles of deterministic optimisation and stochastic optimisation techniques are elaborated along with their advantages and limitations. Four popular stochastic optimisation methods are reviewed along with their properties. In Chapter 3, a few concepts that can benefit optimisation are introduced. In Chapter 4, the new stochastic optimisation algorithm is described from a conceptual point of view. In the

course of studies, two implementations are developed and explained in Chapter 5. To evaluate and validate the algorithm, a few important features of the algorithm are studied in small-scale optimisation problems in Chapter 6. These features involve the management of the stochastic search, the optimisation structure size, the depth of the search, and the selection and coordination of the termination criteria. A few large-scale problems are used to study these features in Chapter 7. In addition, search policies that have impact on the performance are proposed and studied. Comparison with traditional stochastic optimisation methods is performed on a complex application problem. The aim of the comparison is to explore the advantages and limitations of the new method over the stochastic methods. In Chapter 8, some adjustments on the implementation of the new method are selected to meet the requirement of a parallel and distributed computing application. In theory, the computation time should decrease as the number of computing resources increases. The reduction of the computation time is studied in this chapter. Results are also compared with those of the parallel Tabu Search to illustrate the advantages of the new method when applied on parallel and distributed computing environments.

Chapter 2 Optimisation and Techniques

2.1 Problems

The formulation of a mathematical optimisation problem is:

$$\begin{aligned} &Given: \quad f: S \to R, S \subset R^n \\ &sought: \{x^* \in S \mid f(x) \geq f(x^*) \forall x \in S\} \end{aligned} \tag{2.1}$$

where

S : a subset of the Euclidean space R^n specified by the constraints (equality or inequality),

x: the available point in space S,

f: the objective function,

x^*: the optimal solution in space S.

In this thesis, the above formulation is denoted as:

$$\min_{x \in S} f(x) \tag{2.2}$$

In space S, there are some $x' \in S$ following:

$$\begin{aligned} &N_s(x) = \{x \in S : \|x - x'\| \leq \delta\} \\ &x' = f(x') \leq f(x), \forall x \in N_s(x) \end{aligned} \tag{2.3}$$

where

x': the local optima in space S

δ: a small positive value in [0,1].

2.2 Classification of optimisation problems

Optimisation problems can be divided into a number of different classes.

Firstly, optimisation problems can be classified into either continuous or discrete problems. The problems are considered to be continuous if their variables are real and continuous. Otherwise they are discrete problems. With discrete variables in the problems, the formulation of the optimisation problems is outlined as follows:

$$\min_{x,y} f(x,y)$$
$$s.t.\begin{cases} h(x,y)=0 \\ g(x,y)\le 0 \\ x\in X\equiv \Re^n, y\in Y\equiv\{0,1\}^n \end{cases} \tag{2.4}$$

where

x : the continuous variables,

y: the discrete variables,

$f(x,y)$: the objective function,

$h(x,y)$ and $g(x,y)$: the equality and the inequality constraints.

Secondly, the problems can be classified into convex or non-convex problems depending on the convexity of the search space and the objective function. The space is considered to be convex if the closed line segment joining any two available points is in the space. The formulation of the closed line segment is illustrated as:

$$X_{seg} = \{x \mid x = (1-\lambda)x_1 + \lambda x_2, 0 \le \lambda \le 1\} \qquad x_1, x_2 \in S \tag{2.5}$$

where

X_{seg} : the segment line

x_1 : the end point of the line

x_2 : the other end point of the line

λ : a parameter associated with points in the line.

Based on Eq. 2.5, S is the convex set if $X_{seg} \subset S$. The objective function is considered to be the convex function if it satisfies Jensen's inequality. The definition of Jensen 's inequality is:

$$f\left[(1-\lambda)x_1 + \lambda x_2\right] \le (1-\lambda)f(x_1) + \lambda f(x_2) \qquad x_1, x_2 \in S, 0 \le \lambda \le 1 \tag{2.6}$$

where

x_1 and x_2 : two points in S

λ : a parameter between 0 and 1.

The problems that satisfy the above conditions are considered to be convex and have a unique global optimal solution. In contrast, the problems may have only local optimal solutions if the space and the objective function do not satisfy the conditions above.

Optimisation problems can be classified into linear or non-linear problems depending on the linearity of the objective function and the constraints. An optimisation problem is

non-linear if its variables are non-linear or if its objective function is a non-linear function. The variables are considered to be non-linear if the equality and the inequality constraints are non-linear.

2.3 Optimisation methods

Optimisation methods systematically exploit the degree of freedom to minimize or maximize an objective function subject to the constraints. Different optimisation techniques have been developed and applied to a number of application problems. These methods can be classified as deterministic optimisation methods and stochastic optimisation methods.

2.3.1 Deterministic methods

Deterministic methods focus on topological and geometrical methods to solve the optimisation problems. The development of deterministic methods evolves from linear programming (LP), non-linear programming (NLP), mixed-integer linear programming (MILP), and mixed-integer non-linear programming (MINLP).

(a) Linear programming

LP defines the methods to solve the linear optimisation problems. These problems involve the linear objective function that is subject to the linear equality and/or the inequality constraints. From the geometrical point of view, the linear constraints of these problems define a convex polyhedron as the feasible region. Also because the objective functions of these problems are linear, all local optimal solutions are automatically global optimal solutions (Karush, 1939). The typical methods for linear problems involve the simplex problem, and the ellipsoid method (Shor, 1972).

(i) Simplex method

From the geometric point of view, the constraints of the linear problem define a convex polytope. The optimal solution is one of the polytope vertices. The simplex method leverages this insight by rewriting the problems so that one of the vertices can be found easily. Then the method explores the vertices along the edge of the polytope until a local optimal solution is approached. This local optimal solution is also a global optimal solution because of the convexity of the polytope. In the search, there might be multiple adjacent vertexes that improve the optimal solution. Thus, the simplex method applies a pivot rule to determine which vertex to select. An efficient rule can make this method quickly converge to the optimal solutions. Otherwise, the algorithm will spend long time to converge. Klee and Minty (1972) found that the simplex method may visits all vertices before arriving at the optimal vertex.

(ii) Ellipsoid method

The ellipsoid method was the first algorithm developed to solve the linear programs. It works by reducing the problem to a problem of feasibility. This method defines a polytope that is bounded by ellipsoids. The volume of these ellipsoids decreases at each iteration until the centres of the ellipsoids are in the polytope, or until the ellipsoids are too small. Then the central points are the global optimal solutions. This method has been proved to be the polynomial-time solvability of the linear programs and converge much quicker than simplex method.

(b) Nonlinear programming

Optimisation problems, especially engineering ones, often involve non-linear formulations either in the objective functions or in the equality or the inequality constraints. Approaches for nonlinear problems involve Lagrange multiplier method, iterative linearization method, iterative quadratic programming method, and penalty

function method. These methods apply Karush-Kuhn-Tucker (KKT) condition that provides necessary conditions for a solution to be optimal. The overviews of Lagrange multiplier method, generalized reduced gradient method, and sequential quadratic programming method (an iterative linearization method and an iterative quadratic programming method) are illustrated in the following sections.

(i) Lagrange multiplier method

The Lagrange multiplier method can be applied to nonlinear problems formulated by multivariable objective function and constraints. It uses the Lagrange multiplier to locate the optimal solution. The necessary condition for an extremum of an objective function is that the partial derivative of its Lagrange function with respect to variables and the multiplier must be zero. The algorithm then can easily find the optimum that is the same as that of the Lagrange function. The Lagrange multiplier method is limited on a small spectrum of NLP problems since it requires that the domain of a problem is an open set and the objective function and constraints of the problem must have continuous first partial derivative.

(ii) Generalized reduced gradient method

The generalized reduced gradient method is an iterative linearization method that firstly linearizes the problem and successively applies linear programming techniques. At each iteration, this algorithm linearizes the constraints and computes the reduced gradient to determine the search components for variables. The objective function is improved by changing the variables using these search components. Newton's method is then applied to regain feasibility of variables with respect to original constraints. The algorithm is considered to be in convergence if the search components can be arbitrarily reduced to infinite small. A major disadvantage of the generalized reduced gradient method is the requirement of feasibility of both the initial and the intermediated points. Thus, the algorithm has to spend long time to converge.

(iii) Sequential quadratic programming

The sequential quadratic programming method is considered to be the most efficient and powerful algorithm for NLP problems. In this method, the search direction is determined through minimizing the quadratic approximation of the Lagrangian function with linear approximation of the constraints. At each iteration, the Hessian matrix of the Lagrangian function is required to be updated to determine the search direction through a quadratic programming method. Schittkowski (1985) reported that the sequential quadratic programming method outperforms other nonlinear methods in terms of efficiency and accuracy over a large number of nonlinear problems.

(c) Mixed integer programming

Some complicated problems involve discrete binary variables that represent the existence of process units and streams (Achenie & Biegler 1990, Kokossis & Floudas 1990). MILP and MINLP are two techniques devised for these problems. Typical methods for these problems involve Branch and Bound, Generalized Benders Decomposition and outer Approximation.

(i) Branch and Bound

This method is an iterative algorithm. At each iteration, the branch step splits the search space to different sub spaces. The Bound step computes the upper and lower bounds for the global optimal solutions within the sub spaces. Then the algorithm discards the sub spaces if their lower bounds are greater than the upper bounds of the others. The recursion terminates when the search space is reduced to a single element or the upper bound of the search space matches its lower space.

(ii) Generalized Benders Decomposition

The major concept of Generalized Benders Decomposition (Benders 1962, Geoffrion 1972) is to decompose the application problems into the primal and the master sub-problems using the approximation methods. During the processing of this method. the system alternates between the solutions of two sub-problems. The master problems use non-linear duality theory and the Lagrange multipliers obtained in the primal problems to anticipate the integer variables. In addition, the master problem also generates the lower bound for the optimal. The primal problem set these integer variables according to the anticipation and solving the non-linear problems to find the upper bound for the optimal solutions. As the method proceeds, the lower bound increases and the upper bound decreases. The optimal solution can be found if the two bounds are close enough.

(iii) Outer Approximation

Based on the study of Benders and Geoffrion, Duran and Grossman (1986) proposed another method that is similar to Generalized Benders Decomposition. In this method, the application problems are also decomposed to the nonlinear and the mixed integer linear problems to formulate the upper and the lower bounds on the solutions. However, this method uses outer approximation (linearisation) of the nonlinear objective function and the constraints around the primal solutions to generate the mixed integer linear problems. This method is faster than Generalized Benders Decomposition method when the application problems are small-scale but slower when the problems are large-scale (Biegler *et al.*, 1997).

Deterministic methods suffer from a number of shortcomings that restrict its applications to complex engineering problems. In addition, the methods are highly sensitive to the initialisation of variables, especially for the non-linear problems and the non-convex problems. Starting from different initial solutions, these methods are likely to converge to

the different optimal solutions. So for the complex engineering problems, deterministic methods are likely to converge to local optimal or non-optimal solutions or fail to converge at all. In contrast, stochastic optimisation methods can avoid these shortcomings and will be reviewed in the following sections.

2.3.2 Stochastic methods

Stochastic methods refer to the minimization (or maximization) of the objective functions in the presence of randomness in optimisation processes. Recently, stochastic optimisation methods are becoming more and more popular in the engineering domain. In contrast to deterministic methods, stochastic methods follow a statistical random probabilistic driven search to explore the entire search space. In some cases, this random search process can speed up the convergence and make the algorithm less sensitive to the modelling errors. That is because the random search allows for the movements to unexplored areas of the search space that may contain a good solution. The advantages and disadvantages of stochastic methods have been reviewed by Arsham (1998), Fouskakis and Draper (2002), Fu (2002), Gosavi (2003), Michalewicz and Fogel (2000), and Spall (2003). In this thesis, four representative stochastic methods are reviewed involving Simulated Annealing, Tabu search, an Evolutionary method (Genetic Algorithm) and a new probabilistic computational optimisation method (Ant Colony).

(a) Simulated annealing

Simulated annealing algorithm (SA) is based upon the physical analogy of a cooling crystal structure that attempts to arrive at some stable (globally or locally minimal potential energy) equilibrium. Through a slow annealing procedure from a high temperature to the freezing temperature, a crystal can be restructured to the form that has the minimum energy level. This behaviour was simulated by Metropolis (Metropolis et al. 1955) as a meta-heuristic optimisation algorithm and firstly applied to real optimisation

problems by Kirkpatrick (Kirkpatrick et al., 1983) and Cerny (1985). SA was successfully applied to continuous reactions and separation systems (Floquet, Pibouleau, & Domenech, 1997; Marcoulaki & Kokossis, 1999; Meta & Kokossis, 1997, 1998; Cardoso, Salcedo, Feyo de Azevedo, Barbosa, 2000), Batch distillation (Hanke & Li, 2000), the flowsheet optimzation (Painton & Diwekar, 1995; Chaudhuri & Diwekar, 1996,1997), batch scheduling (Das, Cummings, & LeVan ,1990; Patel, Mah, & Karimi, 1991; Wang et. al 1999), energy networks (Dolan, Cummings, & LeVan, 1989, 1990; Maia, Vidal de Carvalho, & Qassim, 1995) and molecule design (Marcoulaki & Kokossis, 2000a, b)

The algorithm is recursive starting from an initial state at high temperature. Evolutions of states are carried out using the Markov process towards both better states as well as worse states according to the acceptance criterion. At each temperature, reversible state transitions (homogenous Markov processes) are performed to equilibrate the system. Temperatures are reduced according to a cooling schedule. Based on an acceptance criterion, transitions to the worse states are increasingly reduced with the decreasing temperatures. SA is likely to converges to the global optimal solutions if the temperature reduces according to $1/\log t$, with T_t close to 0 and t is infinite (T_t: temperature at iteration t). This convergence of SA can be proved by a statistical argument (Marcoulaki & Kokossis, 1999; Aarts & van Laarhoven, 1985).

Markov process

In SA, successive transitions are performed from a current state to a new neighbouring state following the Markov process. The definition of Markov process is:

> *A Markov process ($\{X_n\}$) is a stochastic sequence of events, where the probability of any particular future behaviour of the process, when its current*

state is known exactly, is not altered by additional knowledge concerning its past behaviour (Trivedi, 1982).

The formulation of the Markov property (Taylor and Karlin, 1980) is:

$$\begin{aligned} &T\{x_{n+1} = j \mid x_n = i\} = \\ &T\{x_{n+1} = j \mid x_0 = i_0, \ldots, x_{n-1} = i_{n-1}, x_n = i\}, \forall i_k, i, j \in S, \quad \forall n, k \in \mathrm{N} \end{aligned} \tag{2.7}$$

where

S : the search space,

i_k, i, j : the states in the space,

n: the time sequence.

According to the literature of Iosifescu (1980), the transition matrix ($\{T_{i,j}\}$) is a stochastic matrix of non negative elements in which the sum of entries in each row i is equal to unity. Each element ($T_{i,j}$) of the stochastic matrix stores the transition probability from states i to other states j. Figure 2.1 outlines the flowchart of SA.

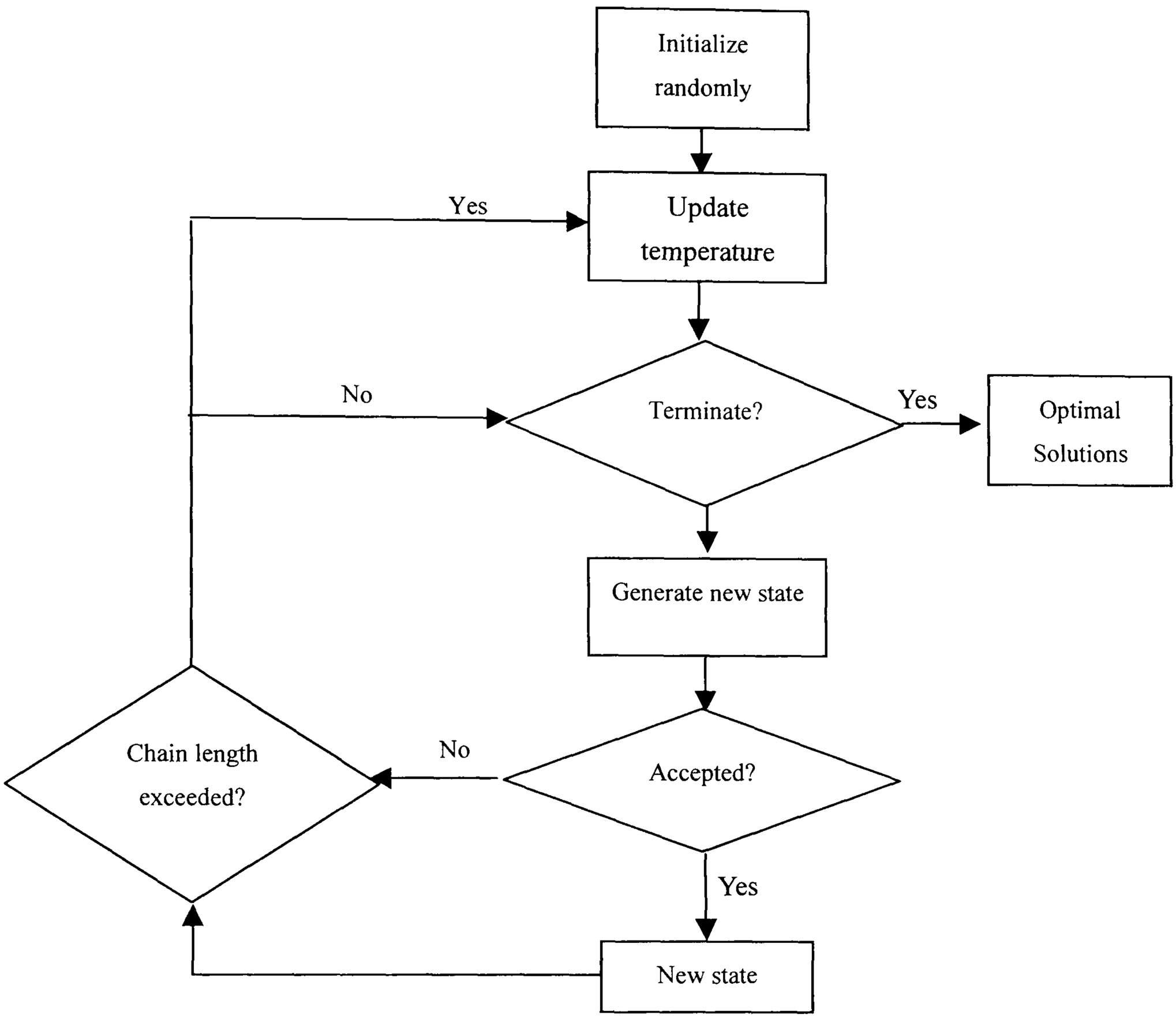


Figure 2. 1 SA

Acceptance criterion

The new generated states are accepted or rejected according to the Metropolis criterion (Metropolis et al., 1953). The major element in the Metropolis criterion is the acceptance probability that is formulated as:

$$P_{i,j} = \exp(-(f_j - f_i)/T) \tag{2.8}$$

where

$P_{i,j}$: the acceptance probability for the transition from state i to state j,

f_i and f_j : the objective function values of state i and state j ,

T : the annealing temperature.

Depending on Eq. 2.8, the Metropolis criterion can be formulated as:

$$P_{i,j} = \min(1, \exp(-(f_j - f_i)/T)) \tag{2.9}$$

Based upon Eq. 2.9, state j can be accepted with the acceptance probability ($P_{i,j}$) equal to 1 if f_j is smaller than f_i. In contrast, state j can be accepted with the acceptance probability ($P_{i,j}$) less than 1 if f_j is larger than f_i. One important feature of the Metropolis criterion is that the acceptance probability for the transition to a worse state (f_j is larger than f_i) is nonzero. This feature can prevent the algorithm from being stuck in local optimal solutions. As the temperature decreases, the acceptance probability of accepting the worse states is reduced. When the temperature is close to zero, only the better states can be accepted and SA becomes the Greedy algorithm (Black, 2005).

Cooling schedule

The cooling schedule involves several important parameters that include:

- Initial temperature.
- Decrement functions.
- Final temperature
- Length of the Markov process.

The initial temperature is usually assigned a high value so that all generated states can be accepted involving both the better ones and the worse ones. In contrast, the final temperature is usually low close to the freezing temperature ($T = 0$) so that only the better states can be accepted in transitions. The decrement function determines the decrement of the temperature at each iteration. In the analogy to a physical process, the system can be easily equilibrated when the temperature is high. With the low temperature, the equilibrium of the system is difficult to approach. An appropriate decrement function allows the temperature to decrease quickly at the high temperatures and slowly at the low temperatures. There are many different cooling schedules proposed and studied in the literature. However, these cooling schedules are generally classified into the static cooling schedules and the dynamic cooling schedules. In the former one, all parameters are predefined and cannot be changed in the process. One of the static cooling schedules is the exponential cooling schedule proposed by Kirkpatrick (Kirkpatrick et al, 1982),

$$T_{t+1} = \alpha T_t \tag{2.10}$$

where

α : a constant close to, but smaller than 1.

In contrast, the dynamic cooling schedules have their parameters adaptively changed in the process. A logarithmic cooling schedule is proposed by Aart and Van Laarhoven (1985). The formulation of the logarithmic cooling schedule is:

$$T_{t+1} = T_t \left[1 + \frac{T_t \ln(1+r)}{3\sigma^t} \right]^{-1} \tag{2.11}$$

where

σ^t : the standard deviation of the state objectives at each iteration,

r : a parameter to control the reducing rate of temperatures.

With a large r, the temperature is reduced quickly and SA could quickly converge to the optimal solutions but with low solution qualities (which is measured by objective value). In contrast, with a small r, temperature is reduced slowly and SA has to spend long computation time to converge to optimal solutions. The length of the Markov process can also affect the performance. In general, SA requires the long Markov process at the low temperatures to achieve the system equilibrium. SA uses either the static Markov process or the dynamic Markov process. With the static Markov process, SA uses the Markov process of the same length throughout. With the dynamic Markov process, SA firstly uses the short Markov process when the temperatures are high, then the length of the Markov process increases as the temperatures decrease. This is because at the high temperatures the system is easy to equilibrate so that the short Markov process is selected. However, at the low temperatures, the system is difficult to equilibrate so that SA uses a long Markov process.

Termination criterion

The SA terminates if one of the following termination criteria (Marcoulaki & Kokossis, 1999; Meta, 1998) are met:

- The temperature falls below the freezing temperature.
- No state can be accepted for a number of iterations.
- The maximum number of iterations have been completed.

Remarks on Simulated Annealing

SA has been proved to be a robust optimisation method even for the complex problems that have a large number of variables and local optima (Romeo et al., 1984; White, 1984)

when compared with the low efficiency and limited reliability of the other global optimisation methods (Dixon et al., 1975; Dixon et al., 1978; Masri et al., 1980; Pronzato et al., 1984). However, it is computationally expensive and requires excessive time to solve complex problems (Linke & Kokossis, 2003b). The most important issues are to address the controls in the parameters, such as the initial temperature, the final temperature, the decrement function, and the length of Markov process. SA has been applied to heat exchanger networks (Dolan et al, 1990), separation (Floquet et al, 1994), flowsheet optimisation (Painton & Diwekar, 1995), reactor network synthesis (Marcoulaki & Kokossis 1999, Cordero et al. 1997) and liquid-liquid extraction (Papadopoulos & Linke, 2004).

(b) Tabu search algorithm

Tabu search (TS) is a member of the family of local search methods. It uses both the memory structure and artificial intelligence to enhance the performance (Glover, 1989, 1990, 1993). Applications of TS are wide and cover the realms of resource planning, telecommunications, VLSI design, financial analysis, scheduling, space planning, energy distribution, molecular engineering, logistics, pattern classification, flexible manufacturing, waste management, mineral exploration, biomedical analysis, environmental conservation and scores of others. Similar to SA, TS is a recursive method. At each iteration, a state transition is carried out from the current solution to one of its best neighbour state. Figure 2.2 outlines the flowchart of TS.

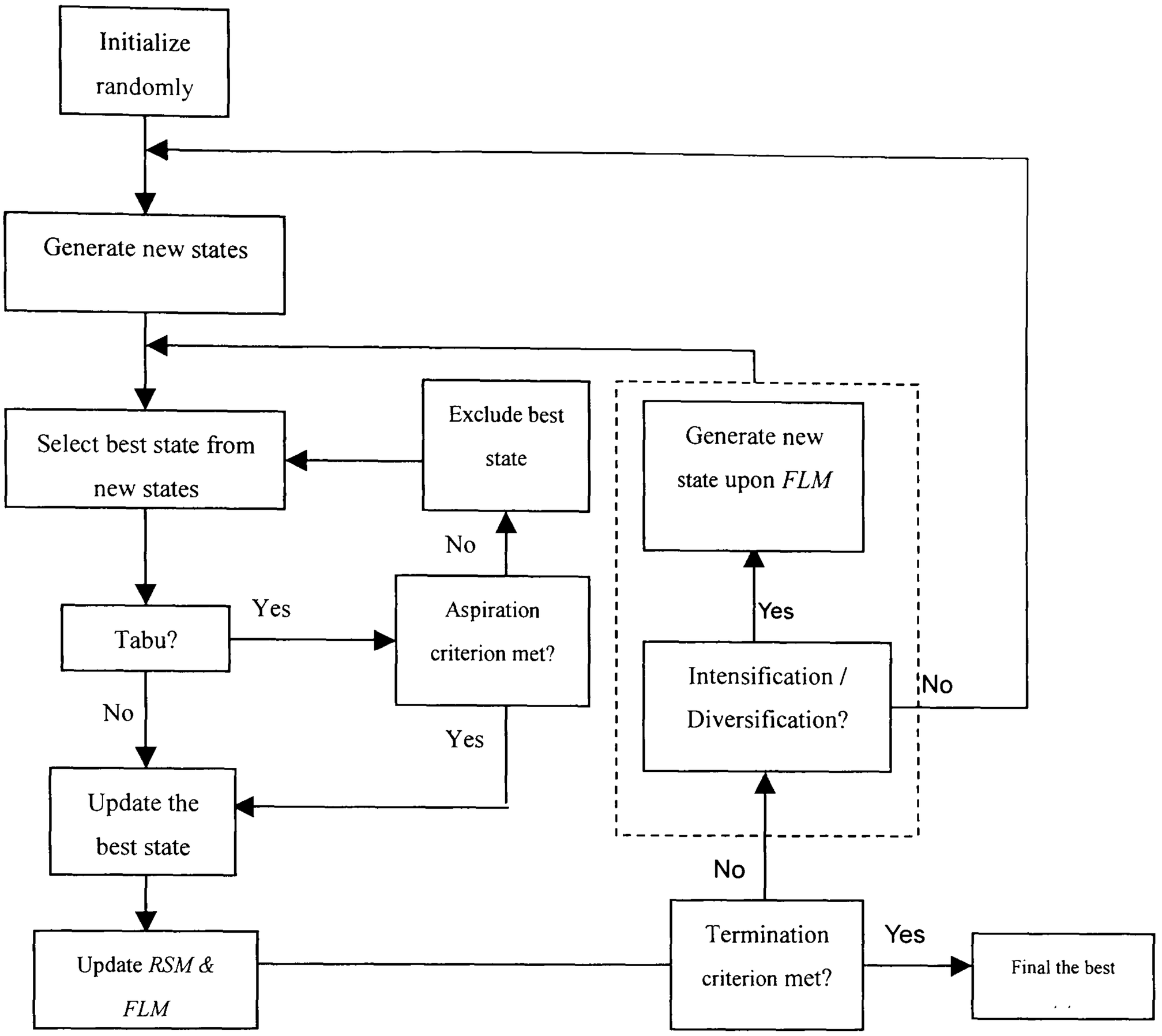


Figure 2. 2 TS

RSM and FLM

TS employs the memories to avoid the solution transitions that might lead to a visited solution. There are two types of memory. One is the recent-based short-term memory (RSM) and the other one is the frequency-based long-term memory (FLM).

RSM is incorporated into TS in the form of a Tabu list. It records the latest transitions that have executed and assigns them with Tabu status. Then the Tabu transitions are excluded to force the search away from the visited states and avoid the search being entrapped into a cycle. The size of the Tabu list (L^{TB}) is a user-specified parameter. It determines the number of recorded transitions. The Tabu list is updated at each transition

and the FIFO (first-in-first-out) discipline is used to limit its size. In contrast to the short-term memory of the Tabu list, FLM enables TS to learn the experience from the past transitions. It records the frequencies of the transitions that have already executed. Depending on these frequencies, the search in TS has bias either to the promising regions that involve the good solutions or to the regions that have never been visited before.

Intensification and Diversification

Depending on FLM, TS applies the Intensification and the Diversification search schemes (Intensification and Diversification) to improve the performance. The idea of Intensification is to pay more efforts on the promising regions where the good solutions are likely to appear. With Intensification, TS can quickly converge to the optimal solutions. In contrast to Intensification, Diversification leads the search away from the regions that have been visited with a high frequency. Thus, the search can explore un-visited regions. To use Intensification and Diversification, TS does Intensification for a number of iterations (L^{IT}) and switches to Diversification for a number of iterations (L^{DV}). Then TS conducts random transitions for a number of iterations (L^{RD}). Deciding the optimal ratio between the intensification period and the diversification period (L^{IT} / L^{DV}) is a challenge for the successful implementation of TS.

Aspiration criterion

The main objective of RSM is to avoid the search re-performing the transitions that have been performed in the recent past. However, some transitions with the Tabu status might lead to some good unvisited states. To allow these transitions, it is necessary to override the Tabu list. The Tabu list can be overrided when:

- The best neighbour in the transition is better than the best state.
- All neighbours in the transitions are Tabu.

Termination criterion

Glover et al. (1993) proposed the termination criteria that include:

- The best state stays same for a certain number of iterations.
- Total number of the iterations is larger than the maximum number of iterations.

TS terminates if either of these conditions are satisfied.

Remarks on Tabu Search

TS is an effective and efficient optimisation method. It can produce the optimal solutions that significantly surpass those obtained by some other methods. However, there are also some challenges in the study of TS.

First and foremost, it is a challenge to decide the size of the neighbourhood at each iteration. If the size were too big, TS has to estimate a large number of neighbour states at each iteration and spend long time. In contrast, TS can quickly converge but to the states with the low qualities, if the size were too small. To find an appropriate neighbourhood size, Wang (Wang et al., 1999) proposed a dynamic neighbourhood size scheme. In such scheme, the search starts with a small neighbourhood size. The size keeps increasing as time wears on. The results presented by Wang (Wang et al., 1999) indicated that TS with the dynamic neighbourhood size can converge to similar optimal solutions to those obtained by TS with the large constant neighbourhood size.

Secondly, deciding the size of Tabu list (L^{TB}) is another challenge. When L^{TB} is too small, TS might be entrapped in the non-optimal region since it executes too many

similar transitions. In contrast, a big L^{TB} allows the search covering the whole search space. However, the search is distracted by exploring the search space and can not quickly converge to the optimal solutions. Based on the studies over the past decades, it is shown that TS converges to the good optimal solution when L^{TB} is set to 7 (Ashley, 2004).

Finally, it is also difficult to determine the optimal ratio between the Intensification period and the Diversification period. However, current applications of TS follow a common discipline. This discipline provides that TS uses more Diversifications at the beginning to explore the entire search space. Then TS changes to more Intensifications to probe into the promising regions to find the global optimal solutions. TS that observes this discipline have been applied to the engineering problems that include plant process design (Wang et al, 1999, Cavin et al, 2004), heat exchanger networks (Lin & Miller, 2004) and reaction/separation systems (Linke & Kokossis, 2003a).

(c) Genetic algorithms

Genetic Algorithm (GA) is an evolutionary method that utilizes the technique inspired by evolutionary biology such as inheritance, mutation, selection and crossover. GA has been applied to a wide range of engineering problems that include dynamic control of processes (Pham, 1998), molecular design (Venkatasubramanian et al, 1994) and so on. Figure 2.3 presents the flowchart of GA.

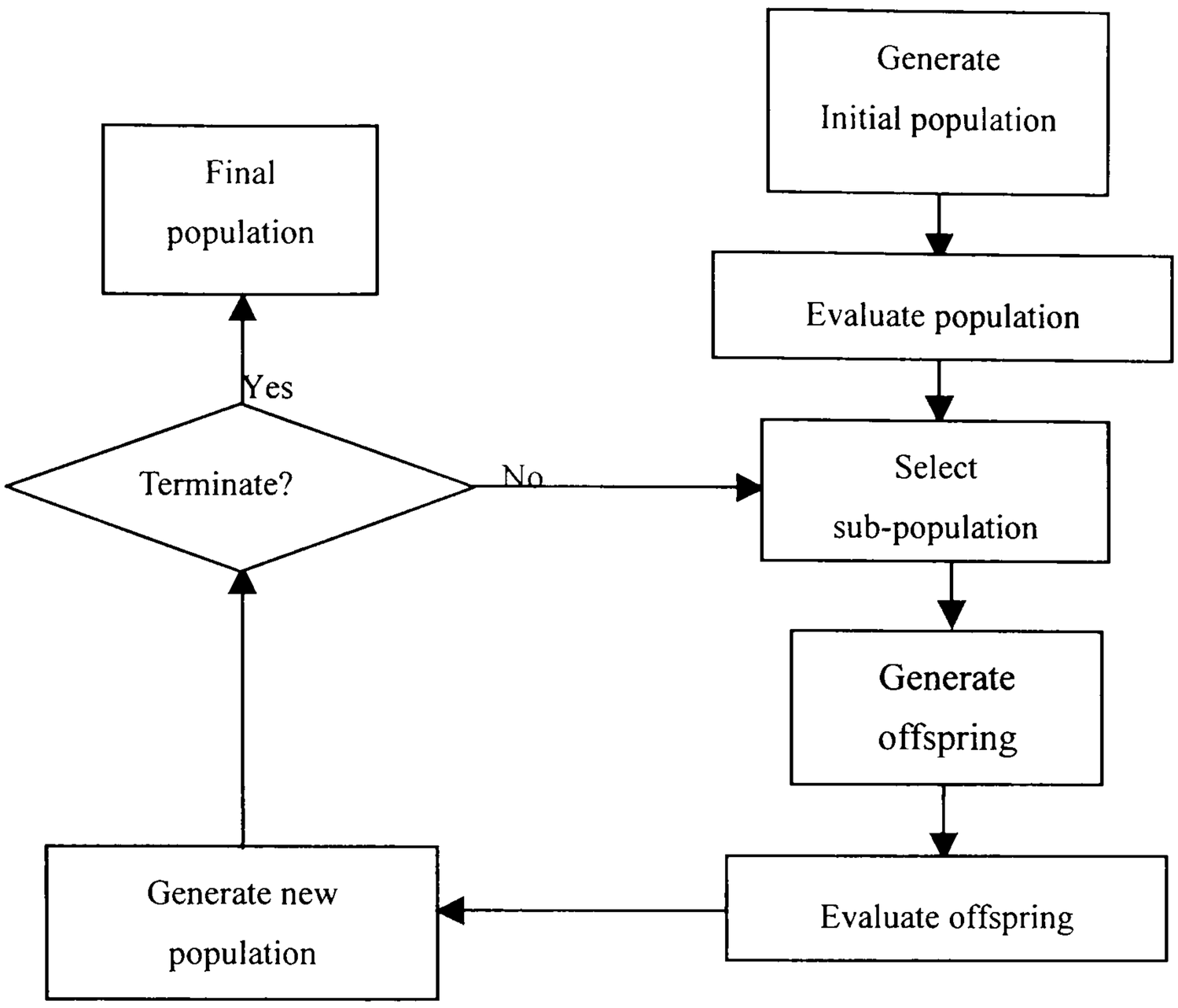


Figure 2. 3 GA

Initial population

Firstly, a number of initial solutions are generated to form the first population. Then the solutions in the population are interpreted into the chromosomes using the binary representation.

Genetic operation

GA employs the operations inspired by the evolutionary biology to generate new populations. These operations involve selection, crossover, and mutation.

Selection operation

The selection operation selects a sub-population from the population to produce offspring. There are a number of different ways to select the sub-populations. These ways include the tournament selection method, the roulette wheel selection method and so on. The

tournament selection method has been proved more efficient than other selection methods.

Reproduction operations

The reproduction operation generates a new population. The operations include the crossover operation and the mutation operation. The idea of the crossover operation is that a biological system combines the genetic genes of the fittest individuals to generate offspring and spread their genetic information over populations. There are numbers of different crossover operations, e.g. the one-point crossover operation, the two-point crossover operation, and the uniform crossover operation. On the other hand, the mutation operation is analogous to the biological mutation. New information is introduced into the population by this operation. A typical mutation operation probably selects an arbitrary bit in an organism strings and changes it. The aim of the mutation operation is to avoid being trapped in the local optimal solutions by introducing the diversities in the evolution of populations.

Termination criterion

The convergence of GA is reflected by the termination criteria that include:

- The number of iterations approaches a maximum number.
- The best solution the stays same for a number of iterations.

Manual inspection of the population can also be used to check the convergence.

Remarks on Genetic Algorithm

GA is a population based mate-heuristic algorithm so that the performance of GA relates to the size of population. With a large population, GA can converge to the good optimal

solutions but requires a long computational time. In contrast, GA can quickly converge but to the optimal solutions with low qualities when the population is small. Carroll (1996) and Goldberg *et al.*, (1992) proposed a rule to estimate the magnitude of the population. The formulation of this rule is:

$$M = \frac{l}{k} x^k \tag{2.12}$$

where

l : the length of the chromosome string,

x : the number of the possibilities for each chromosome,

k : a parameter that defines the average length of the chromosomes

Although Eq. 2.12 gives a good approximation to the optimal population size, population based optimisation methods still require a long computation time to evaluate solutions at each iteration.

(d) Ant colony optimisation

Ant colony optimisation algorithm (ACO) is a new stochastic optimisation method proposed by Dorigo (Dorigo, 1992). ACO has been applied to the engineering problems that include batch scheduling (Jayaraman et al., 2000) and dynamic optimisation of chemical processes (Rajesh et al., 2001). The basic idea of ACO is derived from the behaviour of ants that try to find the shortest path from their nest to food sources. Figure 2.4 illustrates the flowchart of ACO.

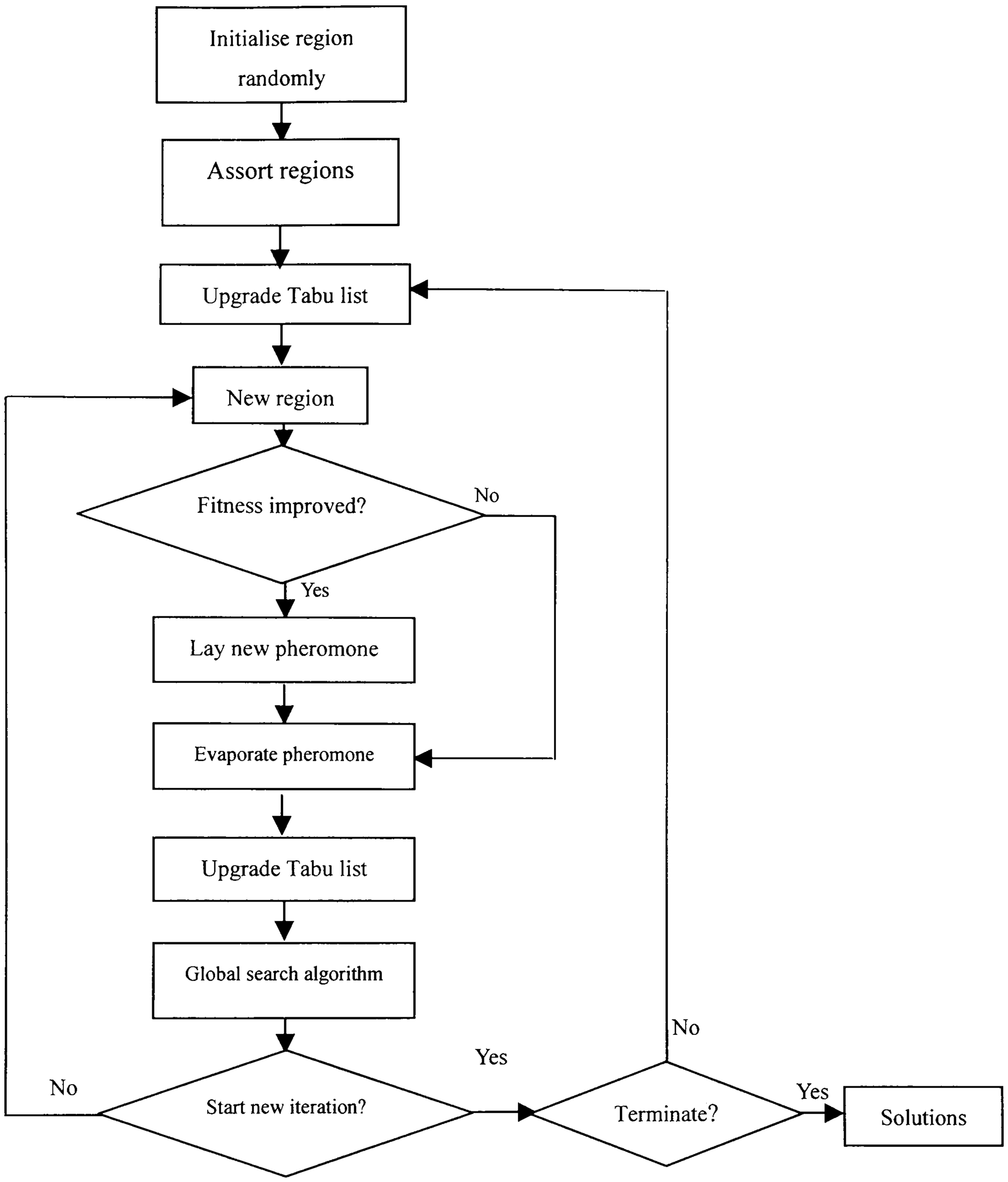


Figure 2. 4 ACO

Initial pheromone

ACO firstly generates a set of initial solutions in different regions. Each region is assigned a small pheromone value (τ_0) calculated based upon the formulation proposed by Gambardella et al., (1999):

$$\tau_0 = \frac{1}{C} \tag{2.13}$$

where

C : the best objective value.

Establish new solution

At each iteration, ants move to a new region according to a probability:

$$P_j = \frac{\tau_j}{\sum_k \tau_k} \qquad j,k \in \mathrm{N} \tag{2.14}$$

where

P_j : the probability for an ant moving to region j,

τ_j and τ_k : pheromones of region j and region k.

Pheromone update

The pheromone of a region increases if an ant passes the region and lays some pheromone. On the other hand, the pheromone of this region also evaporates as time elapses. Bonabeau (Bonabeau, et al. 1999) proposed a pheromone-changing schedule. The formulation of this schedule is:

$$\tau_{j,t+1} = \begin{cases} \tau_{j,t}(1-\rho) + \Delta\tau_j & \text{Fitness improve} \\ \tau_{j,t}(1-\rho) & \text{otherwise} \end{cases} \tag{2.16}$$

where

$\tau_{j,t}$: the pheromone of region j at time t,

ρ : the evaporation rate.

and

$$\Delta\tau_j = \sum_{i=1}^{M} \Delta\tau_i^j \tag{2.17}$$

where

$\Delta\tau_i^j$: the unit pheromone that an ant i lays at region j,

M: the number of ants in ACO.

Termination criterion

Conventionally, ACO terminates after a predefined maximum number of iterations complete. However, it is difficult to find a suitable maximum number. Papadopoulos and Linke (2004) proposed a new termination criterion providing that ACO terminates if the standard deviation of the objectives stays small for a number of iterations. The study reported that such termination criterion could save 86% of CPU time with no compromise to the quality of the optimal solutions.

Remark on ACO

In ACO, the number of ants is a significant parameter. With a large number of ants, ACO can converge quickly but to non-optimal or locally optimal solutions or fail to converge at all . On the other hand, ACO might not produce the expected synergistic effects of cooperation due to pheromone decay (Bonabeau et al., 1999) when ACO employs a small number of ants. Another important parameter is the evaporation rate (ρ). Papadopoulos and Linke investigated the impact of ρ and find ACO can reach the best convergence when $\rho = 50\%$.

2.4 Remarks on optimisation and optimisation methods

Optimisation aims to explore the best solutions to problems by exploiting the degrees of freedom in order to maximise (or minimise) the objective function subject to the equality and / or the inequality constraints. Optimisation problems can be classified based upon the continuity, the linearity and the convexity of the objective function or the constraints. A large number of optimisation methods have been proposed according to the classification of optimisation problems. These methods include both deterministic methods and stochastic methods. The former ones use the topological or the geometrical techniques to solve optimisation problems. By using these techniques, deterministic methods can quickly converge to optimal solutions. However, with these techniques, deterministic methods are limited to a small spectrum of problems, e.g. some methods require the objective functions of application problems differentiable. In addition, deterministic methods might converge to a local optimal solution when the applications are complex and involve non-convex and non-linear formulations. In contrast, stochastic methods follow a statistical random probabilistic driven search. With the randomness in search, stochastic methods can fully explore the search space and exclude the locally optimal solutions. However, with the random search, stochastic methods require a long computation time to converge.

To solve these shortcomings, previous efforts focused either on the optimisation methods themselves or applying advanced computing techniques. Recently, with the development of advanced computing techniques, more and more attention is paid on integrating these techniques to optimisation. In next chapter, a few of these computing techniques are reviewed along with their applications to optimisation techniques.

Chapter 3 Optimisation and computing

Different computing techniques have been applied to optimisation, aiming at exploiting key features and guiding the search efficiently through a parallel use of knowledge acquisition and distributed computing techniques.

3.1 Knowledge acquisition

Knowledge acquisition techniques transform the knowledge available in the world to the forms that can be used by a knowledge-based system. It is also called knowledge discovery that is defined by Fayyad et al., (1996) as the process of identifying valid, novel, potentially useful, and understandable patterns in existing data. Data mining is an important method in knowledge acquisition and was originally utilized in business intelligence and financial analysis. With data mining methods, relationships within the real world data can be identified so that non-statistician can help on decision making (Monk & Wagner, 2006). Moreover, data mining methods can also extract the information from the data generated in experiments. Two Crow Corporation (1999) reviewed the different data mining techniques involving neural networks, decision trees (classification trees), rule induction, K-nearest neighbour and memory-based reasoning, logistics and so on. In addition, several standards for data mining techniques are proposed, e.g. the CRISP-DM standard and the Java Data-Mining Standard.

Data mining methods have been applied to many different fields. In the medical field, data mining techniques are broadly applied to the pharmaceutical development and the construction of ecological models (Dzeroski *et al.*,1997). In the chemistry and chemical engineering fields, data mining techniques are used in discovery of knowledge from the chemical databases (Liang & Gan, 2001), in reaction data mining (Wang *et al.*, 2001), and in process modelling and optimisation applications (Nandi *et al.*, 2003). Ashley and

Patrick (2004) proposed a knowledge driven optimisation method in which the optimisation technique combines with a knowledge acquisition method. This method takes the form of TS and uses the classification tree data mining method to extract the rules. Then the algorithm follows these rules and focuses on the high performance regimes. The method has been proved successful for a number of illustrative engineering problems. It outperforms the robust stochastic optimisation algorithm (TS) in terms of the convergence speed. The study reported the usefulness of integrating the knowledge acquisition into optimisation.

3.2 Parallel and distributed computing

Parallel computing is a form of computation in which many calculations are carried out simultaneously. From the 1990s, parallel computing techniques had been applied on the computing environment in which computers are connected via a network. Distributed computing is a variation of parallel computing. With the distributed computing techniques, the computers in the distributed environment work closely like a single computer. The information is communicated via the network. Parallel computing and distributed computing techniques have been applied to different problems, e.g. engineering problems (Cox, 2002) and optimization problems (Foster et al, 1999; Foster et al, 2001).

In optimisation, parallel and distributed computing techniques can improve the computation capability so that the optimisation method can solve previously unsolvable complex problems. With the improved computation capability, optimisation methods can quickly converge to the global optimal solutions. To apply parallel and distributed computing techniques, the optimisation method firstly breaks the problem into several independent parts so that multiple computers can execute these parts simultaneously. Since stochastic optimisation methods use random probabilistic approaches to determine the search direction, the relations between the current transition and the previous transitions are not as strong as those in deterministic optimisation methods. Thus,

stochastic optimisation methods are more suitable for parallel computing techniques. In the following sections, the applications of parallel and distributed computing techniques to the four stochastic optimisation techniques (SA, TS, GA, and ACO) are reviewed in terms of their classifications, advantages and limitations.

(a) Parallel Simulated Annealing (SA)

Parallel computing techniques have been applied to SA in many different ways. These applications can be classified into the parallel SA with serial transitions, the parallel SA with parallel transitions, and the parallel SA with both serial and parallel transitions depending on the information transition modes (Leite and Topping, 1999).

The parallel SA with only serial transitions requires that the method rigorously follows the normal SA, so it is unlikely to produce a significant speedup on convergence. However, such a scheme is still broadly used since the motivation of SA is derived from the stochastic nature which requires multiple runs to improve the reliability of the results. To implement this form of parallel SA, each computer is assigned its own SA. The initial points (the global information) are firstly distributed to the computers to initialise the SA. Then the computers execute the SA independently with no information transferred between them.

In contrast, the parallel SA with only parallel transitions is implemented as the master-worker paradigm. The workers are assigned by different Markov processes. The Markov processes are launched simultaneously. The master collects the objectives on the workers and then returns the best objective to the workers after all Markov processes finish a transition. This parallel SA might encounter bottlenecks and high communication traffic when a large number of workers are employed and all transfer information with the master at the same time.

The last form of parallel SA is the one with both serial transitions and parallel transitions.

The master-worker computing framework is still selected. The workers do a number of transitions in the Markov processes and then send their best objectives to the master. The master then selects the best one of these objectives and distributes it to workers to restart the Markov processes. Compared to the parallel SA with parallel transitions, this parallel SA reduces the communication between the workers and the master. However, the faster workers have to wait for the other workers to finish their work at each iteration. Thus, the computational resources available in the environment are not fully exploited.

A parallel SA (hybrid parallel SA) was proposed by Mahfoud and Goldberg (1992). It has become increasingly popular and can be found in variety of applications. This parallel SA combines SA with GA to introduce the population-based characteristics into an annealing framework. To implement the hybrid parallel SA, the master-worker computing structure is selected. The reproduction operations (the crossover and the mutation operations) execute on different workers in parallel. The new generated populations are evaluated on these workers and sent back to the master. The master collects the populations from the workers. Then the master rearranges the populations and broadcasts them back to the workers. The temperatures on the master are used to control the convergence of the method. However, the hybrid parallel SA might encounter bottlenecks and high communication traffic if the populations transferred between the workers and the master are large or the environment involves large numbers of workers.

(b) Parallel Tabu Search (TS)

Similar to the parallel SA, parallel computing techniques have been applied to TS in different ways. These applications involve the parallel TS with parallelism in cost function evaluation, the parallel TS with parallelism in problem decomposition, and the TS with parallelism in solution domain explorations.

The first parallel TS delegates evaluations of neighbours to different workers (Chakrapani and Skorin-Kapov, 1993). The master-worker computing structure is used to implement the parallel TS. The master searches the neighbours of the current points and distributes them to the workers, while the workers evaluate these neighbours concurrently and send the results back to the master. Since the most time expensive tasks execute on the workers simultaneously, the computation time can be significantly reduced. However, this parallel TS might encounter high communication traffic since the information is transferred between the master and workers after each move.

The second parallel TS firstly decomposes the application problem to several sub problems. These problems are distributed to the computers. The computers do TS in parallel to solve these sub problems. The best objectives are exchanged between the computers periodically. Glover *et al.*, (1993) applied this form of parallel TS to vehicle routing problems. Since the parallel TS has to firstly decompose the application problems, it is a problem dependent method.

In the last form of parallel TS, the master-worker computing structure is still selected. The workers do TS in parallel. The information that transfers between the workers and the master is a set of parameters that include the length of the Tabu list, the number of the neighbours and so on. To implement this parallel TS, the master generates a set of parameters for each worker after a number of moves. These sets of parameters are sent to workers to control the next moves. This parallel TS is suitable for multiple instruction and multiple data parallel computing with distributed memory (Grainic et al., 1995a; 1995b). However, since the faster workers have to wait for the slower ones, the computation resources cannot be fully exploited.

(c) Parallel Genetic Algorithms (GA)

Since GA is a population-based stochastic optimisation method, it can be easily parallelised. Till now, parallel GA can be found in different applications (Baluja, 1994;

Mariusz and Riccardo, 1999; Cantú-Paz, 1998). These parallel GA can be classified into the ones with parallelism in evaluations and the ones with parallelism in population (Cantú-Paz, 1998).

In the first form of parallel GA, the master-worker computing structure is selected. The workers evaluate the solutions and do mutation operations to generate new solutions in parallel. The master performs the selection and the crossover operations and distributes populations to the workers. In this parallel SA, workers take the most expensive time tasks and execute them in parallel. The overall computation time can be reduced. However, with a large number of workers in the environment, the communication traffic between the workers and the master is high and the reduction on the overall computation time is limited.

The second form of parallel GA distributes the population to the workers. The workers run GA in parallel. This form of parallel GA can be further classified into the parallel GA with distributed memories and the parallel GA with shared memory. In the first one, each computer runs a GA on a sub population and transfers members in this population with those in other populations from time to time. The division of population depends on topology techniques such as the hypercube and the multi-dimensional mesh. This parallel GA was firstly proposed by Cantú-Paz (1998). In the second one, the sub populations are in the same memory and the workers communicate with the memories periodically. Both sub classification of this parallel GA might encounter high communication traffic since the workers have to communicate frequently either with other computers or with the shared memory.

Moreover, yet a further form of parallel GA combines GA with some stochastic optimisation algorithms. Such parallel GA is named as the hybrid GA. Based on the discussion in parallel SA, the studies on the hybrid GA focus on combining parallel GA with SA.

(d) Parallel Ant Colony (ACO)

Most of the parallel ACO follow the idea proposed by Randall and Lewis (2002). In this idea, ants are distributed to different workers. The ants on the same computers are regarded as those from the same colonies. The parallel ACO include the centralized parallel ACO and the decentralized parallel ACO.

The centralized parallel ACO uses the master-worker computing structure. The master collects pheromone from the workers and sends the specified solutions and new generated pheromone to the workers. To implement this parallel ACO, ants are firstly distributed to different colonies on the workers. The master then distributes each worker an initial solution set and pheromone set. The workers then run ACO in parallel for numbers of iterations and generate new solution sets. The master collects these solution sets and pheromone sets and selects the best solutions and generates the new pheromone set. Then the master sends them to the workers to run ACO again. Since the workers have to communicate frequently with the master, the centralized parallel ACO might encounter high communication traffic loading. Piriyakumar and Levi (2002) studied this parallel ACO on the travel salesman problems and found that the communication time could be reduced to some extent by decreasing the frequency of transferring the information between the master and the workers. However, lessening communication frequency could lead the algorithm to the optimal solutions with low qualities.

In contrast to the centralized parallel ACO, the decentralized parallel ACO runs sequential ACO on individual computers. The information (the selected solutions and pheromones) is exchanged between computers after a certain number of iterations. Kruger and Middendorf (1998) studied this parallel ACO on the salesman problems and reported that the decentralized parallel ACO can achieve the optimal solution with high solution quality (objective value of the solutions close to that of the global optimal solution) and spend less time on communication than the centralized parallel ACO. However, the decentralized parallel ACO might in some circumstances encounter high

communication traffic bottleneck since the faster workers cannot exchange the information with the colonies on the slower workers until they all finish work. Another reason is that the information transferred between computers is large when the application problems are high dimensional that involve large numbers of variables.

The application of parallel computing techniques can undoubtedly improve the computation capability. However, with the synchrony and sequential nature in the conventional stochastic methods, computation resources cannot always be fully exploited. Thus the benefit in speeding up the convergence is limited. In addition, the number of available resources within a corporation is finite. With the development of advanced technologies, the parallel and distributed computing applications are no longer limited to the same location with a localised cluster of computers, but are instead being applied on the Grid networks across the Internet.

3.3 Grid technology and applications

Grid computing techniques were first developed to enable resource sharing, job dispatching and data mining in the decade from 1980s to 1990s (Foster & Kesselman, 1999). Recently, Grid computing techniques are used for many different objectives, e.g. integrated networking, communication, computation and information and help executing large-scale, resource intensive and distributed applications (Berman *et al.*, 2003). In Grid computing techniques, a Grid is described as an infrastructure that integrates networking, communication, computation and information to provide a virtual platform for resources sharing, message and command passing, data transfer and mining and task scheduling and dispatching in the fields of business, government, science and industries. Foster and Kesselman (1999) divided a Grid into 4 layers from bottom to top as follows: Fabric layer, Core middleware layer, User level layer and Application layer.

(i) Fabric layer

The Fabric layer provides the resources in Grid. The resources could be CPU power resources, storage, database, network portal, operating systems, etc.

(ii) Core middleware layer

This layer defines the core communication and the authentication protocols required for Grid-specific network transactions. The communication protocols enable the exchange of data between the resources in the fabric layer, whilst the authentication protocols provide the secure mechanisms for verifying the identity of administrators, users and resources.

(iii) User level layer

The user level lay is on top of the core middleware layer. It provides the protocols and services for capturing interactions across resources. This layer is made up of programming tools, resource brokers and deployment devices so that it can implement a wide variety of sharing behaviours.

(iv) *Application layer*

The application layer is the highest layer in a Grid. It is designed for users. This layer consists of the Grid programming models and the Grid application execution environment. The former one is used for users to program the applications, whilst the later one implements the remote controls in the Grid.

Grid computing techniques are firstly developed as meta-computing techniques involving the applications established on Internet protocols. Grids provide application-oriented frameworks with the emerging of Open Grid Services Architecture (OGSA) and shared

virtual systems. Users can access the resources remotely by using the information service and the resource broker layers. End-users do not have to know where the tasks perform and where the data is stored. Grid middleware are software packages that unify different computing and data resources in a specified manner. With these packages, the resources in a Grid can be accessed remotely by client software without incompatibility problems. Globus is a middleware developed as a metacomputing infrastructure toolkit providing capabilities and interfaces for communication, information, resource location, resource scheduling, authentication and data access (Foster & Kesselman, 1997).

In optimisation, Papadopoulos et al. (2005) applied modified TS on a simple Grid (a cluster computer network) in which computers were connected via a local network. Master-worker computing structure was selected in this application. The workers ran TS in parallel and sent objectives to the master periodically. The master then calculated the standard deviation of these objectives and broadcasted it to the workers. This standard deviation determines whether the algorithm selects a random direction or selects the user-defined direction. The user-defined direction is produced by analysing the previous moves using a data mining method. Introducing the user-defined direction can avoid the premature convergence of the optimisation method. Due to the high communication traffic between the master and the workers, this modified TS can gain at most seven times speedup at most when applying it on distributed computing environments.

3.4 Remarks and discussions

Different computing techniques (e.g. knowledge acquisition, parallel computing, and distributed computing) have been integrated in optimisation either to guide the search efficiently or to produce high-throughput computation capacities. To guide the search, optimisation applies knowledge acquisition techniques. However, most of the conventional stochastic optimisation methods do not record and manage the intermediate system information. To produce high-throughput computation capacities, parallel and distributed computing techniques are applied to optimisation. Because of the sequential

nature, it is difficult to apply parallel and distributed techniques to conventional optimisation methods. Thus, the conventional optimisation methods have developed passively exploitation of knowledge acquisition and parallel and distributed computing techniques. In addition, most of the current parallelization follows synchronous mode in which the faster worker has to wait for the slower ones before it starts work. This parallelization suffers from high communication traffic amongst computational resources and can only achieve small benefit in speeding up convergence. Thus, it is a challenge to develop a fully distributed optimisation method in which the computational resources run simultaneously but following asynchronous mode in which computers works as a single computer.

A fully distributed optimisation algorithm that can capitalize on knowledge acquisition techniques is proposed in our research. The concepts of this algorithm are explained in detail in the next chapter.

Chapter 4 The Cascade Optimisation Algorithm

A fully distributed stochastic optimisation algorithm is proposed by incorporating the concepts from the Markov process of SA. It introduces conceptual partitions and pools to store intermediate solutions and corresponding objective values. The partitions and pools grow as the Markov process generates new intermediate solutions. Populations within partitions and pools are distributed periodically according to an external criterion. With the use of partitions and pools, multiple Markov processes can executes simultaneously without interactions. Thus, the new algorithm is suitable for distributed computing technique. The general formulation of optimisation problems takes the form:

$$\min f(x,y) \qquad s.t.\begin{cases} h(x,y)=0 \\ g(x,y)\le 0 \\ x\in X\equiv \Re^n, y\in Y\equiv\{0,1\}^n \end{cases} \tag{4.1}$$

where $f(x,y)$ is the objective function, $h(x,y)$ and $g(x,y)$ are the constraints.

4.1 Notation

$A_{n_p,t}$	Domain of $D^h_{n_p,t}$.
d_t	Density functions in $A_{n_p,t}$.
d_z	A profile of $A_{n_p,t}$.
D	Domain of an optimisation problem.
$D^h_{n_p,t}$	Cascade of n_p pools at time t.
f	Objective function of an optimisation problem.
$f^{\min}_{j,t}$	Minimum f in $P_{j,t}$ at time t.

$f_{j,t}^{\max}$	Maximum f in $P_{j,t}$ at time t.
$f_t^{\min}$	Minimum f in $D_{n_p,t}^h$ at time t.
$f_t^{\max}$	Maximum f in $D_{n_p,t}^h$ at time t.
F	A superset of numerical functions, including objective function.
g_t^L	Markov process defined over time t.
G	Superset of partitions in D.
G_1	The highest partition.
G_{n_p}	The lowest partition.
$G_{j,t}$	Partition j at time t.
h	A numerical function in F.
$h_{j,t}$	A numerical function of $P_{j,t}$
h^{sm}	A measure of object (D, P_j).
$H_{j,t}$	A probability measure of $A_{W,t}$.
i	Index of solutions in partition j. $i = 1,2,....,M_j$
j	Index of partitions from top to bottom. $j = 1,2,...,n_p$
L	The length of Markov process.
M_j	Number of available points in $G_{j,t}$.
n_p	Number of partitions/pools
n_F, n_{ST}, n_d	Integer constants.
P_1	The highest pool.
P_{n_p}	The highest pool.
$P_{j,t}$	Pool j at time t.
Q_t	The population of G_t or $D_{n_p,t}^h$.

$Q_{j,t}$	The population of $G_{j,t}$ or $P_{j,t}$.
r	A parameter in the cooling schedule used to determine the shape of curve between T_1 and T_{n_p}.
R	A random number in [0, 1].
$s_{i,j,t}$	Available point i in partition $G_{j,t}$.
S	Available region of an optimisation problem.
t	Sequence of time periods. $t = 0,1,...$
T_1	The temperature associated with P_1.
T_{n_p}	The temperature associated with P_{n_p}.
T_j	Parameter associated with P_j and is called temperature here.
ε,	A small value in [0, 1].

4.2 Basic concepts

Let S be the feasible region of 4.1:

$$S = \{(x, y) \mid h(x, y) = 0 \wedge g(x, y) \leq 0, x \in X, y \in Y\} \tag{4.2}$$

and D is the domain of 4.1:

$$\text{D= } \{ f(x, y) \mid \forall (x, y) \in S \} \tag{4.3}$$

Let G be the superset of disjoint partitions in S, so that:

$$G \equiv \bigcup_j G_j \subseteq S \tag{4.4}$$

with G_j being ordered sets of finite elements in S:

$$G_j = \{s_{1,j}, s_{2,j}, \ldots\ldots s_{M_j,j}\} \quad s_{i,j} \in S \quad i = 1,2,\ldots,M_j, j = 1,2,\ldots,n_p \tag{4.5}$$

Let F be a superset of numerical functions. For each $h \in F$ including the objective function f of 4.1, $s_{i,j} \in S$ can be mapped so that:

$$s_{i,j} \xrightarrow{h} h(s_{i,j}) \equiv h_{i,j} \in \Re \tag{4.6}$$

Defined over G_j, f can be used to map G_j to $\Re^{M_j}$ so that:

$$f_j : \quad G_j \xrightarrow{f} f(G_j) = \{f(s_{i,j})\}_{i=1}^{M_j} \equiv \{f_{i,j}\}_{i=1}^{M_j} \tag{4.7}$$

and

$$f_j^{'} : \quad G_j \xrightarrow{f} f^{'}(G_j) = \{f(s_{i,j})\}_{i=1}^{M_j} \quad \wedge \quad f(s_{i,j}) \geq f(s_{k,j}) \quad \forall i < k \tag{4.8}$$

Definition 4.1: A set P_j is a pool of G if $\exists G_j \subset G$ so that P_j is the domain of 4.7.

Definition 4.2: A set P_j is an ordered pool of G if $\exists G_j \subset G$ so that P_j is the domain of 4.8

Illustration 4.1:

Let the optimisation problem be:

$$\min f = x_1^2 + x_2^2 + y \qquad s.t. \begin{cases} x_1 + x_2 = y \\ x_1 - x_2 y \leq 0 \\ x_1, x_2, y > 0 \end{cases}$$

with the feasible region:

$$S = \{(x_1, x_2, y) \mid x_1 + x_2 = y \wedge x_1 - x_2 y \leq 0, x_1, x_2, y > 0\}$$

Let A, B, C, D, E, F, G, H, I be available points in *S*, Table 4.1 presents these points.

Table 4. 1 Available points in *S*

Point	x_1	x_2	y
A	37.4	14.6	52
B	20.2	41.8	62
C	1.7	34.3	36
D	16.4	18.6	35
E	12.7	3.3	16
F	35.7	39.3	75
G	0.4	3.6	4
H	1.7	21.3	23
I	52.4	7.6	60

Let the superset of partitions $G = \{G_1, G_2, G_3\} \subseteq S$ with

$$G_1 = \{A, B, C\}$$
$$G_2 = \{D, E, F\}$$
$$G_3 = \{I, H, G\}$$

Figure 4.1 illustrates the partitions *G*.

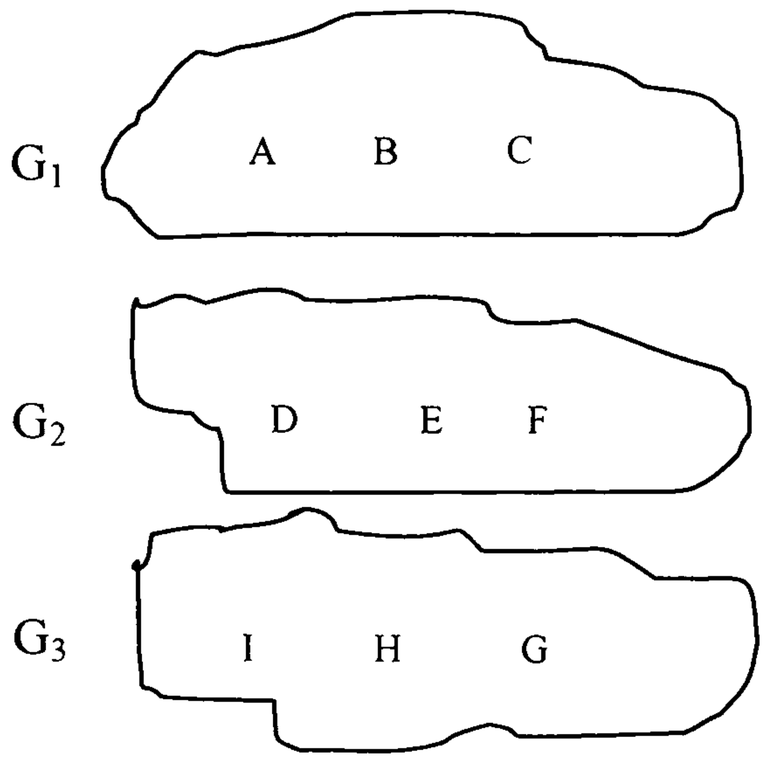


Figure 4. 1 Partitions G

According to f, G_1, G_2 and G_3 are mapped into P_1, P_2 and P_3

$$G_1 \xrightarrow{f} P_1 = \{1665.8, 2218.4, 1215.1\}$$

$$G_2 \xrightarrow{f} P_2 = \{649.8, 189.2, 2894.1\}$$

$$G_3 \xrightarrow{f} P_3 = \{2863.9, 479.5, 17.1\}$$

Based on *Definition 4.1* and *4.2*, P_1 and P_2 are pools and P_3 is an ordered pool.

Lemma A: G_j *is a* σ*-algebra of* G *and* P_j *is a* σ*-algebra of* D.

Proof:

In both 4.7 and 4.8:

- $\emptyset \in G_j$, $\emptyset \in P_j$
- Any union of countable many elements of G_j or P_j is an element of G_j or P_j.

- The complement of any element of G_j in G or P_j in D is an element of G or D.

Lemma B: The objects (G, G_j) and (D, P_j) are measurable spaces.

Proof:

From lemma A, G_j is a σ-algebra of G and P_j is a σ-algebra of D. QED

Definition 4.3: Q_j is the population of G_j iff Q_j is the rank of $P_j(G_j)$:

$$Q_j = rank(G_j) = rank(P_j) \tag{4.9}$$

Let P_j as defined in *Definition 4.1*. Let also an h-metric h_j defined over P_j as a mapping from $\Re^M \to \Re$ so that:

$$h_j : P_j \xrightarrow{h} h(P_j) = h\left(\{f_{i,j}\}_{i=1}^{M_j}\right) \tag{4.10}$$

Lemma C: The function

$$h^{sm}(P_j) = \sum_{i=1}^{M_j} \left| f(s_{i,j}) \right| \tag{4.11}$$

is a measure of the object (D, P_j).

Proof:

h^{sm} is a function: $P_j \xrightarrow{h^{sm}} R \bigcup \{\infty\}$ with values in the extended real numbers such that:

- $h^{sm}(P_j) \geq 0 \quad \forall P_j$
- $h^{sm}(\bigcup_j P_j) = \sum_j h^{sm}(P_j)$

Let $h \in F$ and a set of n_p pools P_j with their h_j metrics. Let an ordered combination $D^h_{n_p}$ so that:

$$D^h_{n_p} = (P_1, P_2, ..., P_{n_p}) \tag{4.12}$$

with

$$h_j > h_k \quad \forall j > k \tag{4.13}$$

Definition 4.4: A combination of n_p pools P_j is called a cascade if $\exists h \in F$ so that $D^h_{n_p}$ follows 4.13.

***Illustration 4.2*:**

Let us apply h_j on P_1, P_2 and P_3 of *Illustration 4.1* so that

$$h_1 = h^{sm}(P_1) = f(A) + f(B) + f(C) = 5099.3$$
$$h_2 = h^{sm}(P_2) = f(D) + f(E) + f(F) = 3733.2$$
$$h_3 = h^{sm}(P_3) = f(G) + f(H) + f(I) = 3360.5$$

Based on *Definition 4.4*, the combination

$$D^h_3 = \{P_1, P_2, P_3\}$$

is a cascade following

$$h_1 > h_2 > h_3$$

Definition 4.5: The cascade domain A_{n_p} *of* $D_{n_p}^h$ *is defined as:*

$$A_{n_p} = \bigcup_{j=1}^{n_p} \{f_{i,j}\}_{i=1}^{M_j} \tag{4.14}$$

Lemma E: $B(A_{n_p}, f_{i,j})$ *is a σ-algebra of* A_{n_p} *and*

$$H_j = \frac{h^{sm}(P_j)}{\sum_{j=1}^{n_p} h^{sm}(P_j)} \tag{4.15}$$

is a probability measure of A_{n_p} .

Proof:

$B(A_{n_p}, f_{i,j})$ is a σ-algebra since

- $\emptyset \in B(A_{n_p}, f_{i,j})$
- Any union of countable many elements of $B(A_{n_p}, f_{i,j})$ is an element of A_{n_p} .
- Each complement of $B(A_{n_p}, f_{i,j})$ is also an element of A_{n_p} .

$H(A_{n_p})$ is a probability measure since

- $H \geq 0$
- H has the countable addition property of h^{sm}

$$H\left(\bigcup_j h^{sm}\right) = \sum_j H\left(h^{sm}\right)$$

- $H(A_{n_p}) = 1$

Illustration 4.3:

The domain of D_3^h in *Illustration 4.2* is

$$A_3 = \{2218.4, 1665.8, 1215.1, 2894.1, 649.8, 189.2, 2863.9, 479.5, 17.1\}$$

Based on 4.15 and *Illustration 4.2*, the probability measures of A_3 are

$$H_1 = H(h^{sm}(P_1)) = \frac{h^{sm}(P_1)}{\sum_{j=1}^{3} h^{sm}(P_j)} = \frac{5099.3}{12193} = 0.418$$

$$H_2 = H(h^{sm}(P_2)) = \frac{h^{sm}(P_2)}{\sum_{j=1}^{3} h^{sm}(P_j)} = \frac{3733.2}{12193} = 0.306$$

$$H_3 = H(h^{sm}(P_3)) = \frac{h^{sm}(P_3)}{\sum_{j=1}^{3} h^{sm}(P_j)} = \frac{3360.5}{12193} = 0.276$$

Let $D_{n_p,k}^h$ a sequence of cascades and $A_{n_p,k}$ the domains of $D_{n_p,k}^h$.

Definition4.5: Let d_k be the density probability function of $A_{n_p,k}$. The d_z *is a "profile" of* $D_{n_p,k}^h$ *iff*

$$\lim_{k\to\infty} d_k = d_z \tag{4.16}$$

Illustration 4.4:

Let the unconstraint optimisation problem be:

$$\min f = x^2$$

Let the sequence of cascade $D_{3,k}^h = \{P_{1,k}, P_{2,k}, P_{3,k}\}$, associated with the sequence of partitions $G_k = \{G_{1,k}, G_{2,k}, G_{3,k}\}$ and

$$G_0 = \{G_{1,0}, G_{2,0}, G_{3,0}\} = [\{x_1\}, \{2x_1\}, \{7x_1\}]$$

Using f,

$$D_{3,0}^h = [\{x_1^2\}, \{4x_1^2\}, \{49x_1^2\}]$$

Let partitions and pools expand by

$$g_k : s_{i,j,k} \to s_{i,j,k+1} = \frac{s_{i,j,k}}{k+2}$$

The $G_{1,k}, G_{2,k}$ and $G_{3,k}$ are given by

$$G_{1,k} = \{x_1, \frac{x_1}{2}, \frac{x_1}{6}, \cdots, \frac{x_1}{(k+2)!}\}$$

$$G_{2,k} = \{2x_1, x_1, \frac{x_1}{3}, \cdots, \frac{2x_1}{(k+2)!}\}$$

$$G_{3,k} = \{7x_1, \frac{7x_1}{2}, \frac{7x_1}{6}, \cdots, \frac{7x_1}{(k+2)!}\}$$

Using f,

$$P_{1,k} = \{x_1^2, \frac{x_1^2}{4}, \frac{x_1^2}{36}, \cdots, \frac{x_1^2}{((k+2)!)^2}\}$$

$$P_{2,k} = \{4x_1^2, \frac{4x_1^2}{4}, \frac{4x_1^2}{36}, \cdots, \frac{4x_1^2}{((k+2)!)^2}\}$$

$$P_{3,k} = \{49x_1^2, \frac{49x_1^2}{4}, \frac{49x_1^2}{36}, \cdots, \frac{49x_1^2}{((k+2)!)^2}\}$$

The domain of $D_{3,k}^h$ is

$$A_{n_p,k}^h = \{x_1^2, \frac{x_1^2}{4}, \frac{x_1^2}{36}, \cdots, \frac{x_1^2}{((k+2)!)^2}\} \cup \{4x_1^2, \frac{4x_1^2}{4}, \frac{4x_1^2}{36}, \cdots, \frac{4x_1^2}{((k+2)!)^2}\} \cup$$
$$\{49x_1^2, \frac{49x_1^2}{4}, \frac{49x_1^2}{36}, \cdots, \frac{49x_1^2}{((k+2)!)^2}\}$$

Using the probability measure H_k as the density function d_k,

$$d_k(A_{n_p,k}) = H_k(A_{n_p,k}) = \left\{ \left[\frac{h^{sm}(P_{1,k})}{\sum_{j=1}^{3} h^{sm}(P_{j,k})} \right], \left[\frac{h^{sm}(P_{2,k})}{\sum_{j=1}^{3} h^{sm}(P_{j,k})} \right], \left[\frac{h^{sm}(P_{3,k})}{\sum_{j=1}^{3} h^{sm}(P_{j,k})} \right] \right\}$$

with

$$h^{sm}(P_{1,k}) = \frac{x_1^2 \sum_{l=1}^{k+2} l!}{((k+2)!)^2}$$

$$h^{sm}(P_{2,k}) = \frac{4x_1^2 \sum_{l=1}^{k+2} l!}{((k+2)!)^2}$$

$$h^{sm}(P_{3,k}) = \frac{49x_1^2 \sum_{l=1}^{k+2} l!}{((k+2)!)^2}$$

The profile of $D_{3,k}^{h}$

$$d_z = \lim_{k\to\infty} d_k(A_{n_p,t}) = \left\{ \left[\frac{1}{54}\right]_{j=1}, \left[\frac{4}{54}\right]_{j=2}, \left[\frac{49}{54}\right]_{j=3} \right\}$$

Definition 4.6: Let $A_{n_p}^{'}$ *and* A_{n_p} *be the domains of* $D_{n_p}^{h'}$ *and* $D_{n_p}^{h}$ *respectively.* $D_{n_p}^{h'}$ *is defined as an inflection of* $D_{n_p}^{h}$ *iff*

$$A_{n_p}^{'} \equiv A_{n_p} \tag{4.17}$$

Illustration 4.5:

The partitions in *Illustration 4.1* is

$$G = \{G_1, G_2, G_3\}$$

with

$$G_1 = \{A, B, C\} \qquad G_2 = \{D, E, F\} \qquad G_3 = \{I, H, G\}$$

Let the partitions $G^{'}$ be

$$G^{'} = \{G_1^{'}, G_2^{'}, G_3^{'}\}$$

with

$$G_1^{'} = \{A, B, D\} \qquad G_2^{'} = \{C, E, I\} \qquad G_3^{'} = \{F, H, G\}$$

Using f, the cascades are

$$D_3^h = \{P_1, P_2, P_3\}$$

$$D_3^{h'} = \{P_1^{'}, P_2^{'}, P_3^{'}\}$$

with

$$P_1 = \{f(\mathrm{A}), f(\mathrm{B}), f(\mathrm{C})\} \qquad P_2 = \{f(\mathrm{D}), f(\mathrm{E}), f(\mathrm{F})\} \quad P_3 = \{f(\mathrm{I}), f(\mathrm{H}), f(\mathrm{G})\}$$

and

$$P_1' = \{f(\mathrm{A}), f(\mathrm{B}), f(\mathrm{D})\} \qquad P_2' = \{f(\mathrm{C}), f(\mathrm{E}), f(\mathrm{I})\} \quad P_3' = \{f(\mathrm{F}), f(\mathrm{H}), f(\mathrm{G})\}$$

Since,

$$A_3 = \{f(A), f(B), f(\mathrm{C}), f(\mathrm{D}), f(\mathrm{E}), f(\mathrm{F}), f(\mathrm{G}), f(\mathrm{H}), f(\mathrm{I})\} = A_3^{'}$$

$D_3^{h'}$ is an inflection of D_3^h.

4.3 Expanding partitions and pools

Let a Markov process

$$g_t^L : S \to S$$

defined over a sequence of time periods t where L is the length of a Markov process.

Let also

$$G_{j,t} = (s_{i,j,1}, s_{i,j,2}, \ldots, s_{i,j,t})$$

so that

$$g_t^L(s_{i,j,k}) = \left(s_{i,j,t+1}, s_{i,j,t+2}, \ldots, s_{i,j,t+L}\right) \in S \quad k = 1,2,\cdots,t \tag{4.18}$$

and $G_{j,t+L}$ is given by:

$$G_{j,t+L} = G_{j,t} \cup g_t^L(s_{i,j,k}) \tag{4.19}$$

with $(s_{i,j,t+1}, s_{i,j,t+2}, \ldots, s_{i,j,t+L})$ observing Markov properties.

Let the sequence of pools $P_{j,t+L}$ be defined over $G_{j,t+L}$, so that

$$P_{j,t+L} = P_{j,t} \cup \{f(s_{i,j,t+1}), f(s_{i,j,t+2}), \ldots, f(s_{i,j,t+L})\} \tag{4.20}$$

Illustration 4.6:

Let the Markov process ($L = 3$) be

$$g_t^3(s_{i,j,k}) = \left(s_{i,j,t+1}, s_{i,j,t+2}, s_{i,j,t+3}\right) \in S \quad k = 1,2,\cdots,t$$

with

$$s_{i,j,t+1} = \frac{s_{i,j,k}}{t+2}, \qquad s_{i,j,t+2} = \frac{s_{i,j,t+1}}{t+3}, \qquad s_{i,j,t+3} = \frac{s_{i,j,t+2}}{t+4}$$

Let the partitions

$$G_0 = \{G_{1,0}, G_{2,0}, G_{3,0}\}$$

with

$$G_{1,0} = \{x_1\} \quad G_{2,0} = \{x_2\} \quad G_{3,0} = \{x_3\}$$

Using *f*, the cascade is

$$D_{3,0}^{h} = \{P_{1,0}, P_{2,0}, P_{3,0}\}$$

with

$$P_{1,0} = \{f(x_1)\} \quad P_{2,0} = \{f(x_2)\} \qquad P_{3,0} = \{f(x_3)\}$$

Let g_0^3 map x_1, so that

$$x_1 \xrightarrow{g_0^3} \left(\frac{x_1}{2}, \frac{x_1}{6}, \frac{x_1}{24}\right)$$

G_0 evolves to G_3

$$G_3 = \{G_{1,3}, G_{2,3}, G_{3,3}\}$$

with

$$G_{1,3} = \{x_1, \frac{x_1}{2}, \frac{x_1}{6}, \frac{x_1}{24}\} \quad G_{2,3} = \{x_2\} \quad G_{3,3} = \{x_3\}$$

Accordingly,

$$D_{3,3}^{h} = \{P_{1,3}, P_{2,3}, P_{3,3}\}$$

with

$$P_{1,0} = \{f(x_1), f(\frac{x_1}{2}), f(\frac{x_1}{6}), f(\frac{x_1}{24})\} \quad P_{2,0} = \{f(x_2)\} \quad P_{3,0} = \{f(x_3)\}$$

If d_z is a profile, $D_{n_p,t}^{h}$ will be in *equilibrium* with respect to d_z *iff*

$$\lim_{t\to\infty} d(A_{n_p,t}) = d_z \tag{4.21}$$

4.4 Outline of the Cascade Optimisation Algorithm

The procedure of the cascade optimisation algorithm is illustrated as follows:

Step 1: Select a number of pools to formulate a cascade (following *Definition 4.1*).

Step 2: Use a Markov process to increase populations in partitions and pools.

The Markov process maps a point in a partition to new points at each iteration. The new points are placed into the partitions to increase their populations.

Step 3: Check whether the cascade follows Eq. 4.13 and develop the inflections *(*following *Definition 4.6)* if required.

Step 4: Check the convergence of the algorithm and iterate with Step 2

The convergence of the algorithm is determined by the termination criteria that address the controls in:

- Population size
- Optimisation progress
- Population feature

The details of the termination criteria will be explained in section 4.5.3. Figure 4.2 outlines the main steps of the algorithm.

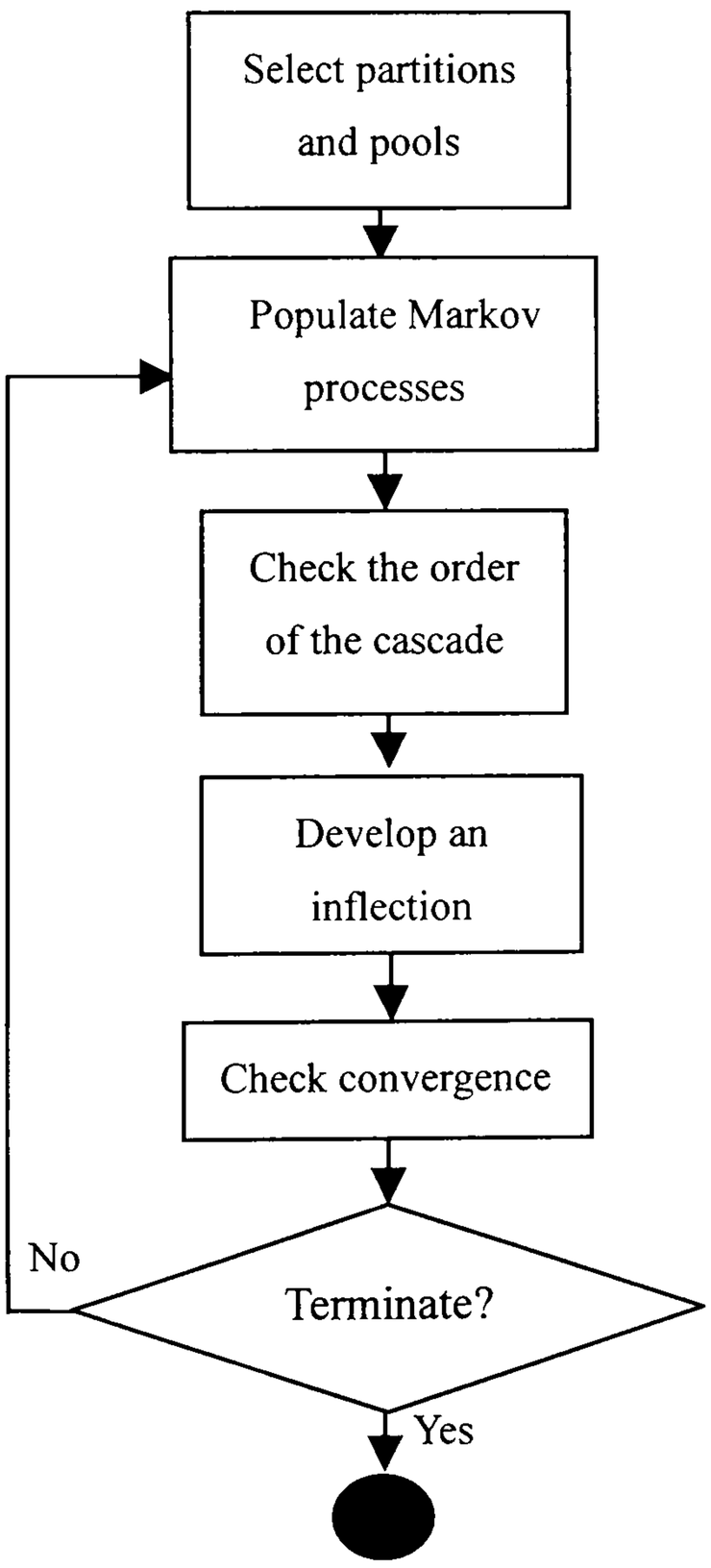


Figure 4. 2 Cascade optimisation algorithm flowchart

4.5 Cascade optimisation algorithm as a Markov process

The cascade optimisation algorithm employs a Markov process to generate new points. The Markov process is similar to that in SA. A global parameter is used as the pool temperature that accounts for the level of uncertainty in each pool, as is the case of temperature in SA. Each pool j is associated with a temperature (T_j) and pools are sorted in decreasing T_j . If the top (T_1) and bottom temperatures (T_{n_p}) are fixed, T_j is calculated on the basis of the following expressions:

$$T_j \equiv f^T(j) = \left(\frac{T_1 - T_{n_p}}{1 - n_p^{\,r}}\right) j^r + \left(T_1 - \left(\frac{T_1 - T_{n_p}}{1 - n_p^{\,r}}\right)\right) \tag{4.22}$$

The parameter r controls the convexity (concavity) of the function that is twice differentiable in $[1, W]$ with its second derivative:

$$f^{T''}(j) = r(r-1)\left(\frac{T_1 - T_{n_p}}{1 - n_p^{\,r}}\right) j^{r-2} \tag{4.23}$$

With different r, Eq. 4.22 can produce different cooling schedules. These cooling schedules can be classified into three major categories:

- With $r > 1$, $f^{T''}(j) < 0$ the cooling schedules are a strictly concave functions.
- With $r < 1$, $f^{T''}(j) > 0$ the cooling schedules are strictly convex functions.

- With $r = 1$, $f^{T''}(j) = 0$ the cooling schedules are linear functions.

Figure 4.3 illustrates the three classes of cooling schedules

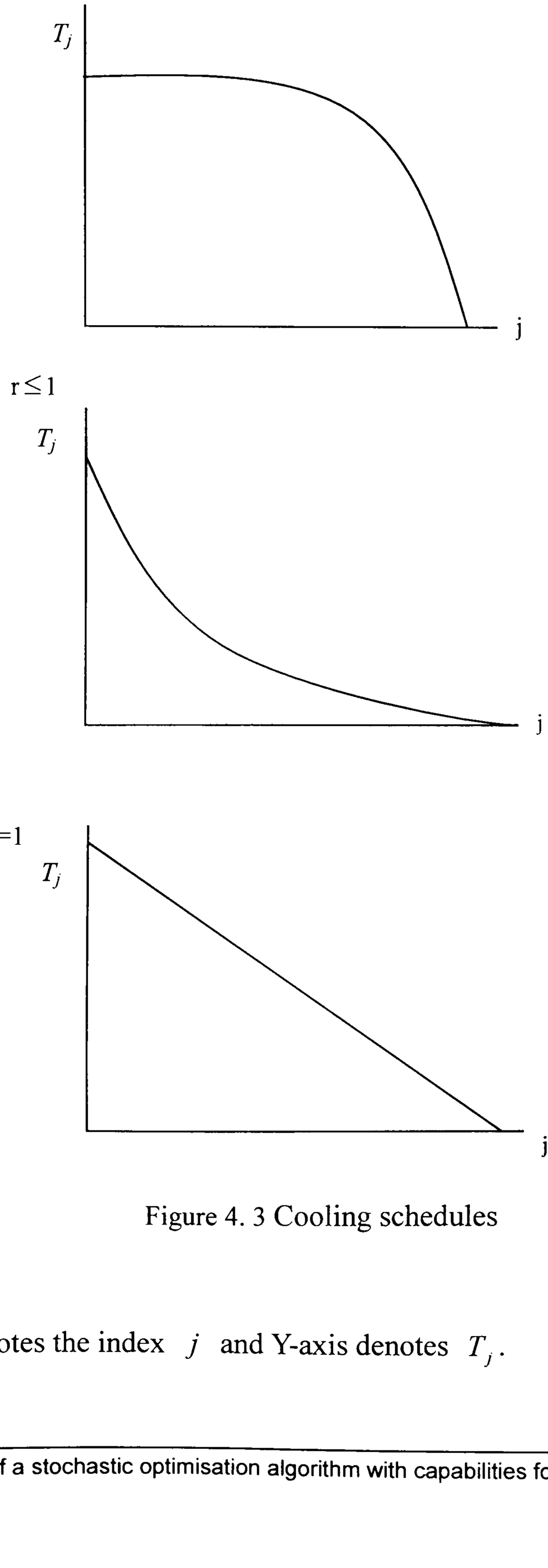


Figure 4. 3 Cooling schedules

Where X-axis denotes the index j and Y-axis denotes T_j.

Illustration 4.7:

To illustrate how T_j controls and manages actions in the algorithm, let us now set up an instance of the algorithm at $t = 0$, using 4 partitions and pools:

$$G_0 = \{G_{1,0}, G_{2,0}, G_{3,0}, G_{4,0}\}$$

Let 4 points

$$x_1 = 12 \quad x_2 = 11 \quad x_3 = 9 \quad x_4 = 8$$

be distributed in partitions, so that

$$G_{1,0} = \{x_1\} \quad G_{2,0} = \{x_2\} \quad G_{3,0} = \{x_3\} \quad G_{4,0} = \{x_4\}$$

Using *f*, the cascade is

$$D_{4,0}^{h} = \{P_{1,0}, P_{2,0}, P_{3,0}, P_{4,0}\},$$

Let $T_1 = 100$, $T_4 = 1$, and $r = 1$. Based on Eq. 4.22, each pool is associated with a temperature:

$$T_1 = 100, \; T_1 = 67 \; , \; T_1 = 34, \text{and} \; T_4 = 1$$

Let a Markov processes (L=3)

$$g_t^3(s_{i,j,k}) = \left(s_{i,j,t+1}, s_{i,j,t+2}, s_{i,j,t+3}\right) \in S \quad k = 1,2,\cdots,t$$

with

$$s_{i,j,t+1} = \frac{s_{i,j,t}}{t+1} + 2$$

4.5.1 Population growth

The Markov process generates points and increases the population in the partitions and pools. Similar to that in SA, the Markov process follows the Metropolis criterion:

$$p_{t,t+1} \equiv P(s_{i,j,t+1} \mid s_{i,j,t}) = \min\left(1, \exp\left(\frac{f_{i,j,t} - f_{i,j,t+1}}{T_j}\right)\right) \tag{4.24}$$

where

$s_{i,j,t}$ and $s_{i,j,t+1}$: the current point and the new point,

$f_{i,j,t}$ and $f_{i,j,t+1}$: objective of $s_{i,j,t}$ and $s_{i,j,t+1}$,

$p_{t,t+1}$: the probability of accepting $s_{i,j,t+1}$.

T_j : the temperature associated with $P_{j,t}$.

With this criterion, new points are accepted according to the rule:

$$s_{i,j,t+1} = \begin{cases} s_{i,j,t+1} & if\ p_{t,t+1} \geq R \\ s_{i,j,t} & if\ p_{t,t+1} < R \end{cases} \tag{4.25}$$

where

R : a random number in [0, 1].

As is apparent, the probability of accepting a new point is controlled by the pool

temperatures. When T_j is large (small), the Markov process follows a high (low) probability and accepts more (less) points.

Illustration 4.8:

Let us revisit the description presented in *Illustration 4.7*. Let the optimisation problem be

$$\min f = x^2$$

The cascade at time 0 is

$$D_{4,0}^{h} = \{P_{1,0}, P_{2,0}, P_{3,0}, P_{4,0}\}$$

with

$$P_{1,0} = \{f(x_1)\} = \{224\} \qquad P_{2,0} = \{f(x_2)\} = \{121\} \qquad P_{3,0} = \{f(x_3)\} = \{81\}$$

$$P_{4,0} = \{f(x_4)\} = \{64\}$$

g_0^3 firstly maps x_4 to x_5:

$$x_4 \xrightarrow{g_0^3} x_5 = \frac{8}{1} + 2 = 10$$

Based on Eq. 4.24,

$$p_{0,1} = 2.3\text{E}-16$$

$p_{0,1} < R$ and x_5 is not accepted. g_0^3 maps x_4 again to x_6:

$$x_4 \xrightarrow{g_0^3} x_6 = \frac{8}{2} + 2 = 6$$

and

$$p_{1,2} = 1$$

$p_{1,2} \geq R$ and x_6 is accepted. g_0^3 then maps x_6 to x_7:

$$x_6 \xrightarrow{g_0^3} x_7 = \frac{6}{3} + 2 = 4$$

and

$$p_{2,3} = 1$$

$p_{2,3} \geq R$ and x_7 is accepted. A set of points are generated by g_0^3

$$x_4 \xrightarrow{g_t^3} (x_6, x_7)$$

G_0 evolves to G_3:

$$G_3 = \{G_{1,3}, G_{2,3}, G_{3,3}, G_{4,3}\}$$

with

$$G_{1,3} = \{x_1\} \quad G_{2,3} = \{x_2\} \quad G_{3,3} = \{x_3\} \quad G_{4,3} = \{x_4, x_6, x_7\}$$

Using f, the cascade

$$D_{4,3}^{h} = \{P_{1,3}, P_{2,3}, P_{3,3}, P_{4,3}\}$$

with

$$P_{1,0} = \{f(x_1)\} = \{144\} \qquad P_{2,0} = \{f(x_2)\} = \{121\} \qquad P_{3,0} = \{f(x_3)\} = \{81\}$$

$$P_{4,0} = \{f(x_4), f(x_6), f(x_7)\} = \{64,36,16\}$$

It is obvious that $p_{t,t+1}$ is large (small) when T_j is large (small). T_1 should be a large value so that all points can be accepted and T_{n_p} should be a small value so that only the points better than the current ones are accepted.

4.5.2 Development of Inflections

The inflections are developed periodically following *Definition 4.6*. To develop the inflections, Let $F_t^{\min}$ and $F_t^{\min}$ be the maximum and minimum objectives in pools at time t. Apparently, the n_p pools have $n_p + 1$ boundaries. Let us assume that the cascade has $n_p + 1$ pools associated with $n_p + 1$ temperatures:

$$\frac{(T_j - T_{j+1})}{(T_1 - T_{n_p+1'})} = \frac{f_{j,t}^{\max} - f_{j,t}^{\min}}{F_t^{\max} - F_t^{\min}} \tag{4.26}$$

Each pool j is associated with two boundaries $f_{j,t}^{\max}$ and $f_{j,t}^{\min}$:

$$f_{j,t}^{\min} = -\frac{\left(F_t^{\max} - F_t^{\min}\right) * (T_1 - T_j)}{(T_1 - T_{n_p+1'})} + F_t^{\max} \tag{4.27}$$

$$f_{j,t}^{\max} = -\frac{\left(F_t^{\max} - F_t^{\min}\right) * (T_1 - T_{j+1})}{(T_1 - T_{n_p+1})} + F_t^{\max} \tag{4.28}$$

Points are distributed among pools following

$$f_{j,t}^{\max} \geq f(s_{i,j,t}) \geq f_{j,t}^{\min} \tag{4.29}$$

Figure 4.4 presents the boundaries in the cascade

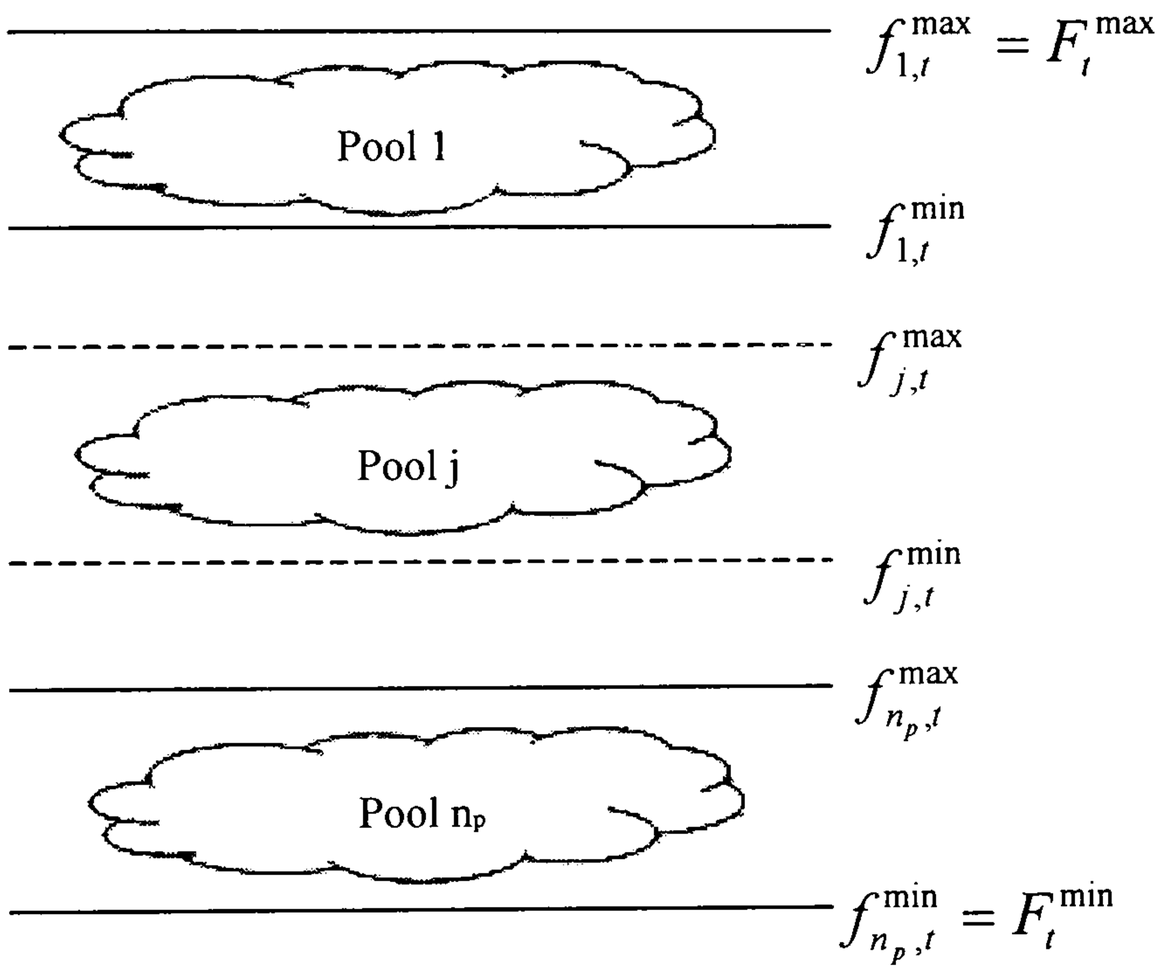


Figure 4. 4 Pools and boundaries

Apparently

$$f_{j-1,t}^{\min} = f_{j,t}^{\max} \geq f_{j,t}^{\min} = f_{j+1,t}^{\max} \tag{4.30}$$

so that

$$f(s_{i,j}) > f(s_{i,k}) \qquad \forall j < k \tag{4.31}$$

The inflections of the cascade can observe 4.13 if

$$Q_{j,t} \geq Q_{k,t} \qquad \forall j < k \tag{4.32}$$

Illustration 4.9:

Let us revisit *Illustration 4.8*, assuming the cascade has 5 pools associated with:

$$T_1 = 100,\ T_2 = 75.25\ ,\ T_3 = 50,\ T_4 = 25.75, \text{and}\ T_5 = 1.$$

$F_3^{\min}$ and $F_3^{\max}$ are:

$$F_3^{\min} = (x_7)^2 = 16 \qquad F_3^{\max} = (x_1)^2 = 144$$

Based on Eq. 4.27 and Eq. 4.28, the boundaries for pools are

$$f_{1,3}^{\max} = 144,\ f_{2,3}^{\max} = 112,\ f_{3,3}^{\max} = 80,\ f_{4,3}^{\max} = 48$$

$$f_{1,3}^{\min} = 112,\ f_{2,3}^{\min} = 80,\ f_{3,3}^{\min} = 48,\ f_{4,3}^{\max} = 16$$

Based on the boundaries, points are redistributed so that

$$f_{1,3}^{\max} = f(x_1) > f(x_2) > f_{1,3}^{\min}$$

$$f_{2,3}^{\max} > f(x_3) > f_{2,3}^{\min}$$

$$f_{3,3}^{\max} > f(x_4) > f_{3,3}^{\min}$$

$$f_{4,3}^{\max} > f(x_5) > f(x_6) = f_{4,3}^{\min}$$

The inflection is developed with the partitions:

$$G_{1,3} = \{x_1, x_2\} \quad G_{2,3} = \{x_3\} \quad G_{3,3} = \{x_4\} \quad G_{4,3} = \{x_6, x_7\}$$

4.5.3 Termination criteria

Based on the discussion in section 4.4, the termination criteria address the controls in

(a) Population size

To control the population size, a minimum population ($Q^{\min}$) and a maximum population ($Q^{\max}$) are proposed. Let Q_t be the population of G_t or $D^h_{n_p,t}$. The termination criterion provides

- The algorithm does not terminate if Q_t is smaller than $Q^{\min}$.
- The algorithm terminates if Q_t is larger than $Q^{\max}$.

The first part of this criterion can avoid premature convergence. For example, if the algorithm terminates when another termination criterion is true but Q_t is smaller than $Q^{\min}$. Search is likely to stop at the non-optimal or the localy optimal solutions. The second part provides that the algorithm terminates when the pools have a large number of population. A large Q^{mas} ensures the convergence but requires long computation time.

(b) Optimisation progress

The cascade optimisation algorithm also converges if the optimisation system does not progress any further (no better solution can be explored). Let ε be a value between 0 and 1 and n_F be an integer. The termination criterion provides that the cascade optimisation algorithm terminates if :

$$\left|F_t^{\min} - F_{t+L}^{\min}\right| \leq \varepsilon$$

for n_F iterations.

(c) Population feature

As the population is distributed in pools, the cascade optimisation algorithm might have several features, some of which reflect the convergence. Pool level and cascade level are two of these features.

(i) Pool level

The cascade optimisation algorithm is considered to be in convergence if the pool level is constant. The pool level is reflected on the standard deviation of the population of pools. Let σ_t be the standard deviation and n_{ST} be an integer. The termination criterion provides that the algorithm terminates if :

$$\left\|\sigma_t - \sigma_{t+L}\right\| \leq \varepsilon$$

for n_{ST} iterations

(ii) Cascade level

The cascade level is reflected on the density function ($d_t(A_{n_p,t})$). The cascade is in equilibrium if $d_t(A_{n_p,t})$ approximates to d_z, Let the change of $d_t(A_{n_p,t})$ be:

$$\left\|d_t - d_{t+L}\right\| = \left\|d_t(A_{n_p,t+L}) - d_t(A_{n_p,t})\right\| \qquad (4.33)$$

$\|d_t - d_{t+L}\|$ approaches 0 if the cascade is in equilibrium. Let n_{ST} be an integer. The termination criterion provides that the cascade optimisation algorithm terminates if :

$$\|d_t - d_{t+L}\| \leq \varepsilon$$

for n_d iterations.

4.6 The Cascade Optimisation Algorithm: Formulation

The cascade optimisation algorithm uses n_p partitions ($G_{j,t}$) and pools ($P_{j,t}$). The set of n_p $P_{j,t}$ are combined as a cascade ($D^h_{n_p,t}$). Given the top temperature (T_1) and the bottom temperature (T_{n_p}), each $P_{j,t}$ is associated with a pool temperatures (T_j) following Eq. 4.22. The Markov process (g_t^L) increases the population in $G_{j,t}$ and $P_{j,t}$.

The cascade optimisation algorithm (COPT) is outlined as:

Step 1: Place an initial point and its objectives into $G_{1,t}$ and $P_{1,t}$.

Step 2: Maps a point in $G_{j,t}$ to new points. Place the new points and their objectives to $G_{j,t}$ and $P_{j,t}$.

Step 3: Check whether $D^h_{n_p,t}$ follows Eq. 4.13 and develop an inflection if required.

Step 4: Check the termination criteria described in section 4.5.3 and iterate with Step 2.

4.7 Remarks and discussions

4.7.1 COPT

The partitions and pools are introduced into COPT to distribute populations. A Markov process maps a point in a partition to new points at each iteration. The inflections of the cascade are developed and the convergence of COPT is checked from time to time. The layout of Figure 4.5 presents the structure of COPT.

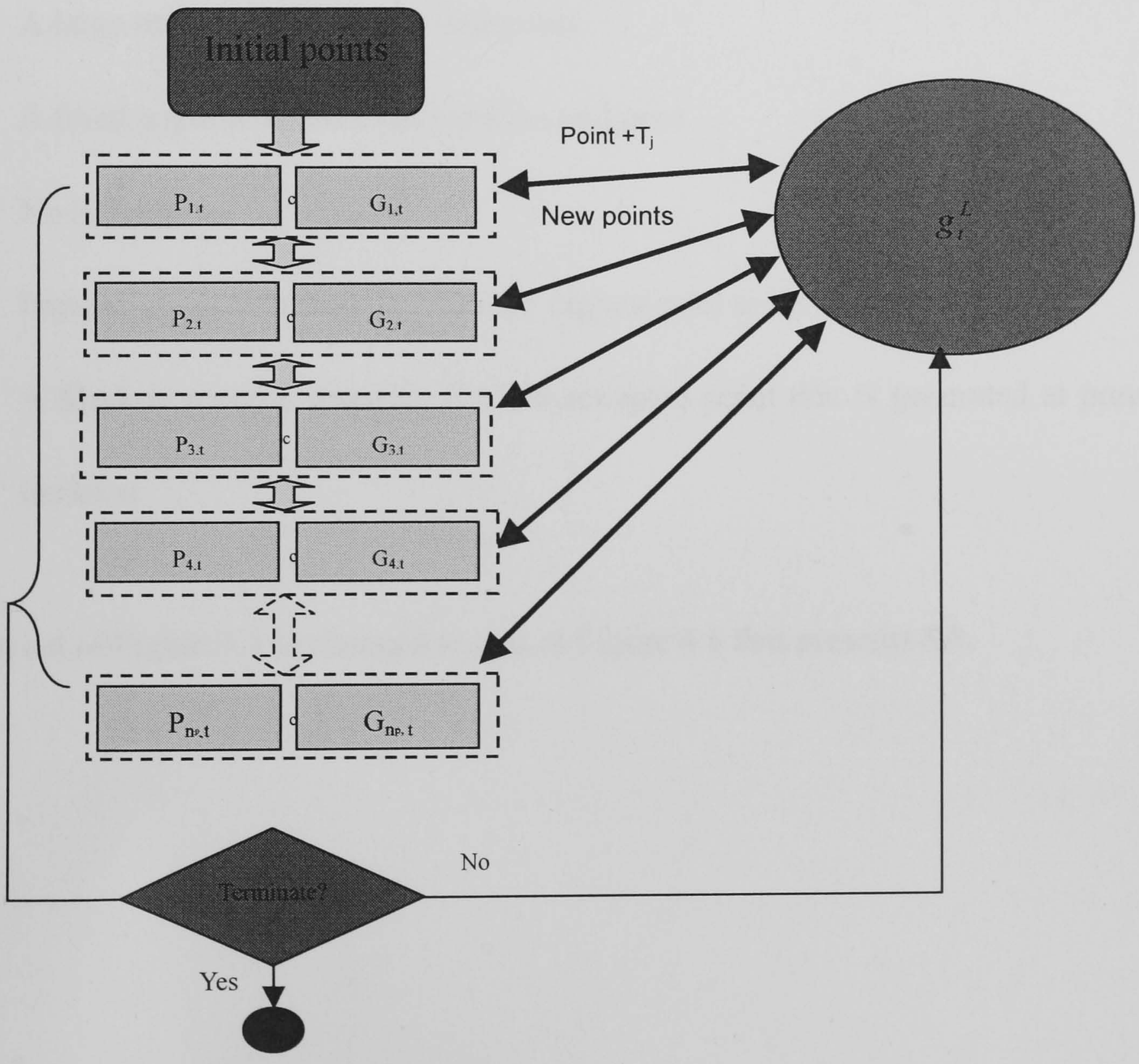


Figure 4. 5 COPT

Based on the structure, multiple Markov processes can be launched simultaneously so that COPT can be effectively applied on parallel and distributed computing environments. With the Markov process, it is closely related to the stochastic methods. The similarities are illustrated in the following sections.

4.7.2 Analogies between COPT and SA

Let us consider a COPT structure featuring:

- A large number of partitions and pools
- A fixed population for each partition and pool
- No inflections
- Population growth starting from the highest pool to the lowest pool
- A Markov process mapping the last accepted point that is generated at previous iteration

The layout of Figure 4.5 is changed to that of Figure 4.6 that presents SA.

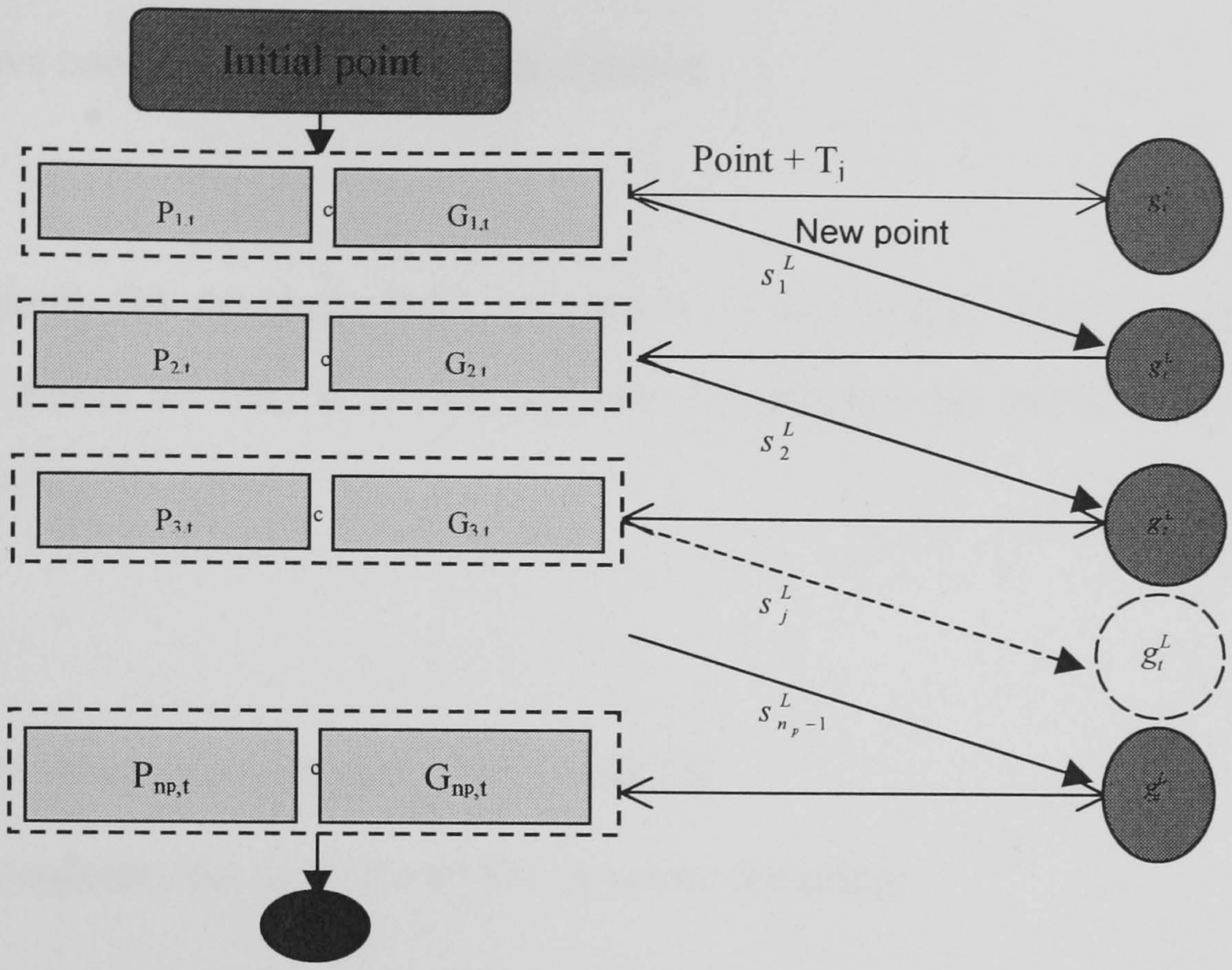


Figure 4. 6 SA

s_j^L: the last accepted point in G_j

Initially, a point and its objective are placed into the $G_{1,t}$ and $P_{1,t}$. g_0^L maps this point to new points following the Metropolis criterion. T_1 determine the acceptance probability in this criterion. The new points and their objectives are placed into the $G_{1,t}$ and $P_{1,t}$. g^L maps s_1^L to new points. T_2 determines the acceptance probability in the Metropolis criterion. Points and their objectives are placed into the $G_{2,t}$ and $P_{2,t}$. The algorithm then iterates until when $G_{n_p,t}$ and $P_{n_p,t}$ are filled by the new generated points and their objectives. Analogies between COPT and SA are strengthened as they both

- Use the Markov process that follows the Metropolis criteria.
- Have cooling stages (pools account for T_j).
- Have cooling stages sorted in decreasing T_j.

As is apparent, SA develops from the highest to the lowest cooling stages. With the sequential nature in SA, it is not suitable for running on parallel and distributed environments

4.7.3 Analogies between COPT and TS

Let us now consider the case of a COPT structure featuring:

- A fixed population for each partition and pool
- No inflections
- New points generated out of a small subset of each pool

The layout of Figure 4.5 is then changed to that of Figure 4.7 that presents TS.

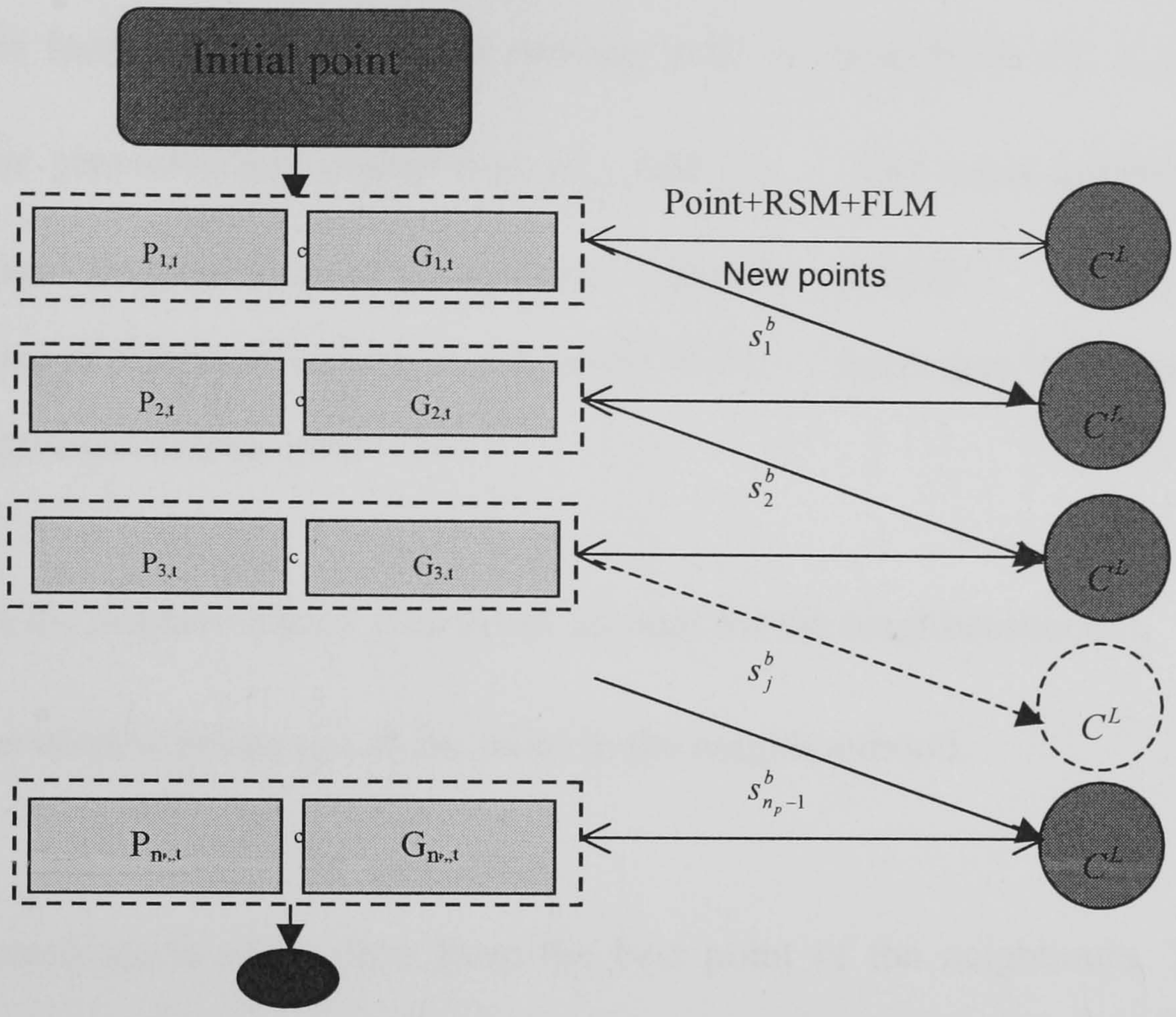


Figure 4. 7 TS

C^L : a local search in which L neighbours are generated

s_j^b : the best point in G_j and

$$f(s_j^b) \le f(s_{i,j}) \qquad \forall i$$

Initially, a point and its objective are placed into $G_{1,t}$ and $P_{1,t}$. C^L generates L neighbours of this point. The neighbours and their objectives are placed into $G_{1,t}$ and $P_{1,t}$.

C^L again generates L neighbours of s_1^b and places them and their objective into $G_{2,t}$ and $P_{2,t}$. $G_{1,t}$ is the recent-based short-term memory (RSM) used to prevent C^L from generating the same points as the ones in the RSM. C^L again generates the neighbours of s_2^b and places them and their objectives into $G_{3,t}$ and $P_{3,t}$. $G_{2,t}$ is the RSM for this local search. TS keeps running until the neighbours of $s_{n_p-1}^b$ and their objectives are generated and placed into $G_{n_p,t}$ and $P_{n_p,t}$. The super partitions G_t are used as the frequency-based long-term memory (FLM). COPT performs intensifications and diversifications based on the FLM. Analogies between COPT and TS are strengthened as they both:

- Have the neighbourhood (partitions account for the neighbourhoods).
- Generate new points out of the point in the neighbourhood.

The local search starts every time from the best point of the neighbours. TS has the sequential nature and is therefore not suitable for the parallel and distributed applications.

Chapter 5 Algorithm implementation as a master-worker paradigm

5.1 Master-worker architecture

The master-worker paradigm is common in parallel and distributed computing applications that execute on the environments involving a single master and a numbers of workers. The workers execute the computational tasks and the master dispatches the tasks to the workers and collects results. The global information is transferred between the master and the workers periodically. To avoid bottlenecks and reduce the burden of communication with the master, one has to build the workers as independent from each other as possible.

5.2 Master and worker tasks in COPT

The Markov process increases the population of the pools. Multiple Markov processes can be launched simultaneously to increase the population. To implement COPT, the workers undertake the Markov process (Tsk^{MC}) that is the time expensive task. On the other hand, the master implements the partitions and pools and executes the tasks that include:

- Development of inflections (Tsk^{Inf})
- Checking the convergence of COPT (Tsk^{Tm})

Tsk^{Inf} and Tsk^{Tm} follow different time schedules. Figure 5.1 and Figure 5.2 outline the worker process and the master process respectively.

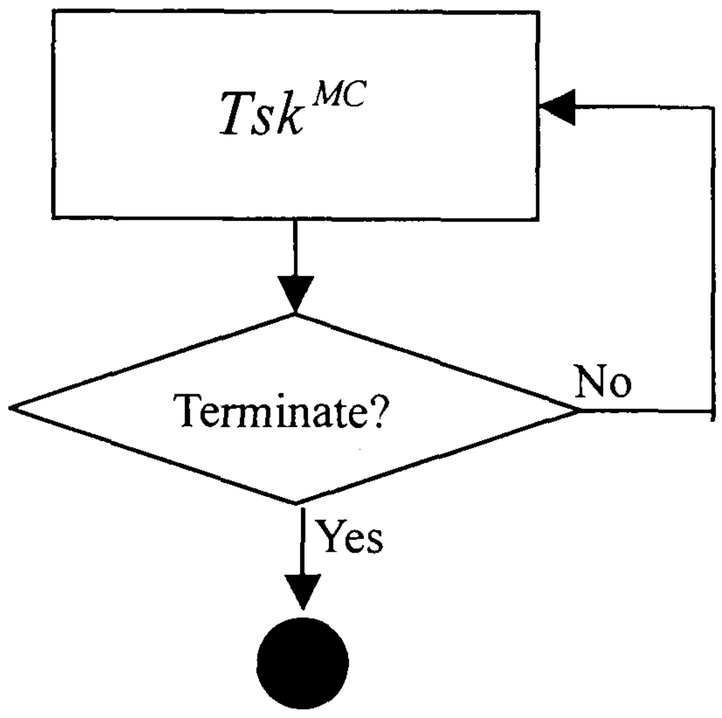
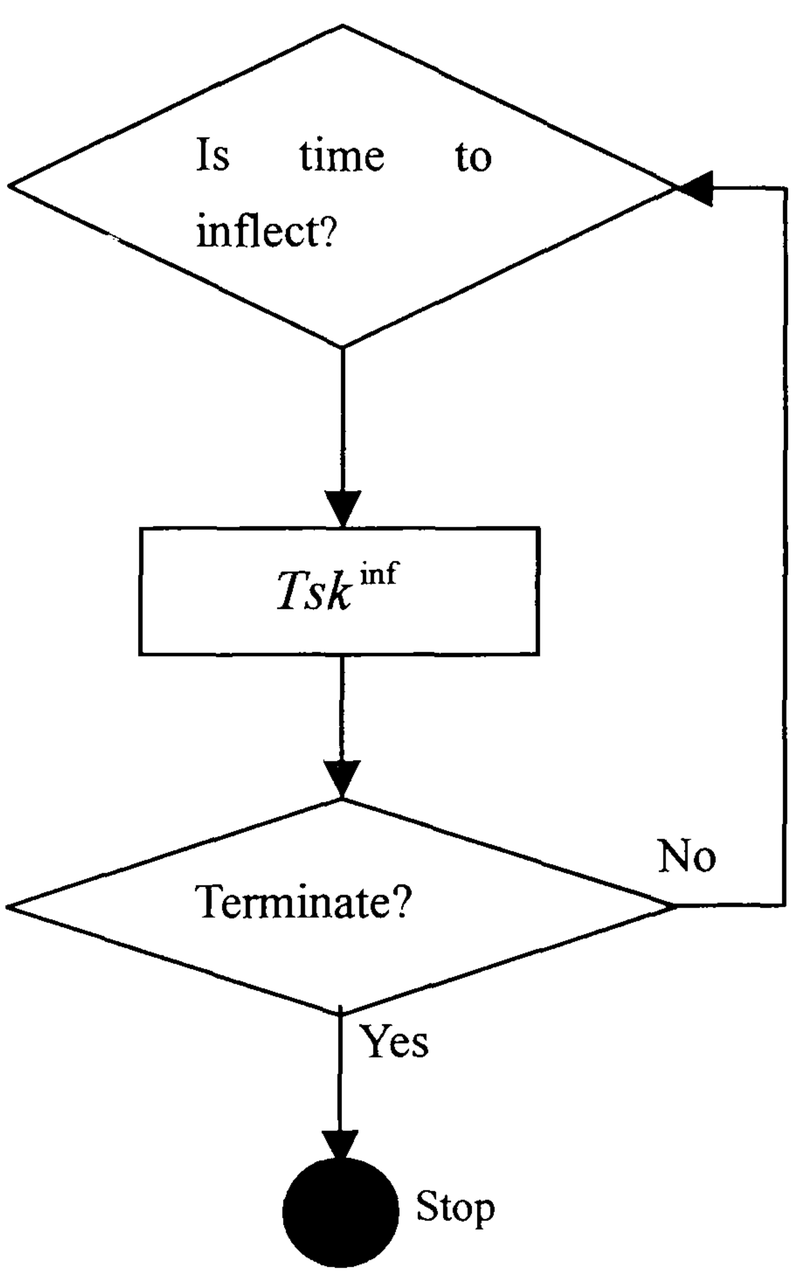


Figure 5. 1 Worker process



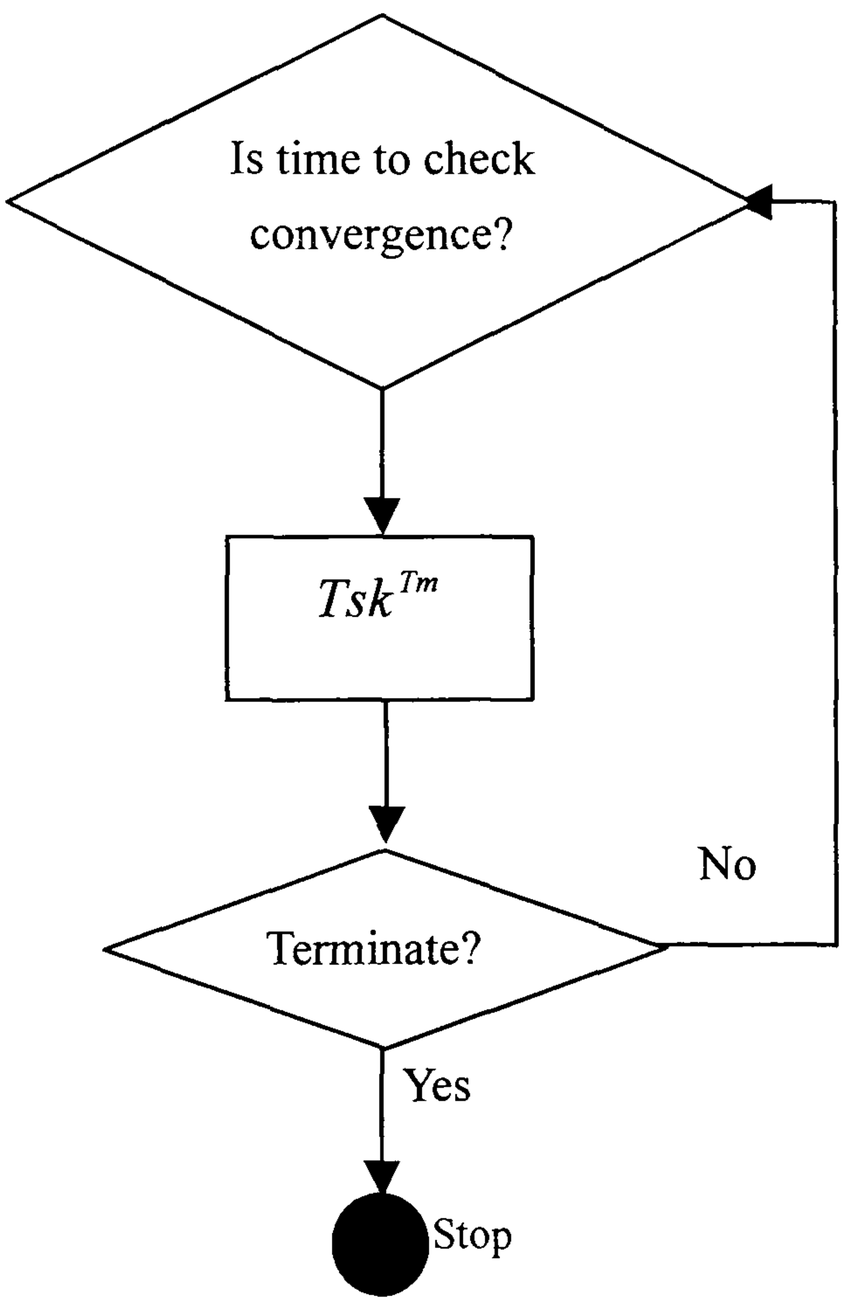


Figure 5. 2 Master process

There are two tasks that transfer the information between the master and workers:

- Exporting information from the master to workers (Tsk^{Ep})

- Importing information from workers to the master (Tsk^{Ip})

5.3 Communication

The information transferred between the master and workers involves:

- Information related to points (point-related information)
- Control commands for workers (command-related information)

The point-related information includes:

- M-W information that involves a selected point and its objective and transfers from the master to workers

- W-M information that involves new points and their objectives and transfers from workers to the master

The command-related information involves:

- Initiation commands (i.e. start COPT)
- Termination commands (i.e. terminate COPT)

Figure 5.3 outlines the information transferred in the course of COPT.

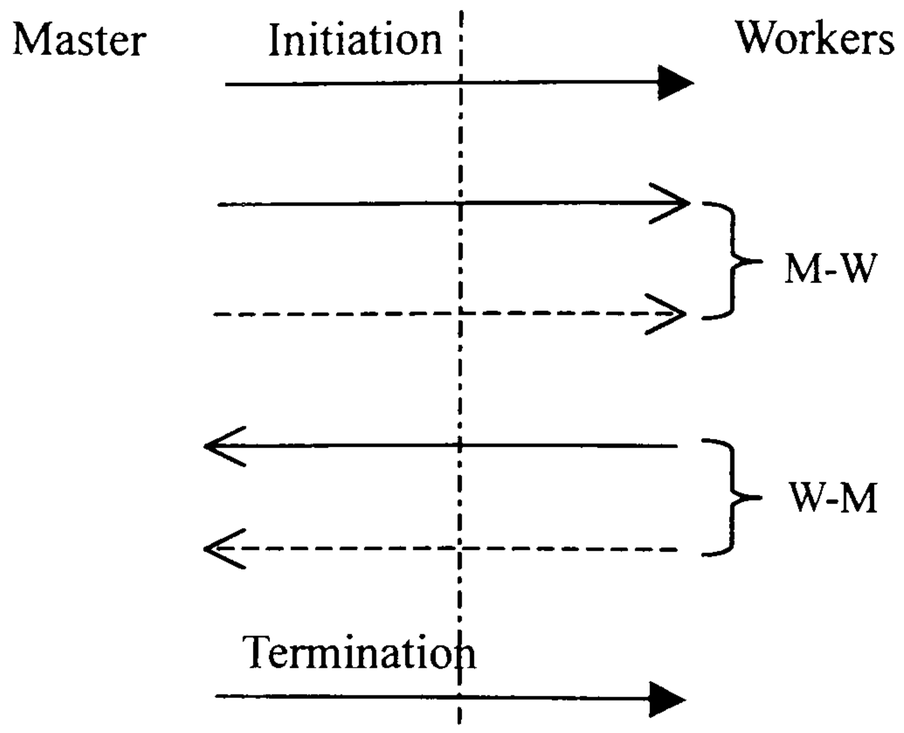


Figure 5. 3 Information transferred

Based on the figure, COPT develops from the top to the bottom.

Initially,

— The master sends the initiation commands to the workers.

In the course of COPT,

— Workers take the M-W information from the master.

— Workers send the W-M information to the master.

Finally,

— The master sends the termination command to workers, if the termination criteria are satisfied.

5.4 Implementation of COPT

COPT is implemented as the master-worker paradigm in two different ways. For implementation, the computing environment firstly includes a master and a worker. In

the following sections, the implementations are explained in terms of the partitions and pools and tasks ($Tsk^{\inf}$, Tsk^{Tm}, Tsk^{Ep}, Tsk^{Ip}, and Tsk^{MC})

5.4.1 Implementation of partitions and pools

The partitions and pools are developed by tables in a database. The database is implemented on the master. *pool ID* and *point ID* are two columns that identify the partitions/pools and the points in them. Let us revisit the notations described in Chapter 4. The *pool ID* and the *point ID* correspond to the index j and the index i.

First partitions and pools method

In the first implementation, the database involves *pool_feature*, *cascade*, and *partitions* tables. Figure 5.1 presents the partitions and pools.

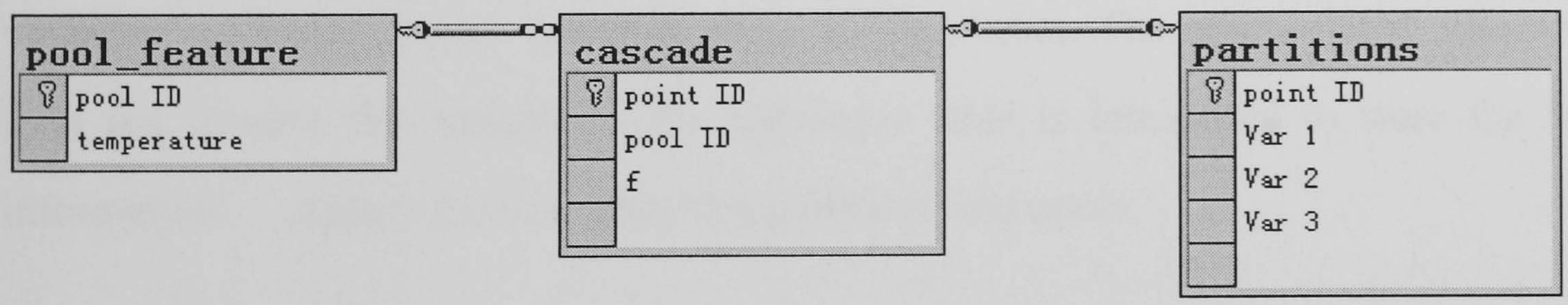


Figure 5. 4 First implementation of the partitions and pools

The *pool_feature* and the *cascade* tables implement the pools. The first one stores the pool temperatures (T_j). It involves *pool ID* and *temperature* columns. The latter one stores the objectives in pools. It involves *point ID*, *pool ID* and f columns. The f column stores the objectives. The *pool ID* column is the candidate key connecting to that in the *pool_feature* table. The *partition* table implements the partitions and stores the variables

of points. Beside the *point ID* column, the *partitions* table involves the columns that store the variables. These columns are *Var X* where *X* is the index of the variables. In Figure 5.4, three variables are stored in *Var 1*, *Var 2*, and *Var 3* columns in the *partitions* table. The *point ID* column is the primary key connecting to that in the *cascade* table.

The *partitions* table in this implementation stores the variables. If the application problem has *n* variables, the *partitions* table should have *n VarX* columns. With a large *n* employed, the table would grow quickly. In the end, the *partitions* table might become too big to manage. The size of the M-W information and the W-M information is large since they involve all the variables of the points. COPT therefore encounters bottlenecks and high communication traffic. In addition, this database for a particular problem may be not suitable for other problems since the numbers of the variables for these problems may be different.

Second partitions and pools method

In the second implementation, the database does not have the *partitions* table. The variables are stored in files (solution files) on the worker. The point-related information does not involve the variables. An additional table is introduced to store the M-W information. Figure 5.5 illustrates this partitions and pools.

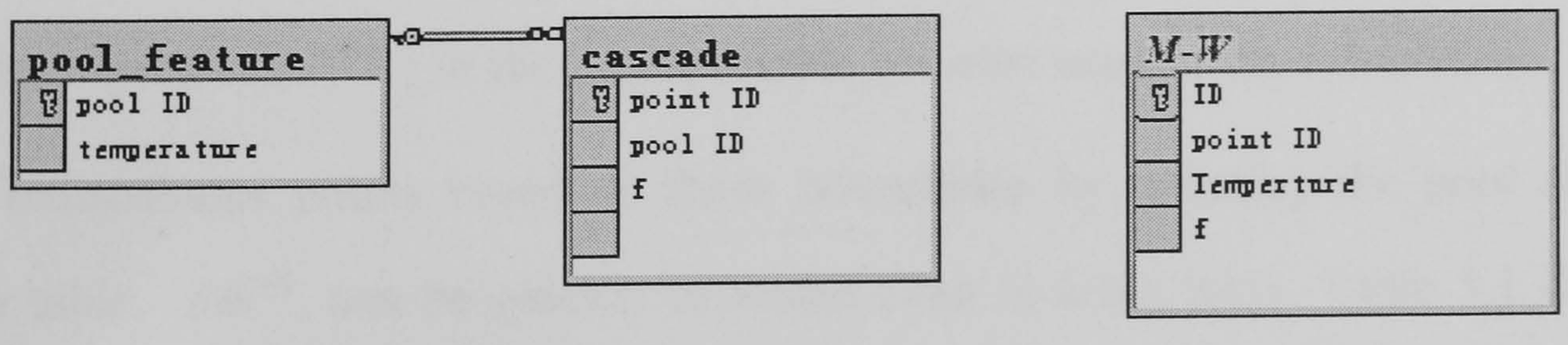


Figure 5. 5 Second implementation of the partitions and pools

The database now involves *pool_feature*, *cascade,* and *M-W* tables. The *pool_feature* table and the *cascade* table are the same as those in the first implementation. The *M-W* table stores the M-W information and comprises *point ID*, *temperature* and *f* columns. ID is another column that is the primary key identifying the different M-W informations.

This database stores only the objectives so that it is small even if COPT is applied to problems that have large numbers of variables. In addition, with the absence of the *partition* table, the implementation is suitable for different applications. The M-W information and the W-M information do not involve variables so that they become much smaller than those in the first implementation.

5.4.2 Implementation of $Tsk^{\inf}$

$Tsk^{\inf}$ develops inflections of the cascade by periodically referring to *Definition 4.6*. The implementation of $Tsk^{\inf}$ is illustrated as follows:

First implementation of $Tsk^{\inf}$

$Tsk^{\inf}$ is implemented on the worker. $Tsk^{\inf}$ runs after Tsk^{MC} at each iteration. The worker calculates the boundaries ($f_{j,t}^{\max}$ and $f_{j,t}^{\min}$). Based on Eq. 4.22, the pool temperatures (T_j) in the *pool_feature* table, the maximum objective ($F_t^{\max}$) and the minimum objective ($F_t^{\min}$) in the *cascade* table are also used in the calculation. Then the master redistributes points based on these boundaries by updating the *pool ID* in the *cascade* table. $Tsk^{\inf}$ can be quickly executed even in a big table. Table 5.1 illustrates the CPU time spent for $Tsk^{\inf}$.

Table 5. 1 Times for $Tsk^{\inf}$

Size (rows)	Time (s)
10	0.013
100	0.034
1000	0.166
10000	0.336

Based on this table, $Tsk^{\inf}$ spends only 0.336 seconds when the *cascade* table involves 10000 rows.

Second implementation of $Tsk^{\inf}$

$Tsk^{\inf}$ is now implemented on the master. The master calculates the boundaries of pools and redistributes the points in the pools. $Tsk^{\inf}$ executes on a computer thread and follows independent time schedule. The time schedule for $Tsk^{\inf}$ in the first implementation is not feasible because the master now cannot catch the time when Tsk^{MC} finishes. To keep the order of the cascade $Tsk^{\inf}$ executes as frequently as possible. A new schedule is proposed that COPT develops the inflections every time after the pools has a new point. $Tsk^{\inf}$ now is parallel with and independent of Tsk^{MC} on the worker.

5.4.3 Implementation of Tsk^{Tm}

Based on the description in section 4.5.3, termination criteria are related to the population in pools (Q_t), and the change of the best objective ($\left|F_t^{\min} - F_{t+L}^{\min}\right|$), the standard deviation of the populations ($\|\sigma_t - \sigma_{t+L}\|$), and density function ($\|d_t - d_{t+L}\|$). Tsk^{Tm} checks these values periodically and determines the convergence.

First implementation of Tsk^{Tm}

Tsk^{Tm} is implemented on the worker. The worker extracts Q_t, $\left|F_t^{\min} - F_{t+L}^{\min}\right|$, $\left\|\sigma_t - \sigma_{t+L}\right\|$ and $\left\|d_t - d_{t+L}\right\|$ from the partitions and pools and determines the convergence. Similar to the first implementation of Tsk^{Inf}, Tsk^{Tm} use ODBC to extract these values. This task is launched after $Tsk^{\inf}$ at each iteration.

Second implementation of Tsk^{Tm}

Similar to the second implementation of Tsk^{Inf}, Tsk^{Tm} is implemented on the master. The master extracts Q_t, $\left|F_t^{\min} - F_{t+L}^{\min}\right|$, $\left\|\sigma_t - \sigma_{t+L}\right\|$ and $\left\|d_t - d_{t+L}\right\|$ and determines the convergence. Tsk^{Tm} executes on a computer thread and follows a new time schedule. The schedule is based on a new time unit that is defined as the period in which a new point is accepted and placed into partitions. The new schedule provides that the master checks the convergence every time after L new points are placed into partitions.

5.4.4 Implementation of Tsk^{Ep}

Tsk^{Ep} selects the M-W information from partitions and pools and transfers it to the worker. The implementations of Tsk^{Ep} is illustrated as follow:

First implementation of Tsk^{Ep}

Tsk^{Ep} is implemented on the worker and executes before Tsk^{MC} at each iteration. With the *partition* table, the M-W information involves the variables of the selected points. The worker takes the M-W information and initializes the Markov processes. Since the Markov process observes the Metropolis criterion that requires the pool

temperatures, the M-W information involves the pool temperatures. For example, if Tsk^{Ep} selects the point 3 from partition 4, the objectives and variables where the *point ID* is 3 and *pool ID* is 4 are selected from the *cascade* table and the *partition* table. The temperature of pool 4 (T_4) is also selected from *pool_feature* table. The worker selects the M-W information from the master and starts Tsk^{MC}. Tsk^{Ep} in this implementation is not suitable for executing on the parallel and distributed computing environments. The master would become too busy to deal with the queries from the workers when a large number of workers intent to fetch the M-W informations at the same time.

Second implementation of Tsk^{Ep}

In this implementation, Tsk^{Ep} are separated into two subtasks that include:

- The master selects the M-W information and stores it in the output files.
- The worker takes the output files from the master and finds the corresponding solution files.

The M-W information involves the objectives of the points and the temperature of the pool where the points are selected. This information is firstly stored in the *M-W* table and then transformed to the content in the output files (named as *Export*). Similar to other tasks in the second implementation, this subtask executes on a computer thread following a new schedule. Based on the schedule, this subtask executes every 1 second so that the M-W information can be updated as frequently as possible. Figure 5.6 illustrates the master putting M-W information into an ouput file in Tsk^{Ep}.

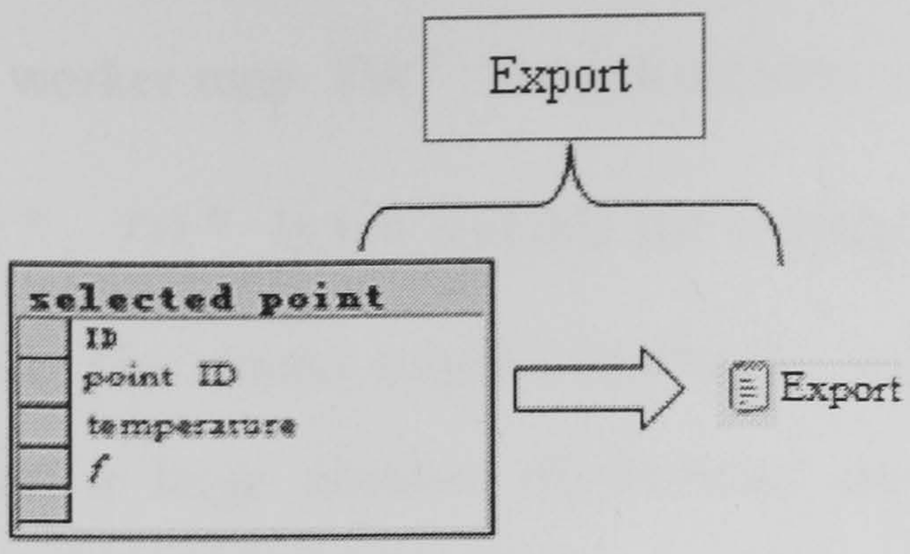


Figure 5. 6 Master storing M-W information into an output file

The other subtask executes on the worker. It takes the output files from the master and finds the corresponding solution files to start Tsk^{MC}. To find the solution files, the name of each solution file is appended by the *point ID* of the selected solution (*SoX* where *X* is the *point ID*). The worker reads the *point ID* in the output files and finds out the solution files. Figure 5.7 illustrates the worker finding the solution file.

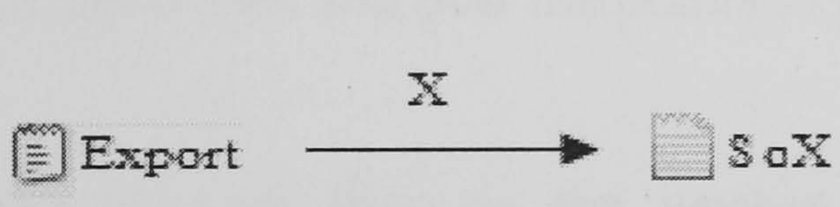


Figure 5. 7 Worker finding the solution file of point X

The subtasks executes before Tsk^{MC} at each iteration.

5.4.5 Implementation of Tsk^{Ip}

First implementation of Tsk^{Ip}

Tsk^{Ip} is implemented on the worker. The worker sends the W-M information to the master and stores them into partitions and pools. The W-M information involves the objectives and the variables of the new points that are stored in the *cascade* and the

partitions tables. The worker runs Tsk^{Ip} every time after Tsk^{MC}. Similar to the first implementation of Tsk^{Ep}, Tsk^{Ip} is not suitable for running on parallel and distributed computing environments. The master might become too busy to deal with the queries from the workers when a large number of workers are intent to send the W-M information at the same time.

Second implementation of Tsk^{Ip}

Similar to the second implementation of Tsk^{Ep}, Tsk^{Ip} is divided into two subtasks that include:

- The worker sends the input files to the master.
- The master reads the input files and puts the points in partitions and pools.

Variables are stored in the solution files on the worker. The W-M information involves the objective and *point ID* of a point and is stored in the input files (named as *Import*). Then the worker sends them to the master. The input files are associated with the solution files and their names are also appended by the point IDs (*ImportX* where X is the point ID).

In the second subtask, the master reads the input files and places the objectives in the *cascade* tables as presented in Figure 5.8.

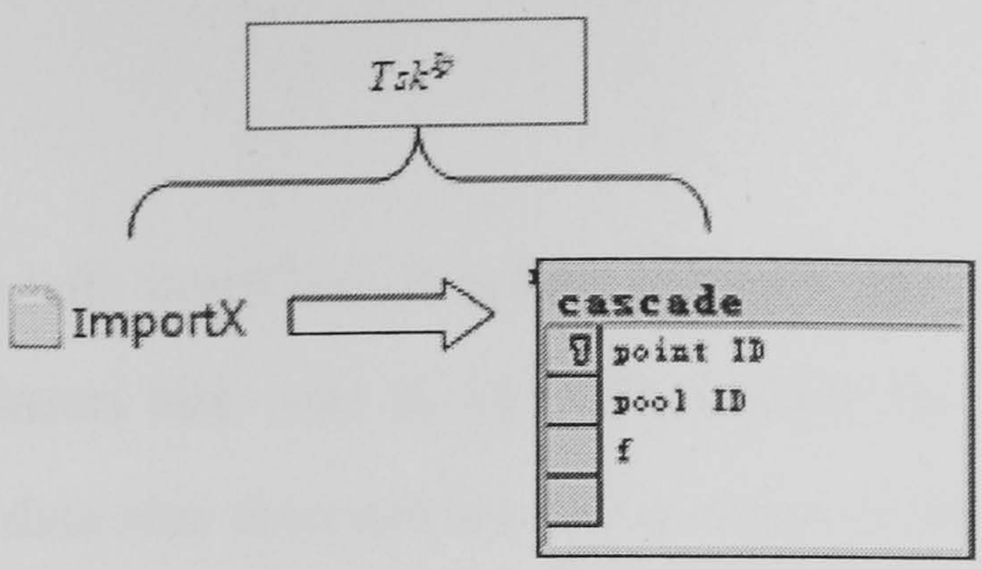


Figure 5. 8 Master storing W-M information into the partitions and pools

The bulk insert task of Microsoft SQL server 2000 is used to store the W-M information . It has been proven to be the quickest way to insert data. Table 5.2 shows the CPU time spent for inserting different size data into database by using the traditional SQL queries.

Table 5. 2 Time spent by using SQL queries

Size (Kb)	Time (s)
10	1.234
100	3.437
1024	60.656
10240	665.359

It is obvious that increasing the data size increases the time. This method has to spend a long time to insert a large amount of data into the database. For example, it has to spend 665.4 seconds to insert 10Mb data into the database. As the queries execute on the worker, the increasing time would increase the overall time spent for COPT. Table 5.3 illustrates the clock time spent by using the bulk insert task.

Table 5. 3 Time spent by using the bulk insert task

Size (Kb)	Time (s)
10	0.281
100	0.328
1024	0.61
10240	1.453

Based on the results, the bulk insert task is orders of magnitude quicker than SQL queries. For example, the bulk insert task only spends 1.5 seconds for inserting 10Mb data. In addition, increasing the data size does not lead to an unacceptable increment on the time spent.

Each of these subtasks has an independent time schedule. The first subtask follows a schedule where the worker sends the input file every time after a new point is generated and stored in the files (the solution files and the input files). On the other hand, the second subtask executes as long as the master obtains a new input file.

5.4.6 Implementation of Tsk^{MC}

First implementation of Tsk^{MC}

Tsk^{MC} is implemented on the worker. The worker runs the Markov process that follows the Metropolis criterion.

Second implementation of Tsk^{MC}

The worker takes Tsk^{MC} and follows the steps:

Step 1: *Selection of initial solution* extracts the output files and finds the corresponding solution files based on the description in the second implementation of Tsk^{Ep} in section 5.4.4. Then the worker reads the solutions in the files.

Step 2: *Development of new solutions* runs the Markov process following the Metropolis criterion.

Step 3: *Storage of solutions stores* the newly accepted solutions in the files (the solution files and the input file).

Figure 5.9 illustrates the worker generating new points and storing them in files.

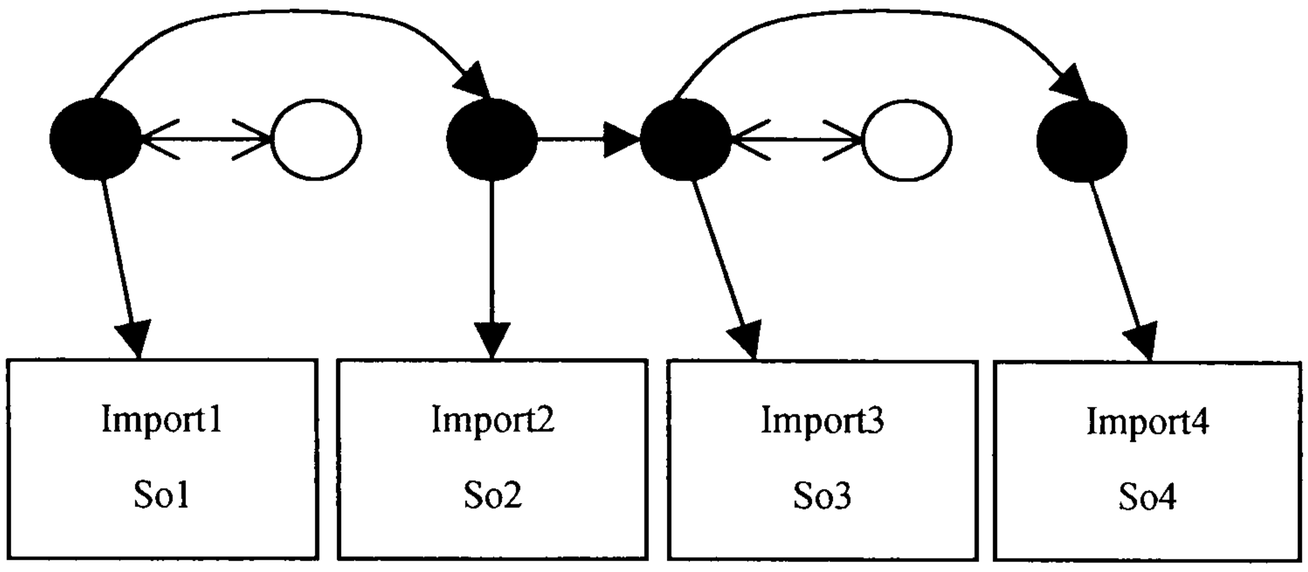


Figure 5. 9 Worker generating new points and storing them in files

The worker generates four points whose point IDs are 1, 2, 3, and 4. Then the worker stores these points in paired files namely: Import1 and So1, Import2 and So2, Import3 and So3, Import4 and So4 files respectively.

5.4.7 Relations between the tasks

In the first implementation, all tasks execute sequentially on the worker so that COPT is difficult to be parallelized. In addition, with a large numbers of workers employed, the master would become too busy to deal with the queries from the workers.

In the second implementation, the tasks ($Tsk^{\inf}$ and Tsk^{Tm}) execute on the master. Tsk^{Ep} and Tsk^{Ip} are separated into two subtasks where one executes on the master and the other runs on the worker. All the above tasks are associated with specific tables and follow different time schedule. Figure 5.10 presents the relation of computer threads, tasks and tables.

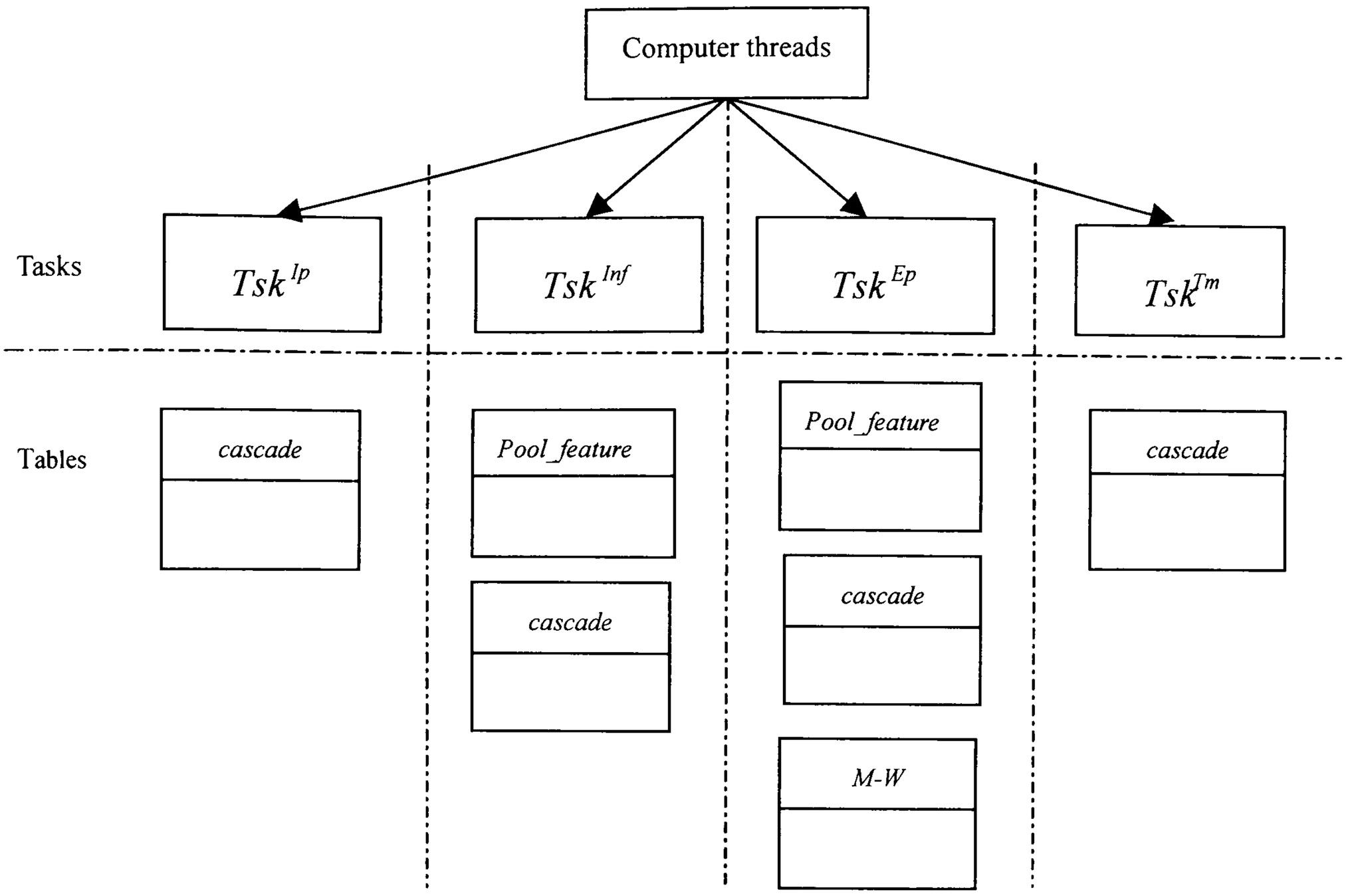


Figure 5. 10 Relations of computer threads, tasks, and tables

5.5 Computing implementation

The master runs on Windows 2000 server on which the partitions and pools are implemented as the tables in database. The database is managed by Microsoft SQL server 2000. The worker runs on Windows XP professional on which the tasks are programmed using Fortran programming language. In the first implementation, all tasks are coded on the worker and executed sequentially. To communicate with and manage the database on the master, the worker uses Fortran SQL library to call Open Database Connectivity (ODBC). In the second implementation, the tasks ($Tsk^{\inf}$, Tsk^{Tm}, Tsk^{Ep}, and Tsk^{Ip}) are implemented by intrinsic functions of the SQL server, running on independent computing threads of the master. The worker now only undertakes Tsk^{MC} which is coded by Fortran programming language.

5.6 Remarks and Discussions

There are two different implementations for COPT, in which the partitions and pools are developed by the tables of a database. The database in the first implementation includes the *partition* table that stores the variables of points. The size of the database, in such case, increases quickly if the application problem involves a large number of variables. The partitions and pools are not universal since the *partition* table has to change when COPT is applied to different applications. In addition, the amount of information transferred between the master and workers is large since it involves the variables. In contrast, the database in the second implementation does not involve the *partition* table. The growth of the database is very small even if the application problem involves a large number of variables. Without the *partition* table, the partitions and pools are suitable for different applications. The information transferred between the master and the worker is now much smaller than that in the first implementation.

COPT involves several tasks and is developed differently in the two implementations. In the first implementation, the worker undertakes all tasks and runs them sequentially. COPT does not follow the requirement of a master-worker paradigm. With a large number of workers employed, the master would become too busy to deal the queries from workers. In the second implementation, the master undertakes $Tsk^{\inf}$ and Tsk^{Tm} and half of Tsk^{Ep} and Tsk^{Ip} while the worker focuses on Tsk^{Mc}. The tasks on the master are independent and follow different time schedules. COPT is implemented as the master-worker paradigm and can be applied to the parallel and distributed computing environments.

In the following chapter, COPT is studied on several simple problems in terms of the impact of its important features and the selection and coordination of the termination criteria. The performance of COPT reflects on the solution quality (measured by objective value) and computation time.

Chapter 6 Evaluation and validation

6.1 Features and problems

COPT has several features that can affect its performance. These features involve:

- The management of stochastic search
- The size of optimisation structure
- The depth and intents of search
- The selection and coordination of different termination criteria

The management of stochastic search relates to the inflection of COPT which is implemented as redistributing the populations of pools. The distribution of population may adopt different distribution functions. With the pool temperatures, the Markov process follows different acceptance probabilities. The distribution function therefore has an impact on the performances. The size of optimisation structure is determined by the number of partitions (pools) (n_p). As n_p increases, the population of a partition (pool) decreases so that the structure size has an apparent impact on the performance. The length of Markov process (L) determines the depth of search. Compared to a short Markov process, a long Markov process might explore the search space more easily. The termination criteria involve four different parts. The selection and coordination of the termination criteria also have an impact on the performance.

6.2 Benchmark problems

COPT is applied to four test problems (Floudas, 1999). These problems arose in literature

studies involving a wide spectrum of problems including:

- Quadratic assignment problems
- Heat exchanger network synthesis problems
- Phase and chemical equilibrium problems
- Process design problems

The four test problems are listed as:

- TP1 is a non-convex quadratic problem
- TP2 is a generalized geometric programming problem
- TP3 is a twice differential nonlinear programming problem
- TP4 is a quadratically constraint problem

The formulations of TP1- TP4 are given in the Appendix A.

6.3 Implementation and performance measurements

The performance of COPT is measured by its solution quality and the computation time required to converge. The solution quality is accounted by the objective of the optimal solution achieved ($F^{\min}$). The computation time is measured differently in the two implementations. In the first implementation, the tasks are implemented sequentially on the worker. The computation time is accounted by the worker's CPU time that is sum of the CPU times for all tasks, so that:

$$T^{CPU} = \sum_{t=1}^{5} T_t^{CPU}$$

T^{CPU} = the worker's CPU time

T_1^{CPU} = the CPU time for Tsk^{MC}

T_2^{CPU} = the CPU time for Tsk^{Inf}

T_3^{CPU} = the CPU time for Tsk^{Tm}

T_4^{CPU} = the CPU time for Tsk^{Ep}

T_5^{CPU} = the CPU time for Tsk^{Ip}

On the other hand, $Tsk^{\inf}$, Tsk^{Tm}, and part of Tsk^{Ep} and Tsk^{Ip} execute on the master and are independent of Tsk^{MC} in the second implementation. The computation time is the maximum of the CPU times for the tasks, so that

$$T^{CPU} = \max\{T_1^{CPU}, \sum_{t=2}^{5} T_t^{CPU}\}$$

Compared to the other tasks, Tsk^{MC} is the most time expensive one. Table 6.1 illustrates the comparison between T_1^{CPU} and $\sum_{t=2}^{5} T_t^{CPU}$. COPT is applied to TP1 and terminates when the population of pools (Q_t) approaches 1000.

Table 6. 1 CPU times for the tasks

T_1^{CPU}	10.787
$\sum_{t=2}^{5} T_t^{CPU}$	2.513

Based on the above table, T_1^{CPU} is about 5 times larger than $\sum_{t=2}^{5} T_t^{CPU}$. Thus, T_1^{CPU} is the overall computation time.

Depending up the discussions in Chapter 5, COPT is implemented on a computing environment with a master and a worker. The configurations of the master and the worker are presented in Table 6.2

Table 6. 2 Configurations of the master and the worker

Master	**CPU**	**Ram Memory**
Prise-sql	Intel(R) Core(TM) 2.13GHz	2Gb
Worker	**CPU**	**Ram Memory**
Grid 1	Intel(R) Pentium(R) 1.20GHz	512Mb

6.4 The management of stochastic search

Distribution functions can be influenced by the pool boundaries. These pool boundaries are associated with the pool temperatures. The cooling schedule presented in Eq.4.22 determines the pool temperatures. Since the parameter r controls the convexity (concavity) of the cooling schedule, it might have impacts on distribution functions. With different r values, three classes of cooling schedules are produced:

- Linear cooling schedule ($r = 1$)
- Convex cooling schedules ($r > 1$)

- Concave cooling schedules ($r < 1$)

Each class of cooling schedules is associated with a form of distribution functions, involving:

- Linear distribution
- Convex distributions
- Concave distributions

The distribution of population that follows the above distribution functions are illustrated in the following sections.

(i) *Linear distribution*

The Linear distribution is associated with the linear cooling schedule. In the linear cooling schedule, the temperature difference of adjacent pools is constant. Based on Eq. 4.26, the difference of the boundaries of a pool (defined as pool size) is constant. The distributions of population of ten pools at three periods are extracted and illustrated in Figure 6.1. The three periods account for:

- Initial phase ($Q_t = 100$)
- Intermediate phase ($Q_t = 500$)
- Final phase ($Q_t = 1000$)

Q_t : the population of pools at time t

Q: the population size

j: index of a pool

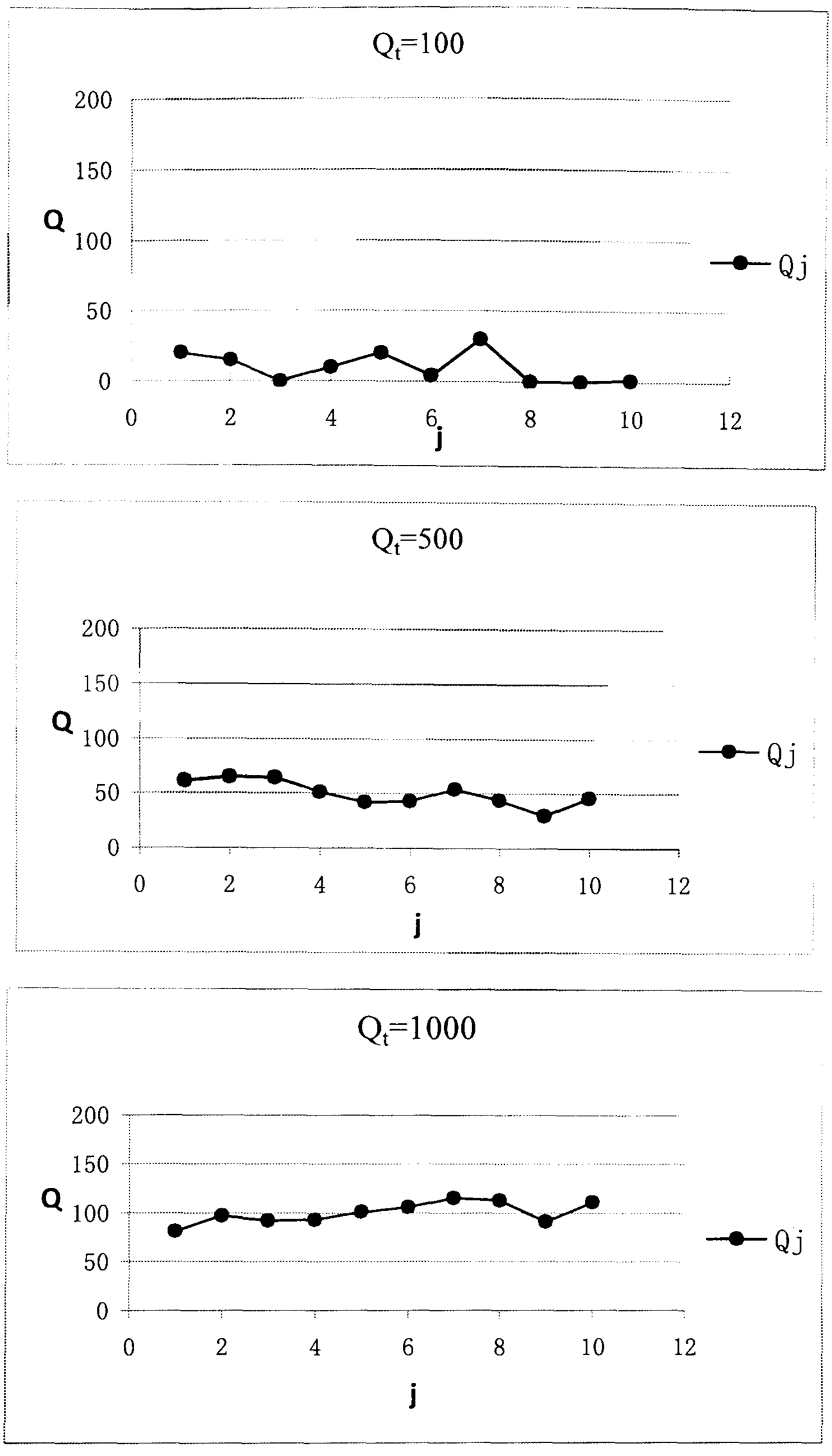


Figure 6. 1 Linear distributions

Based on the above figures, the linear distribution tries to evenly distribute the population into pools with no bias. With the use of inflection, each pool is distributed by intermediated solutions within the same quality level (defined by its lower bound $f_{j,t}^{\min}$ and upper bound $f_{j,t}^{\min}$), increasing from the top pool to the bottom pool. The linear distribution produces the quality levels which can be distributed by almost the same amount of intermediate solutions. The Markov process that starts from an

intermediate solution within a pool follows the acceptance probability controlled by the corresponding pool temperatures. Thus, cascade optimisation algorithm that uses linear distribution can execute the Markov process that follows different acceptance probabilities and starts from the same number of intermediate solutions within different quality levels.

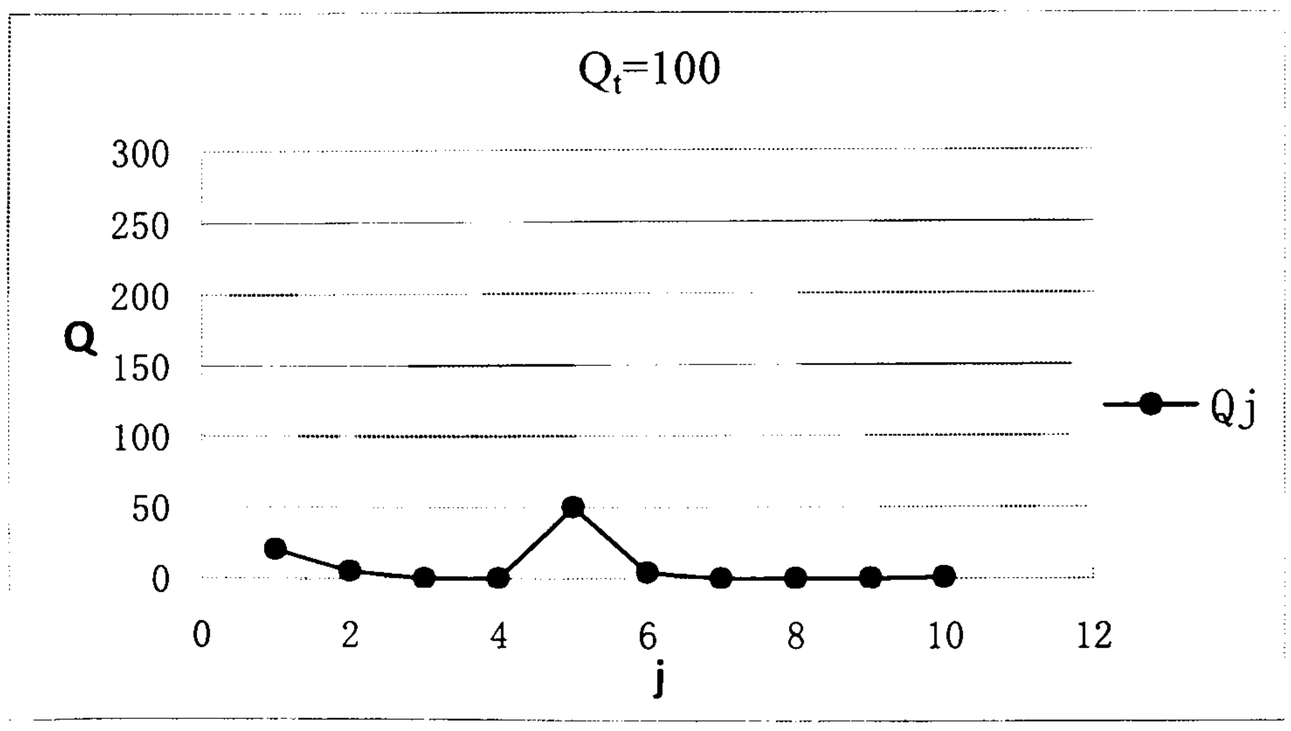
(ii) Concave distribution

The concave distribution is associated with the concave cooling schedule. In the concave cooling schedule, the temperature difference of adjacent pools follows:

$$T_j - T_{j+1} > T_k - T_{k+1} \qquad \forall j < k$$

T_j and T_k : the temperatures of pool j and pool k

Based on Eq. 4.26, the size of pools decreases from the top pool (pool 1) to the bottom pool (pool n_p). Figure 6.2 presents a concave distribution (with r=-0.1) at the three periods.



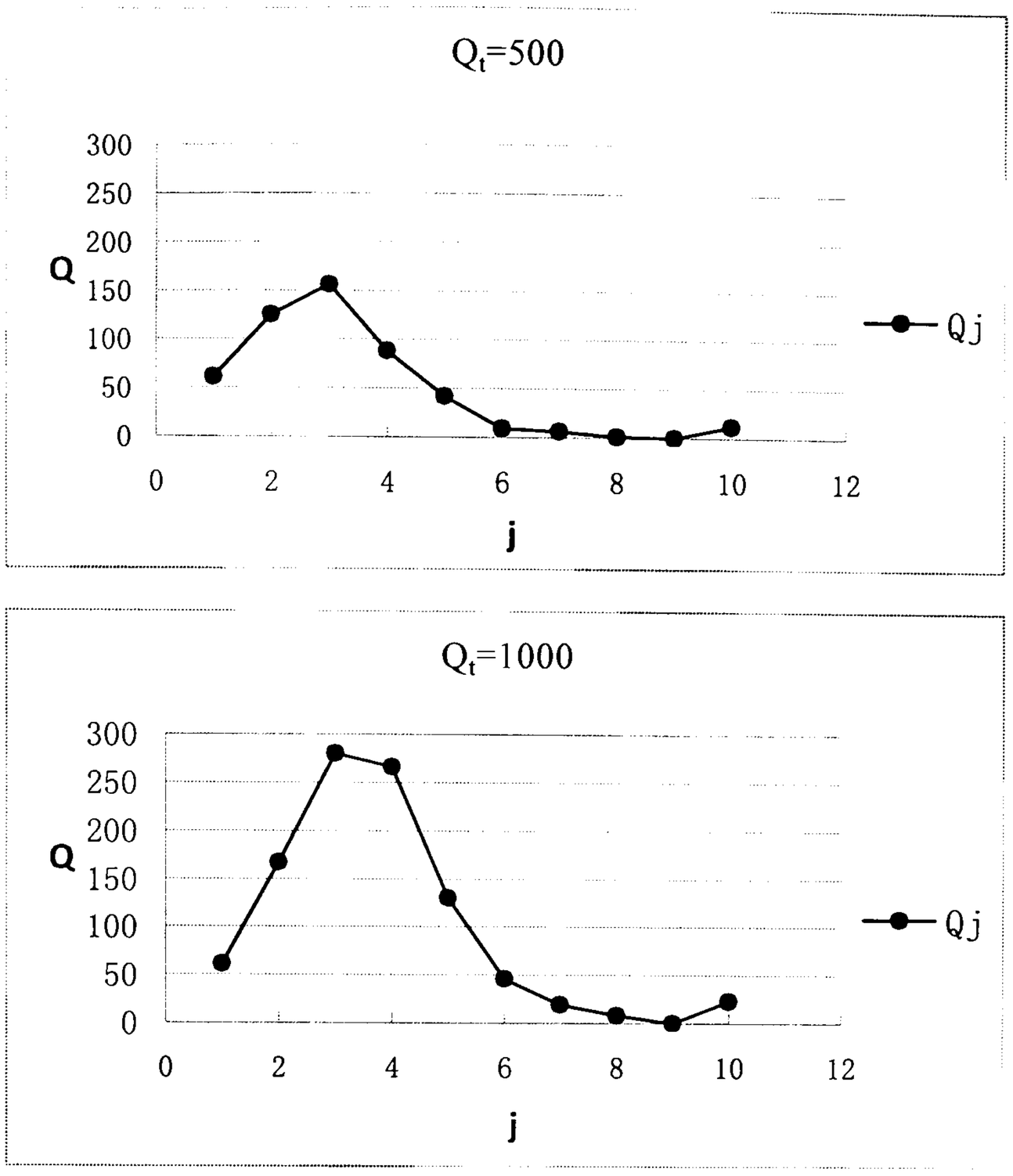


Figure 6. 2 Concave distributions

As the figures illustrate, the population of a higher pool (the top pool or a pool close to the top pool) is larger than that of a lower pool (the bottom pool or a pool close to the bottom pool). With the inflection, each pool is assigned by a quality level increasing towards the bottom pool. With inflection under a concave distribution, the number of intermediate solutions within a low quality level is larger than the number of intermediate solutions within a high quality level.

The Markov process for a higher pool can launch from a large number of intermediate solutions in the overall space. Such Markov process follows a high acceptance probability and can easily explore the search space. With inflection, the promising areas penetrate downwards to lower pools. In contrast, the Markov process for a lower pool focuses on a small number of good intermediate solutions in promising areas. Such Markov process now follows a low acceptance probability and searches deeply for the optimal solution within the promising areas. Thus, the cascade optimisation algorithm

with concave distribution is likely to explore the search space and converge to the optimal solution.

As r decreases, the concavity of a cooling schedule increases. Two concave cooling schedules are selected as:

- The one with a small concavity $(r = -0.1)$
- The one with a large concavity $(r = -10)$

The corresponding distributions of population at the last period are extracted and illustrated in Figure 6.3.

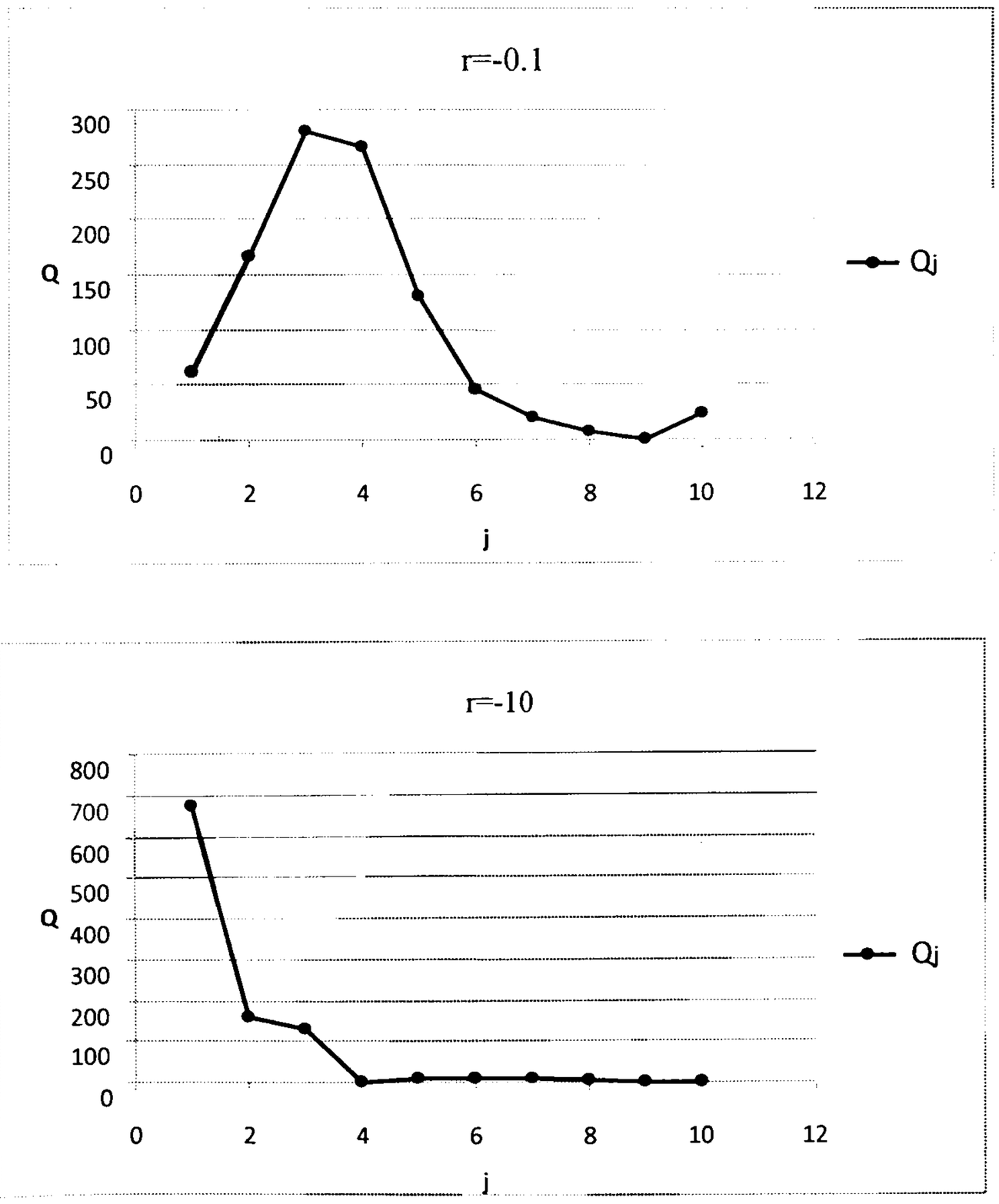


Figure 6. 3 Concave distributions with different r

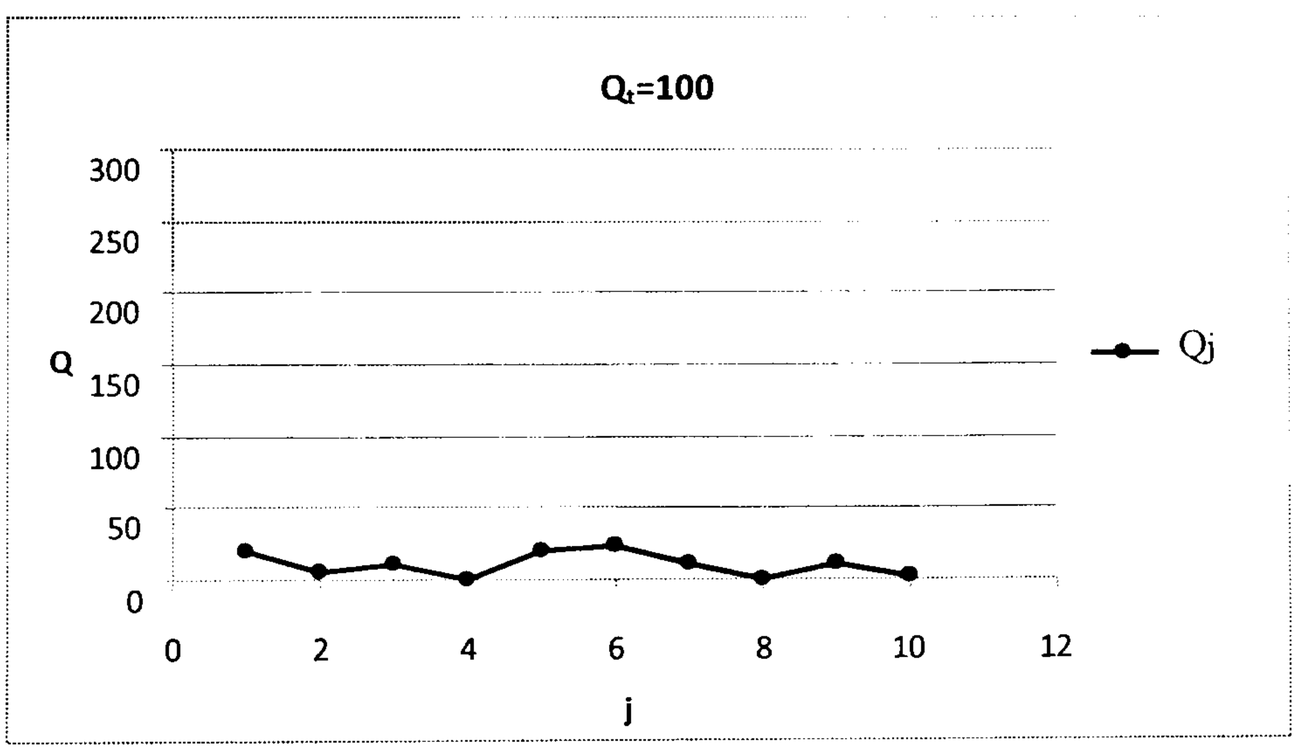
Based on the figures, the bias of distributions moves up towards the top pools (P_1) as the concavity increases. Finally, the most of the population is distributed to P_1 .

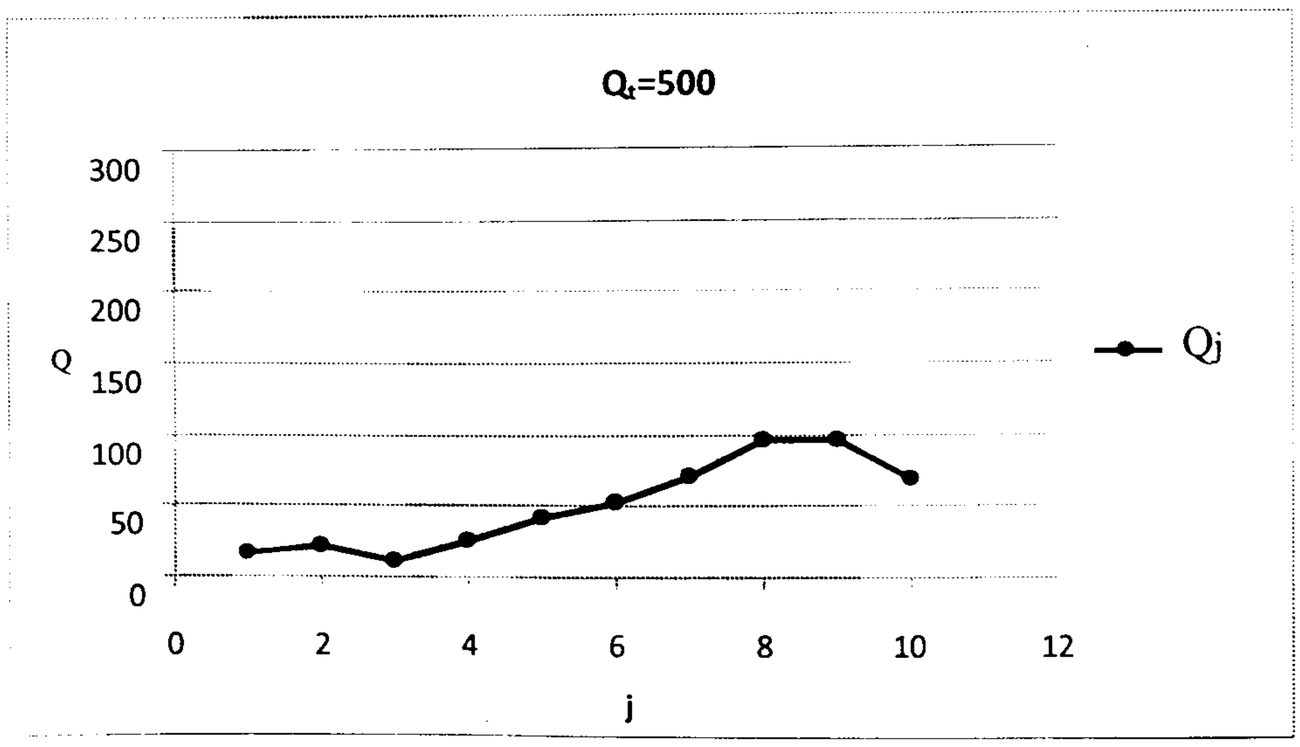
(iii) Convex distribution

The convex distribution is associated with the convex cooling schedule. In the convex cooling schedule, the temperature difference of adjacent pools follows:

$$T_j - T_{j+1} < T_k - T_{k+1} \qquad \forall j < k$$

Based on Eq. 4.26, the size of pools increases from the top pool to the bottom pool. Figure 6.4 presents a convex distribution at the three periods.





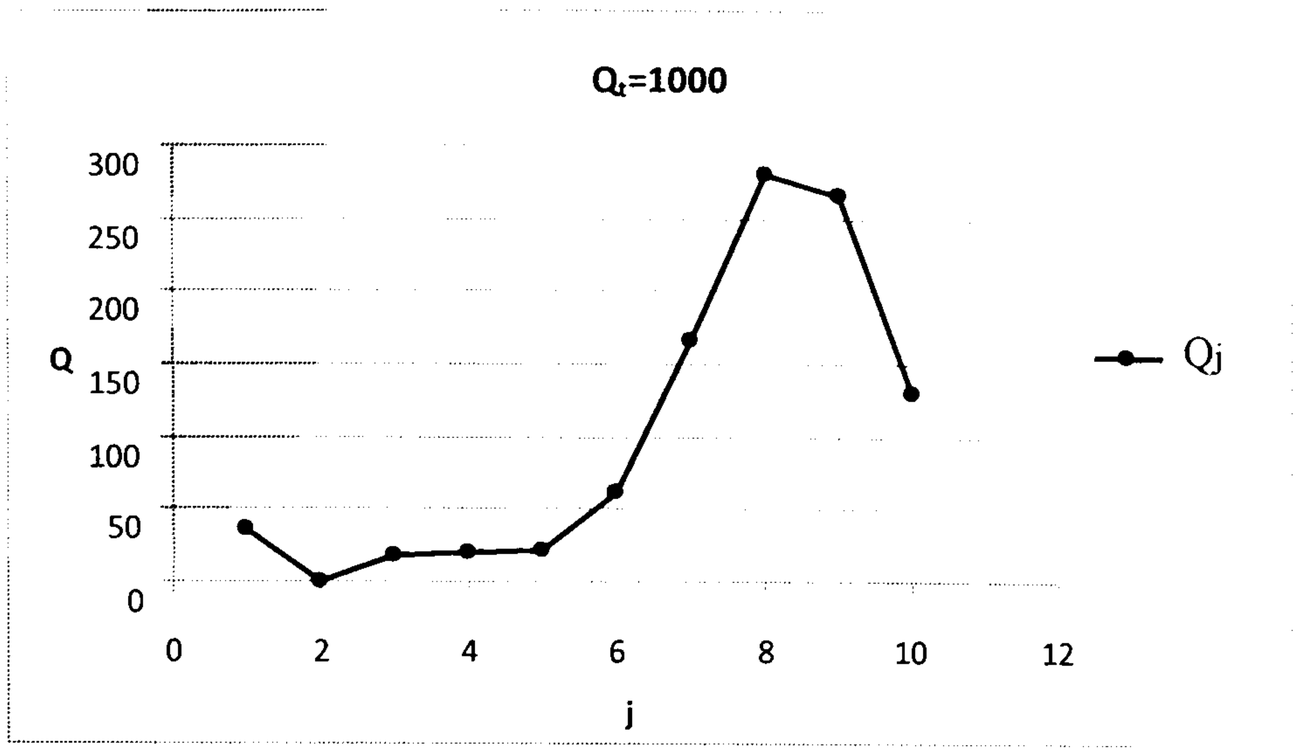


Figure 6. 4 Convex distributions

The distribution of population has bias to lower pools. This bias becomes more and more obvious as COPT proceeds. In contrast to concave distributions, the number of intermediate solutions within a low quality level is smaller than the number of intermediate solutions within a high quality level. The Markov process for a lower pool is launched from a large number of good intermediate solution in promising areas, whilst the Markov process for a higher pool focuses on a small number of points. In such case, the search cannot easily explore the search space and is likely to be distracted by too many intermediate solutions within high quality levels.

As r increases, the convexity of a cooling schedule increases. Two convex cooling schedules are selected as:

- The one with a small convexity $(r = 2)$
- The one with a large convexity $(r = 10)$

The corresponding distributions of population at the last period are extracted and illustrated in Figure 6.4.

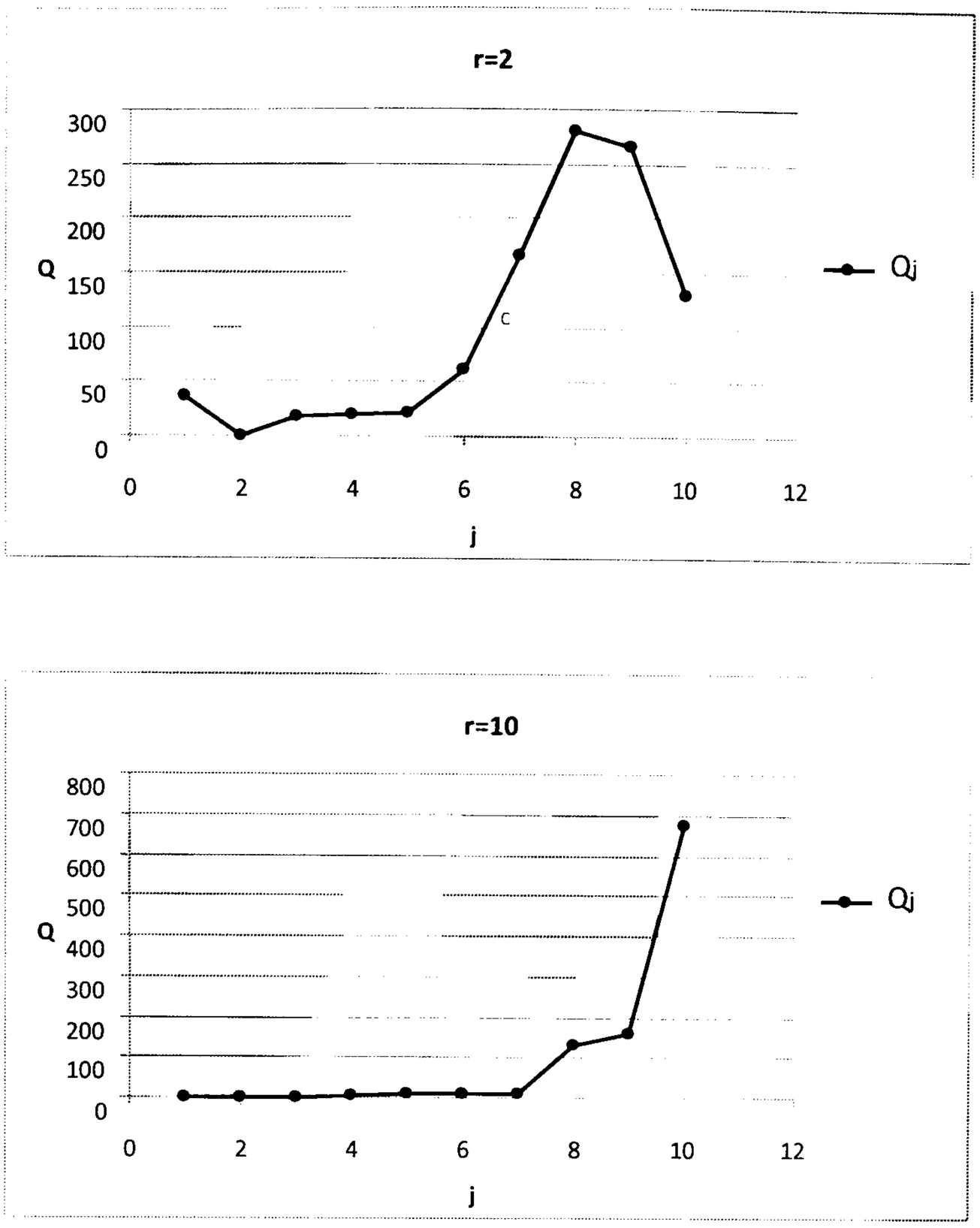


Figure 6. 5 Convex distributions with different *r*

The bias of distributions moves downward to the bottom pools (P_{n_p}) as the convexity increases. Finally, the most of the population is distributed to P_{n_p} .

To study the effect of *r* and the distribution functions, the cooling schedules with nine *r* values from -10 to 20 are selected. The top pool temperature (T_1) should be a large value so that the Markov process launched from the top pool can accept all new points. In contrast, the bottom pool temperature (T_{n_p}) should be a small value so that the Markov process launched from the bottom pool can only accept the new points better than the current points. So $T_1 = 100$ and $T_{n_p} = 0.01$ are selected. A Markov process with the fixed length (L=10) is selected to generate new points. The set of experiments concludes convergence on the basis of the population of pools so that COPT terminates when the

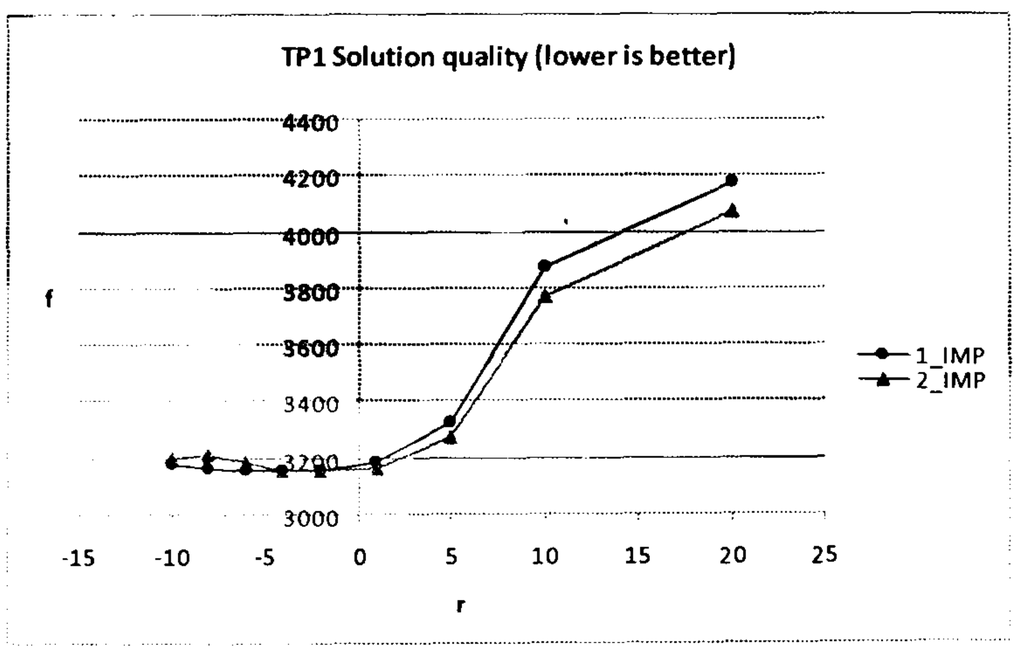
population of pools (Q) approaches a maximum population (Q^{max}) (the complete set of termination criteria are discussed in Section 6.7). The test problems are small-scale so that the stochastic search can quickly generate a large population. Q^{max} is assigned a large value so that

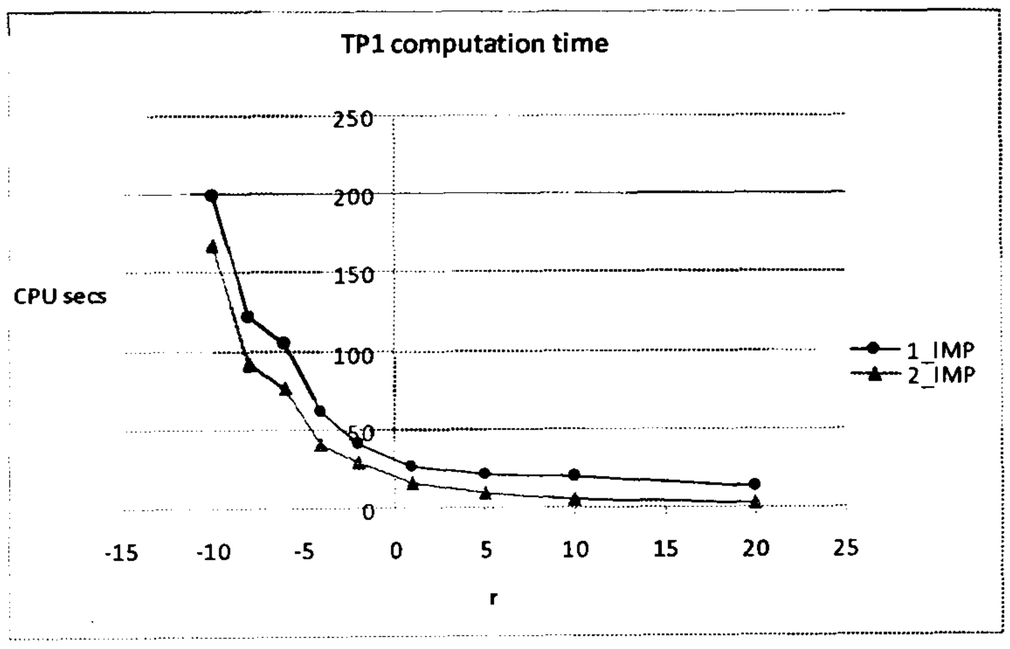
$$Q^{max} = 2000$$

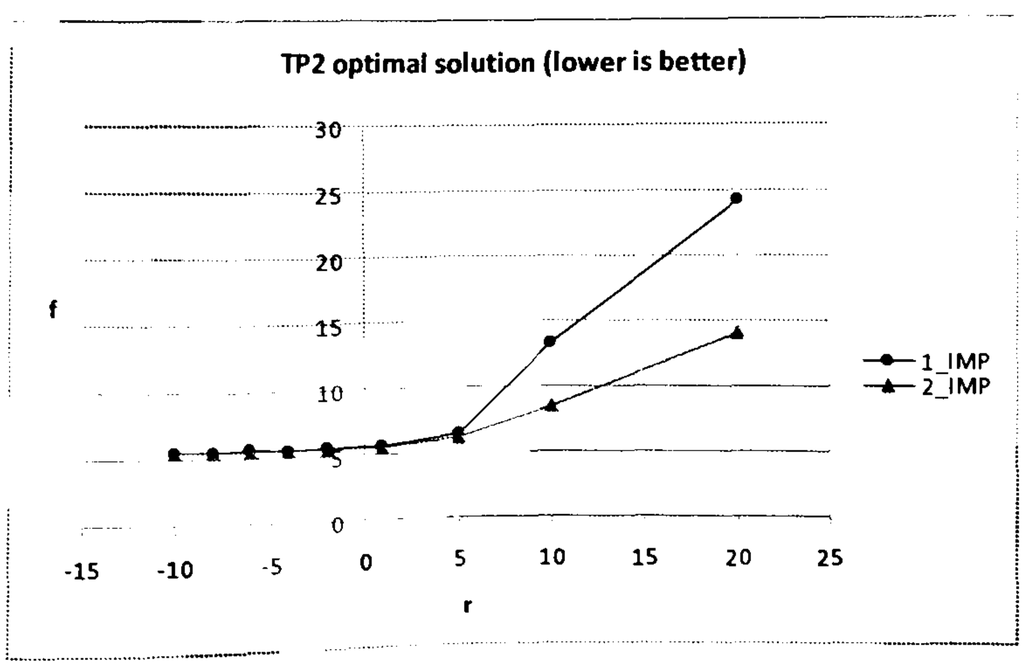
If the population is distributed in a large number of pools, the impact of distribution functions can be easily extracted. So the number of pools is fixed as:

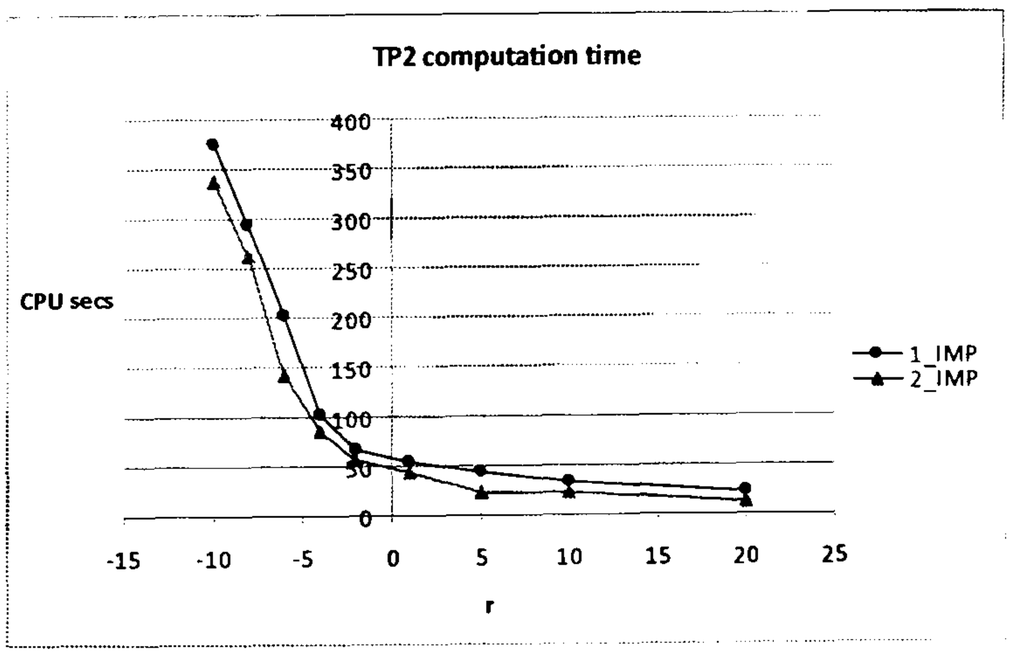
$$n_p = 100$$

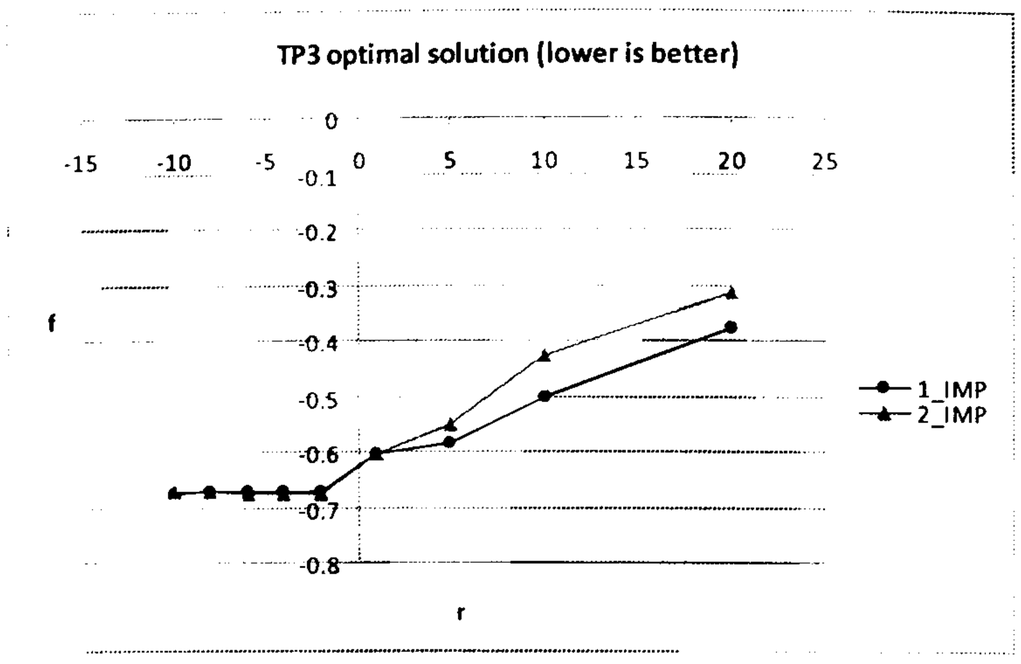
10 statistical runs with different initial points are selected to test each distribution function. The results are the averages of the 10 runs. Figure 6.6 presents the performances of COPT.

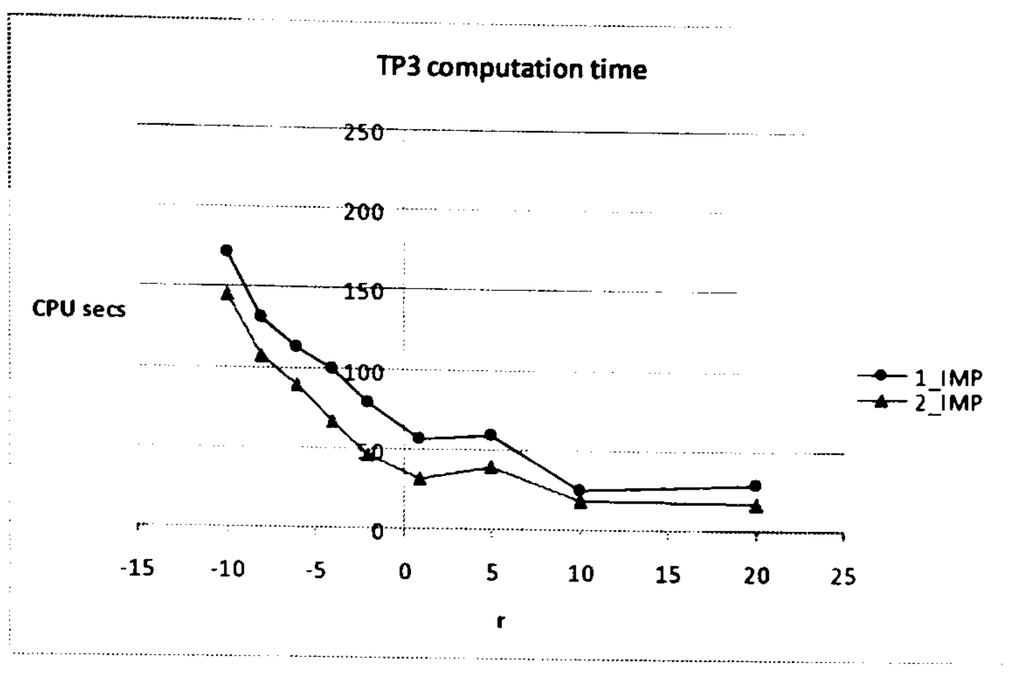


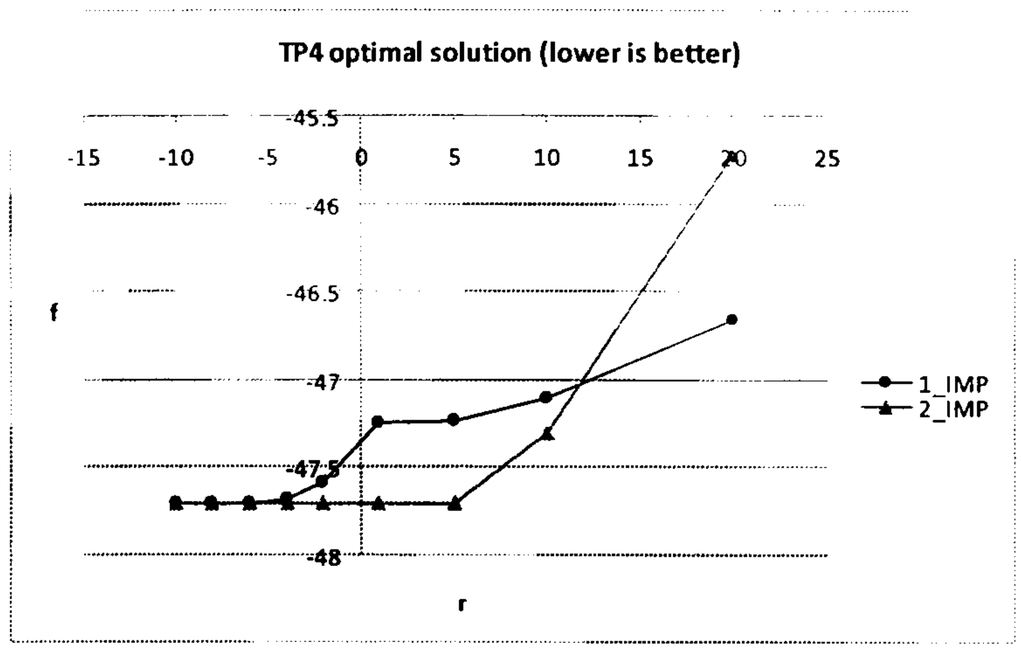












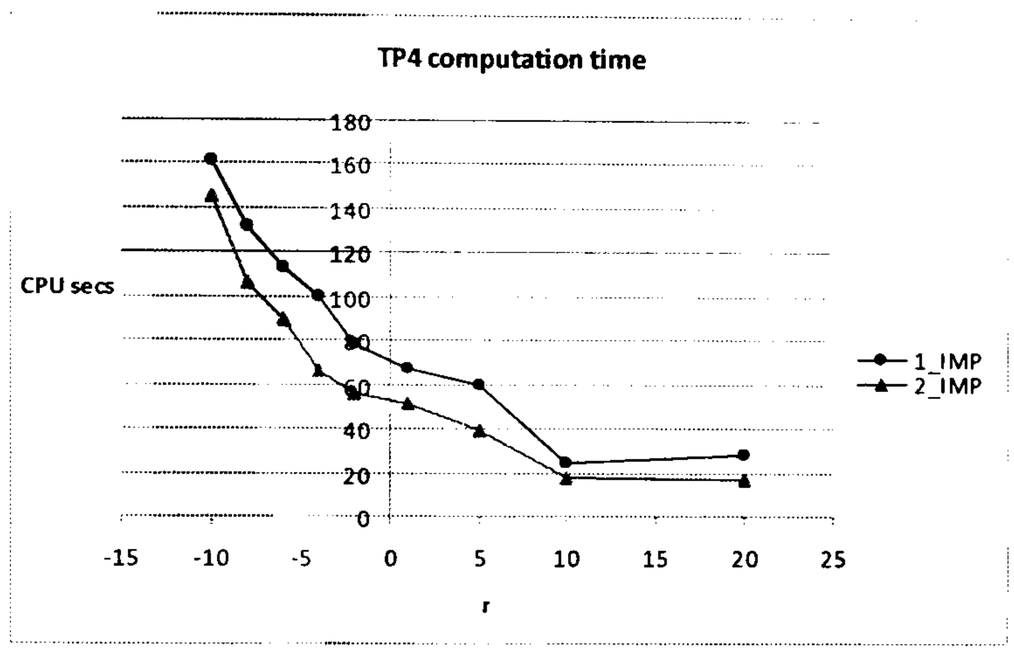


Figure 6. 6 Performance with different distribution functions

f: objective values

1_IMP: the first implementation

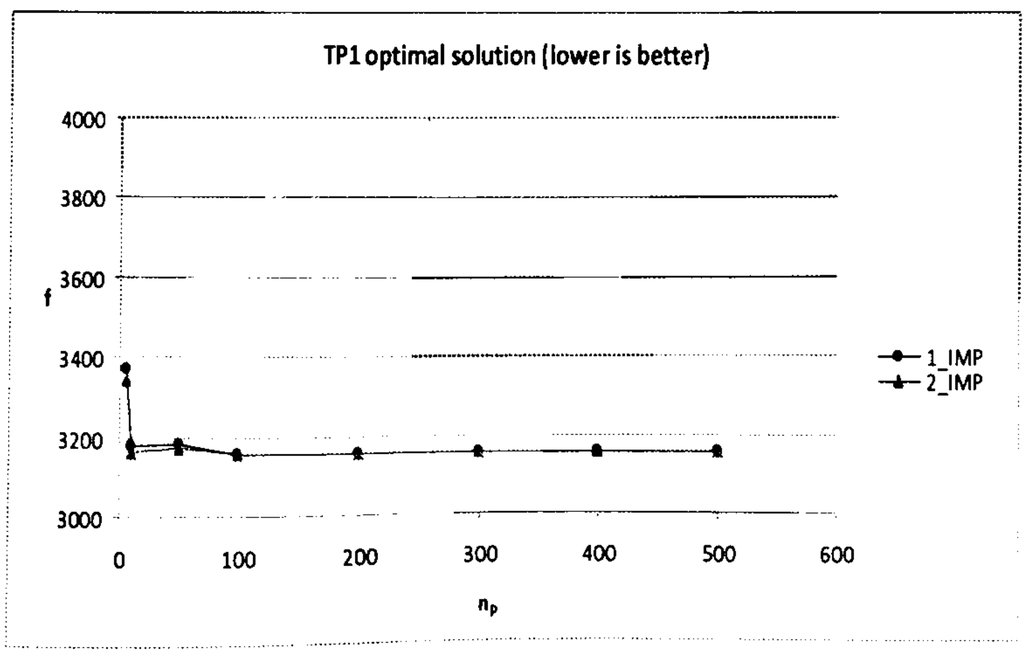
2_IMP: the second implementation

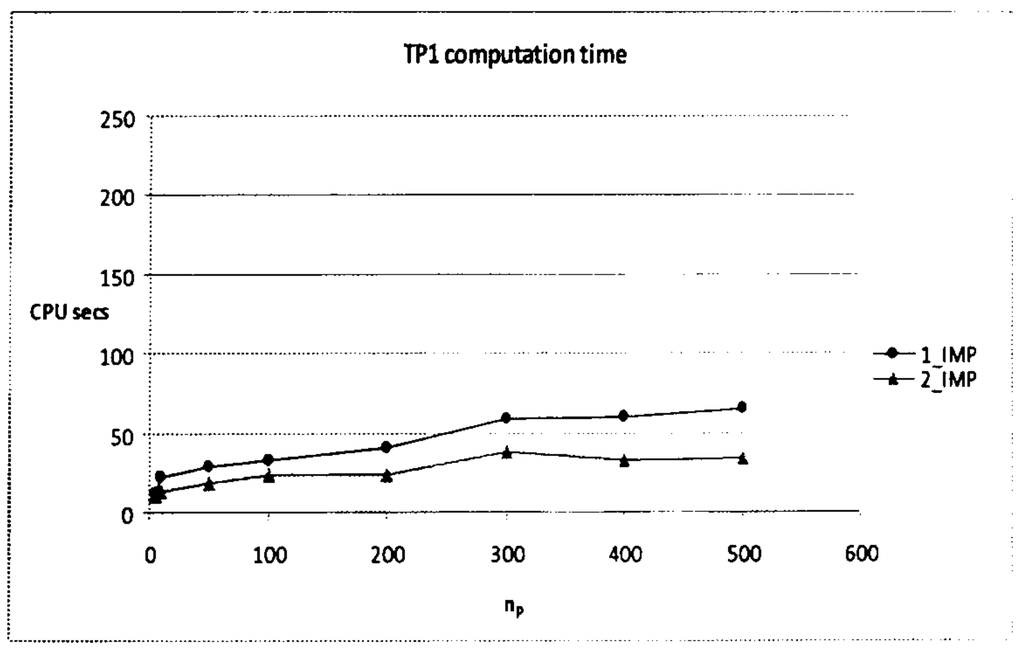
The distribution of population has an increasing bias moving toward to the bottom pool as *r* increases. This bias decreases the solution qualities of COPT. Meanwhile, the computation time decreases with this bias. In addition, the computation time for the first implementation is larger than that for the second implementation. That is because COPT that uses the second implementation launches the tasks simultaneously so that the computation time is not sum of the CPU times for the tasks but the CPU time for Tsk^{MC} . Based on the results, adopting the concave distribution with *r* close to -2 can give the best tradeoff between the solution quality and the computation time.

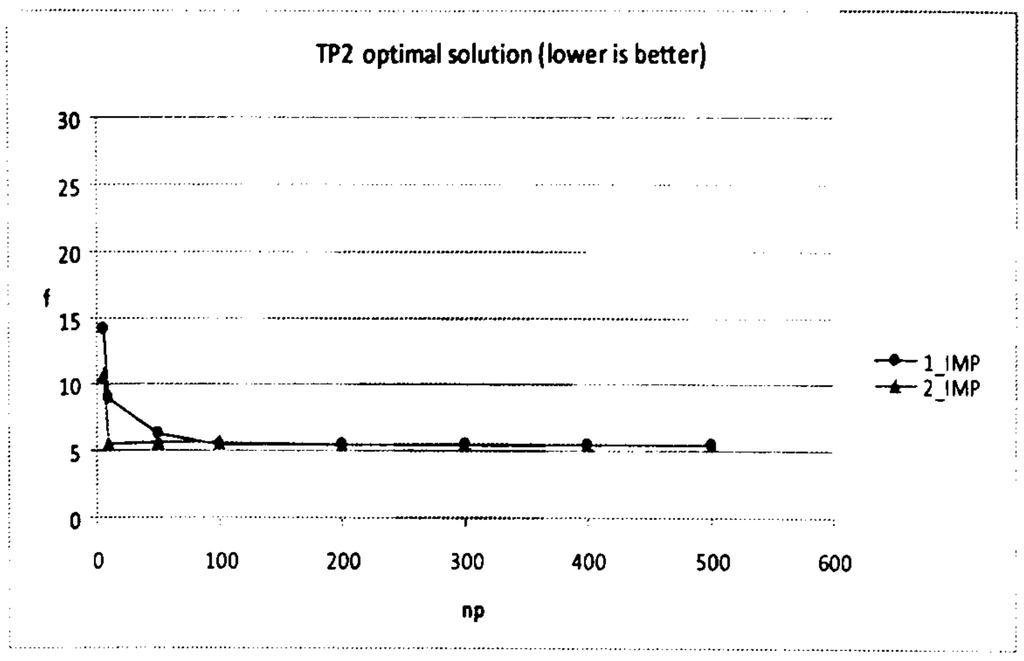
6.5 Size of optimisation structure

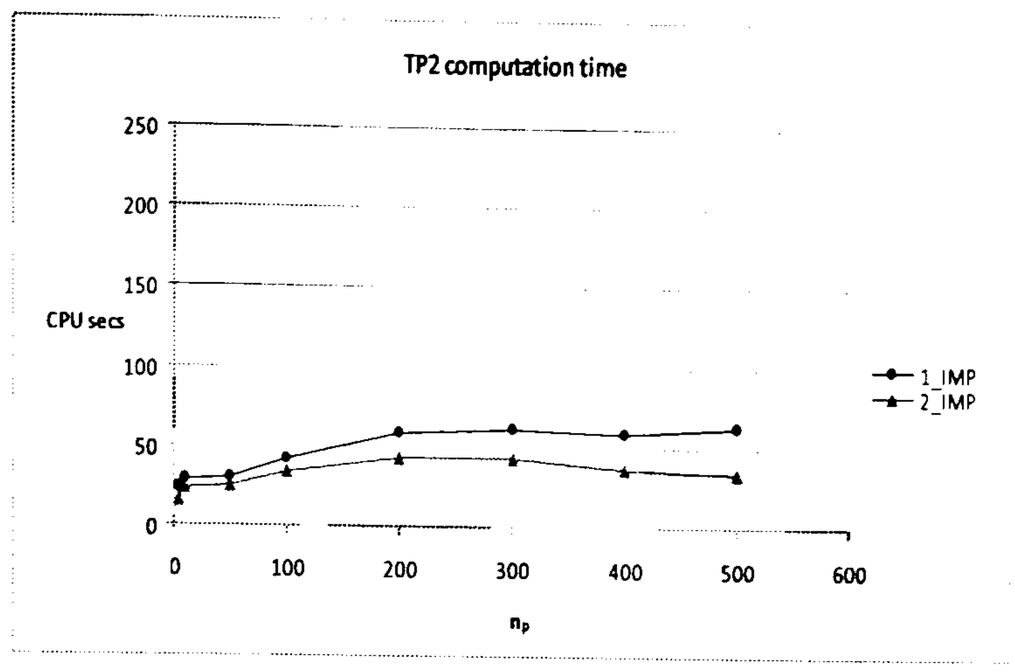
The size of optimisation structure is determined by the number of pools (n_p). The Markov process launched from a pool follows an acceptance probability. With a large n_p, the Markov process follows a large number of probabilities so that the search might converge to the optimal solution. In theory, increasing n_p increases the CPU time for Tsk^{Inf} and Tsk^{Tm}. The computation time for the first implementation increases with n_p. In contrast, the computation time for the second implementation is the CPU time for Tsk^{MC}. The impact of n_p on computation time depends on the Tsk^{MC}.

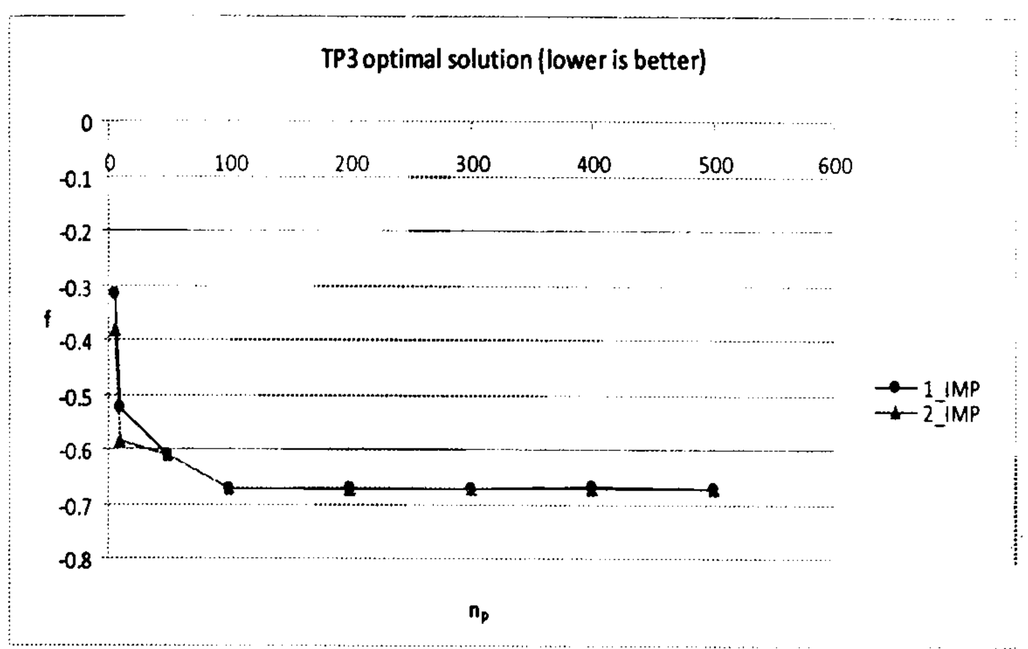
T_1, T_{n_p}, L, and $Q^{\max}$ are set the same as those in the study of distribution functions. The concave distribution function (r=-2) is selected. Eight different structure sizes are selected with n_p from 5 to 500. Figure 6.7 illustrates the performances. Similar to the study in distribution functions, the results are the average of 10 stochastic runs.

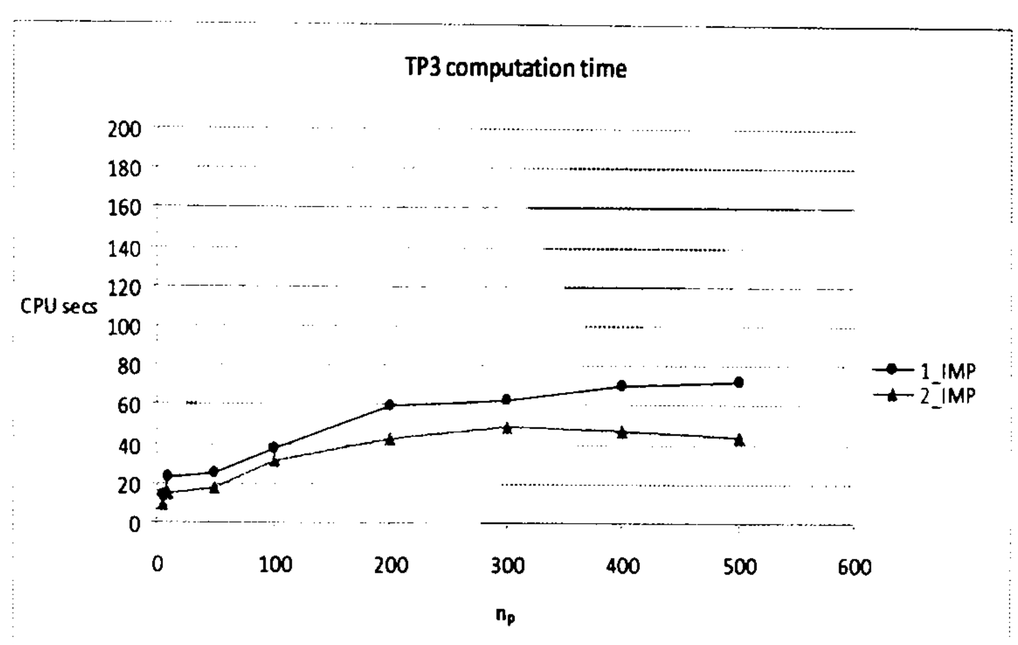


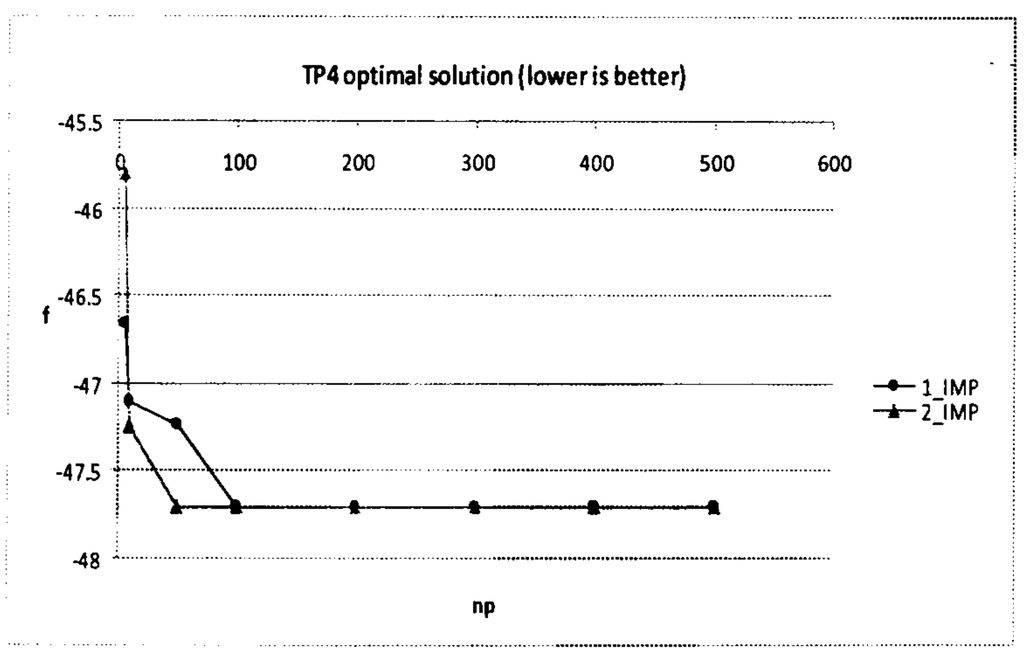












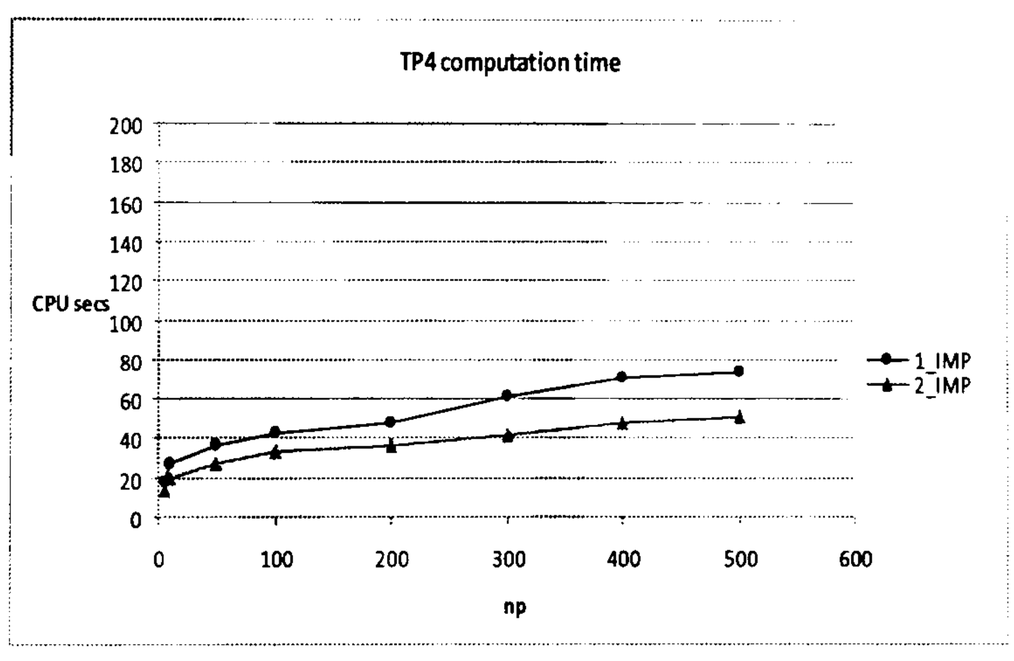


Figure 6. 7 Performance with different structure sizes

Based on the figures, increasing structure size increases the solution qualities. The incremental benefit diminishes as n_p is larger than 100. Table 6.3 presents the standard deviations of $F^{\min}$ ($\sigma_{F^{\min}}$) in two ranges ($n_p < 100$ and $n_p \geq 100$) when using the first implementation.

Table 6. 3 Standard deviations of $F^{\min}$

Test problems	$\sigma_{F^{\min}}$ $(n_p < 100)$	$\sigma_{F^{\min}}$ $(n_p \geq 100)$
TP1	109.164	0.539
TP2	4.019	0.00261
TP3	0.150	0.00248
TP4	0.297	0.000332

Let us take TP1 for example. $\sigma_{F^{\min}}$ for $n_p < 100$ is about 170 times larger than that for $n_p \geq 100$.

The computation time increases with the structure size. That means that COPT with a large optimisation structure size requires a long computation time to converge. To check the degree of increment on computation time, an increasing ratio is introduced. This ratio is measured by the percentage of the time difference between $n_p = 5$ and $n_p = 500$ over the time of $n_p = 500$ (r^{Ins}). Table 6.4 illustrates the ratio for the two implementations.

Table 6. 4 Increasing ratios

Test problems	r^{Ins} (1_IMP)	r^{Ins} (2_IMP)
TP1	0.814	0.632
TP2	0.632	0.419
TP3	0.808	0.683
TP4	0.762	0.668

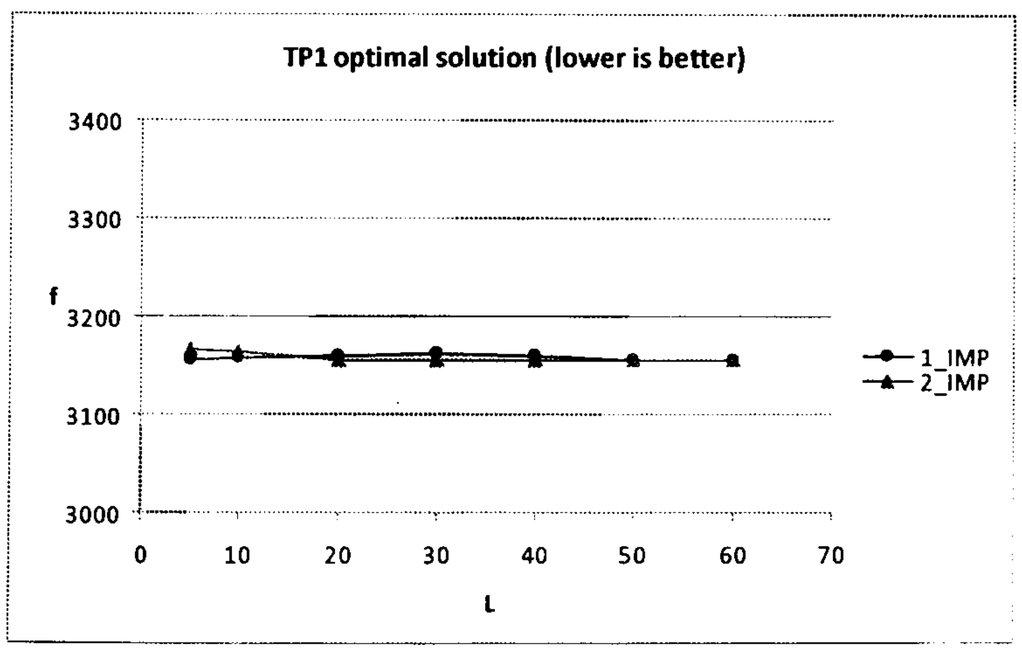
As the table illustrates, the two implementations have large increasing ratios (the smallest one is 42%). However, COPT can be applied on the parallel and distributed

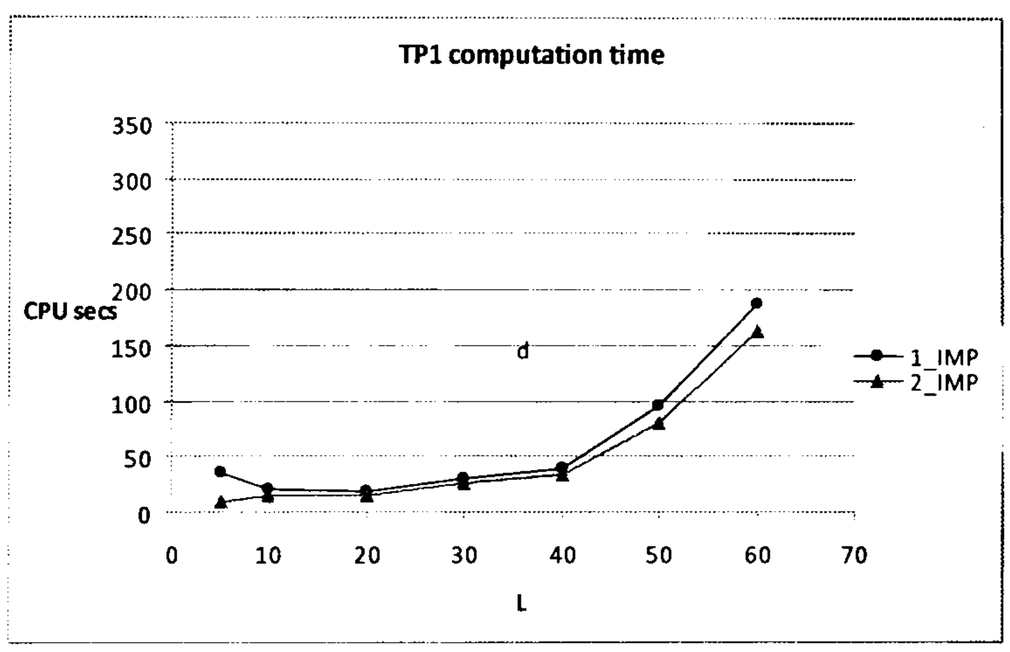
computing environment so that the resulting increment can be reduced.

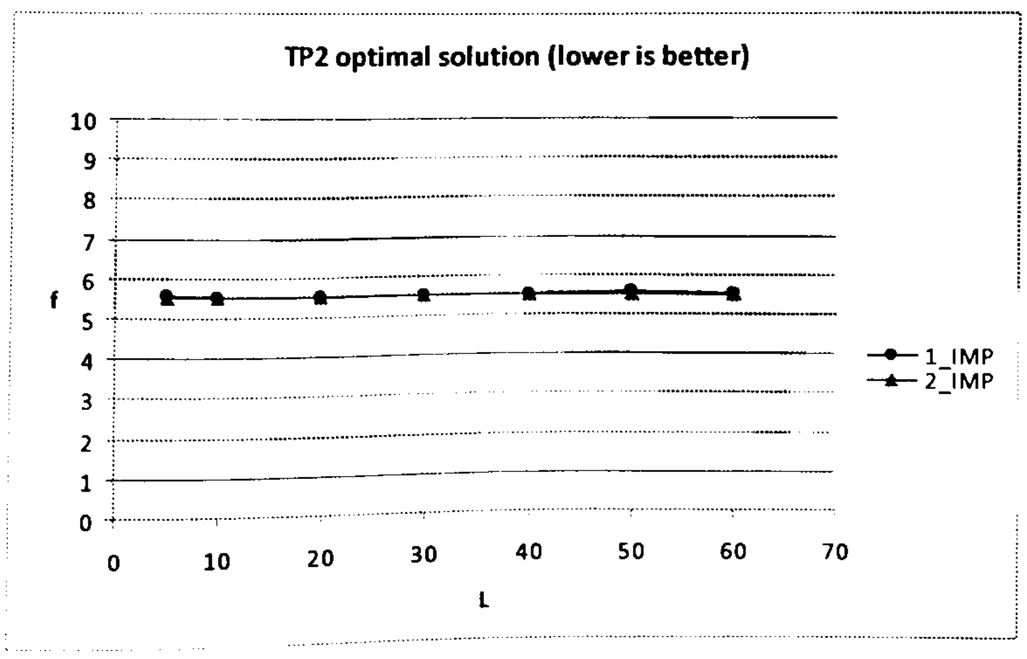
6.6 Depth of search

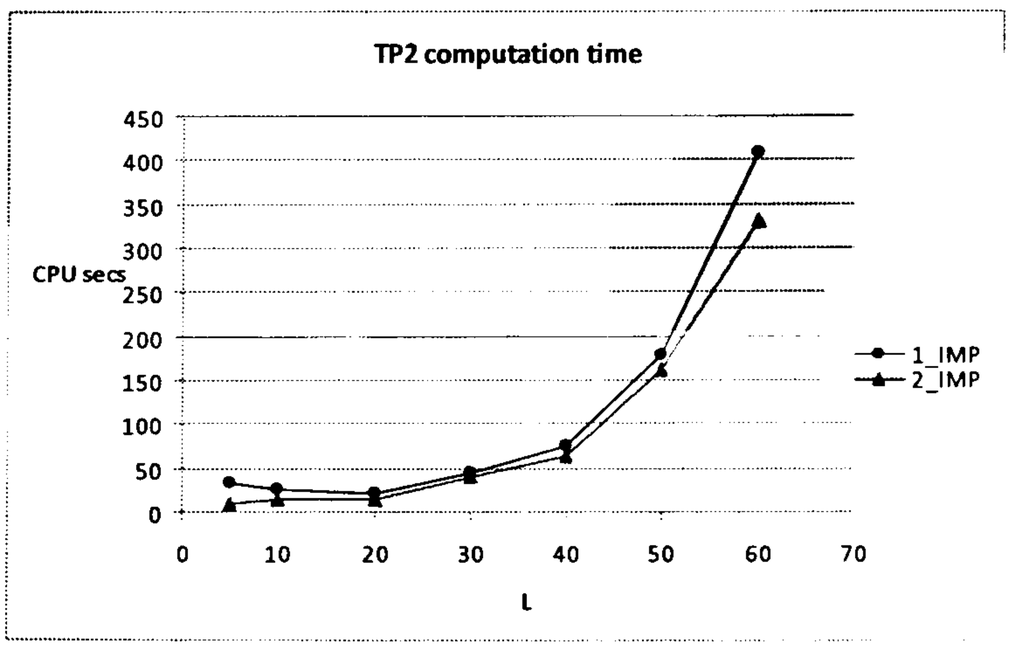
The depth of search is reflected on the length of Markov process (L). A long Markov process can explore more points than a short Markov process. COPT with a long Markov process can easily converge to the optimal solution but at the expense of a long computation time.

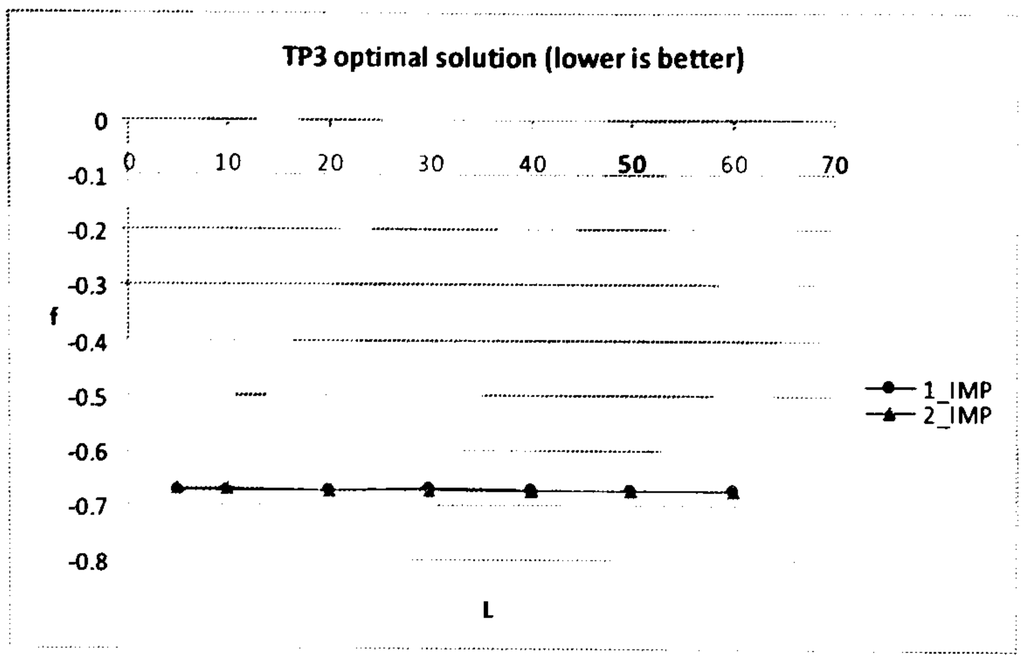
To study the depth of search, parameters (T_1, T_{n_p}, r, and $Q^{\max}$) are set same as those in the study of the size of optimization structure. $n_p = 100$ is selected. Seven different search depths are selected with L from 5 to 60. The results are the averages of 10 statistical runs. Figure 6.8 presents the performances with different search depths.

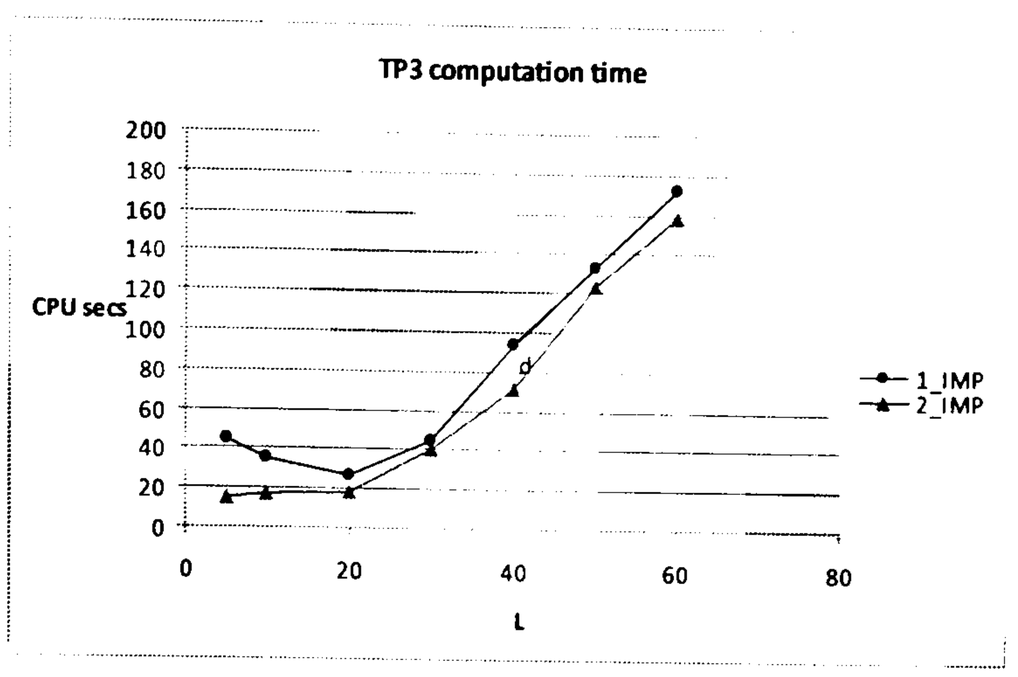


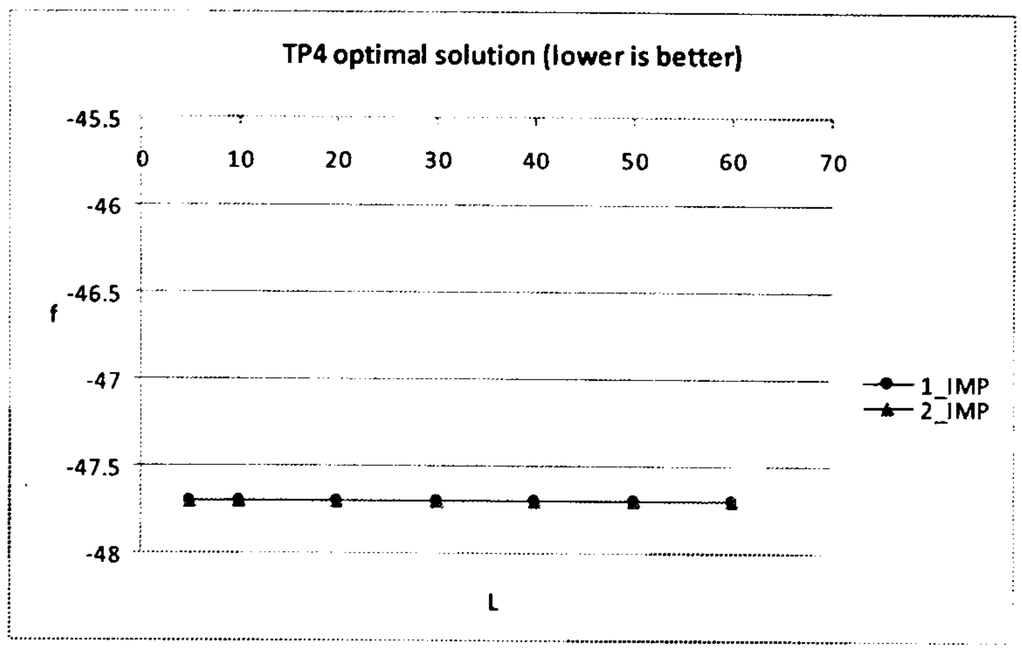












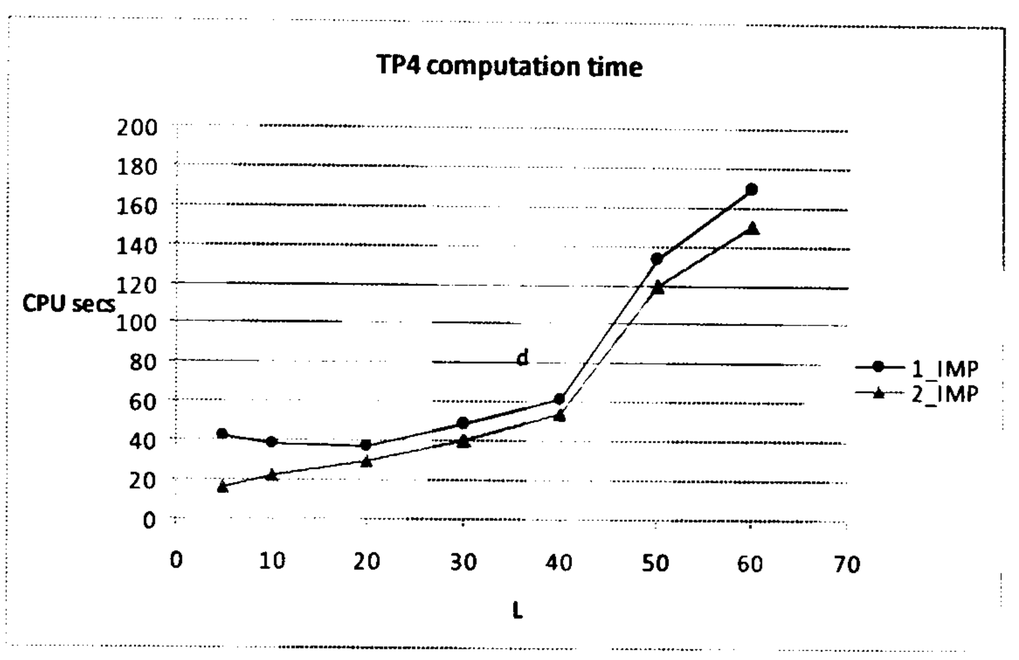


Figure 6. 8 Performance with different search depths

Based on these results, COPT cannot achieve obvious benefit in solution quality by increasing search depths. A percentage of convergences is introduced to test the benefits. Let us take the results of TP1 for example. Table 6.5 presents the percentages of convergences (p^{cvg}).

Table 6. 5 Percentages of convergences

L	p^{cvg} (1_IMP)	p^{cvg} (2_IMP)
5	92.4%	94.6%
10	92.6%	93.4%
20	94.6%	99.6%
30	99.6%	98.5%

40	99.7%	100%
50	100%	99.6%
60	99.6%	99.7%

Based on the table, the search depth has a small effect on convergence (the minimum $p^{cvg} = 92.4\%$).

For the first implementation, the computation time decreases firstly and then increases as L increases. The computation time involves the CPU time for Tsk^{MC} (T_1^{CPU}) and Tsk^{Ep} (T_4^{CPU}). T_1^{CPU} is large when COPT uses a long Markov process. However, T_4^{CPU} is small since M-W information is transferred to the worker only a few times. The increasing search depth increases T_1^{CPU} but decreases T_4^{CPU} . Initially, the decreasing rate of T_4^{CPU} is larger than the increasing rate of T_1^{CPU} so that the computation time decreases. Later on, the decreasing rate is smaller than the increasing rate so that the computation time increases. For the second implementation, T_1^{CPU} is the computation time and cannot be affected by T_4^{CPU} . So the computation time increases as the search depth increases. Since the search depth does not have a significant impact on the solution quality, the fixed search depth (L=10) is selected for the following studies.

6.7 Termination criteria

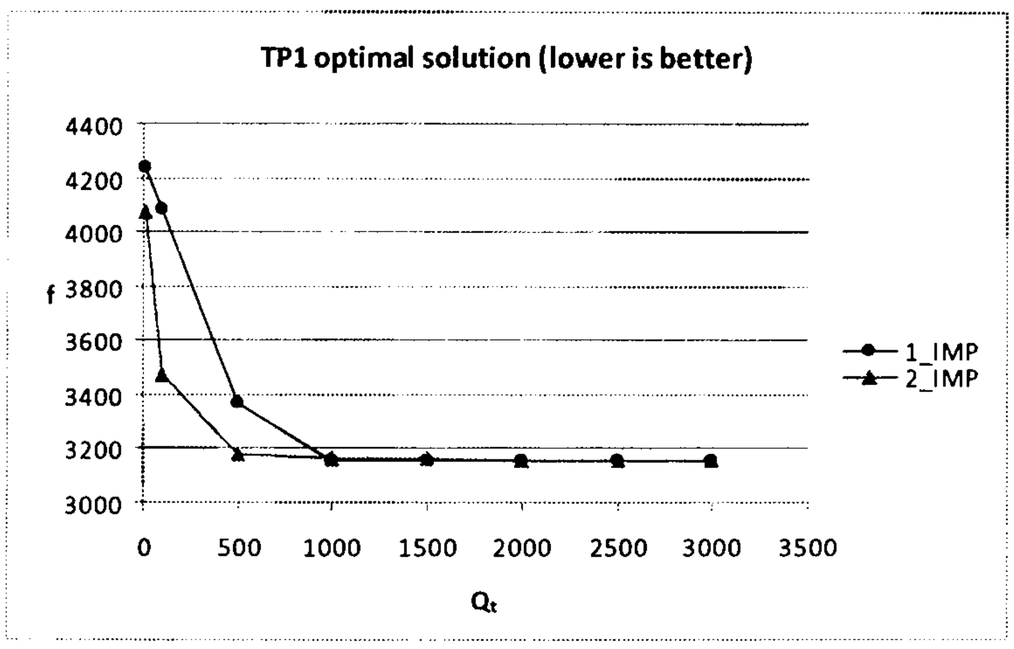
The proposed termination criteria involve:

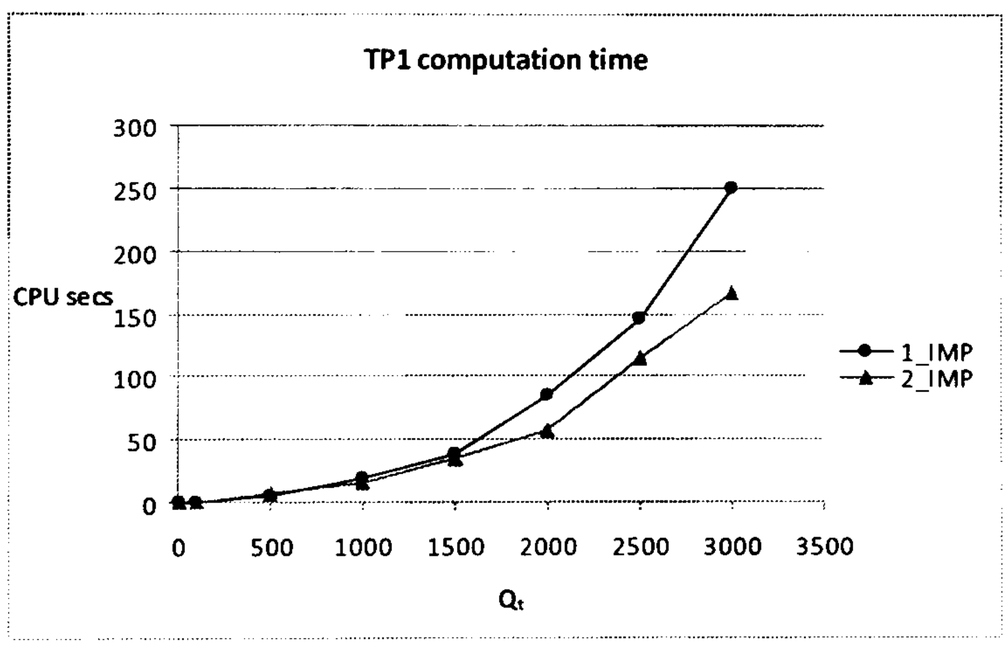
(i) Population size controls

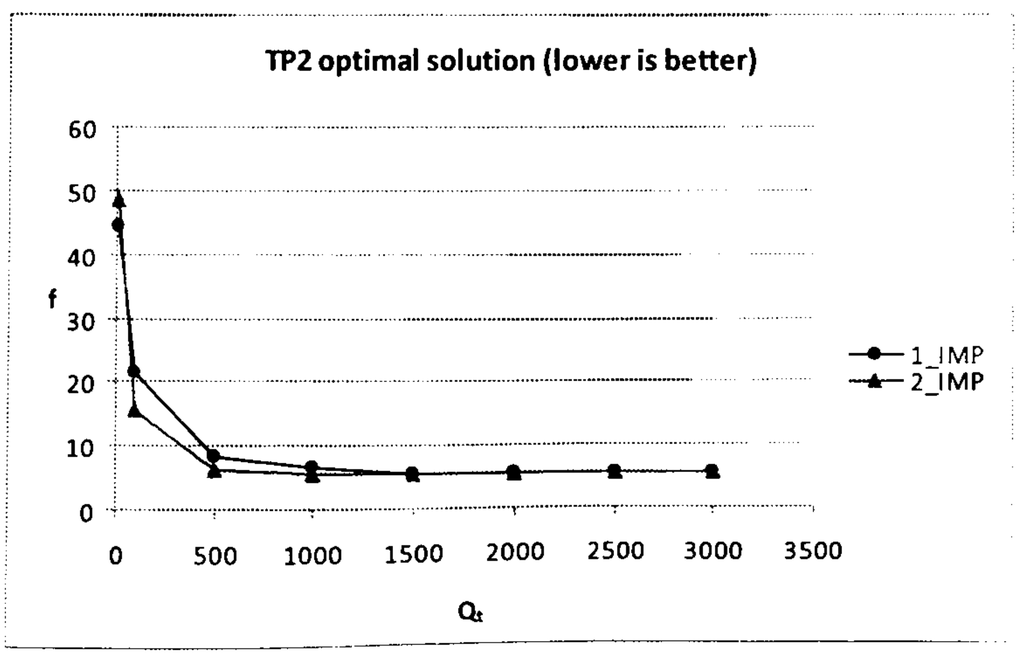
This termination criterion proposes the minimum population (Q^{min}) and the maximum population (Q^{max}). It provides:

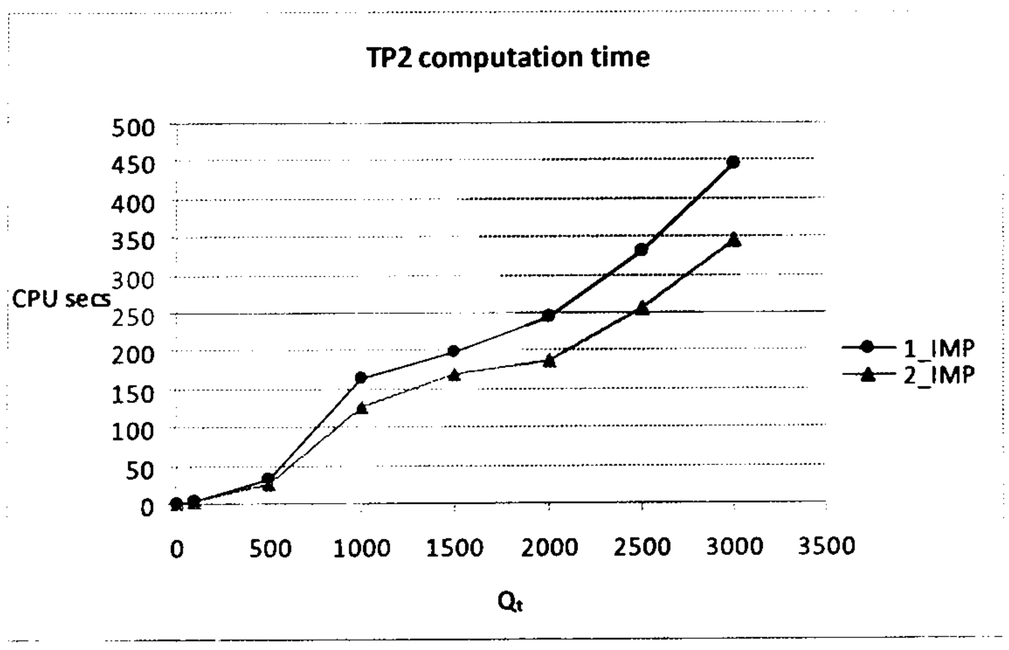
- COPT does not terminate if Q_t is smaller than Q^{min}.
- COPT terminates if Q_t is larger than Q^{max}.

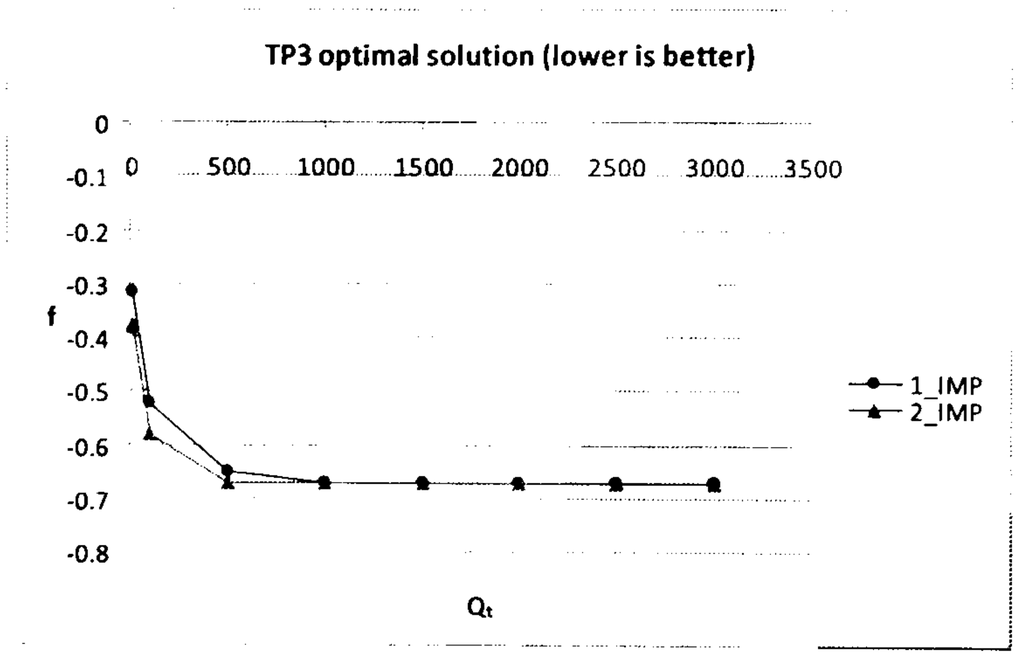
To study this termination criterion, parameters (T_1, T_{n_p}, r, n_p, and Q^{max}) are set same as those in the study of search depth. The Markov process (L=10) are selected. Eight Q_t from 10 to 3000 are selected to terminate COPT. Figure 6.9 presents the performances.

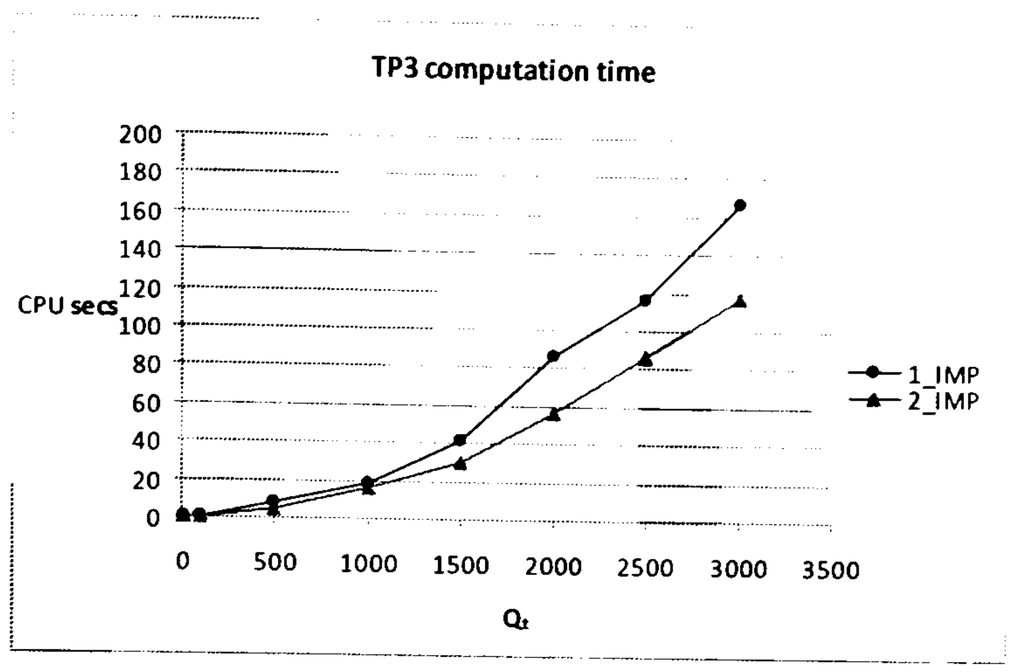


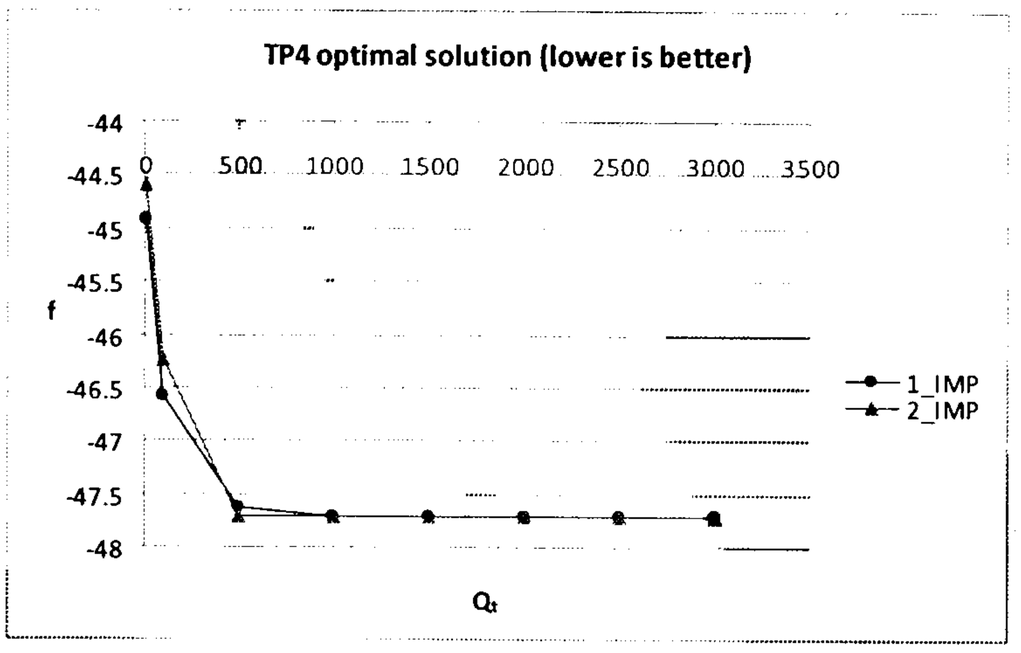












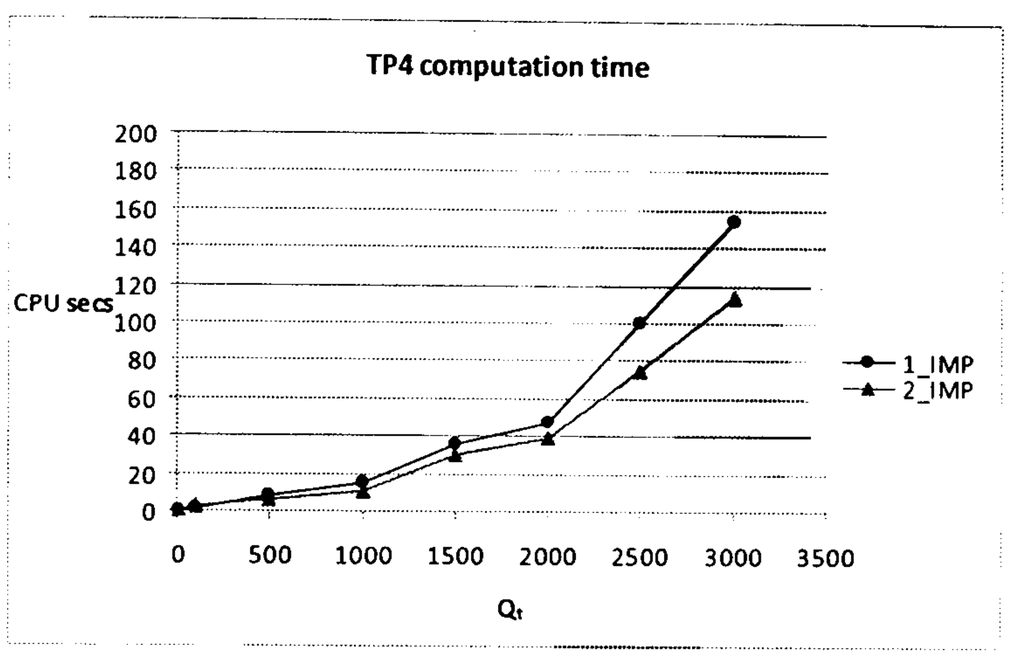


Figure 6. 9 Performance with different Q_t

Based on the results, increasing Q_t increases the solution quality. The incremental benefit diminishes when Q_t is larger than 1000. COPT can rarely converge to non-optimal or local optimal solutions if $Q_t \geq 1000$. Table 6.5 illustrates the standard deviations of $F^{\min}$ ($\sigma_{F^{\min}}$) ($Q_t < 1000$ and $Q_t \geq 1000$) when using the first implementation.

Table 6. 6 Standard deviation of $F^{\min}$

Test problems	$\sigma_{F^{\min}}$ $(Q_t < 1000)$	$\sigma_{F^{\min}}$ $(Q_t \geq 1000)$
TP1	567.229	1.292
TP2	18.311	0.418
TP3	0.167	0.000701
TP4	1.365	0.000772

Let us take TP1 again for example. $\sigma_{F^{\min}}$ for $Q_t < 1000$ is about 438 times larger than that for $Q_t \geq 100$. Thus, $Q^{\min}$ is set to 1000. On the other hand, $Q^{\max}$ is set to 3000 to ensure that COPT converges to the optimal solution.

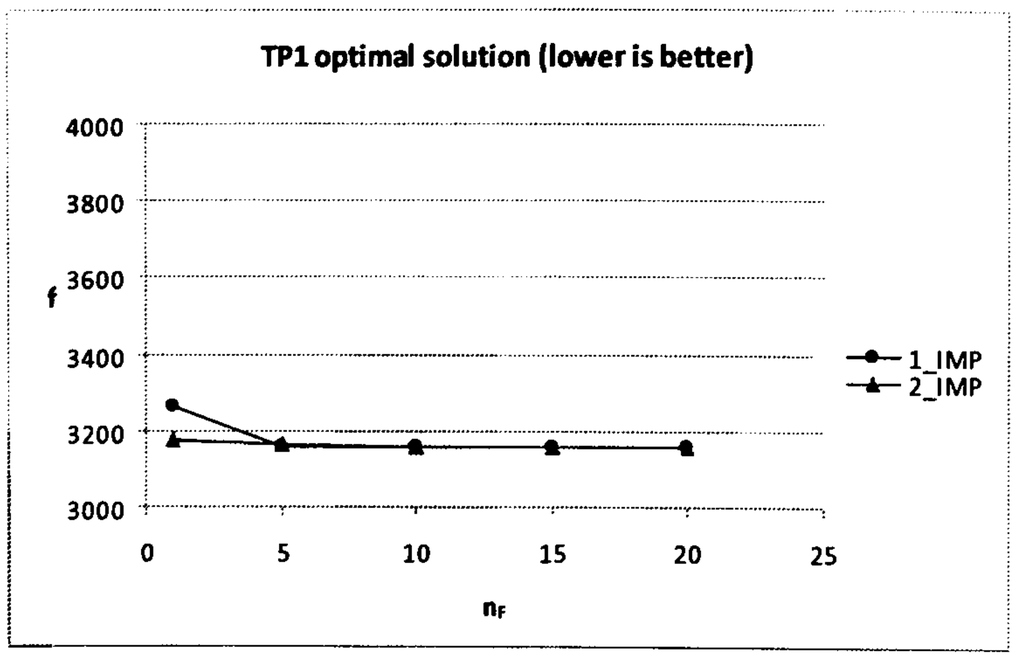
On the other hand, the computation time increases with Q_t. In addition, the difference between the computation times for the two implementations also increases. That is because for the first implementation the computation time involves the CPU times for Tsk^{Ep} and Tsk^{Ip}. These two CPU times increase since more and more W_M information and M_W information are transferred as Q_t increases.

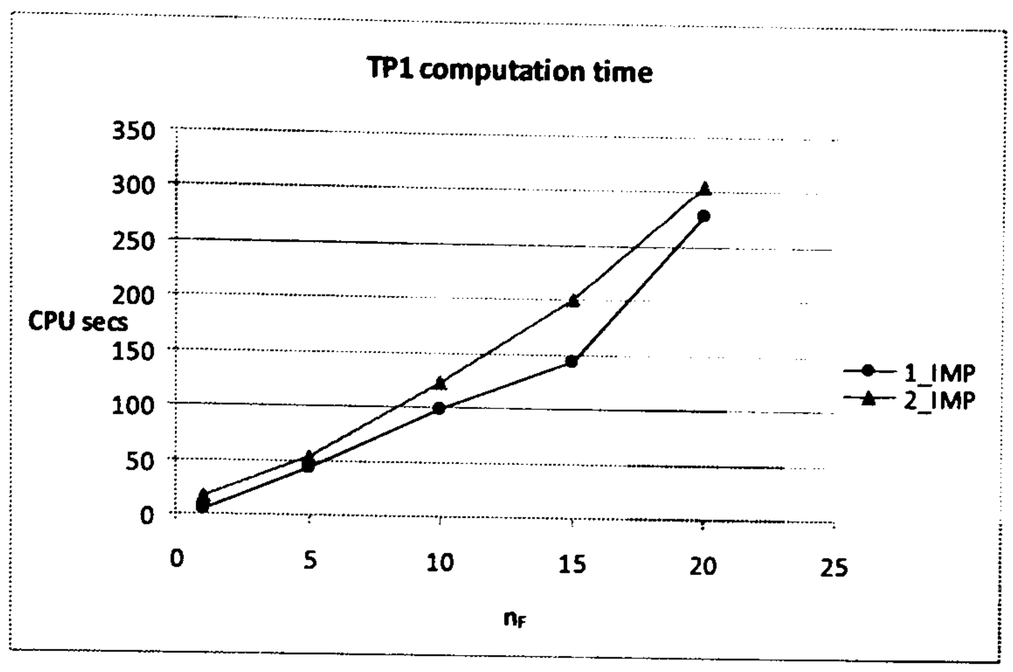
(ii) Optimisation Progress controls

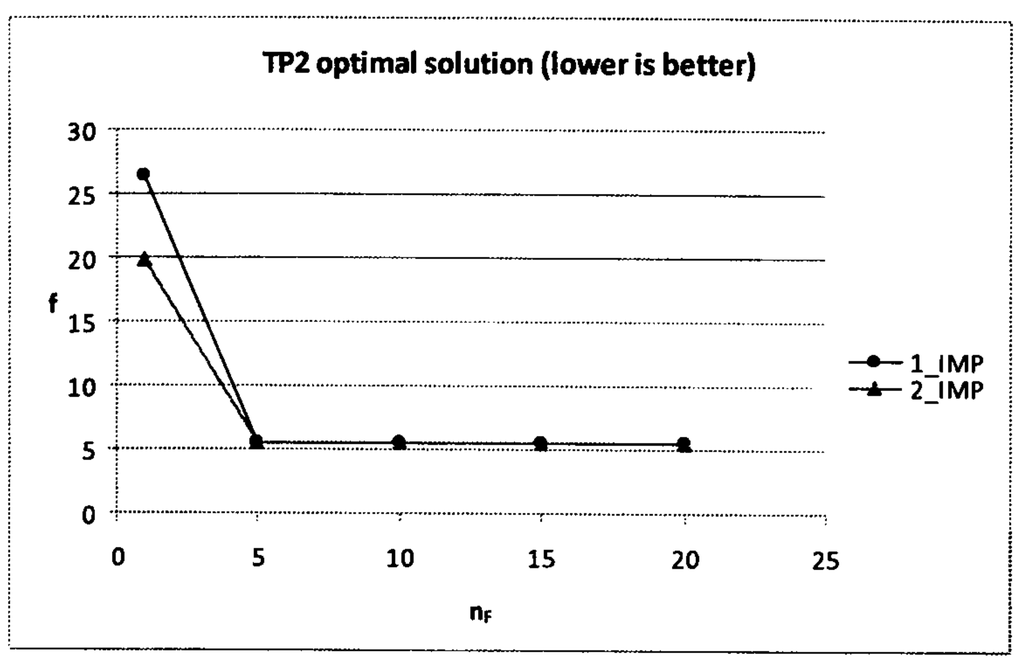
This termination criterion provides that COPT terminates if the best objective stays same:

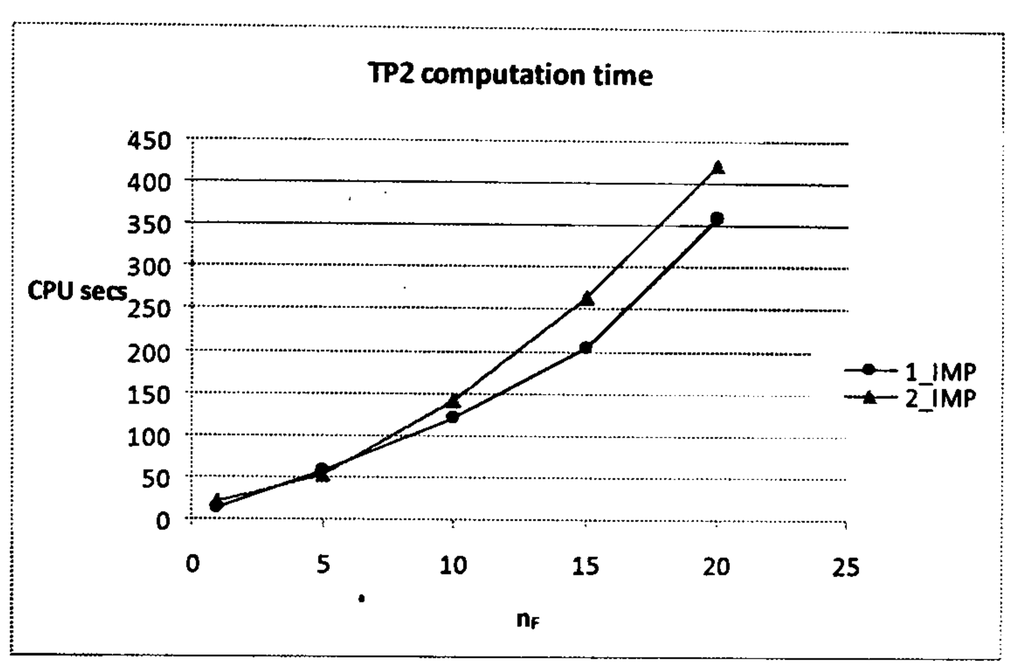
$$\left| F_t^{\min} - F_{t+L}^{\min} \right| \leq \varepsilon$$

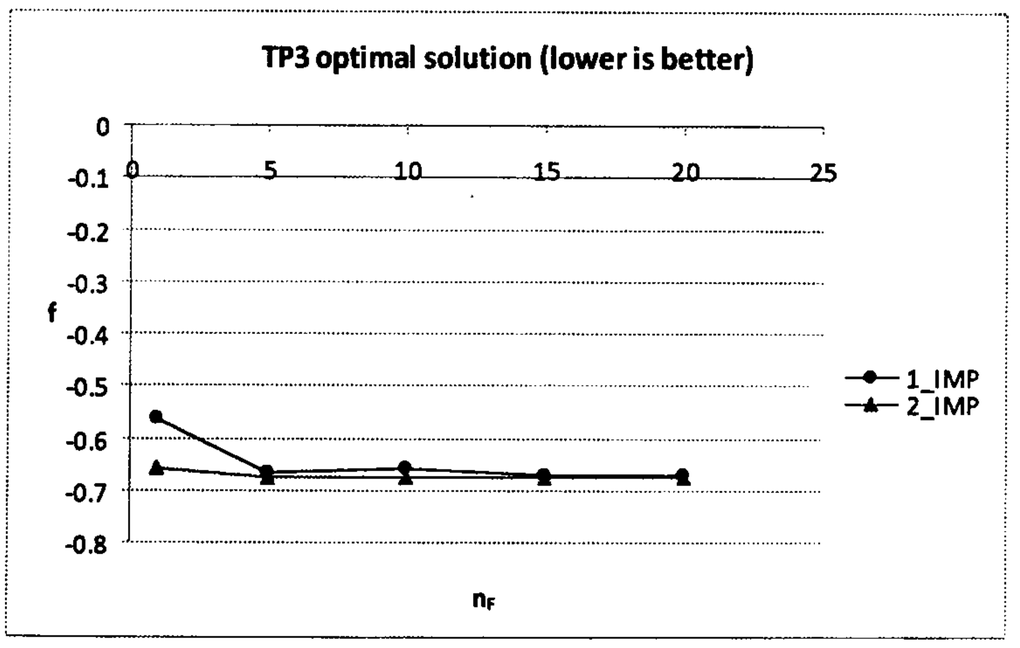
for a number of iterations (n_F). For the first implementation, $F^{\min}$ is checked every time after the Markov process. For the second implementation, $F^{\min}$ is checked every time after L new points are placed into partitions. $\varepsilon = 0.1$ is selected to determine the accuracy of the global optimal. Parameters (T_1, T_{n_p}, r, L, n_p, and $Q^{\max}$) are set same as those in the study of population size controls. Figure 6.10 presents the performances with different n_F.

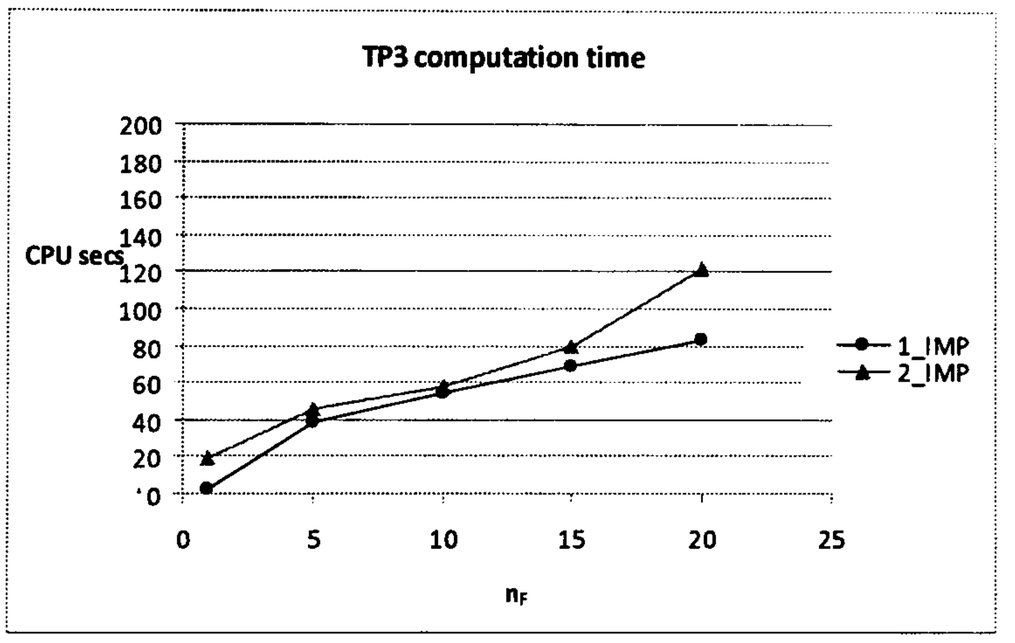


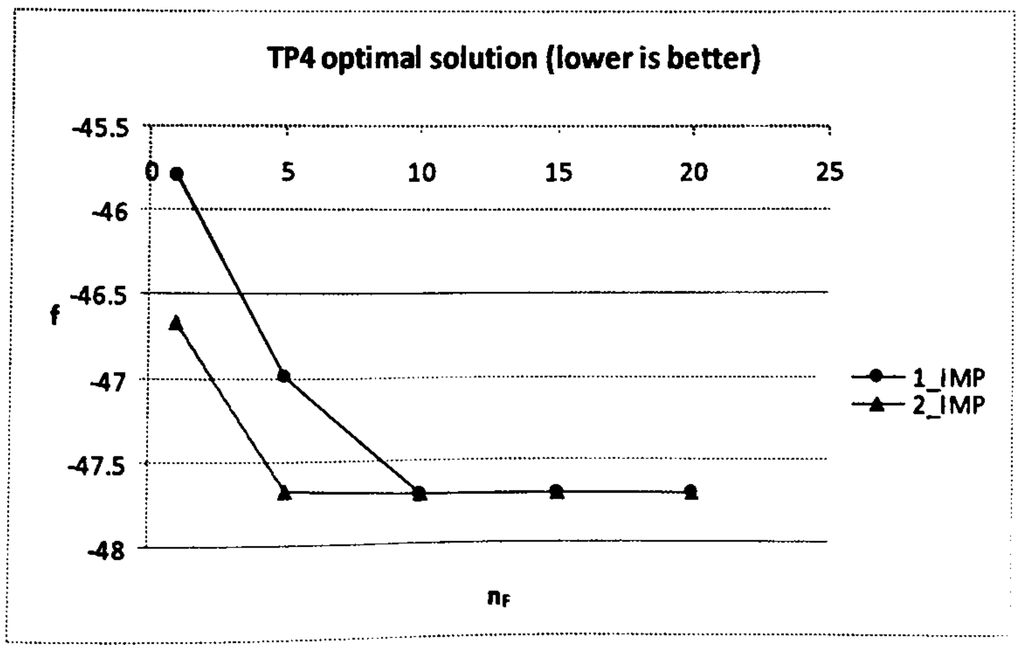












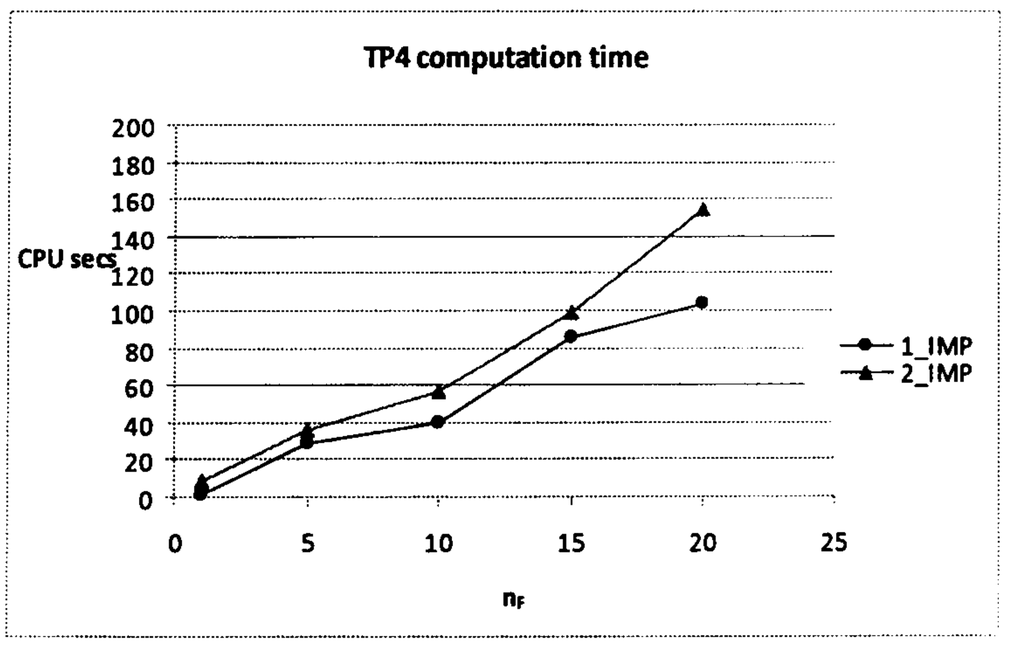


Figure 6. 10 Performance with different n_F

Based on the results, increasing n_F increases the solution quality. Meanwhile, the computation time increases with the n_F. Adopting $n_F = 10$ can give the best tradeoff between the solution quality and computation time.

(iii) Population feature control – pool level

The pool level is reflected on the standard deviation of the population of pools (σ_t). Figure 6.11 presents σ_t as COPT proceeds.

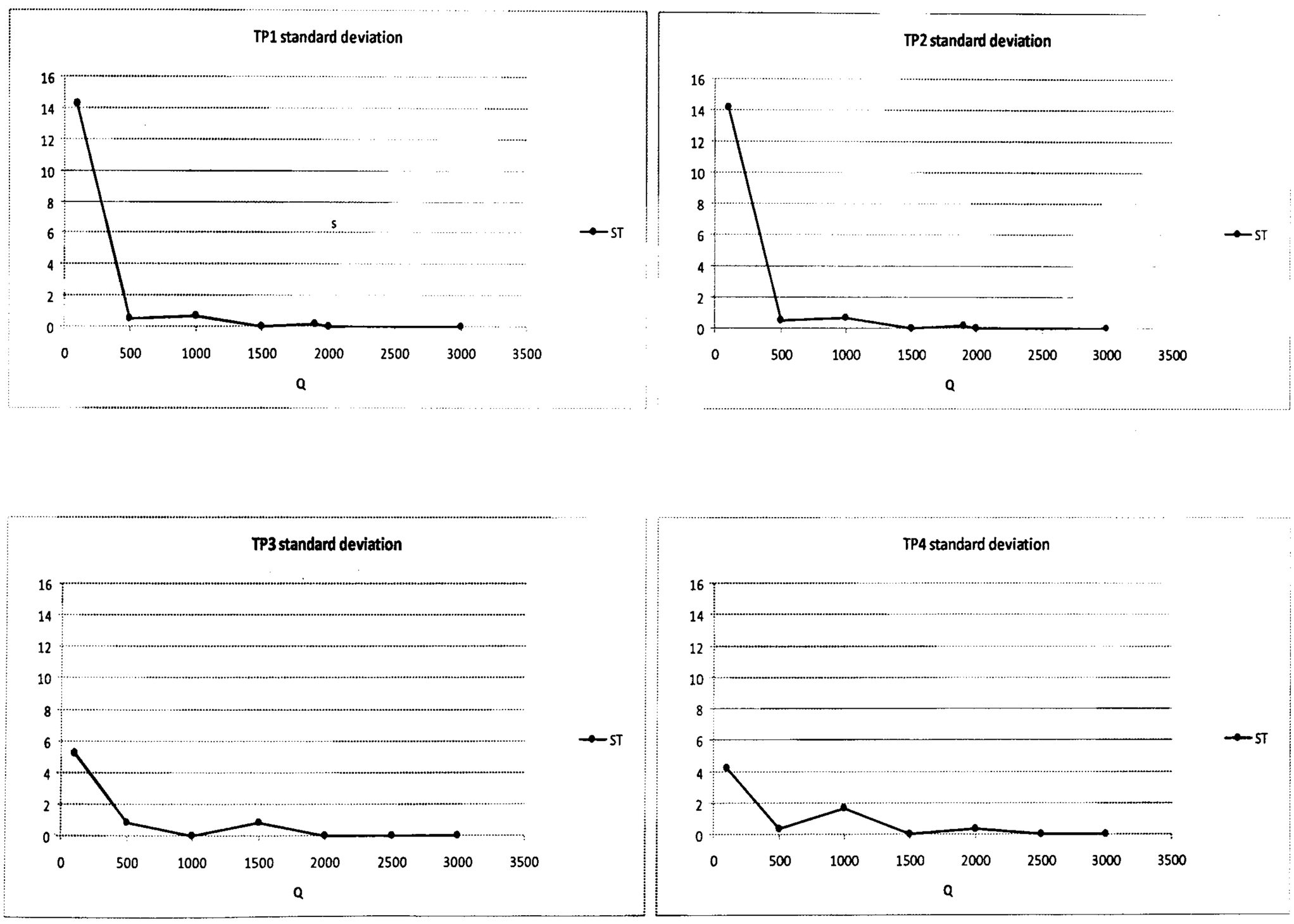


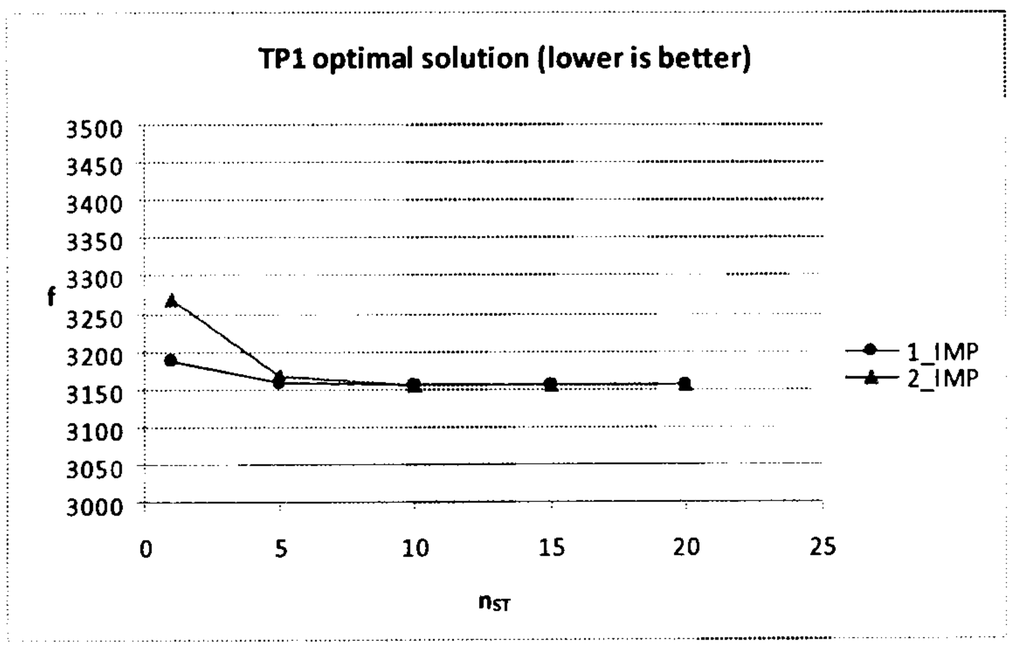
Figure 6. 11 Standard deviations of the population of pools

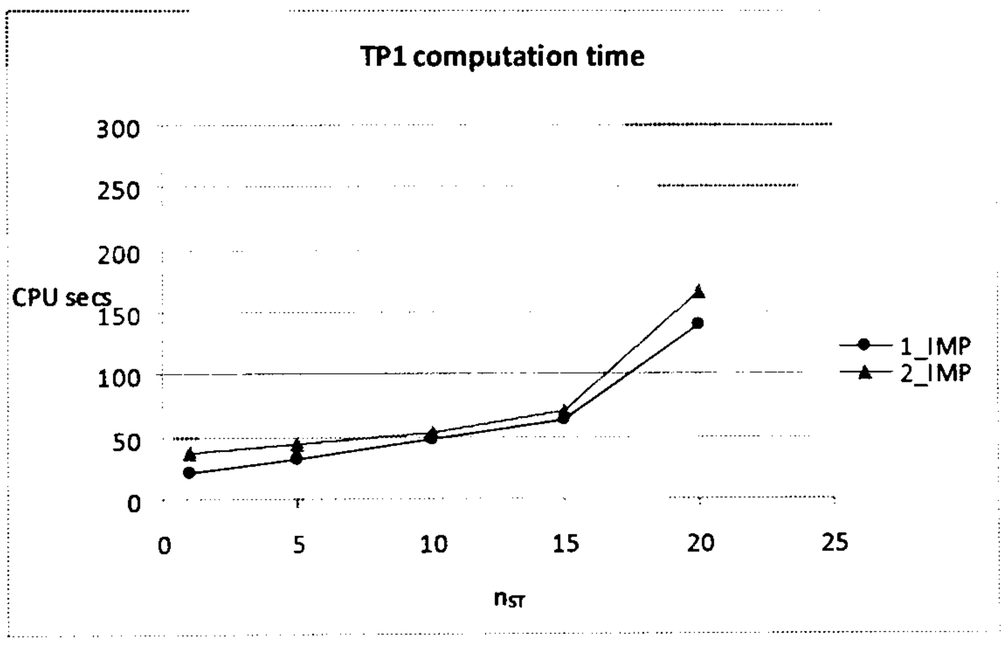
ST : the standard deviation

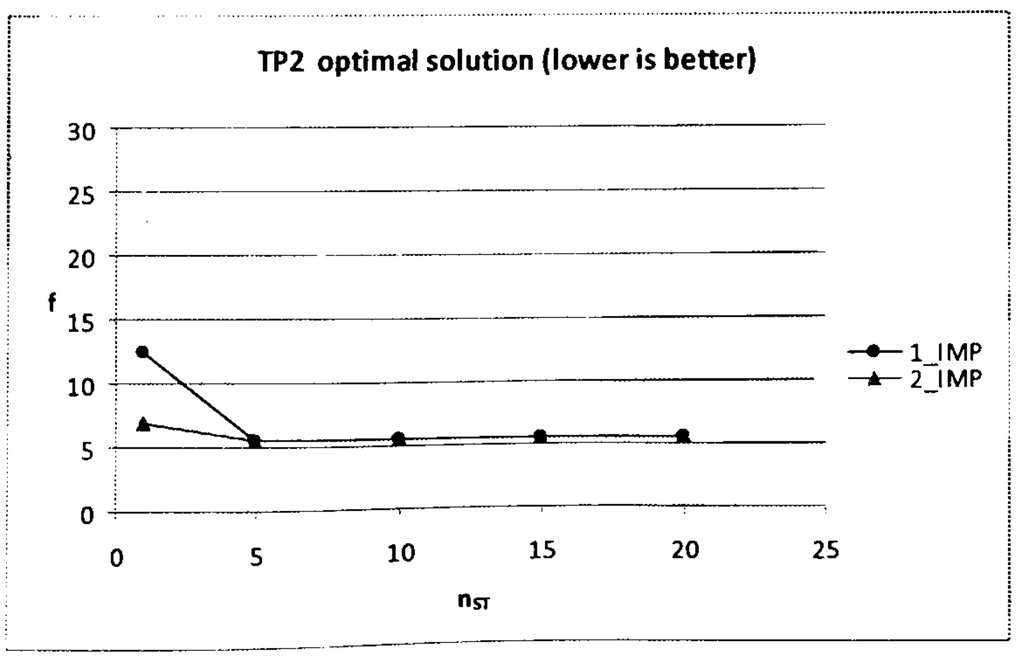
As the figures illustrate, σ_t decreases as COPT develops so that the population has the decreasing pool level. A termination criterion related to the pool level is proposed. In this criterion, COPT terminates if the standard deviation stays the same:

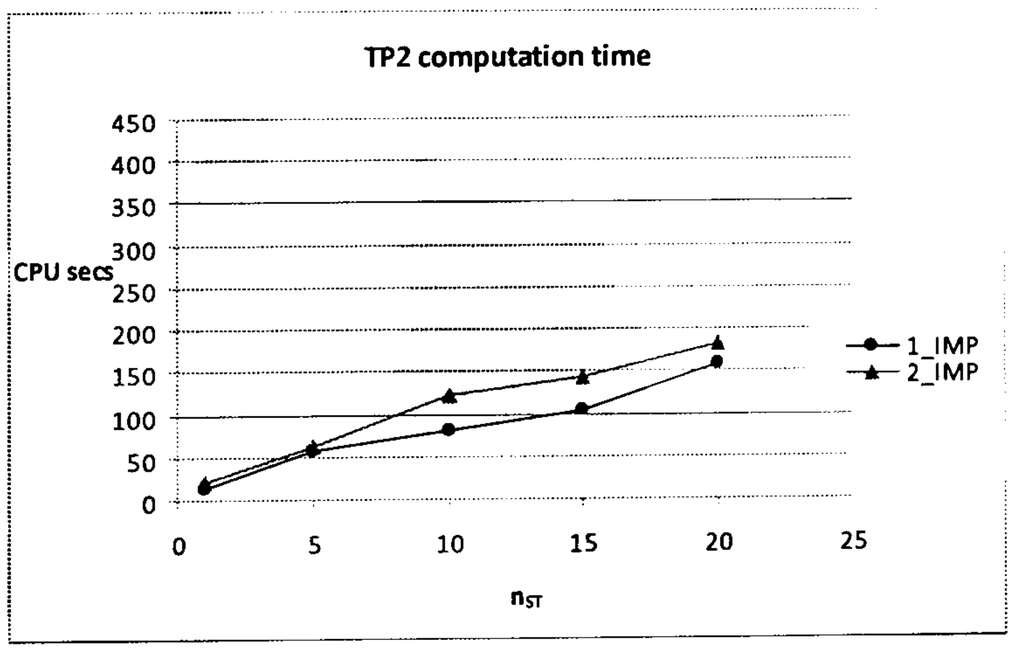
$$\left\|\sigma_t - \sigma_{t+L}\right\| \leq \varepsilon$$

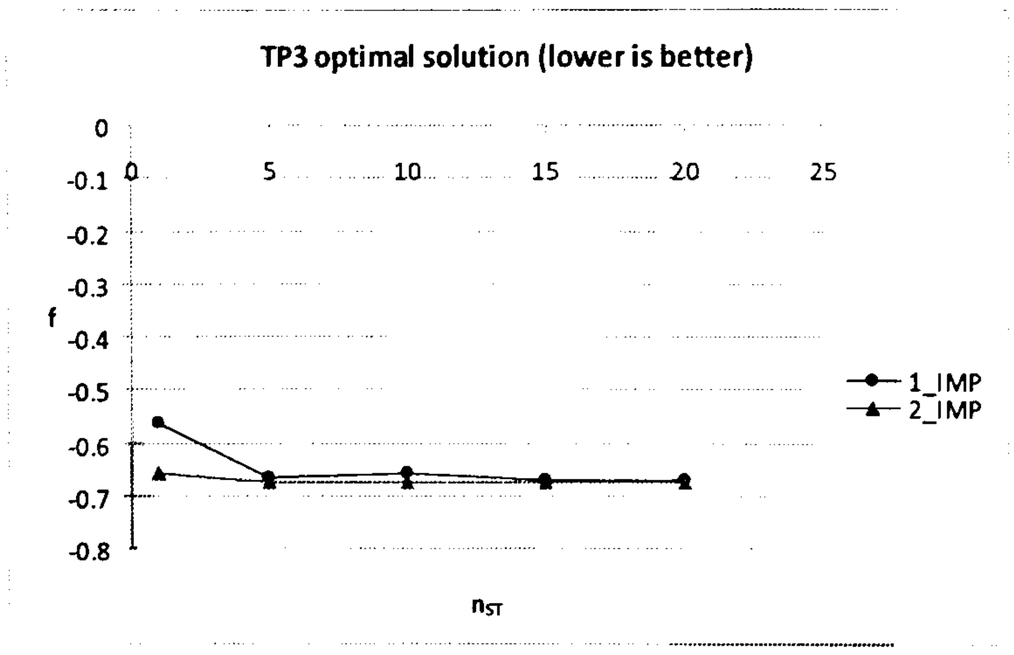
for n_{ST} iterations. To study the effect of n_{ST}, $\varepsilon = 1$ is selected to determine the accuracy of the standard deviation. Parameters (T_1, T_{n_p}, r, L, n_p, and $Q^{\max}$) are kept the same as those in studies of the previous termination criteria. Figure 6.12 presents the performances with different n_{ST} .

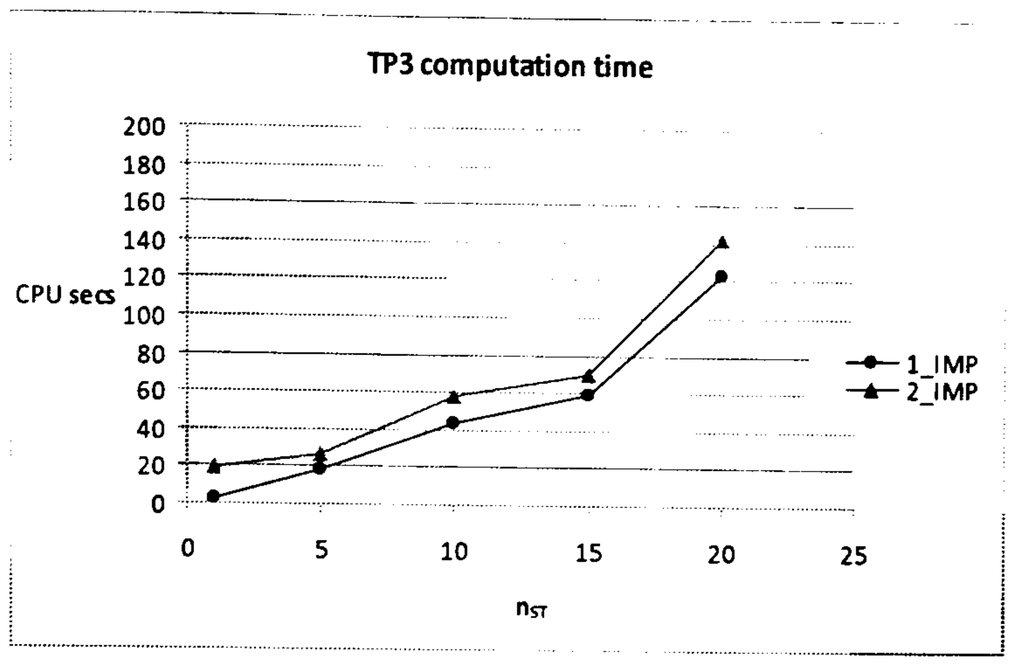


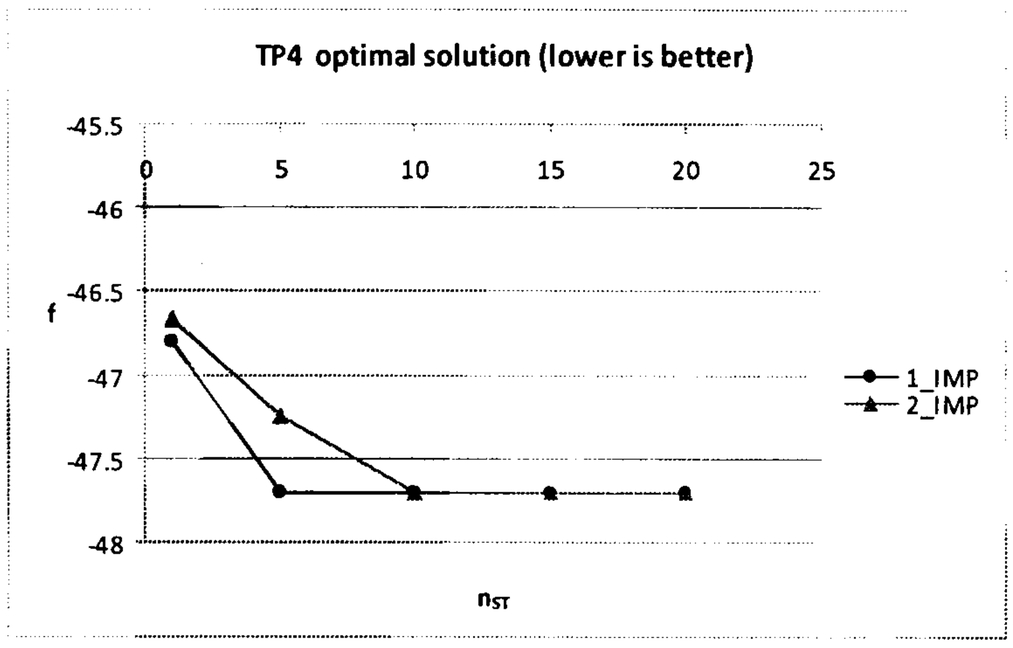












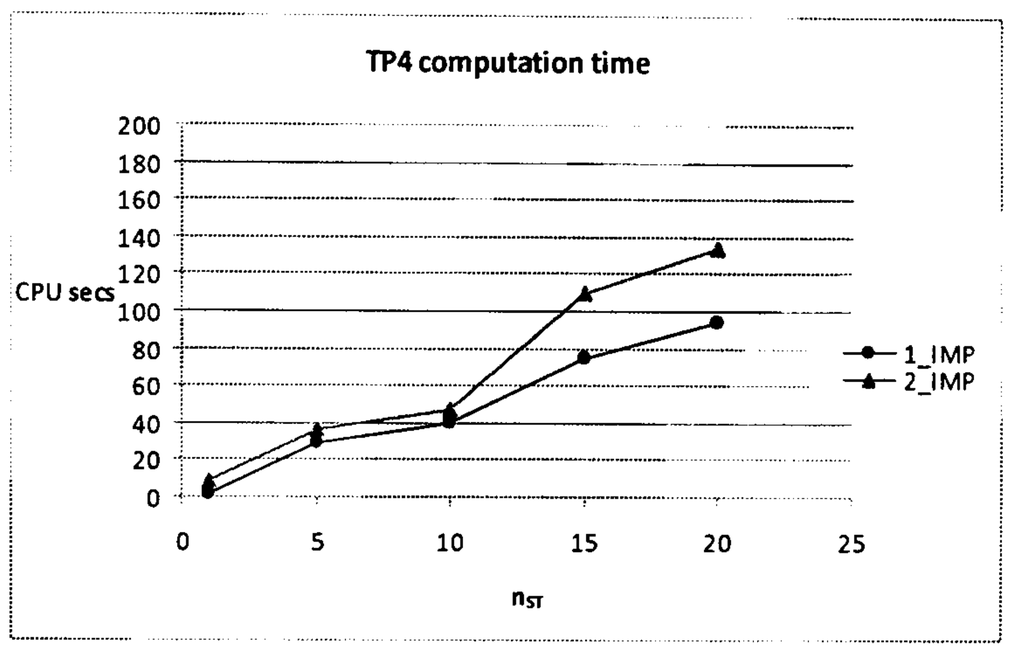


Figure 6. 12 Performance with different n_{ST}

The solution quality increases as n_{ST} increases. Meanwhile, the computation time also increases. Based on the results, $n_{ST} = 10$ gives the best tradeoff between the solution quality and computation time.

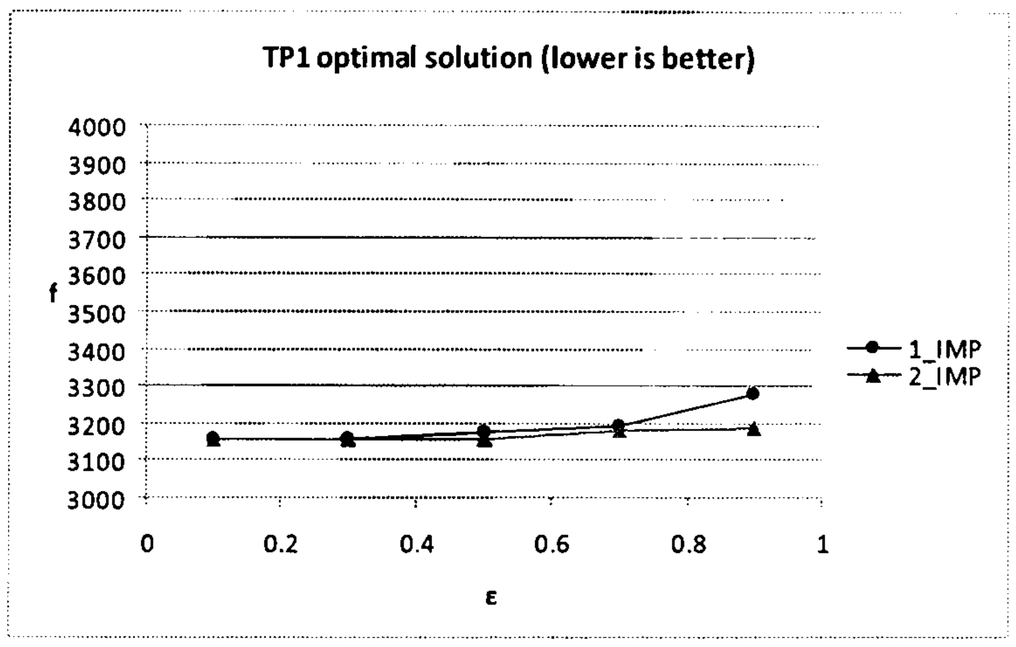
(iv) Population feature control – cascade level

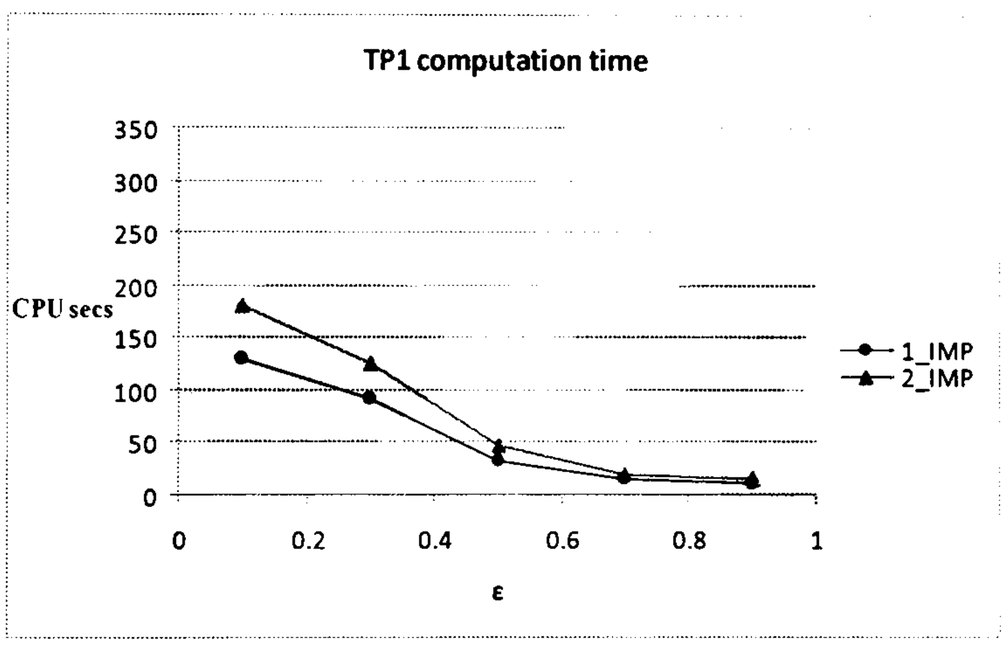
Another feature is the cascade level that relates to the density functions of pools. Based on the discussion in Chapter 4, the cascade is considered to be in equilibrium if the density function stays same:

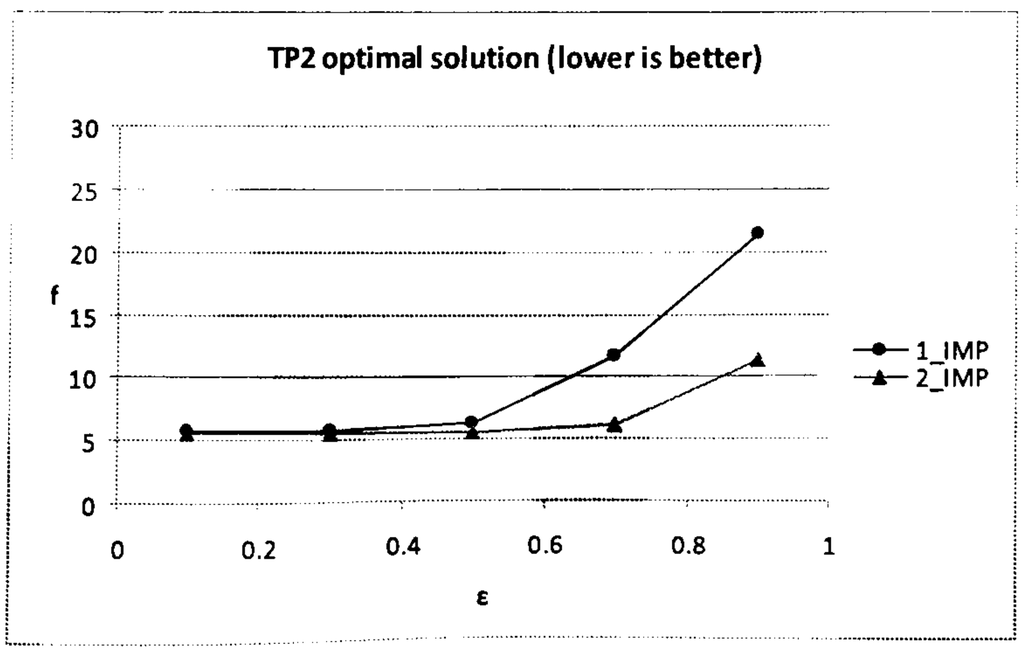
$$\left\| d_t - d_{t+L} \right\| \leq \varepsilon$$

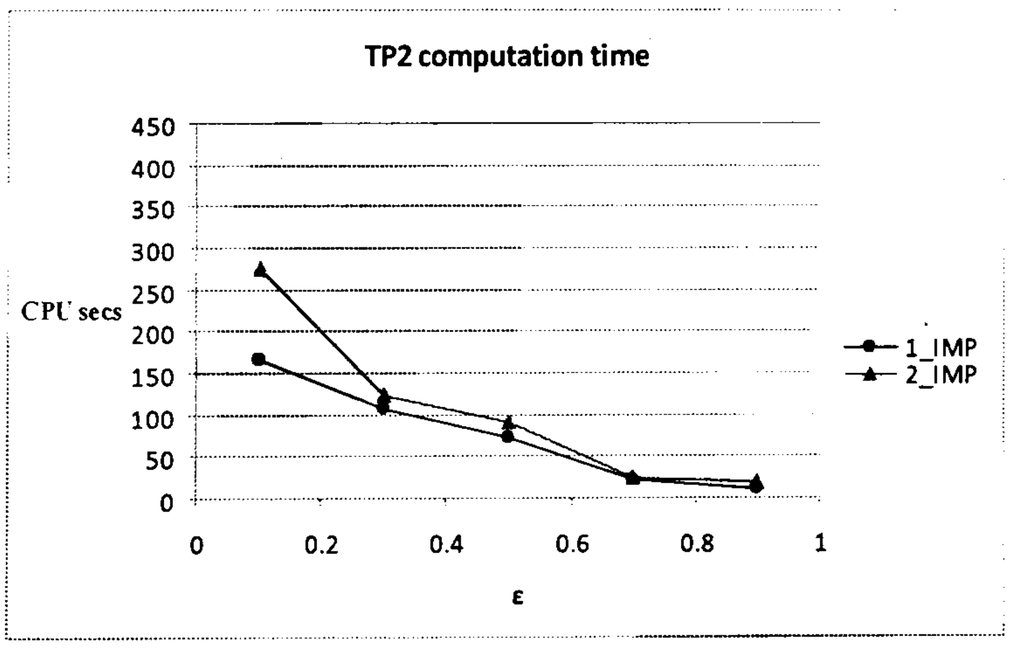
for n_d iterations. The cascade level of population in such case stays the same and COPT converges.

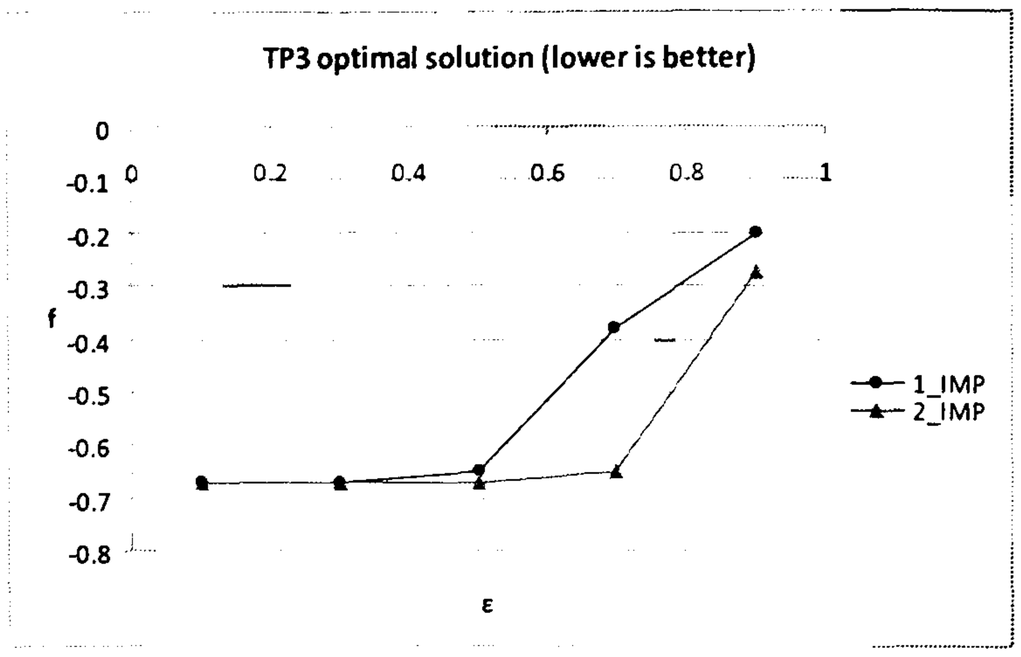
To study the effect of ε , $n_d = 3$ is selected. Parameters (T_1, T_{n_p}, r, L, n_p, and $Q^{\max}$) are kept the same as the those in studies of the previous termination criteria. With large value of ε close to one, the termination criterion cannot respond to even a large $\|d_t - d_{t+L}\|$ so that COPT quickly converges but to a solution with a low quality. With a small ε close to 0, the termination criterion is sensitive to a small $\|d_t - d_{t+L}\|$ so that COPT can converge to a good solution but spends a long time. Figure 6.13 presents the performance with different ε .

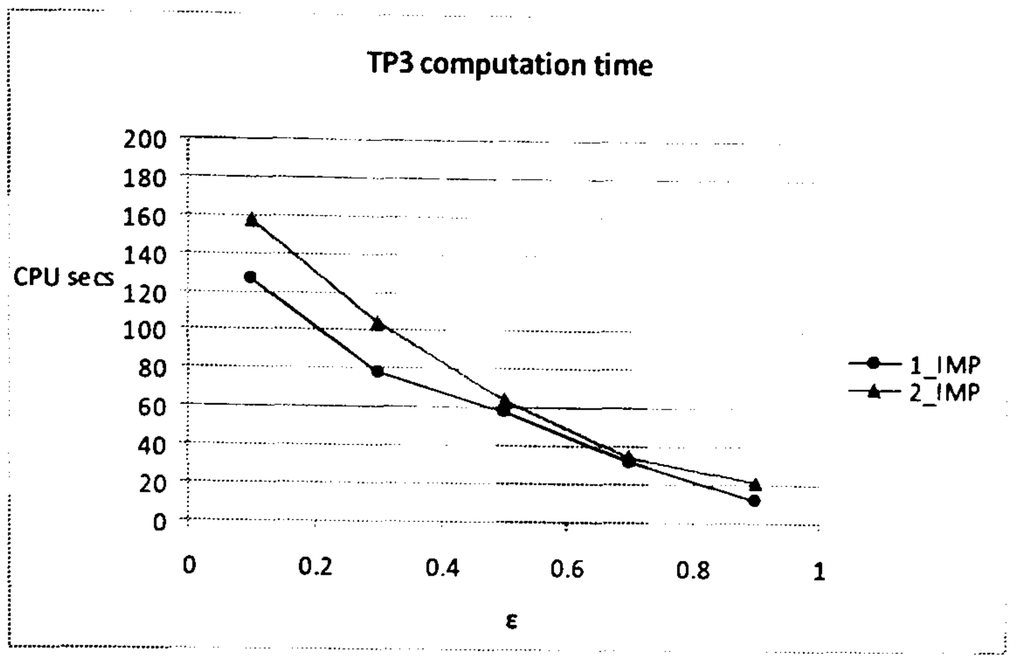


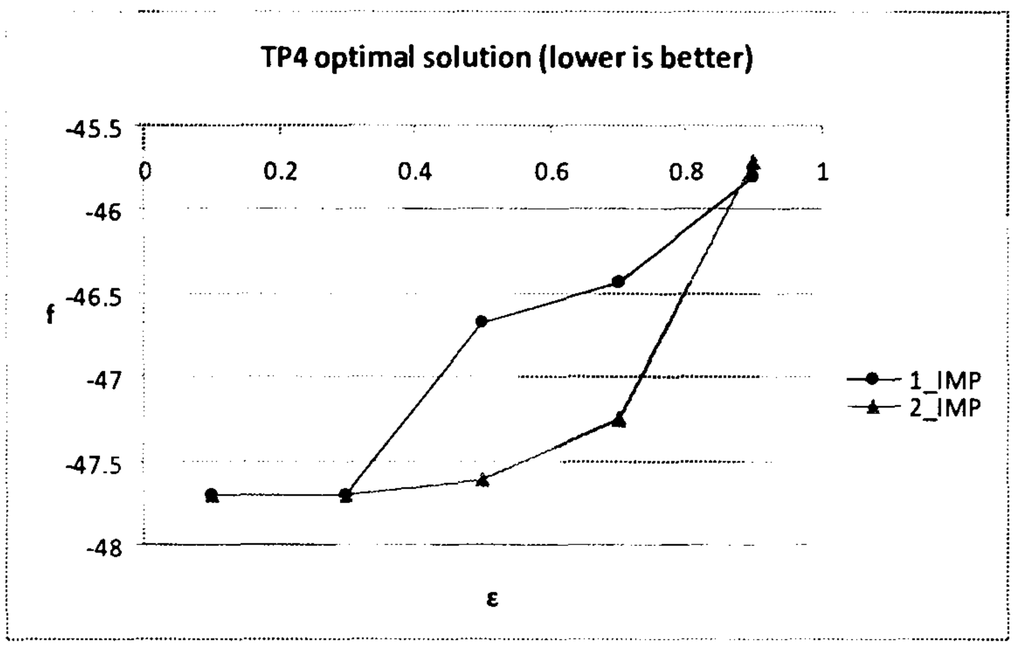












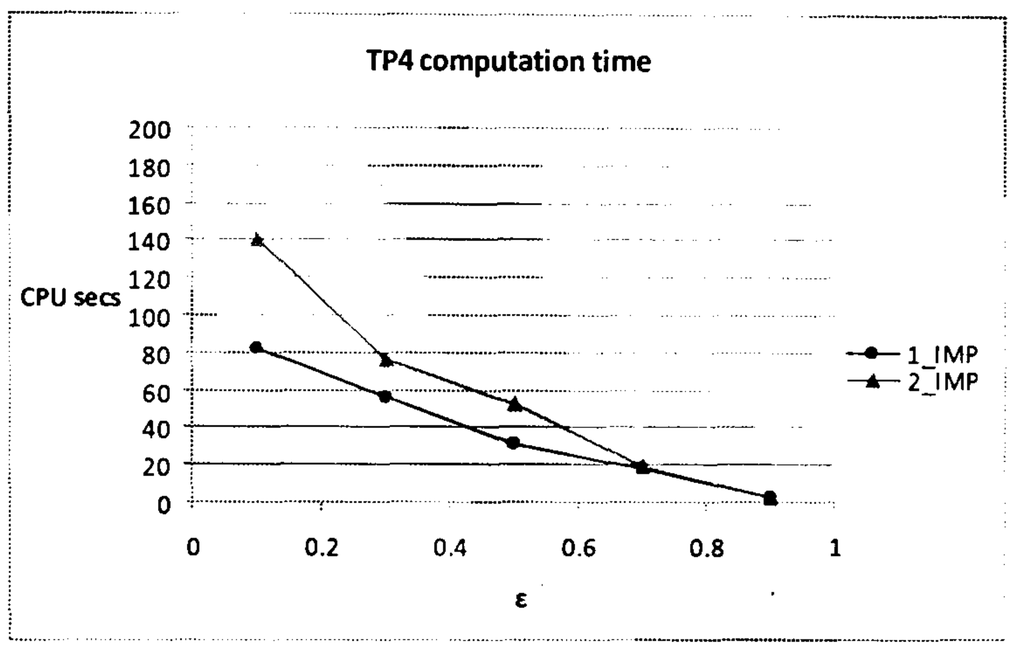
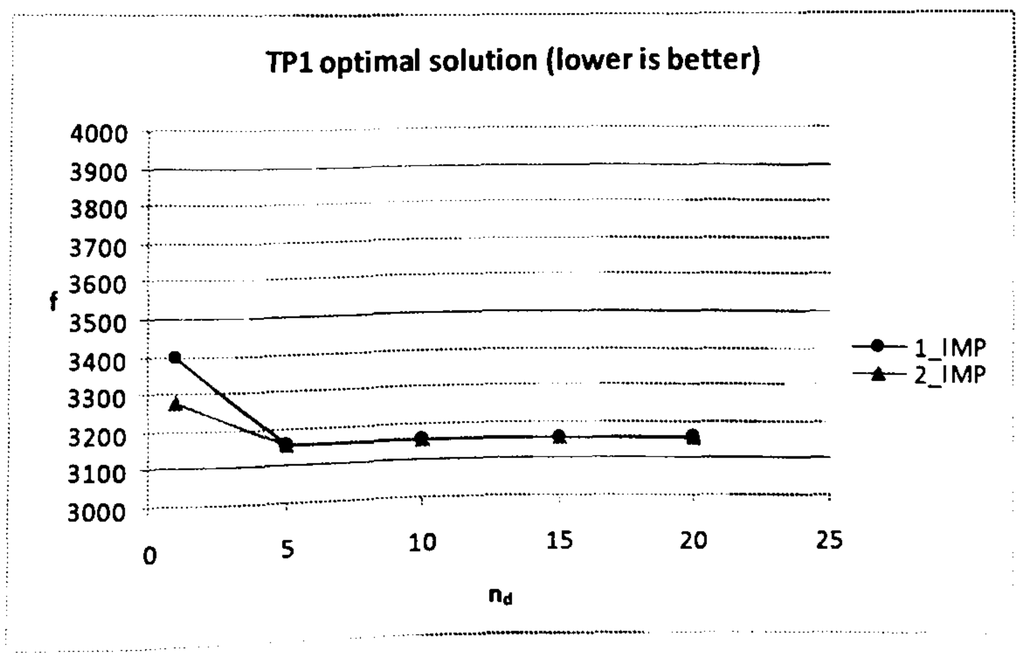
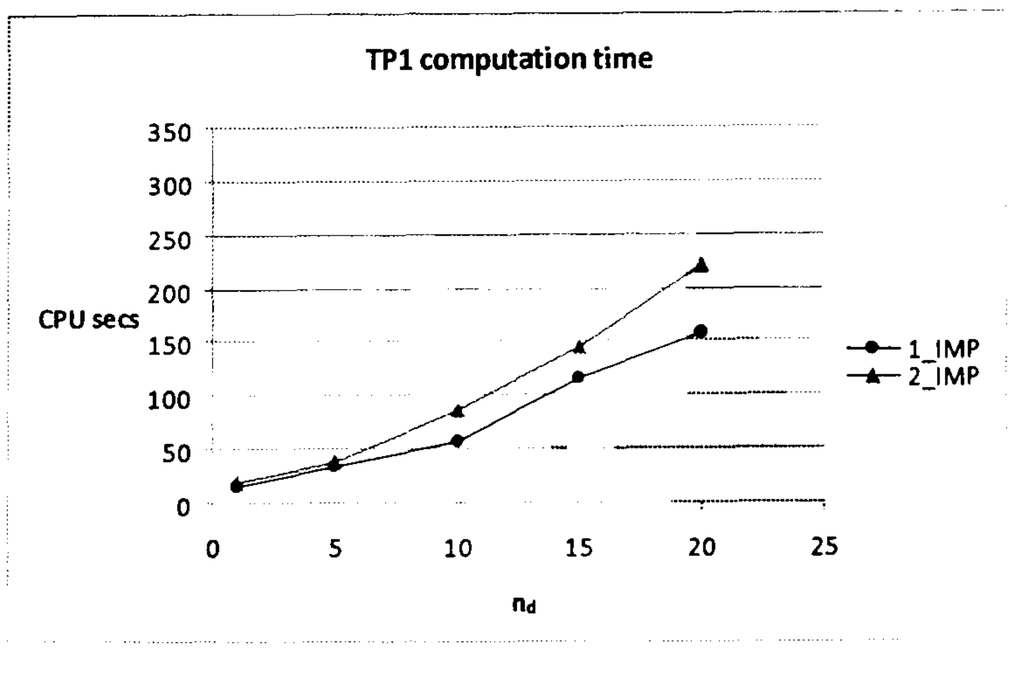
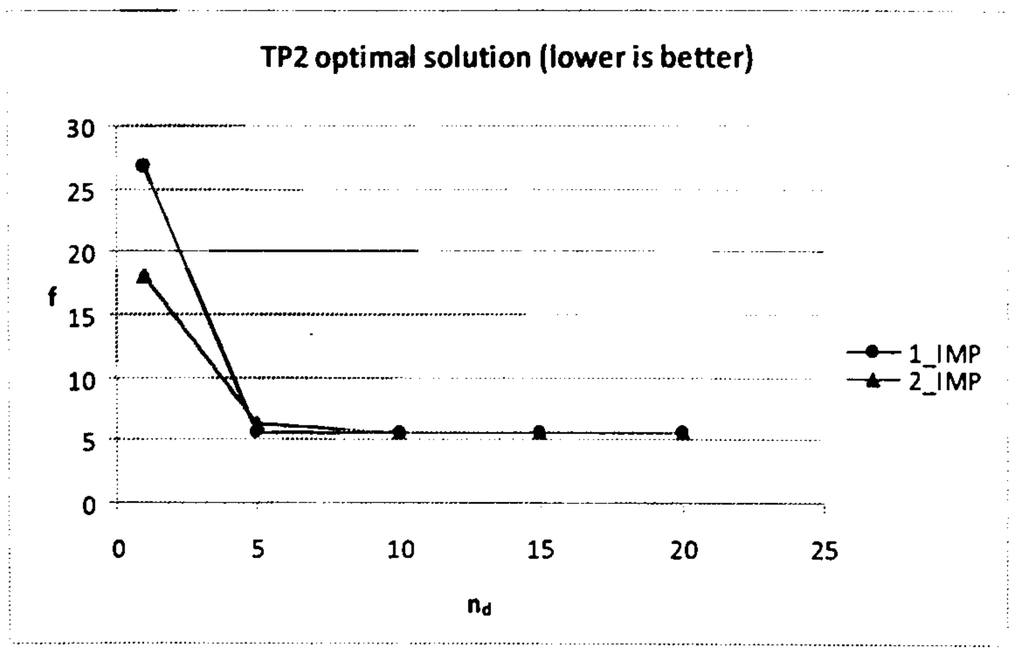


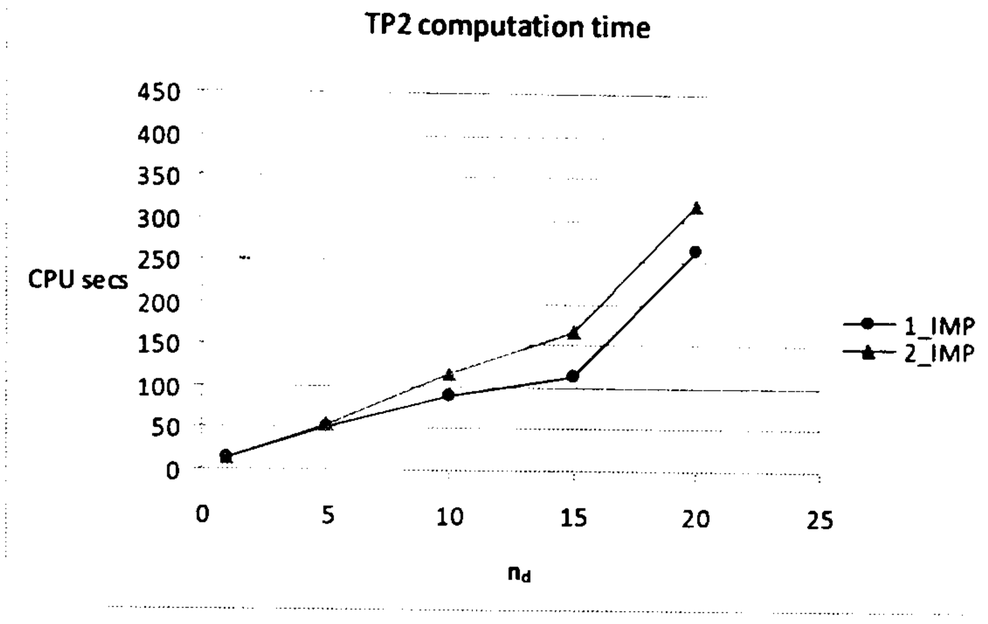
Figure 6. 13 Performance with different ε

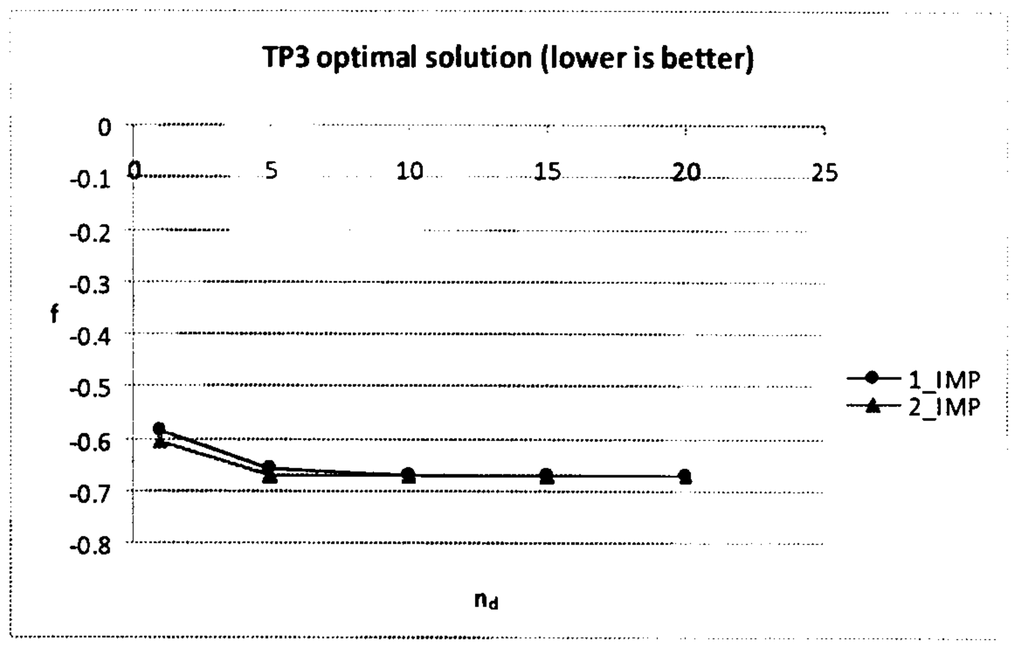
As the figures illustrate, increasing ε decreases the solution quality and the computation time. Based on the result, $\varepsilon = 0.3$ is selected to approach the best tradeoff between the computation time and solution quality. n_d is another parameter in this termination criterion. The performance when using the termination criterion with five n_d from 1 to 20 is extracted and illustrated in Figure 6.14.

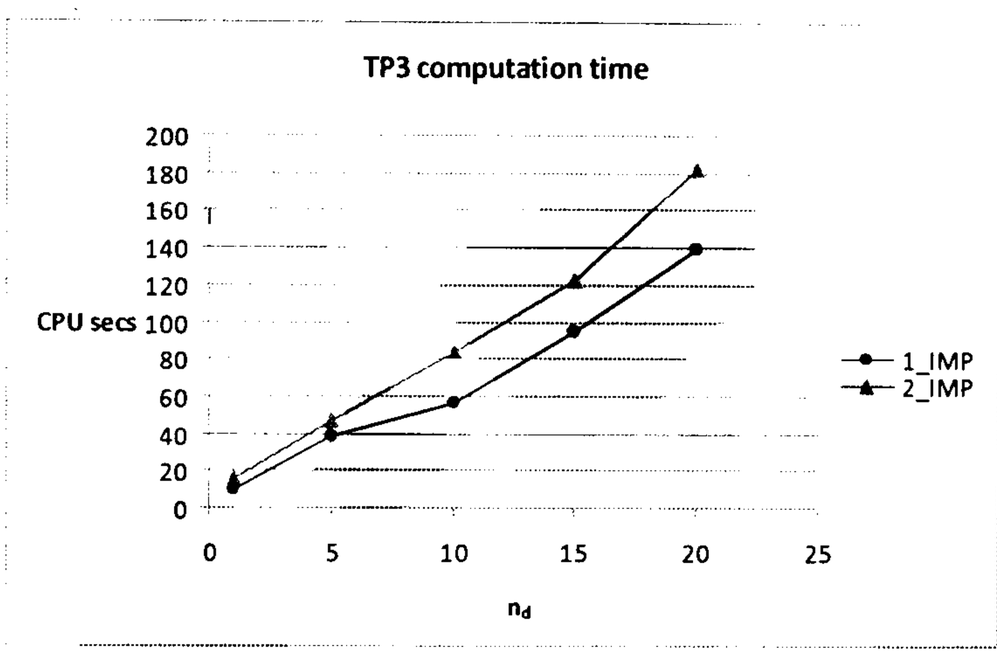


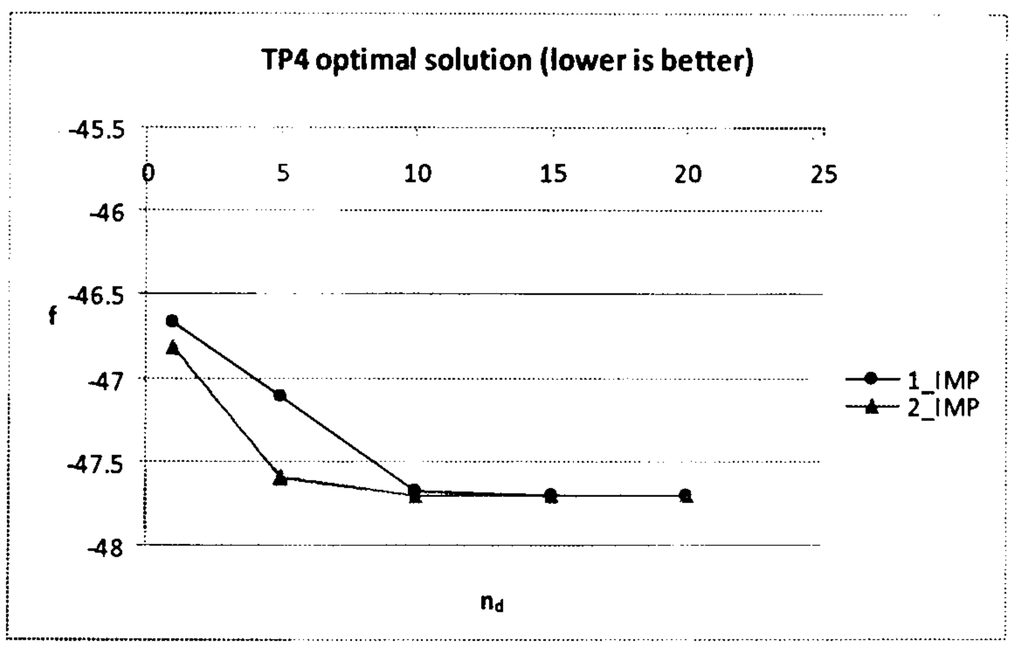












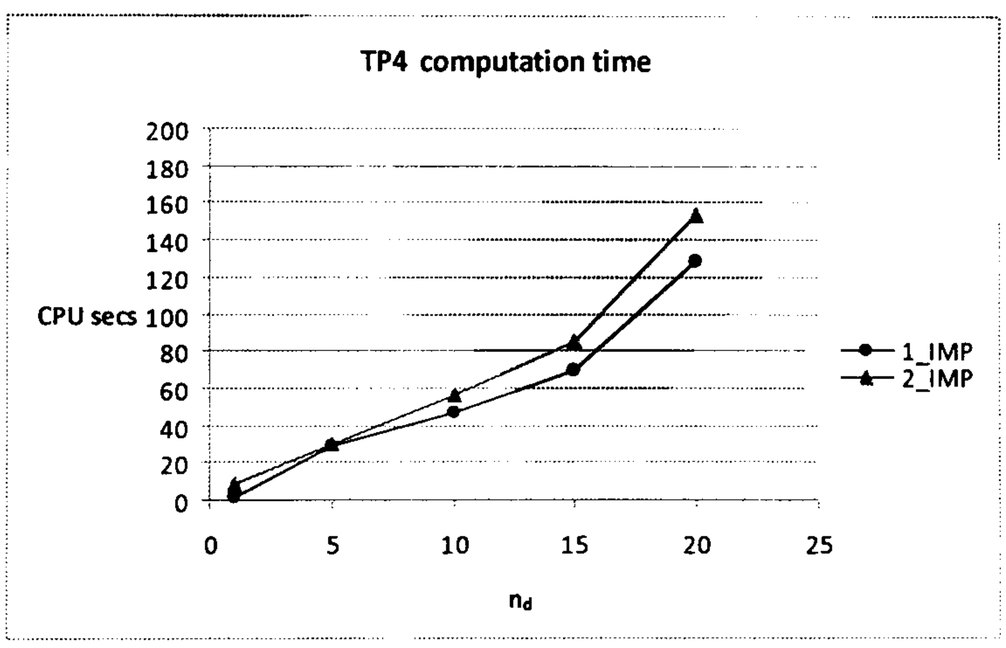


Figure 6. 14 Performance with different n_d

The solution quality increases as n_d increases. Meanwhile, the computation time increases. The termination criterion with $n_d = 10$ is adopted to give the best tradeoff between the solution quality and computation time. The research related to the population features is not complete. There might be some other preferred population features to choose. These features should be investigated to relate the characteristics of the problems.

6.8 Remarks and discussion

COPT has three important features that involve the management of stochastic search, size of optimisation structure, and the depth of search. These features have impact on the performance of COPT. The performance is measured by the solution quality and the computation time. Based on the results, COPT with a convex distribution can converge to good optimal solutions but require a long computation time. Increasing structure size increases the solution quality and meanwhile increases the computation time. The search depth does not have significant impact on solution quality but does on the computation time. The termination criteria relate to the controls in the population size controls, optimisation progress, and the population features. To control the population size, a minimum population $Q^{\min}$ is proposed to prevent the algorithm from premature convergence. The algorithm terminates when the population approaches a maximum population $Q^{\max}$. Depending on the study of the small-scale problems, $Q^{\min}$ and $Q^{\max}$ were set to 1000 and 3000 respectively. To control the optimisation progress, the algorithm terminates if $F_t^{\min}$ stay the same for a number of iterations (n_F). Increasing n_F can increases the solution quality and the computation time. Based on the results, terminating the algorithm when $\left|F_t^{\min} - F_{t+L}^{\min}\right| \le 0.1$ for 10 iterations can give the best tradeoff between the computation time and solution quality. To control the population feature, we control both the pool level and the cascade level. The pool level and the cascade level are measured by the standard deviation of population of pools (σ_t) and the density function of the domain (σ_t) respectively. The algorithm terminates if $\|\sigma_t - \sigma_{t+L}\| \le 0.1$ for a number of iterations (n_{ST}) or $\|d_t - d_{t+L}\| \le \varepsilon$ for a number of iterations (n_d). Increasing n_{ST} and n_d increases the solution quality and the computation time. The algorithm with a small ε can converge to a good optimal solution but require a long computation time. Depending on the study of the

small-scale problems, setting both n_{ST} and n_d to 10 and ε to 0.3 can give the best tradeoff between computation time and solution quality.

All the above studies are based on the four small-scale problems. In next chapter, COPT is studied on large-scale problems. Since the Markov process follows different acceptance probabilities, the form of distributing acceptance probabilities in search (search policy) is another important feature. The impact of search policies is studied as well as the above features. The comparison between the performances of COPT and a conventional stochastic method is explored on the application of a complex engineering problem.

Chapter 7 Large-scale applications

7.1 Problems

In this chapter, the studies of COPT involve:

- Features and termination criteria
- Search policies
- Performance on complex problems

The features and the termination criteria described in Chapter 6 are now studied in engineering problems. The Markov process follows different acceptance probabilities determined by pool temperature. The form of distributing acceptance probabilities to the Markov process might lead to different performance. For example, COPT that uses the Markov process with a high acceptance probability might perform differently with the one that uses Markov process with a low acceptance probability. Search policies are proposed to decide this form. The impacts of search policies on the performance of COPT are studied in this chapter. COPT is also studied on complex problems, in comparison with the stochastic methods (TS).

7.2 Benchmark problems

(a) Reaction engineering optimisation

Engineering problems applied in the studies include:

Problem 1

Problem 1 is lactose production that is the hydrolysis of lactose by β-galactosidase. It has

six components that include:

- β-galactosidase (that is the enzyme)
- lactose and glucose
- galactose with α and β forms
- gluconic acid
- deactivated enzyme

This problem involves a main reaction where lactose is hydrolyzed into glucose and β-galactosidase as a by-product, a reaction that deactivates the enzyme, as well as several other side reactions which produces additional by-products. The kinetics include 4 linear equations and one non-linear fraction. The objective is to maximize the outlet concentration of Glucose. More details of this problem have been illustrated in Appendix B. The optimal structure of reactor network for this reaction problem is a series of PFRs, which is in agreement with published results (Marcoulaki and Kokossis, 1999).

Problem 2

Problem 2 is Van de Vusse reaction scheme that consists of a combination of parallel and serial reactions (Marcoulaki and Kokossis, 1999; Kokossis and Floudas, 1990). This reaction scheme includes 3 components that are represented by A, B, and C and 3 reactions. These reactions include a first-order main reaction that produces the desired product, a first-order reaction in series with the main reaction and consuming the desired product, and a second-order reaction in parallel with the main one. The kinetics includes 2 linear and 1 quadratic equations. The objective is to maximize the concentration of component B. More details of this problem have been illustrated in Appendix B. The optimal structure is composed by a CSTR (V=10.13 L) followed by a PFR (V=15.14 L). Chitra and Govind (1985) found the same optimal configuration with a CSTR (V=11.211 L) followed by a PFR (V=16.81 L). Kokossis and Floudas (1990) found an optimal configuration composed by a CSTR (V=11.382 L) followed by a PFR (V=16.811 L).

(b) Process synthesis applications

The purpose of process synthesis applications is to design flow sheets using superstructure schemes that include all different possibilities for mixing and processing. A comprehensive superstructure is presented in Figure 7.1 (Papadopoulos & Linke, 2004). The superstructure consists of all possible combinations of *n* reactor units. The reactor type is modelled as:

- Ideal stirred tank reactors (CSTR)
- Plug flow reactors (PFR)
- Distributed side stream reactor (DSSR)

Figure 7.2 illustrates the reactor representation/options where PFR is approximated as a cascade of equal volume sub-CSTRs (Kokossis & Floudas, 1990).

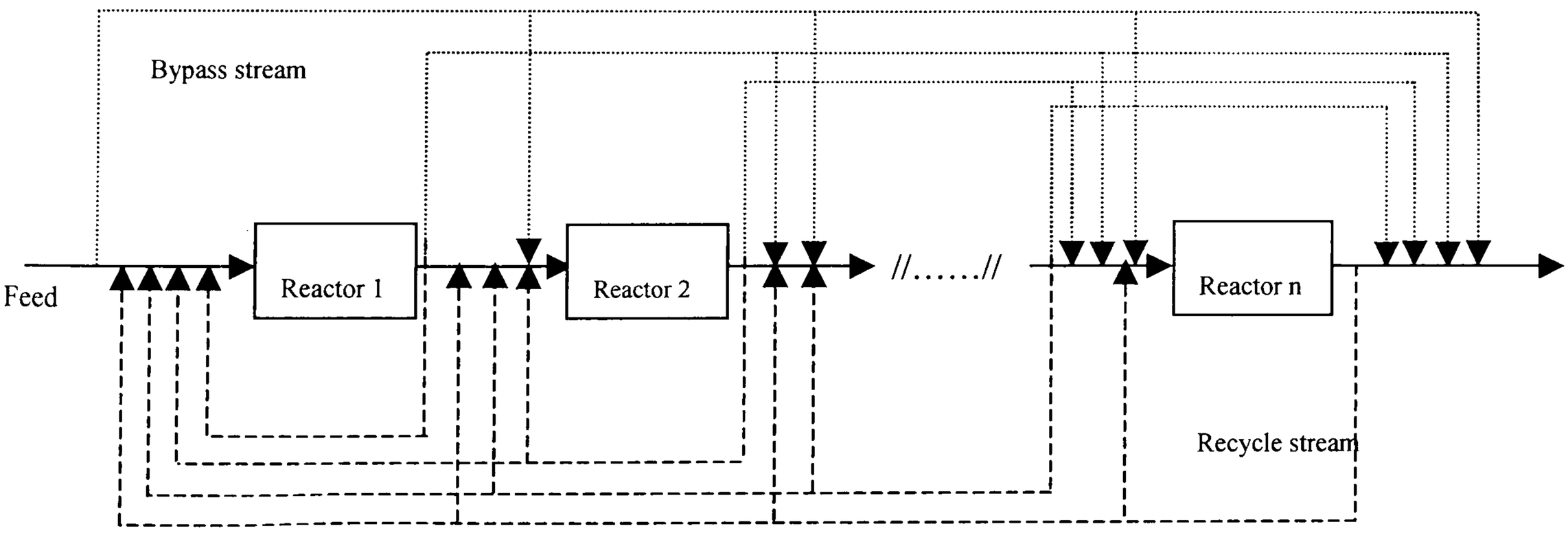


Figure 7. 1 Superstructure representation

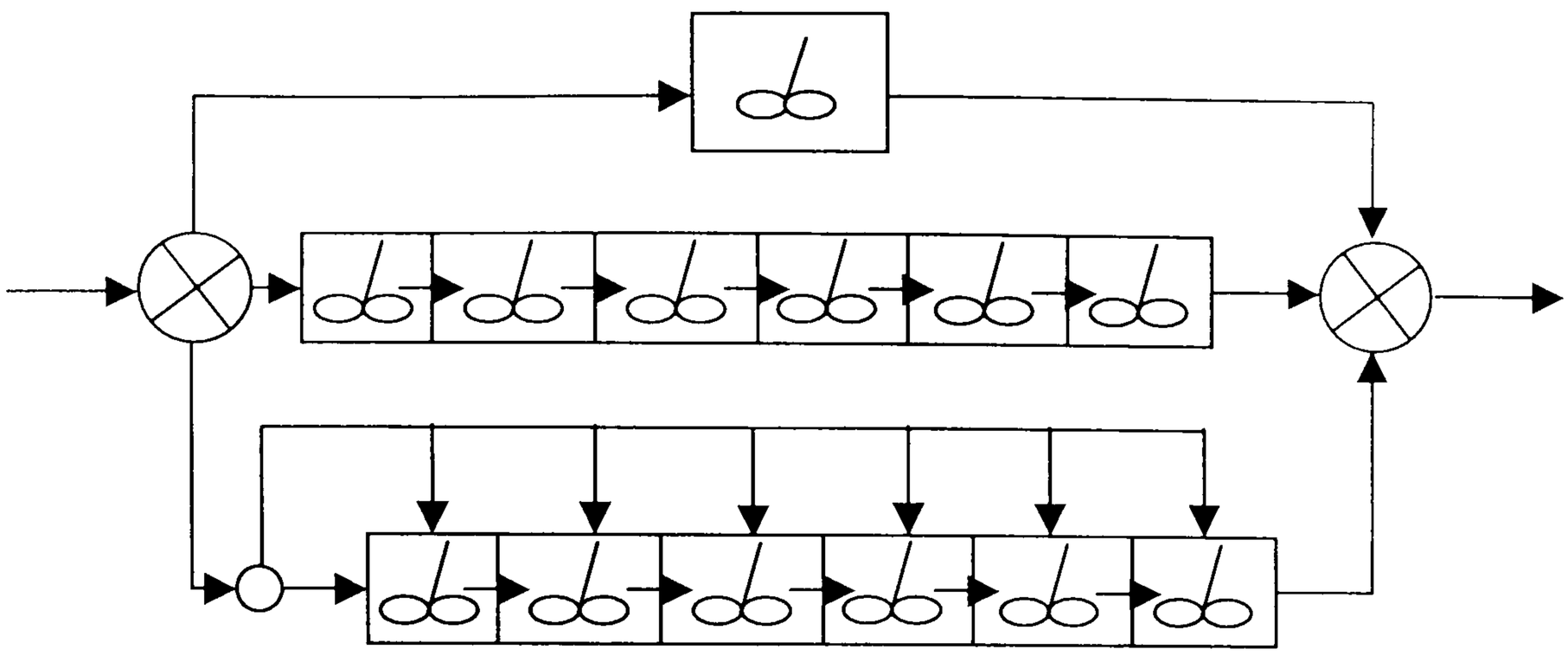


Figure 7. 2 Reactor representation/options

The use of the superstructure is presented in combination with reactions described in problem 3 to optimise reactor efficiency and select the reactor types appropriate to integrate. This superstructure problem has been solved by TS (Glover 1989, 1990, 1993) and SA (Kirkpatrick *et al.* 1983, Aarts and van Laarhoven 1985) and GA (Holland,1975).

Problem 3

Problem 3 is related to a Biocatalytic reaction process (Saccharomyces Cerevisiae production). In the problem glucose is converted to ethanol at the presence of the biocatalyst, namely Saccharomyces cerevisiae. The components of this problem involve:

- intracellular glucose
- intracellular pyruvate
- intracellular acetaldehyde
- acetate
- intracellular ethanol
- enzyme (Acetaldehyde dehydrogenase)
- nicotinamide adenine dinucleotide
- adenosine triphosphate

According to Lei, etal. (2001), this process can be characterised with 12 reactions that can be divided into three groups. The first group comprises 3 reactions that lead to the production of ethanol. The second group includes 3 reactions which complete with those in the first group. The third group consists of 6 reactions representing the mechanism of the growth of Saccharomyces cerevisiae. The kinetic model involves 10 nonlinear fractions and 1 linear equation. The objective is to maximize the production of ethanol. More details of this problem have been illustrated in Appendix B. Ashley (2004) reported that the optimal configurations are the combinations of plug flow and mixing. In her studies, both Tabu search and a rule-based search are used to explore the optimal configurations. The best Tabu structure features a feed bypass, while the best rule-based structure features a recycle.

7.3 Optimisation Studies

The experiments aim to explore the ability of the algorithm to cope with denoting problems. The studies explored:

- The efficiency of COPT to provide quick convergence to a good solution
- The impact of COPT features on the performance
- The implication of selecting different convergence criteria

(i) Performance and settings

The assessment is based on the solution quality and the computation time required to converge. Using the experience accumulated in Chapter 6, experiments made advantages of the second implementation as described in section 5.4.1. A single worker and a single master are involved in the computing environment with the configurations illustrated in Table 6.2. The CPU time for Tsk^{MC} (T_1^{CPU}) illustrated in section 6.3 represents the

computation time and the best objective (F^{max}) in pools reflects the solution quality.

(ii) COPT features

The features of COPT include:

- The management of stochastic search
- The size of the optimisation structure
- The depth and intents of search

Following the studies in Chapter 6, COPT terminates when the population of pools (Q_t) approaches a maximum Q^{max} .

(iii) Termination criteria

The termination criteria address controls in the population of intermediate solutions, the optimisation progress, and the distribution of intermediate solutions in pools. In the population control, COPT does not terminate if Q is smaller than a minimum Q^{min} and terminates if Q is larger than a maximum Q^{max}. In the optimisation progress control, COPT terminates if $\left|F_t^{max} - F_{t+L}^{max}\right| \le \varepsilon$ for n_F iterations. In the distribution control, COPT terminates if $\left\|\sigma_t - \sigma_{t+L}\right\| \le \varepsilon$ for n_{ST} iterations or if $\left\|d_t - d_{t+L}\right\| \le \varepsilon$ for n_d iterations.

(iv) Search policies

To study the impact of search policies with a, now, much large number of pools, pools are clustered into floating, middle, and settling ranges. The floating range is related to high probabilities to accept new solutions, while the settling range is related to the low probabilities to accept new solutions. These ranges are associated with different parameters. Figure 7.1 illustrates the three ranges with 6 pools.

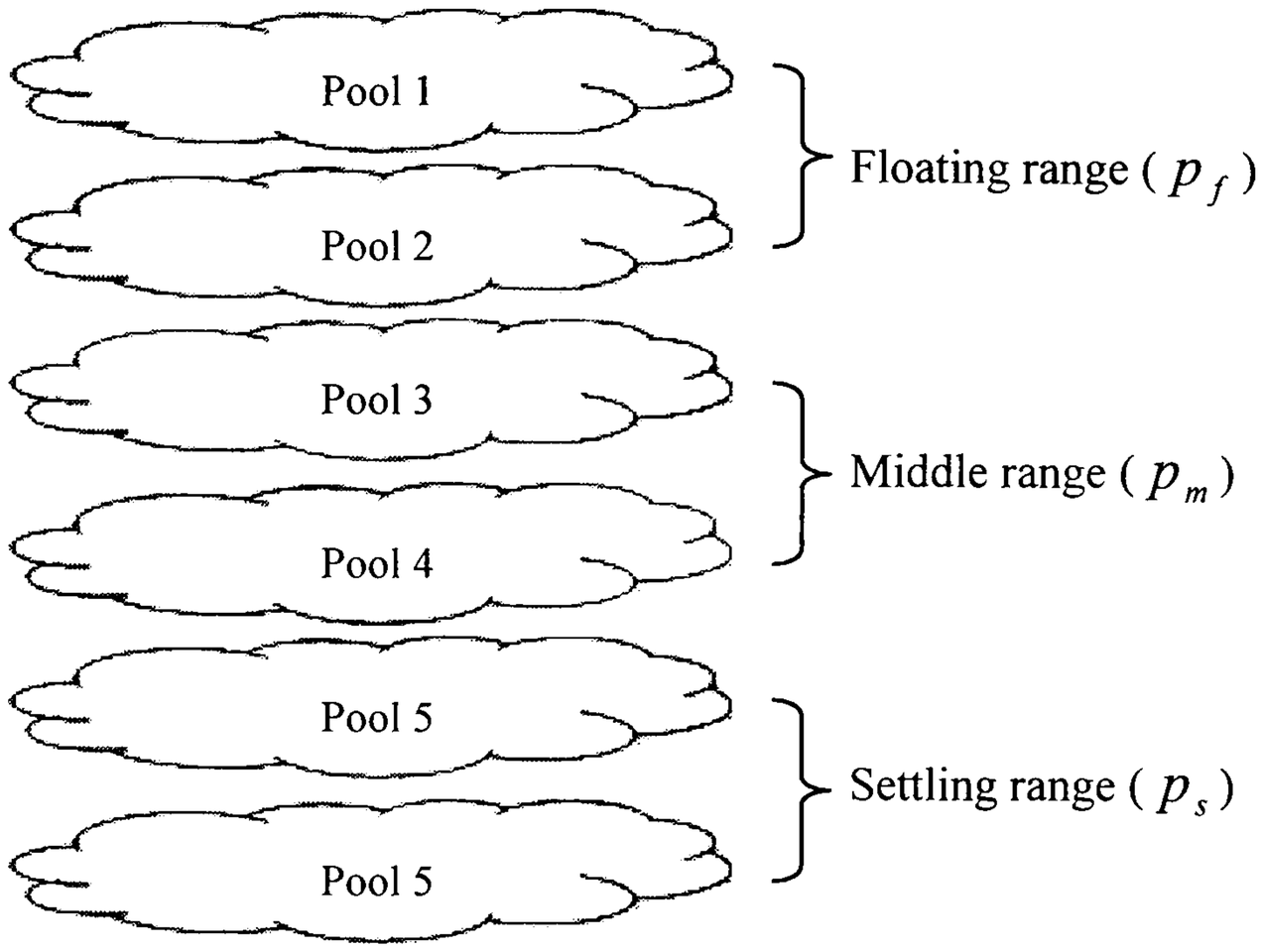


Figure 7. 3 Three ranges with 6 pools

The three parameters accounting for the probabilities to select pools from different ranges are denoted by p_f, p_m, and p_s. Choices of the parameters for seven different cases are presented in Table 7.1.

i^{cb}: index of different combinations

Table 7. 1 Choice of p_f, p_m, and p_s

i^{cb}	p_f	p_m	p_s
I	0.75	0.125	0.125
II	0.5	0.25	0.25
III	0.33	0.33	0.33
IV	0.25	0.5	0.25
V	0.125	0.75	0.125
VI	0.25	0.25	0.5
VII	0.125	0.125	0.75

7.4 Results

(a) Management of stochastic search

The management of stochastic search is related to the distribution of population of pools. Following the experiences in Chapter 6, the distribution of population can adopts different distribution functions that are associated with cooling schedules. Based on Eq.4.22, the cooling schedules are controlled by parameter *r*. To study the effect of distribution function, nine different *r* values are selected for the cooling schedules ($T_1 = 100$ and $T_{n_p} = 0.01$). The Markov process has the fixed length (*L*=10). The number of pools $n_p = 100$. $Q^{\max} = 1500$ is selected. Ten statistical runs with different initial reactor network structures are tested for each distribution function. Results represent the averages of the tests. Figure 7.4 presents the COPT performances.

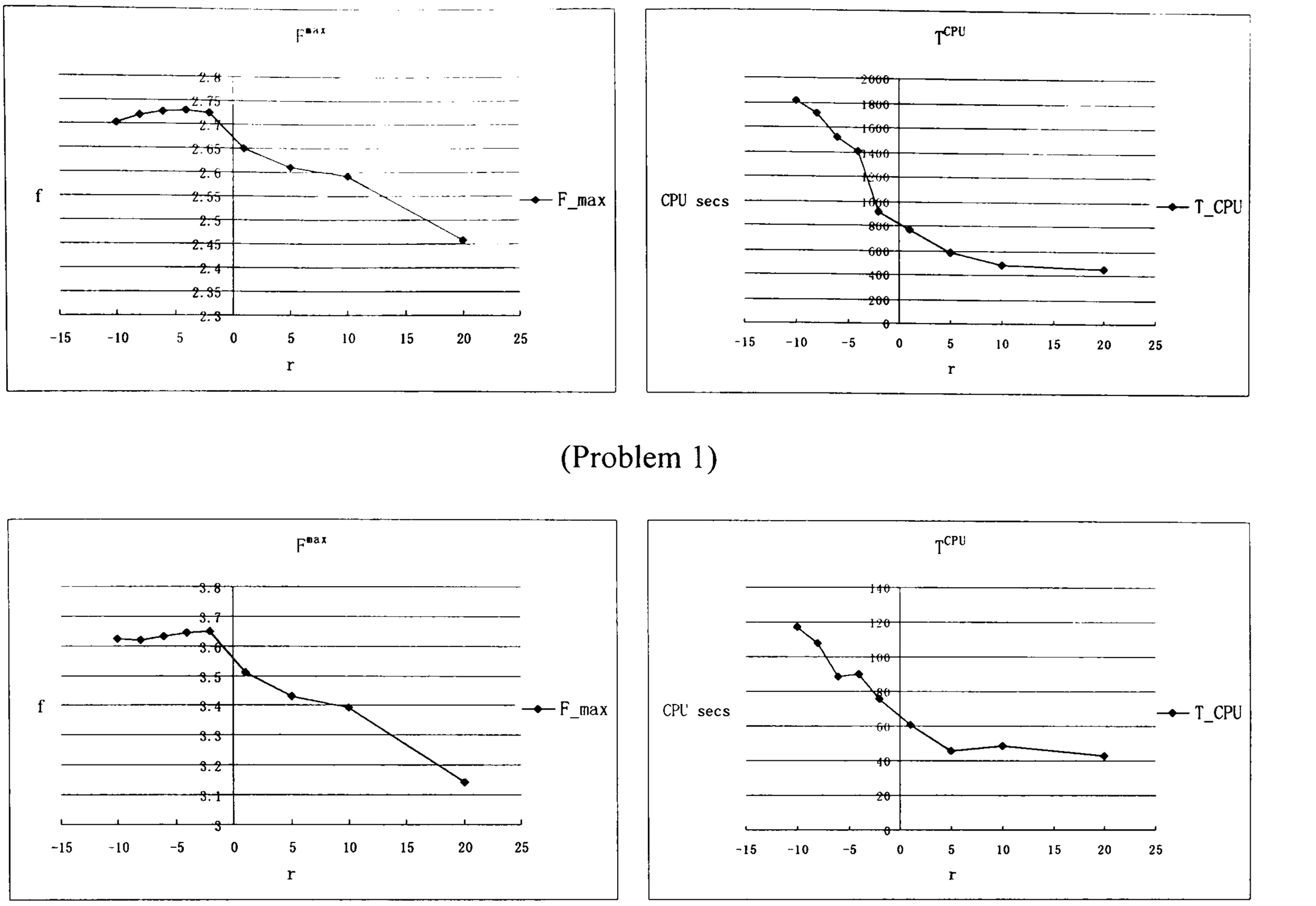


Figure 7. 4 Performance with different distribution functions

F_max : the solution quality F^{max}

T_CPU: the computation time T_1^{CPU}

Based on the results, increasing r decreases the solution quality and also the computation time. COPT with a concave distribution can converge to a good solution but requires a long computation time. Adopting the concave distribution (r=-2) can give the best tradeoff between the solution quality and the computation time. The effect of r and the distribution function on engineering problems is consistent with that on earlier small-scale optimisation problems.

(b) Size of Optimisation Structure

Following the experiences in Chapter 6, the size of optimisation structure is reflected on the number of pools. To study the effect of the number of pools, the parameters (T_1, T_{n_p}, L, and Q^{max}) are kept the same as in the study of distribution functions. The concave distribution function (r = -2) is selected to give good performance. Different numbers of pools ranging from 5 to 500 are selected. Results represent the average of 10 stochastic tests. Figure 7.5 illustrates the performance with different structure size.

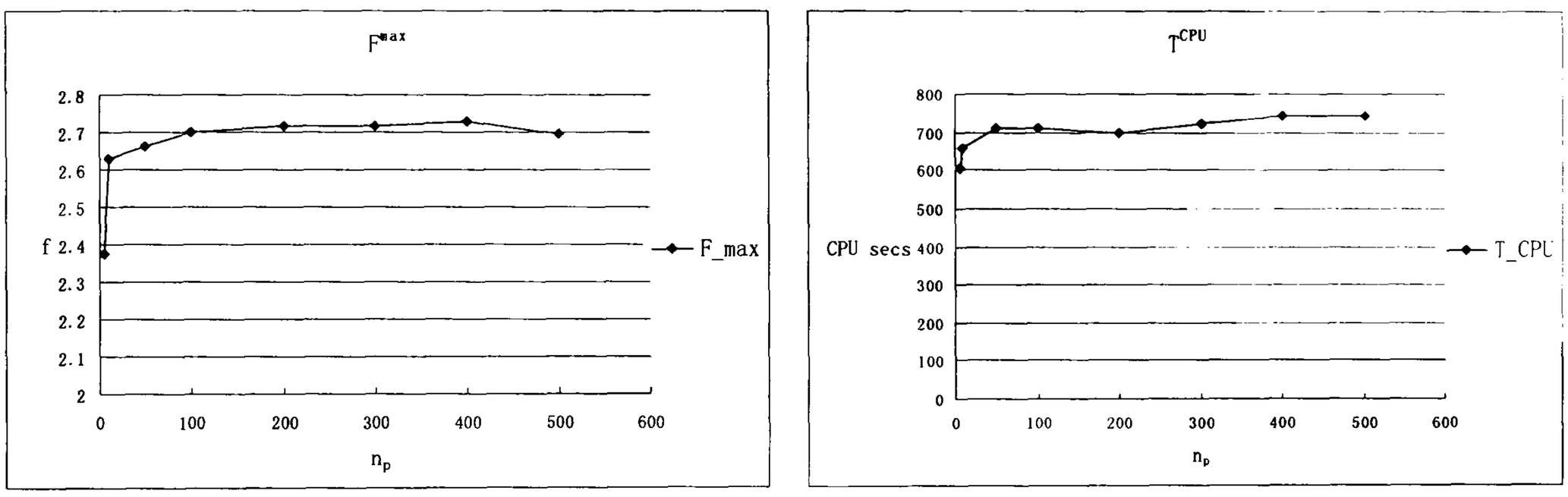


Figure 7. 5 Performance with different structure sizes

Based on the results, increasing the structure size increases the solution qualities. Consistent with the analysis in 6.5, the incremental benefit diminishes as n_p exceeds 100. The standard deviations of $F^{\max}$ ($\sigma_{F^{\max}}$) ($n_p < 100$ and $n_p \geq 100$) are presented in Table 7.1.

Table 7. 2 Standard deviations of $F^{\max}$

Test problems	$\sigma_{F^{\max}}$ $(n_p < 100)$	$\sigma_{F^{\max}}$ $(n_p \geq 100)$
1	0.158	0.0126
2	0.354	0.0400

Following the methods in section 6.5, the increasing ratios of computation time is calculated and presented in Table 7.3.

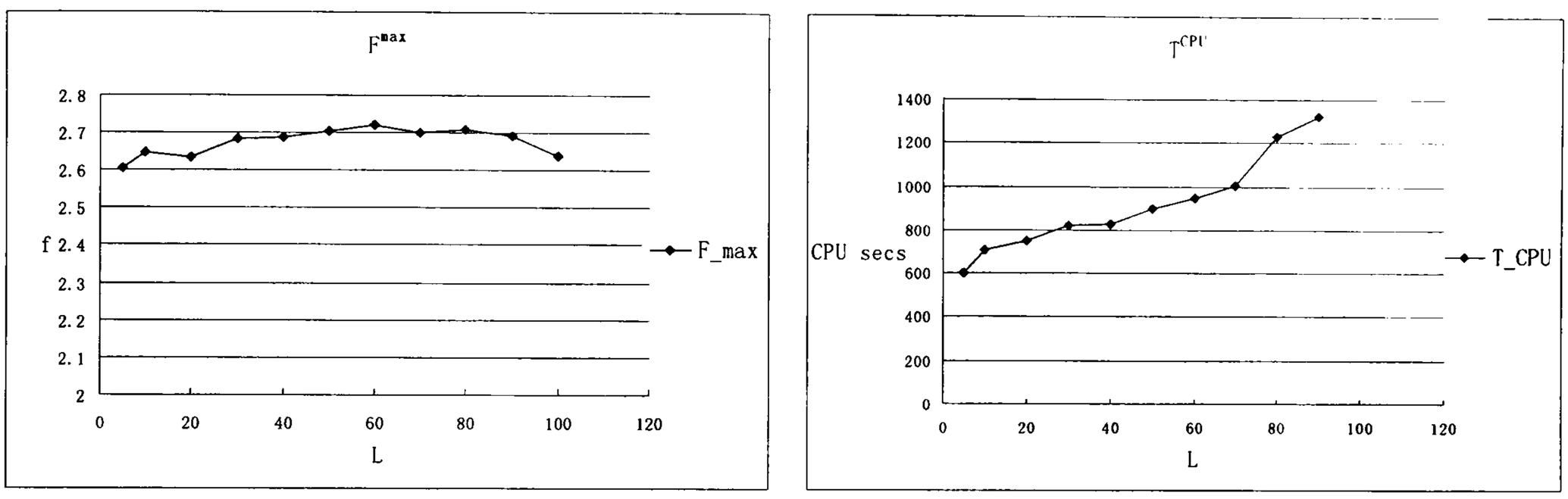
Table 7. 3 Increasing ratios

Test problems	r^{Ins}
1	0.186
2	0.214

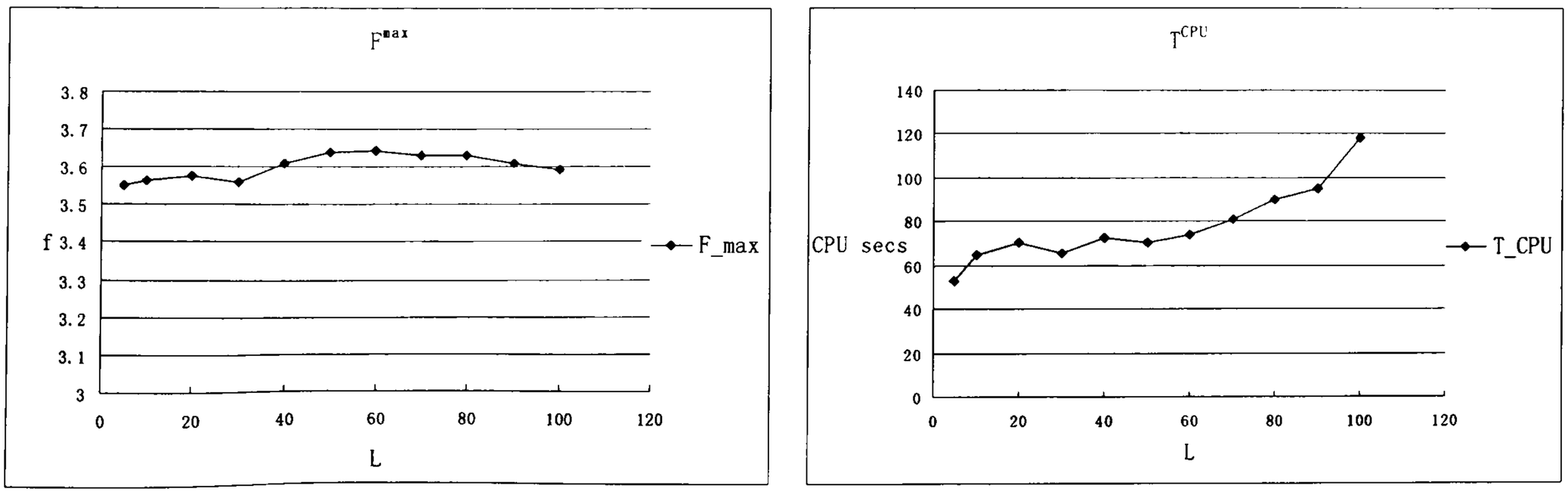
Based on the results, the increasing ratios are smaller than the ones produced on small-scale problems (the largest one is 21% that is smaller than that (41%) illustrated in table 6.4). The best tradeoff between the computation time and solution quality occurs with 100 pools. This value of 100 pools is then selected for the following studies. Since COPT can be applied on both parallel and distributed computing environments. the resulting increment of computation time can be reduced.

(c) Depth of search

Following the experiences in Chapter 6, the depth of search is reflected on the length of Markov process. To study the depth of search, the parameters (T_1, T_{n_p}, r, and $Q^{\max}$) are kept the same as in the study of optimisation structure size and $n_p = 100$. Compared to the studies on small-scale problems, a larger range of search depths are selected ranging from 5 to 100. Figure 7.6 presents the performance.



(Problem 1)



(Problem 2)

Figure 7. 6 Performance with different search depths

The results show that with increasing L the solution quality firstly increases and then decreases. However, both the incremental benefit and decremental loss are small. Table 7.4 presents the percentages of the convergences (p^{cvg}) for the problem 1.

Table 7. 4 Percentages of convergences

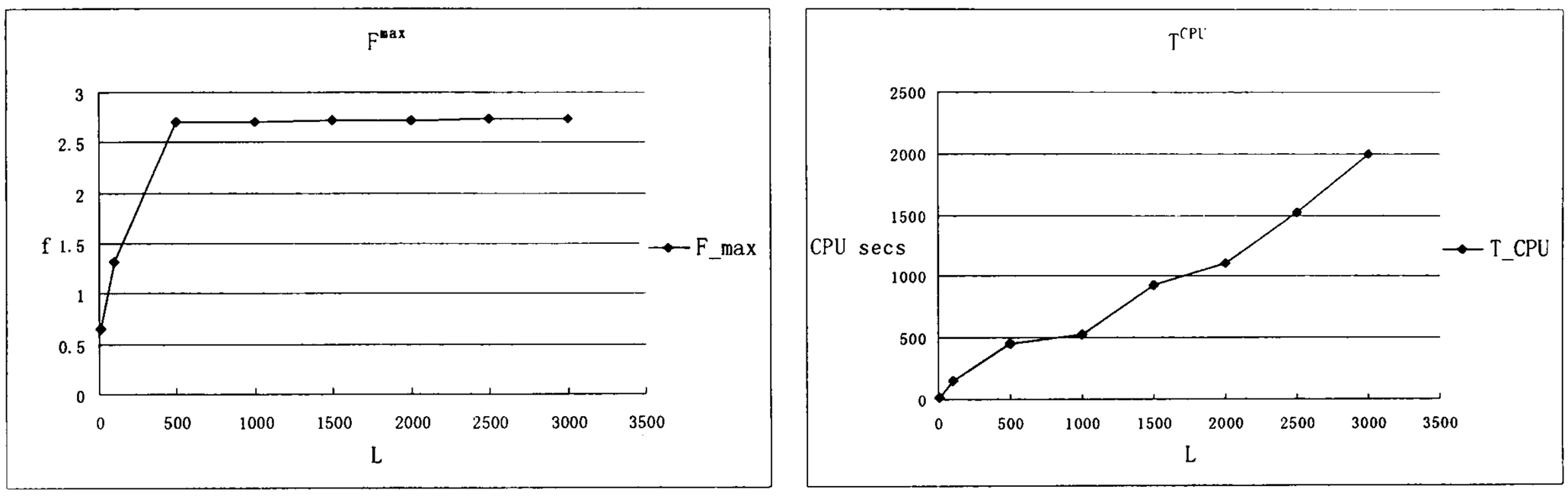
L	p^{cvg} (%)
5	95.7%
10	97.2%
20	96.8%
30	98.6%
40	98.8%
50	99.5%
60	100%
70	99.3%
80	99.6%
90	99.0%
100	96.9%

Based on the results, the depth of search has a small impact on the convergence of COPT (the minimum $p^{cvg} = 95.7\%$). In contrast, the computation time increases as L increases so COPT with a long Markov process requires a long computation time to converge. Adopting the Markov process (L=50) can give the best tradeoff between the computation time and solution quality and it is this value that is selected for the following studies.

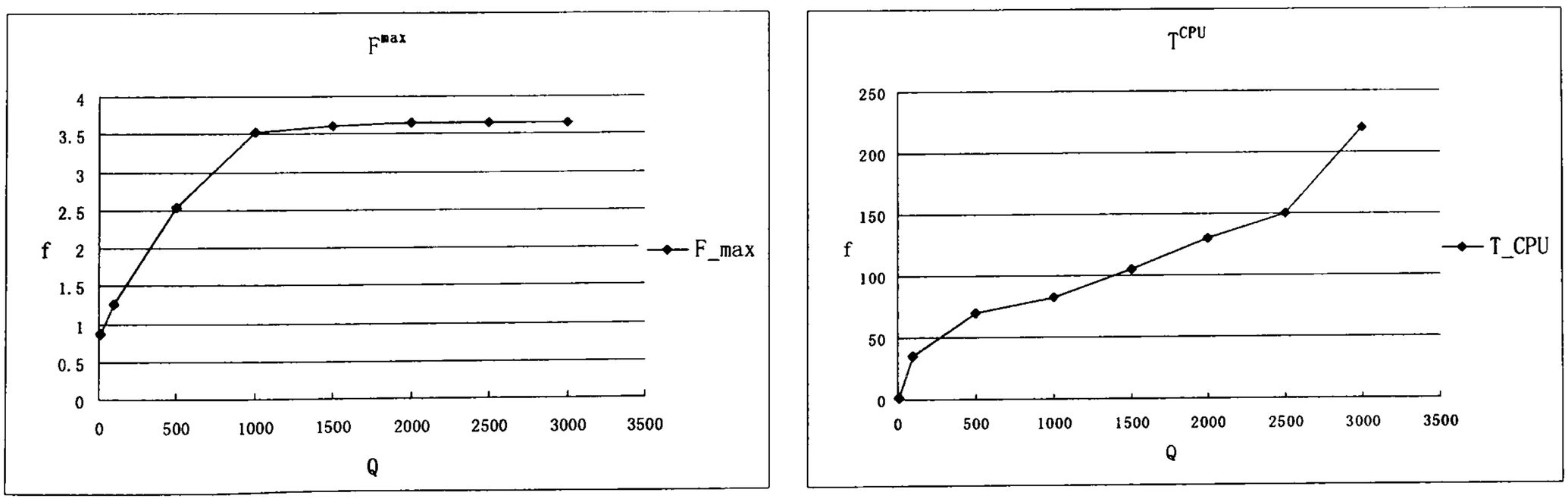
(d) Termination criterion

(i) Population size control

Following the studies in Chapter 6, the population size control is related to the control of $Q^{\min}$ and $Q^{\max}$. To study this termination criterion, parameters (T_1, T_{n_p}, r, and n_p) are set the same as in the study of search depths. The Markov process (L=50) is selected. Eight Q_t from 10 to 3000 are selected to terminate COPT. Each result is the average of 10 statistical runs. Figure 7.7 presents the performance.



(Problem 1)



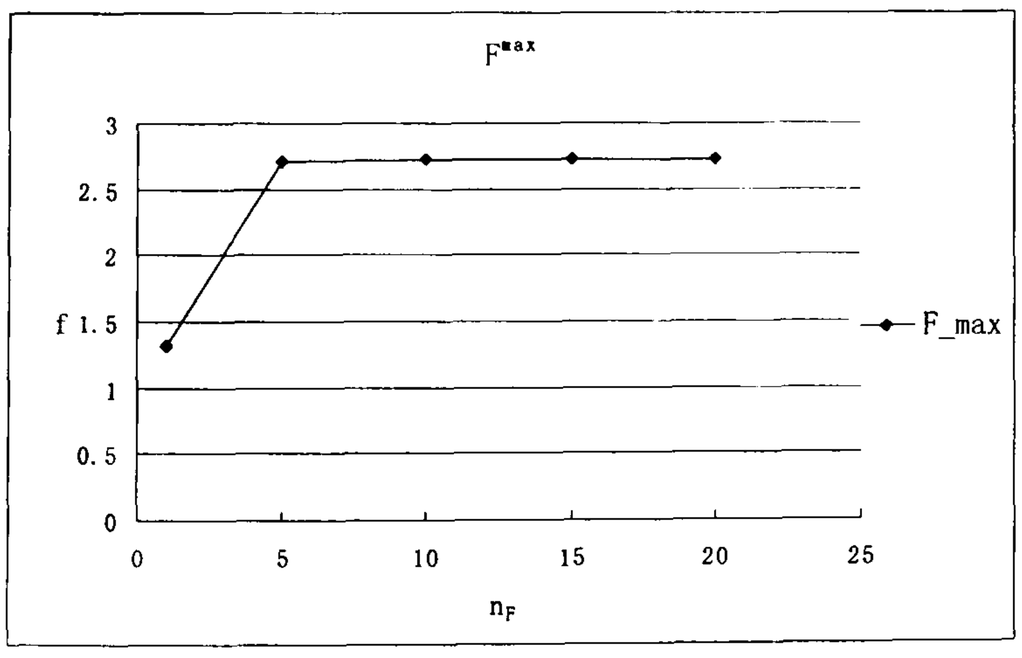
(Problem 2)

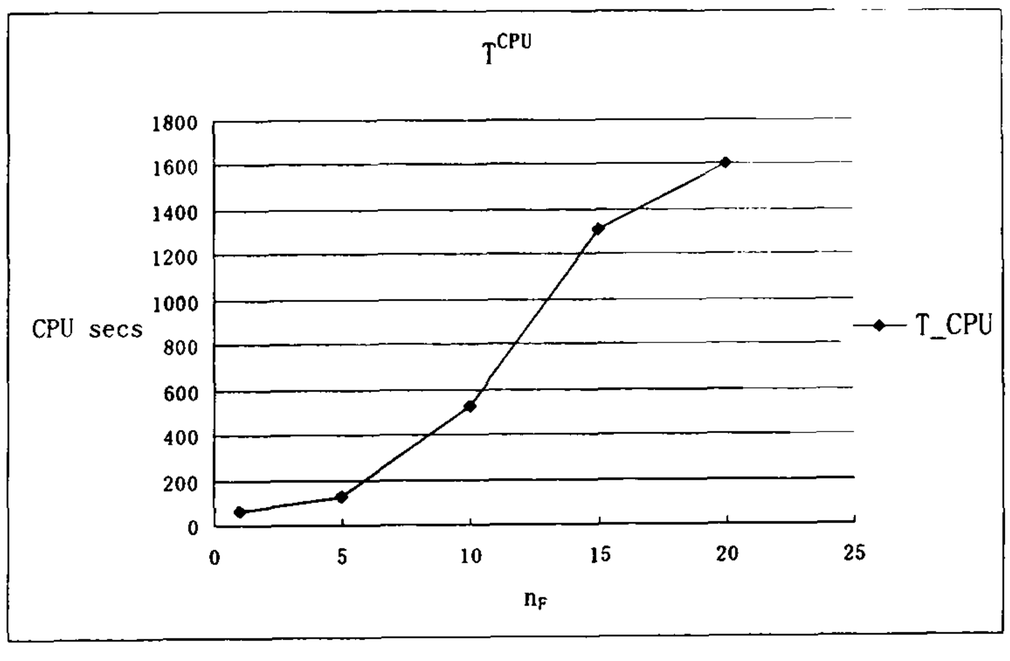
Figure 7. 7 Performance with different Q_t

Based on the results, the increasing Q_t increases the solution quality. The incremental benefit diminished as Q_t is larger than 1000. This is consistent with the study on the small-scale problems described in section 6.6. COPT can rarely converge to non-optimal or local optimal solutions if $Q_t \geq 1000$ so that $Q^{\min}$ is set to 1000. On the other hand, $Q^{\max}$ is set to 3000 to ensure that COPT converges to the optimal solution.

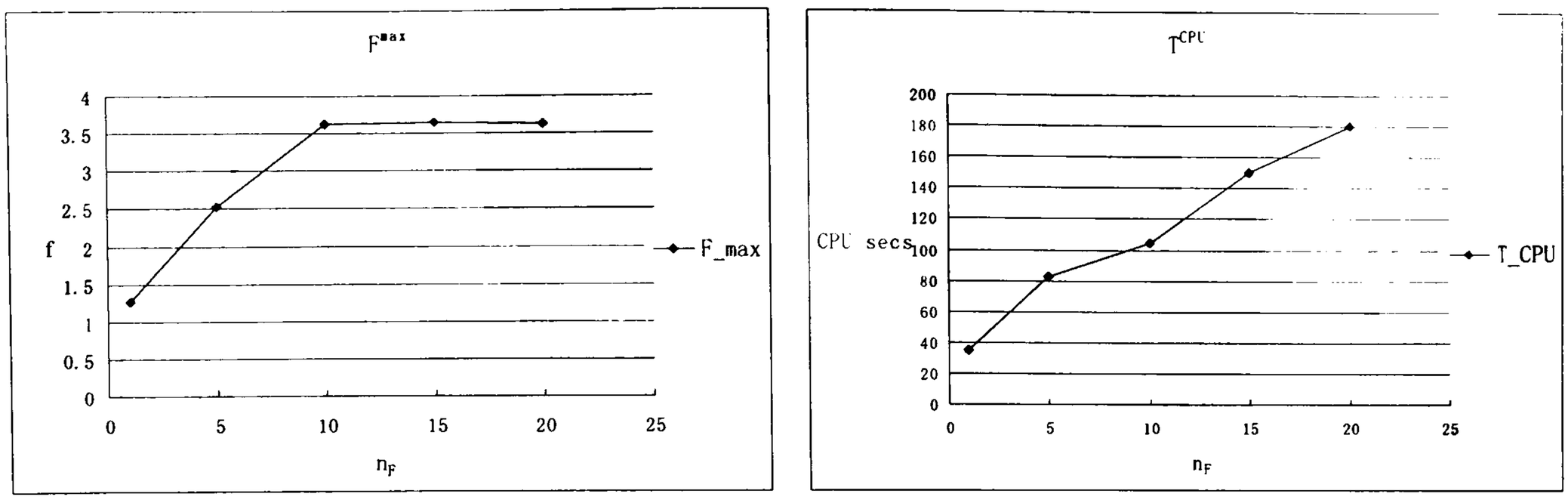
(ii) Optimisation Progress controls

In this termination criterion, COPT terminates if $\left|F_t^{\max} - F_{t+L}^{\max}\right| \leq \varepsilon$ for n_F iterations. $\varepsilon = 0.1$ is selected to determine the accuracy of the optimal solution. To study the effect of n_F, parameters (T_1, T_{n_p}, r, n_p, and L) are set same as those in the study of population size control. Figure 7.8 presents the performance when using different n_F from 1 to 20.





(Problem 1)



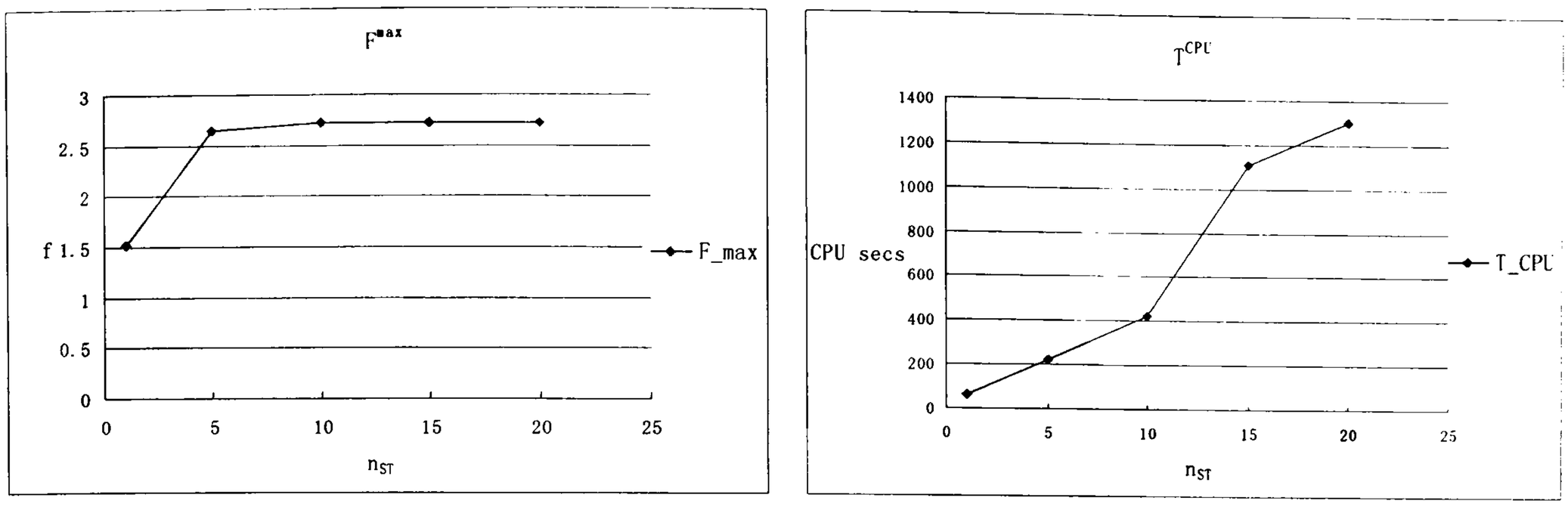
(Problem 2)

Figure 7. 8 Performance with different n_F

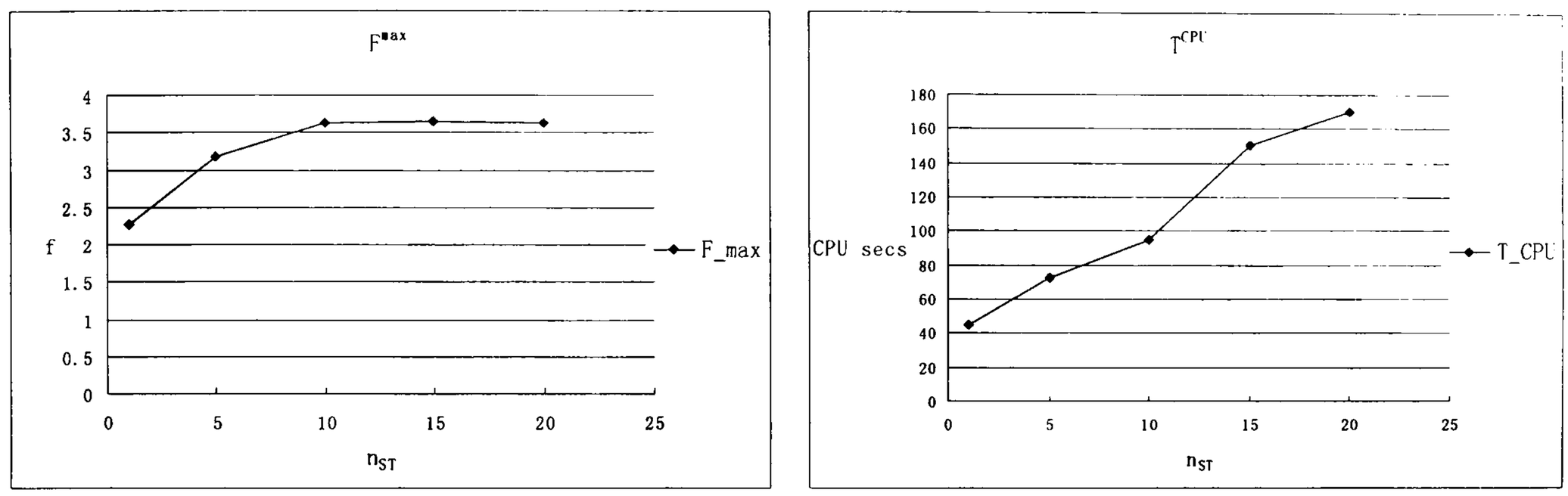
The results show that the solution quality is increased by increasing n_F. Meanwhile, the computation time also increases with increasing n_F. Adopting $n_F = 10$ can give the best tradeoff between the solution quality and computation time. This is consistent with the studies on small-scale problems illustrated in section 6.7.

(iii) Distribution control – pool level

Following the studies in section 6.7, the pool level is evaluated by the standard deviation of the population of pools (σ_t). COPT terminates if $\|\sigma_t - \sigma_{t+L}\| \le \varepsilon$ for n_{ST} iterations. $\varepsilon = 1$ is selected to determine the accuracy of the standard deviation. The parameters (T_1, T_{n_p}, r, n_p, and L) are kept constant. Figure 7.9 presents the performance when using different n_{ST} for the termination criterion.



(Problem 1)



(Problem 2)

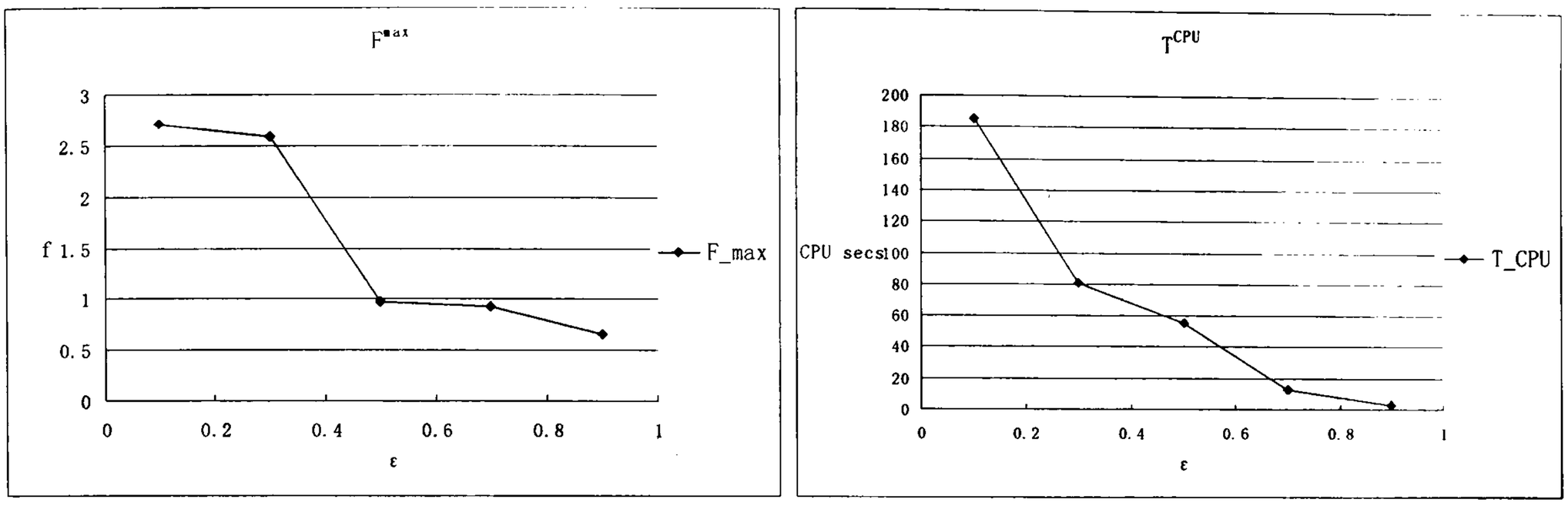
Figure 7. 9 Performances with different n_{ST}

The results show that the solution quality increases as n_{ST} increases. Meanwhile, the computation time also increases. Based on the results, the termination criterion with $n_{ST} = 10$ is adopted to give the best tradeoffs between the solution quality and computation time. This is consistent with the studies of pool levels in section 6.7.

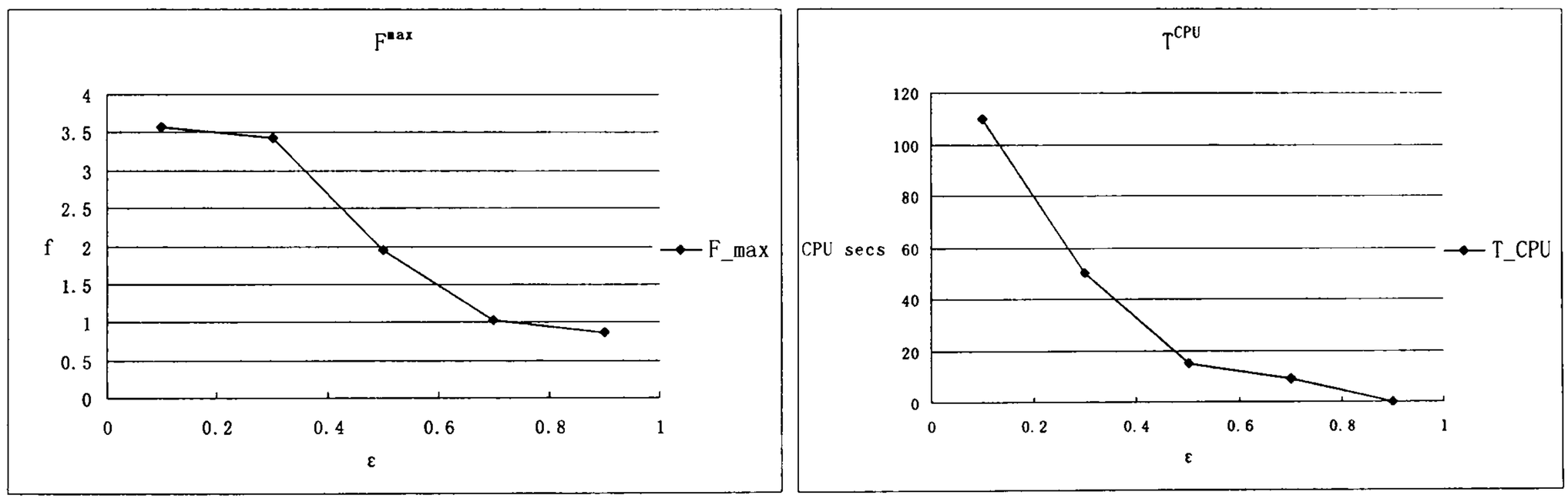
(iv) Distribution control – cascade level

Following the experience in Chapter 6, the cascade level is evaluated by the change of density function. In this termination criterion, COPT terminates if $\|d_t - d_{t+L}\| \le \varepsilon$ for n_d iterations. To study the effect of ε, $n_d = 3$ is selected. The parameters (T_1, T_{n_p},

r, n_p, and L) are kept constant. Figure 7.10 presents the performance when using different ε from 0.1 to 0.9.



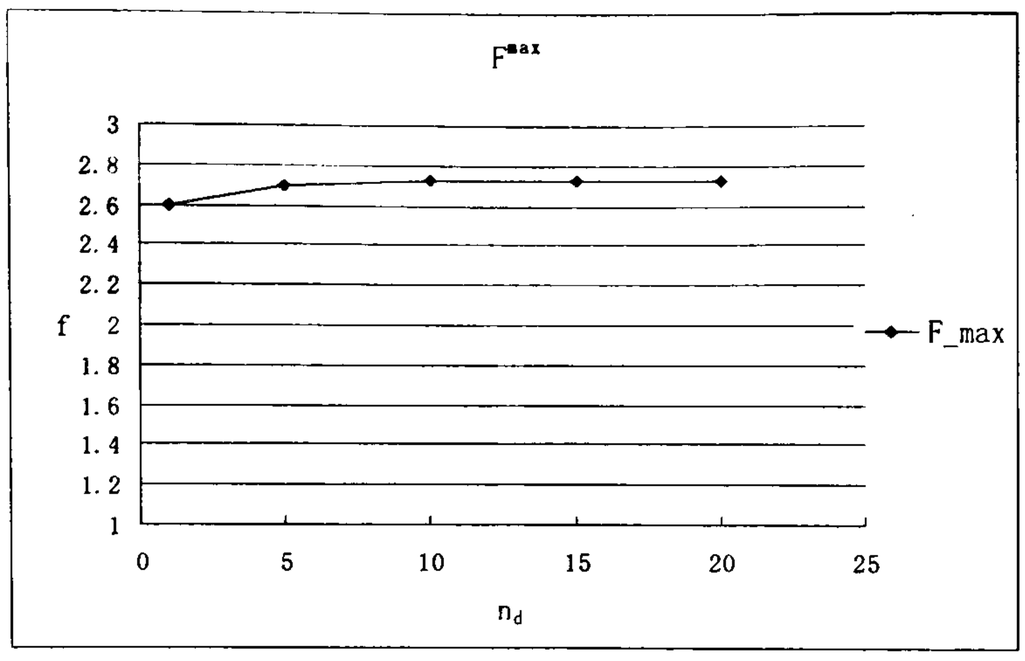
(Problem 1)

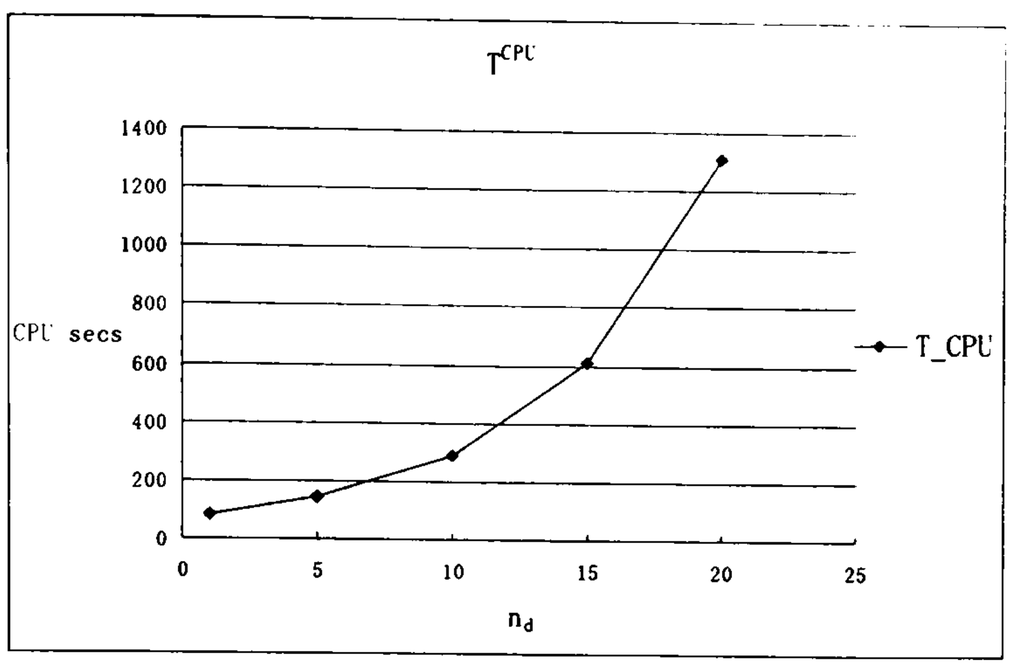


(Problem 2)

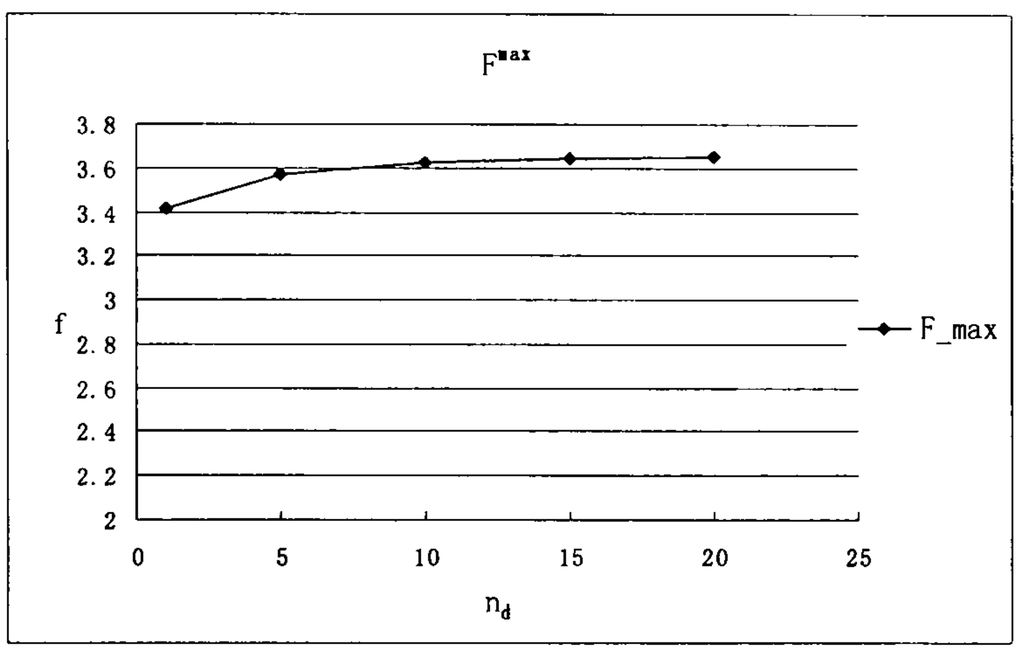
Figure 7. 10 Performances with different ε

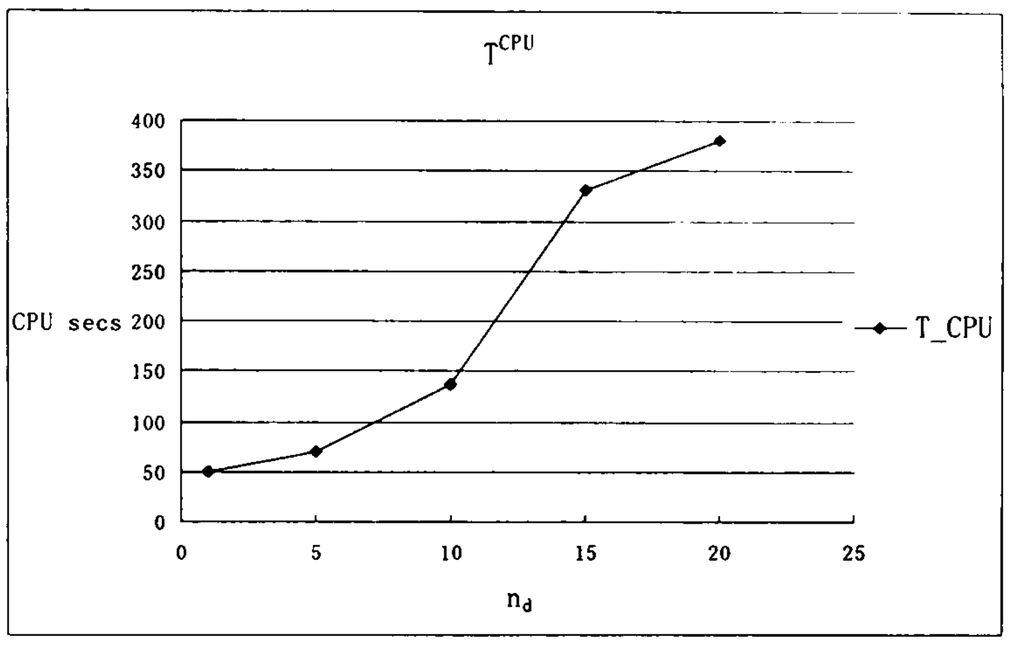
The results show that increasing ε decreases the solution quality and also the computation time. Adopting $\varepsilon = 0.3$ can give the best tradeoffs between the computation time and solution quality. Then $\varepsilon = 0.3$ is selected to study n_d . Figure 7.11 illustrates the performance when using different values of n_d from 1 to 20.





(Problem 1)





(Problem 2)

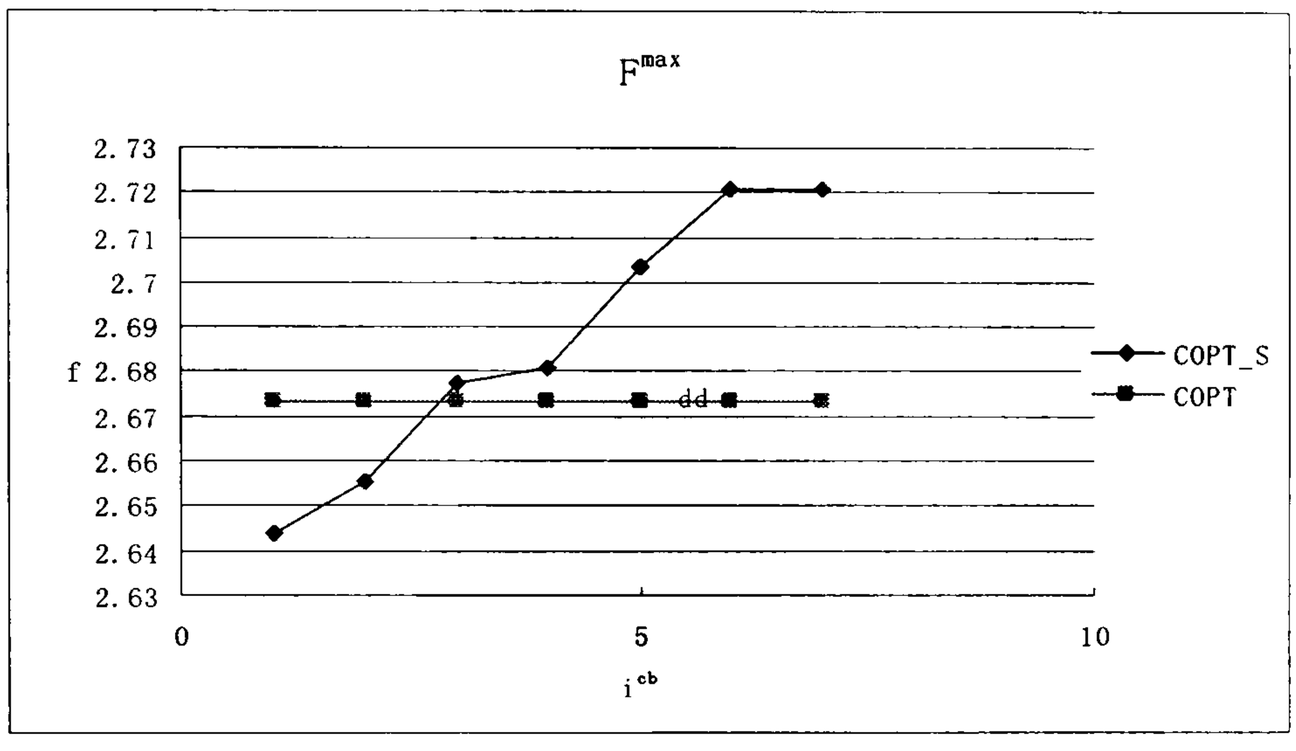
Figure 7. 11 Performance with different n_d

The results show that increasing n_d can increase the solution quality. Meanwhile, the computation time also increases. Adopting $n_d = 10$ can give the best tradeoff between the solution quality and computation time. This is the same trend off as was found in the study of n_d in section 6.7.

(e) Search policies

Depending on the above study, the termination criteria selected are: COPT does not terminate if Q_t is smaller than 1000 and terminates if Q_t is larger than 3000, or if $\left|F_t^{\max} - F_{t+L}^{\max}\right| \le 0.1$ for 10 iterations, or if $\left\|\sigma_t - \sigma_{t+L}\right\| \le 1$ for 10 iterations, or if

$\|d_t - d_{t+L}\| \leq 0.3$ for 10 iterations. Comparison between the performances of the normal COPT and the COPT that uses the choices in Table 7.1 (COPT_S) is performed to study the search policies. The parameters (T_1, T_{n_p}, r, n_p, and L) are set the same as in the earlier study of termination criteria. The results are the averages of 10 statistical runs that use different initial structures. Figure 7.12 illustrates the comparison.



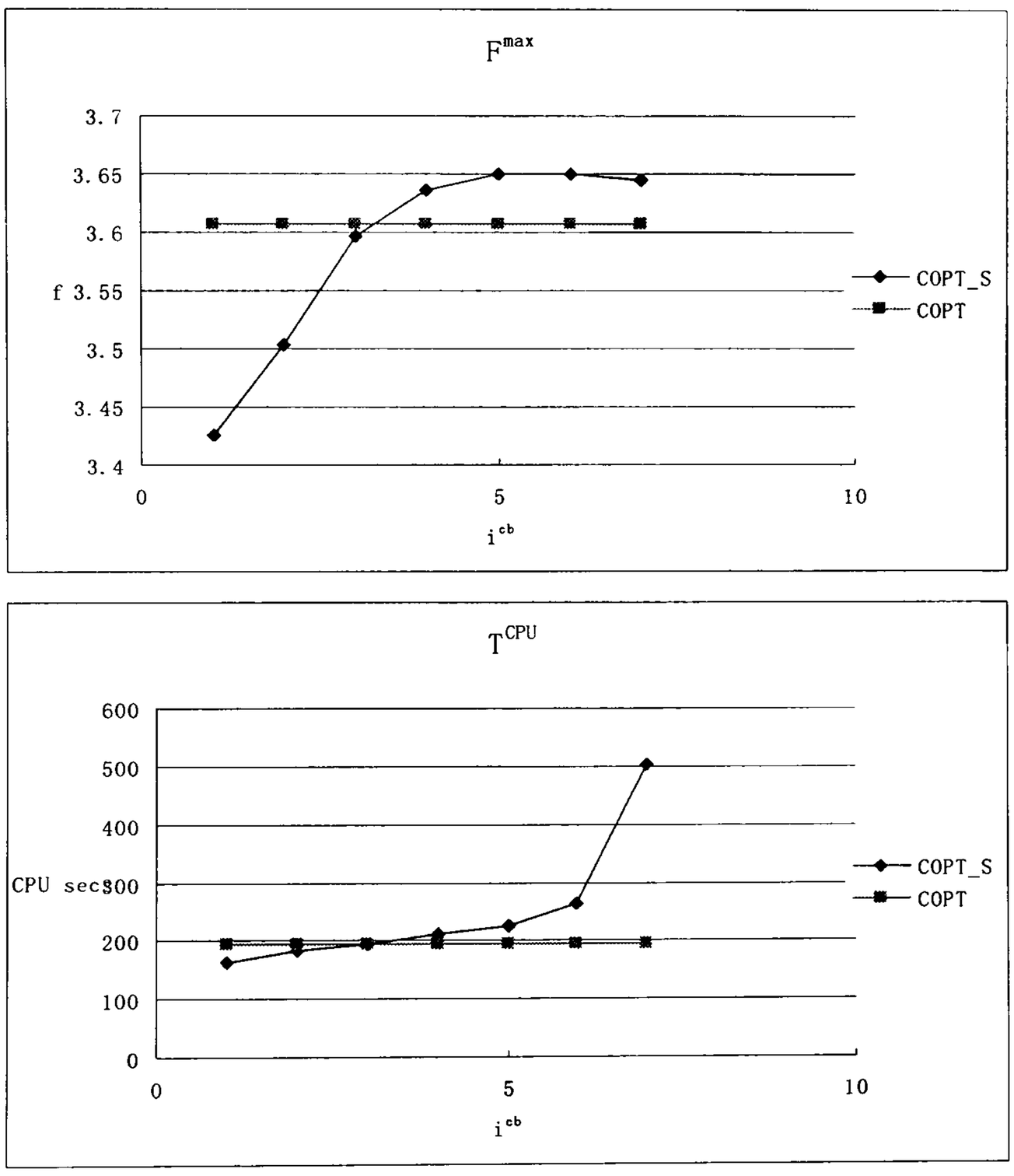
(Problem 1)

Figure 7. 12 Comparison between the performances of COPT and COPT_S I

For COPT_S, the solution quality increases with an increasing bias to the settling range. Meanwhile the computation time increases with the bias. This reflects that COPT can converge to a good solution if the search has a bias to the settling range. Compared to the normal COPT, COPT_S can converge to a better solution with a bias to the middle

range or the settling range (choice IV- VII). However, COPT_S requires longer computation time than COPT with the bias. Based on the figures, the choice VI ($p_1 = 0.25, p_2 = 0.25, p_3 = 0.5$) gives the best tradeoffs between the computation time and solution quality.

The search policies are also studied in problem 2. Figure 7.3 presents the comparison between the performances of COPT and COPT_S.



(Problem 2)

Figure 7. 13 Comparison between the performances of COPT and COPT_S II

The conclusion is consistent with the study of problem 1. The solution quality increases with an increasing bias on settling range. Meanwhile, the computation time increases. The results show that the choice VI can give the best tradeoff between the computation time and the solution quality.

7.5 Comparison with other methods

Comparison between the performances of COPT and a conventional stochastic method is performed on a more complex problem (problem 3) than problem 1 and problem 2. Parameters (T_1, T_{n_p}, r, n_p, and L) and the termination criteria are set the same as in the study of search policies. The search policy with choice VI is selected as the optimum.

Referring to the studies by Ashley (2004), parameters and termination criterion of TS are fixed, so that:

- Neighborhood size is set to 7.
- The Tabu list that has a single entry is selected.
- Termination criteria are set at un-improving for 50 iterations or maximum of 500 iteration occurs.

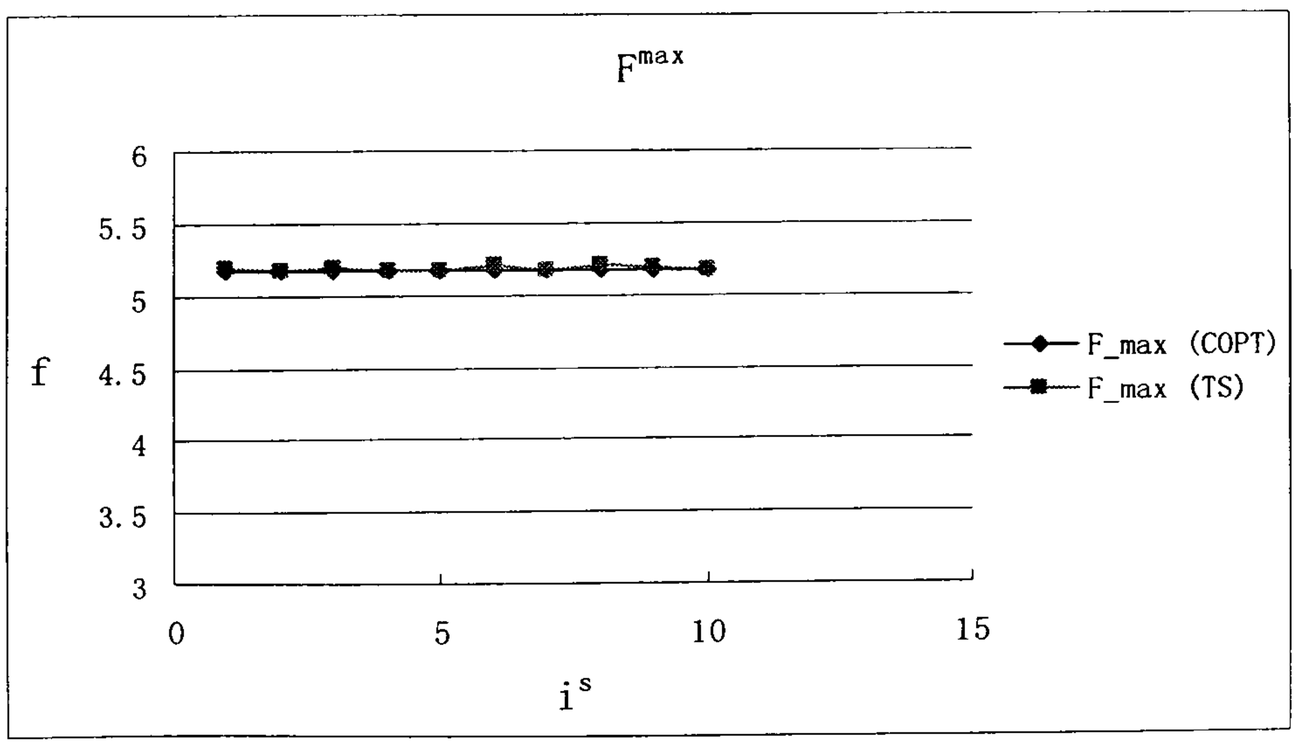
COPT runs on the computing environment involving a master and a worker. The configurations of the master and the worker are as in Table 6.2. TS runs on the worker. The solution quality and computation time for TS are evaluated by the best objective ever found and the worker's CPU time respectively. Table 7.5 illustrates comparison between the performances of the two methods. These results are averages of 10 runs using different initial structures.

i^s : index of initial network structure

Table 7. 5 Comparison between COPT and TS

	COPT		TS	
i^s	$F^{\max}$	T^{CPU}	$F^{\max}$	T^{CPU}
I	5.180	59765.432	5.187	116864.701
II	5.174	55009.087	5.172	100396.202
III	5.175	12142.612	5.200	24279.421
IV	5.174	34782.156	5.173	84837.643
V	5.167	43789.348	5.172	76189.532
VI	5.166	45721.413	5.201	69201.233
VII	5.174	32142.265	5.166	50132.212
VIII	5.170	39123.574	5.201	66125.521
IX	5.170	32152.613	5.190	55284.311
X	5.174	61923.557	5.180	125264.719

Depending on the table, the performances of the two methods are further illustrated in Figure 7.4.



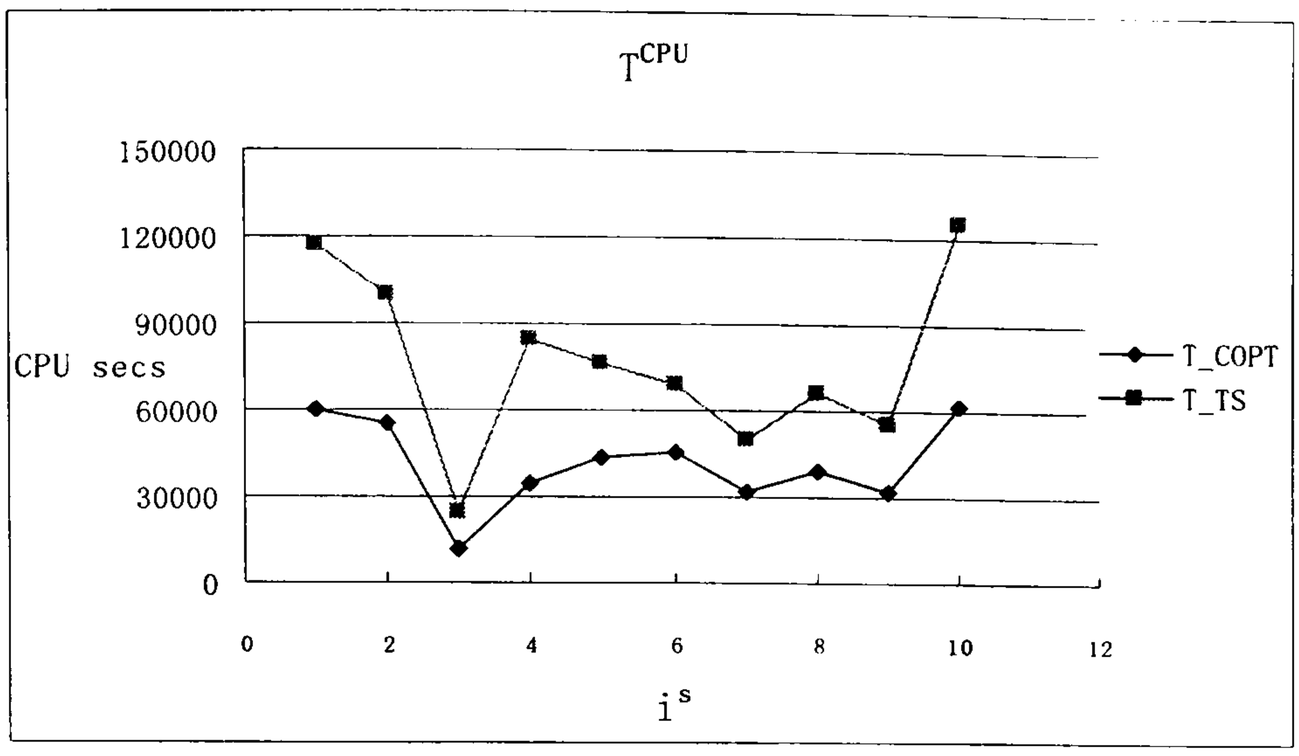


Figure 7. 14 Comparison between COPT and TS

F_max (COPT): the solution quality of COPT

F_max (TS): the solution quality of TS

T_CPU (COPT): the computation time for COPT

T_CPU (TS): the computation time for TS

Based on these results, we can say that COPT can achieve the same solution quality more quickly than TS. For example, the computation time for COPT that uses the second initial structure is twice as long as that for the TS.

7.6 Remarks and Discussions

COPT is studied on large-scale problems in this chapter. The impacts of the features are consistent with the earlier studies on small-scale problems. COPT with concave distribution can lead to better optimal solution but spend longer computation time than that with convex distribution. Increasing structure size (the number of pools) increases the solution quality and the computation time. The search depth (Markov process length) does not have apparent impact on solution quality but the computation time. COPT with a long Markov process requires a long computation time to converge. Based on the study

of large-scale problems, adopting r=-2, $n_p = 100$, L=50 can give the best tradeoff between the computation time and solution quality. Similar to the study of small-scale problems, the termination criterion related to the control in population size defines a minimum population ($Q^{\min}$) and maximum population ($Q^{\max}$). Based on the results, $Q^{\min}$ and $Q^{\max}$ are set to 1000 and 3000. COPT can also terminate if $F_t^{\min}$ stay the same for a number of iterations (n_F). Increasing n_F can increases the solution quality and the computation time. Based on the results, terminating the algorithm when $\left|F_t^{\max} - F_{t+L}^{\max}\right| \le 0.1$ for 10 iterations can give the best tradeoff. The last termination criterion related to the control in pool level and cascade level. To control the pool level, the algorithm terminates if the standard deviation of population of pools stay same ($\|\sigma_t - \sigma_{t+L}\| \le 0.1$) for a number of iterations (n_{ST}). On the other hand, to control the cascade level, COPT terminates if the density function of the domain stay same ($\|d_t - d_{t+L}\| \le \varepsilon$) for a number of iterations (n_d). Increasing n_{ST} and n_d increases the solution quality and the computation time. The algorithm with a small ε can converge to a good optimal solution but require a long computation time. Similar to the study of small-scale problems, setting both n_{ST} and n_d to 10 and ε to 0.3 can give the best tradeoff between computation time and solution quality.

Search policies are proposed to determine the form of distributing acceptance probabilities in the search. The pools are clustered into floating, middle, and settling ranges. Each range is assigned by a probability to select pools from this range. Based on the results, the search policies appear to be an open problem that could benefit by customization. The solution quality increases when there is an increasing bias to the settling range. Meanwhile the computation time increases with this bias. The search

policy with choice VI ($p_1 = 0.25, p_2 = 0.25, p_3 = 0.5$) can give the best tradeoffs between the computation time and solution quality. In addition, COPT is also studied on a complex problem. The comparison between COPT and TS is performed and results illustrate that COPT can converge to a solution with similar solution quality more quickly than TS.

The previous computing environment includes only a single worker. However, the parallel and distributed environment should have multiple workers. To apply COPT on this environment, the implementation requires some minor adjustments. In theory, the convergence can be sped up by increasing the number of workers. In the next chapter the benefits in speeding up convergence are studied and the benefits of COPT and stochastic method are compared.

Chapter 8 Parallel and Distributed computing

8.1 Introduction

COPT, in comparison to TS, SA, etc, has a large numbers of features that are parallel and independent:

- Each Markov process, previously part of a longer process, is now independent.
- The management of intermediate solutions between Markov processes can be done in parallel with the development of solutions.

So COPT has obvious potential to distribute. Tasks to distribute include:

- The development of new solutions (launch of Markov processes)
- Communication with pools (analysis and management of intermediate solutions)

The above tasks are related to Tsk^{MC} described in section 5.2.

Tasks related to storage and management of solutions include:

- Input data into pools
- Export data from pools
- Development of Inflections
- Checking the termination

The above tasks are related to Tsk^{Ip}, Tsk^{Ep}, $Tsk^{\inf}$, Tsk^{Tm} described in section 5.2.

8.2 Methodology

Following the discussions in section 5.2, COPT has a master process and worker processes. The master process is to:

Step 1: Initialization Select a number of partitions and pools. Generate a number of initial points for workers (N workers have at least N initial points).

Step2: Data extraction (Tsk^{Ep}) Select a set of points and store them in output files. The output files are associated with different workers.

Step 3:Storage (Tsk^{IP}) Read input files and inserts the new points into pools.

Step 4: Manage population ($Tsk^{\inf}$) Develop the inflections of cascade following Definition 4.6.

Step 5: Check convergence (Tsk^{Tm}) Check the convergence of COPT following the termination criteria described in section 7.4.2

Step 2, 3, 4, and 5 are independent and following different time schedules. On the other hand, the worker processes execute Tsk^{MC} following:

Step 1: Selection of initial points Fetch the output files from the master and read the initial point in the files.

Step 2: Development of intermediate solutions Run Markov processes simultaneously to generate new solutions.

Step 3: Storage and management of solutions Store new solutions into input files and send them to the master.

Figure 8.1 presents the network structure of COPT on parallel and distributed computing environments.

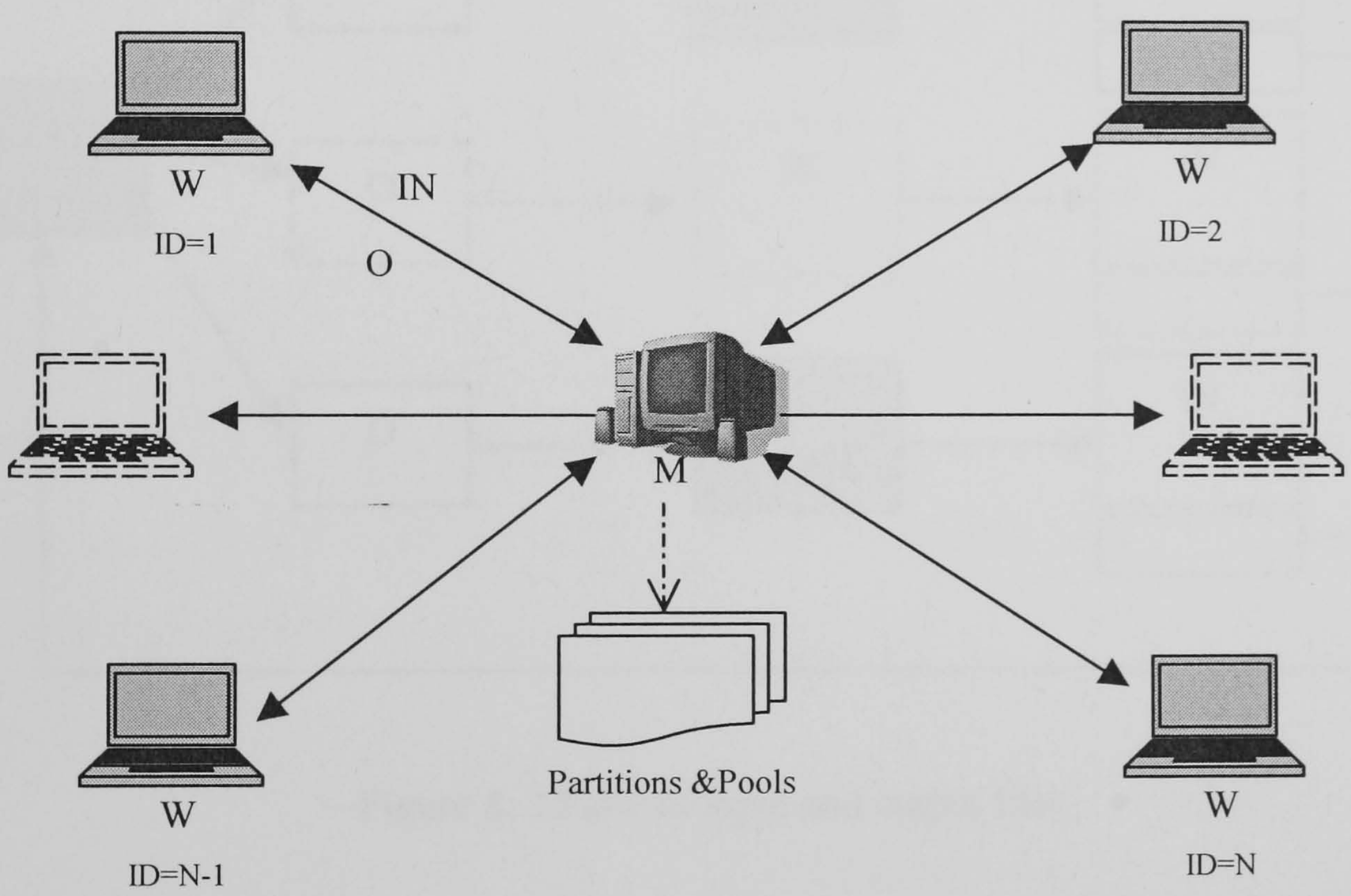


Figure 8. 1 Network structure of COPT

M: master computer

W: worker computers

IN: input file

O: output file

ID: identities of worker computers (from 1 to N)

Each worker has a unique ID (worker ID) different to those of other workers. Solutions generated by workers are stored in the partitions and pools together with their differing worker IDs. The partitions and pools are implemented as tables, similar to those in section 5.4.1. Tables now involve an additional column that store the worker IDs. Input and output files are communicated between the master and workers. Figure 8.2 outlines the flow of input and output files in COPT

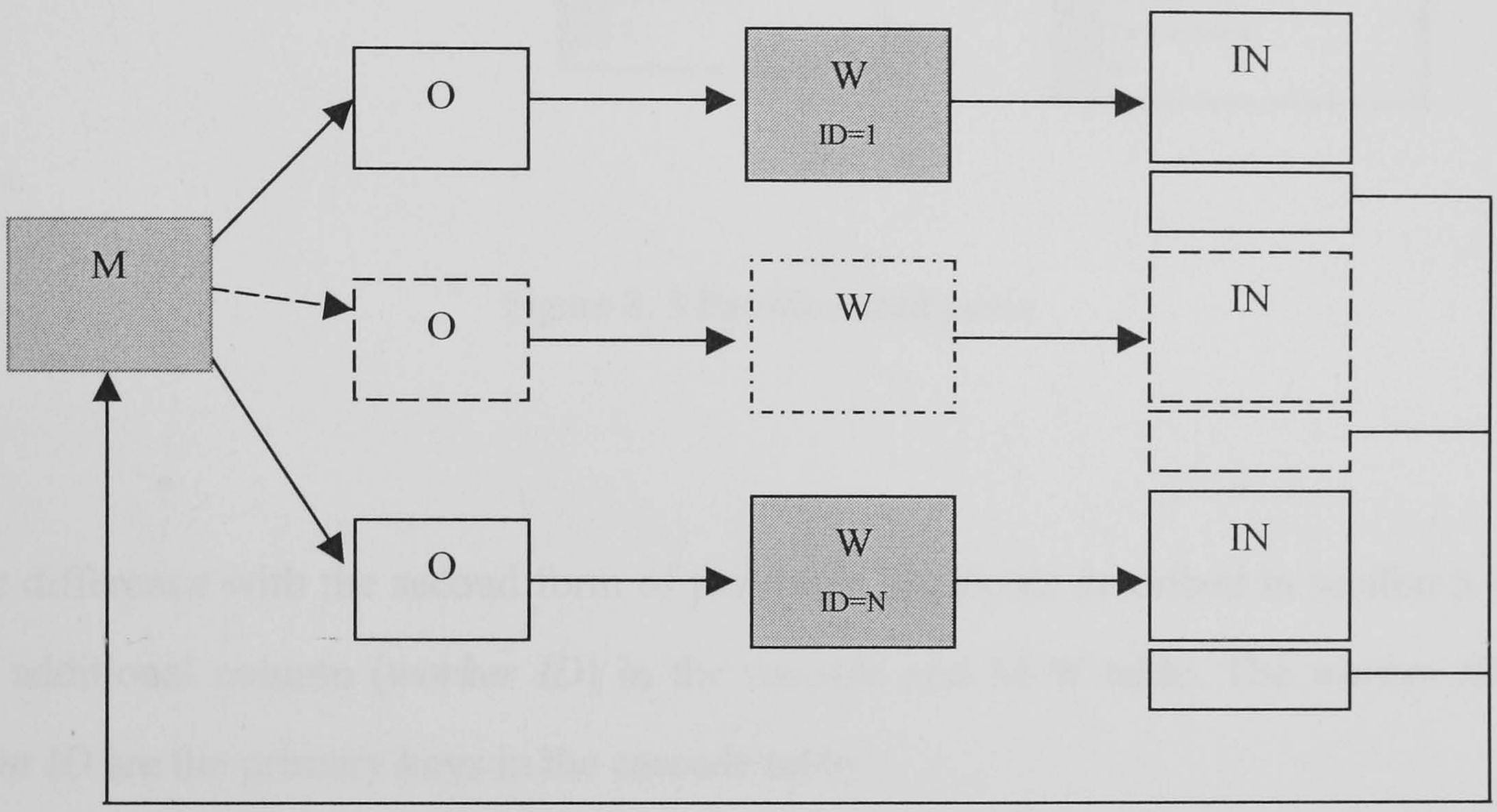


Figure 8. 2 Flow of input and output files

Input and output files are associated with different workers. The ID is appended in the name of input and output files so that workers and master can identify the files. The master extracts M-W information (as described in section 5.3) and puts it into output files.

Workers store new solutions into input files with different names and send them, actually part of the files that involve W-M information as described in section 5.3, to the master. The master then places the new solutions into partitions and pools.

Partitions and pools

Due to the presence of worker IDs, points in partitions are identified by point IDs and worker IDs. For example, point 3 is generated by worker 2 so that its worker ID and point ID are 2 and 3 respectively. Figure 8.3 presents the partitions and pools.

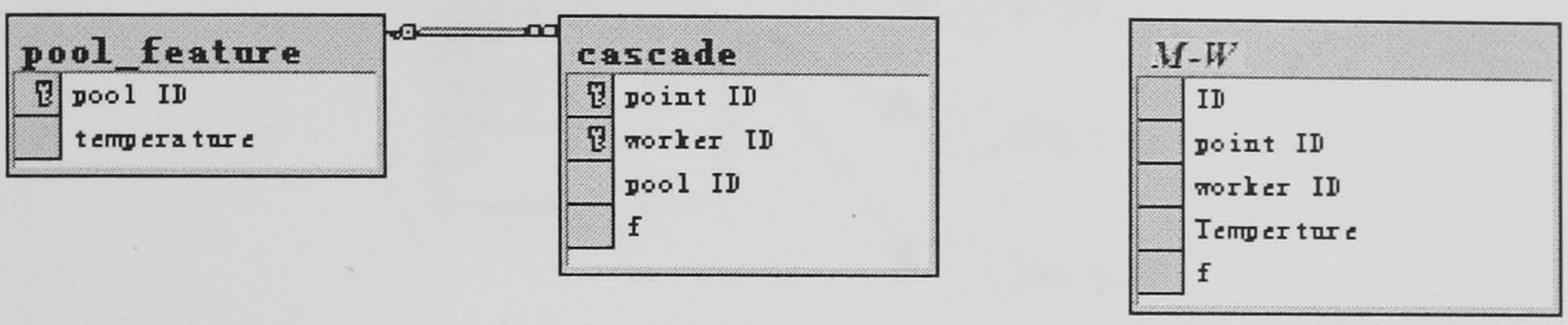


Figure 8. 3 Partitions and pools

The difference with the second form of partitions and pools described in section 5.4.1 is the additional column (*worker ID*) in the cascade and M-W table. The *worker ID* and *point ID* are the primary keys in the cascade table.

Data extraction and storage

Data extraction and storage on the master are now implemented in different ways due to the presence of multiple workers. The differences are explained as follows:

(i) Extraction on the master

The master extracts M-W information from the partitions and pools and stores it in the *M-W* table. The information, different from that described in section 5.3, involves the worker ID. Then the master writes this information into output files. The output files are associated with different workers and appended by worker ID in names. Figure 8.4

illustrates the data extraction from the master. Let us assume that three workers (1, 2, and 3) are present in the computing environment.

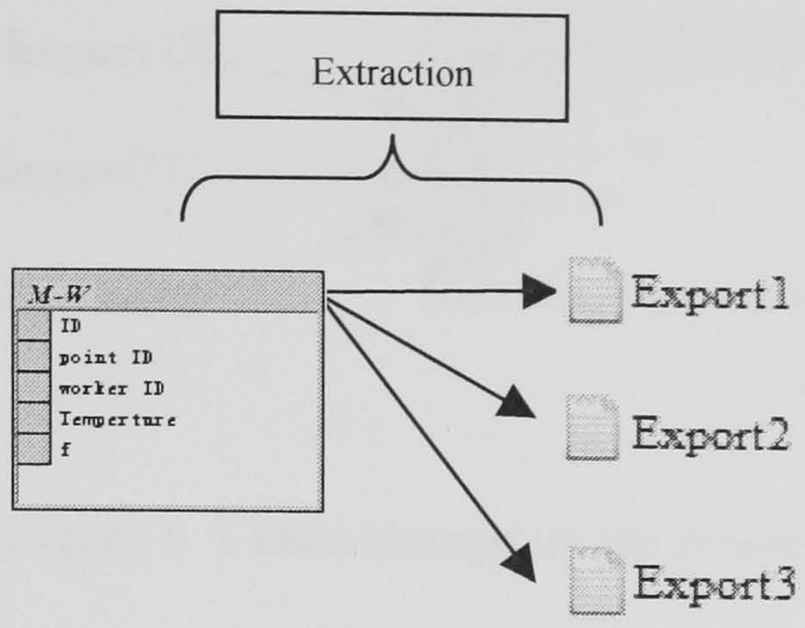


Figure 8. 4 Data extraction from the master

Similar to Tsk^{Ep} described in section 5.4.4, the output file is named as *Export.* The master generates three output files (*Export1, Export2, Export3*) associated with the three workers (1, 2, and 3).

(ii) Storage on the master

Input files, different from the one in the second implementation described in section 5.4.5, are appended by worker ID in the name. The W-M information in the input files involves worker IDs. The master places the W-M information into the cascade table. Figure 8.5 presents the data storage on the master. Similar to the description in (i), the environment has three workers (1, 2, and 3).

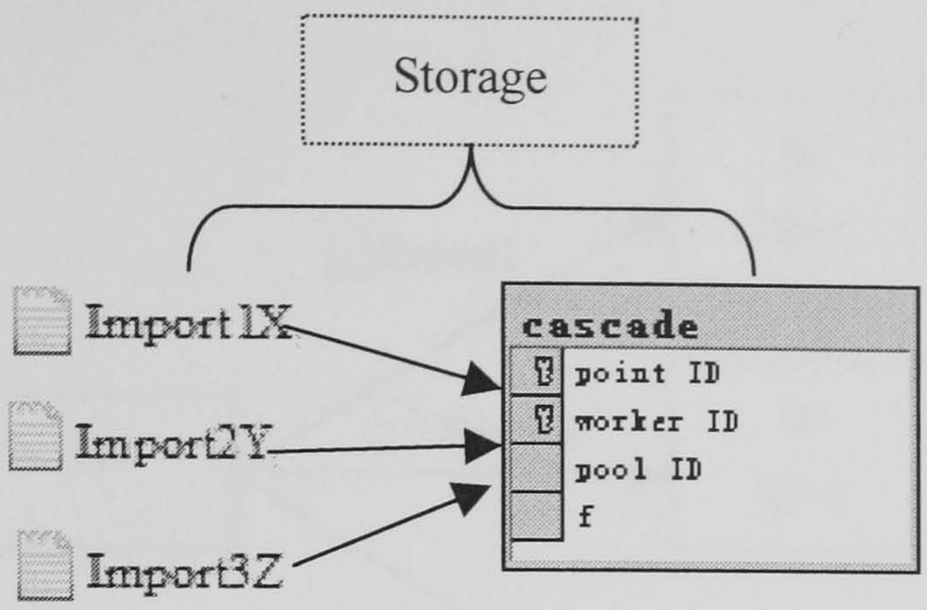


Figure 8. 5 Data storage on the master

Input files (*Import1X, Import2Y, and Import3Z*) are generated respectively by workers 1, 2, and 3. Beside the worker ID, the input files are also appended with the point ID (*X, Y,* and *Z* are the point IDs). This is similar to the second implementation described in section 5.4.6. The M-W information from the input files is stored into the *cascade* table by the master.

(iii) Extraction on workers

Since the output files are associated with workers, workers take the specific output files from the master and read the M-W information from them. Figure 8.6 presents three workers (1, 2, and 3) taking their output files from the master.

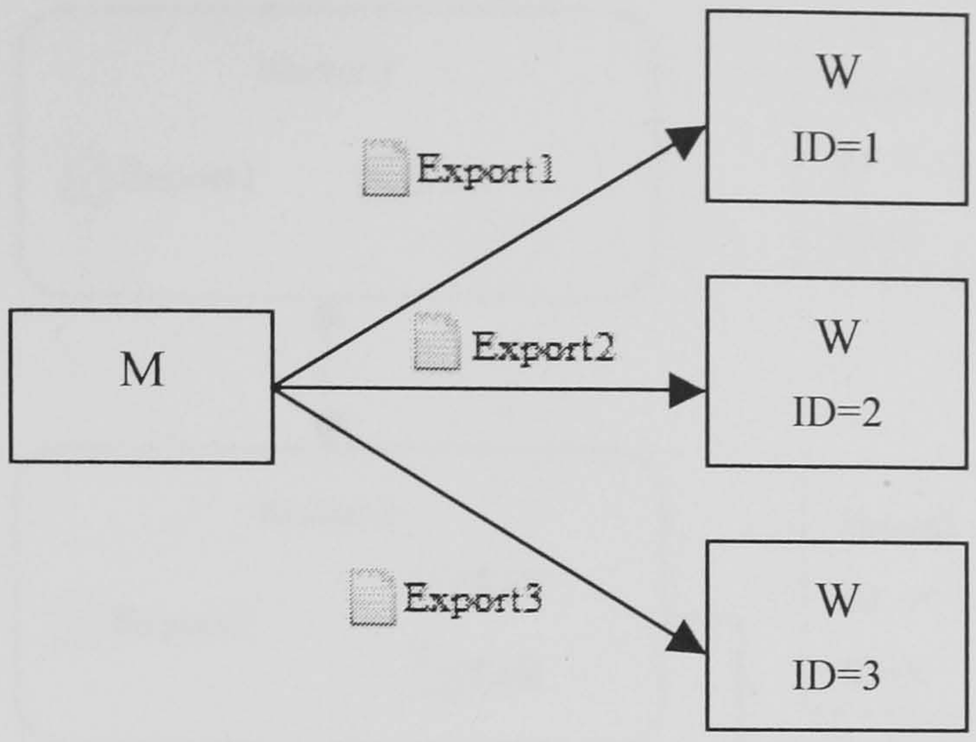


Figure 8. 6 Workers taking output files

The figure represents how workers (1, 2, and 3) take the output files (*Export1, Export2,* and *Export3*) from the master.

M-W information in output files involves worker ID and point ID, which help workers to find the corresponding solution files. Solution files, similar to the ones in the second implementation described in section 5.4, store variables and differ in names. Their names are appended by point ID (SoX where X is point ID). Figure 8.7 represents the three workers finding corresponding solution files.

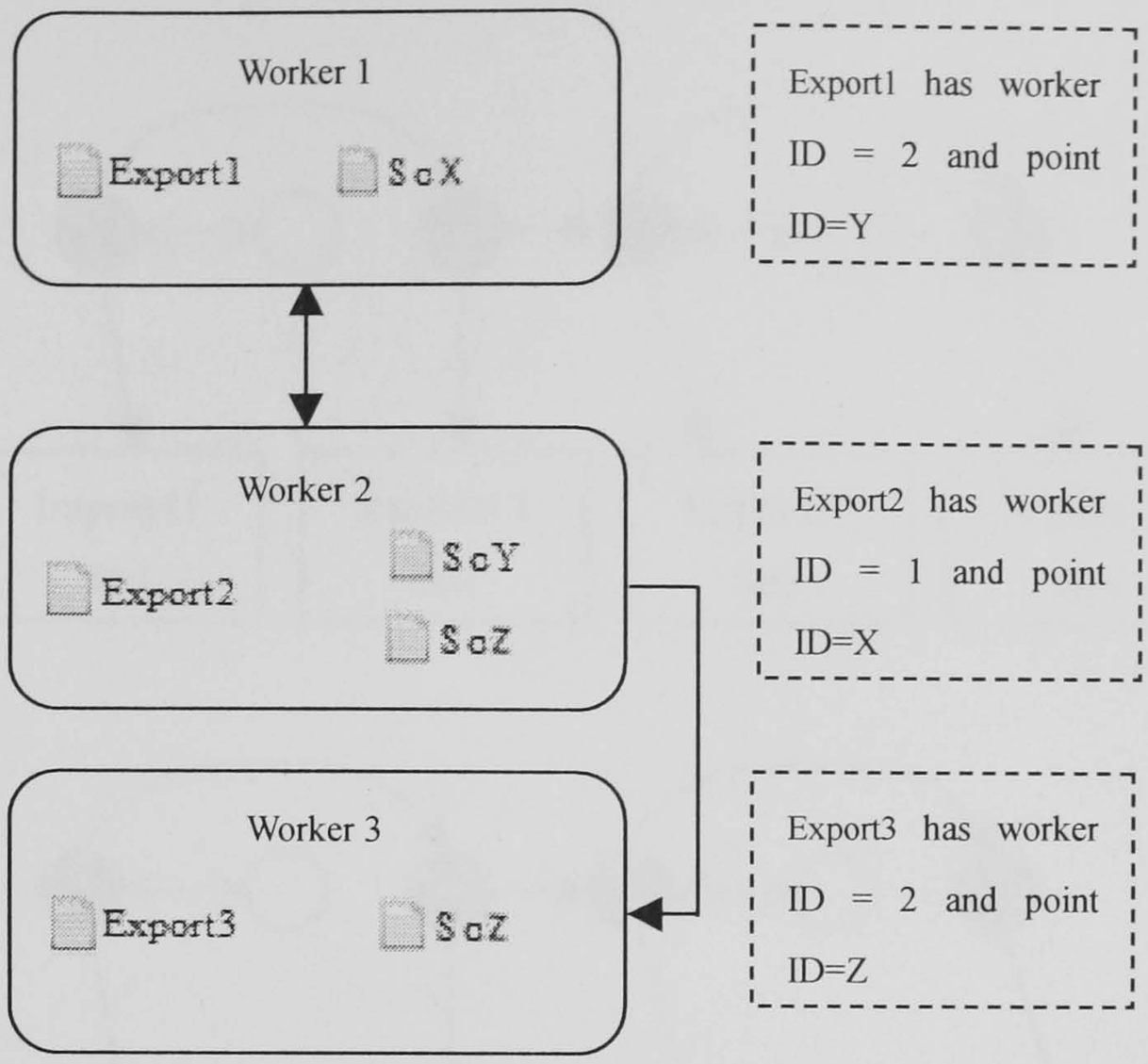


Figure 8. 7 Workers finding solution files

As the figure illustrates,

- *Export*1 has worker ID = 2 and point ID = Y so that worker A reads *SoY* in worker 2.
- *Export*2 has worker ID = 1 and point ID = X so that worker A reads *SoX* in worker 1.
- *Export*3 has worker ID = 2 and point ID = Z so that worker A reads *SoZ* in worker 2.

(iv) Storage on workers

Then workers read the solution files and start Markov processes. The Markov processes follow the Metropolis criterion and generate new points. New points are stored in

solution and input files. Figure 8.8 illustrates workers storing new points into files. Two workers (1 and 2) are represented in the figure.

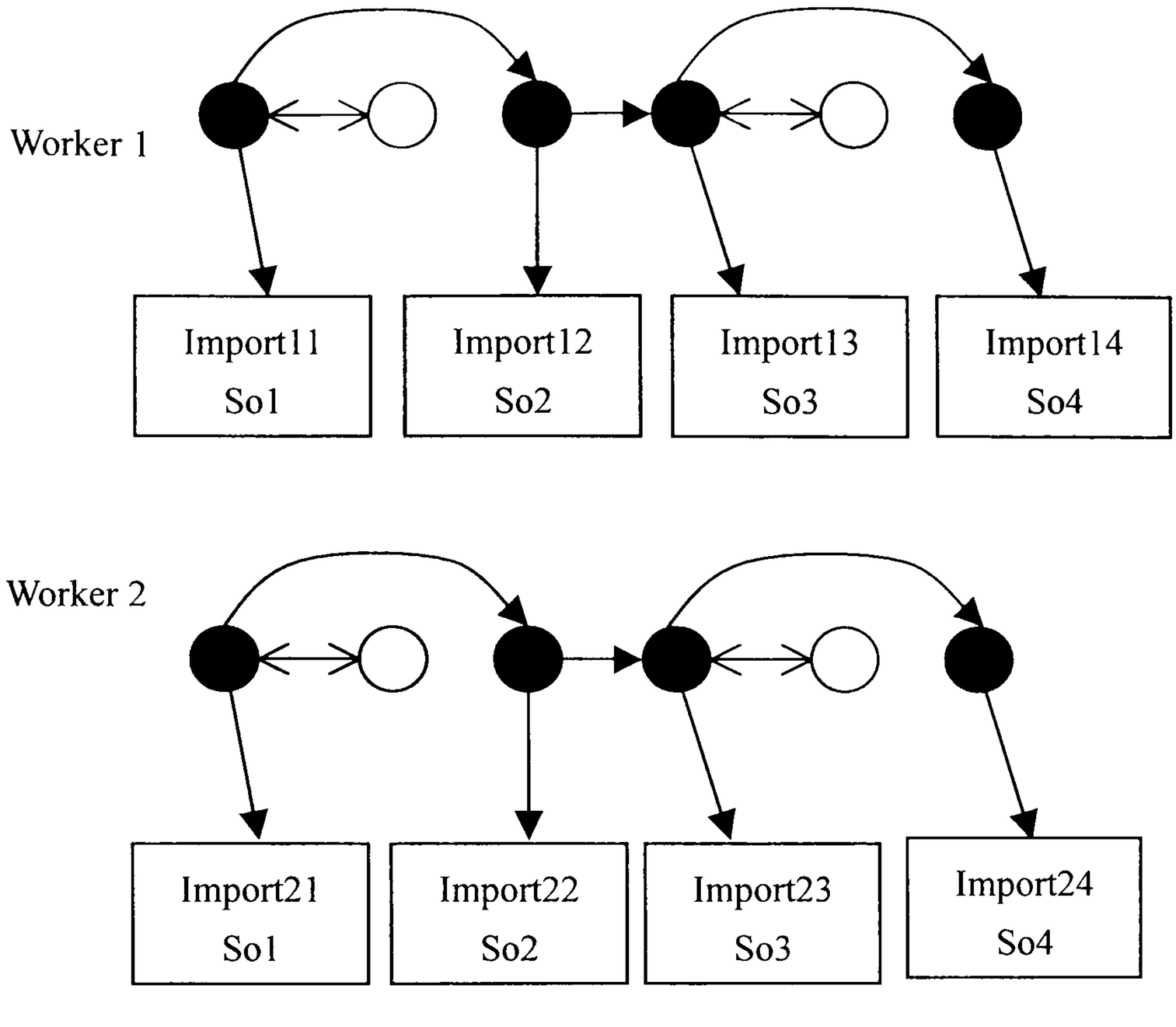


Figure 8. 8 Workers storing new points into files

As the figure illustrated,

- Worker 1 generates points 1, 2, 3, 4 and stores them in files import11 and So1, import12 and So2, import13 and So3, import14 and So4.
- Worker 2 generates points 1, 2, 3, 4 and stores them in files import21 and So1, import22 and So2, import23 and So3, import24 and So4 files.

8.3 Synchronous mode and asynchronous mode

With multiple workers are employed, COPT can be executed either in synchronous or in asynchronous modes.

(i) Synchronous mode

Workers are parallelized to run the Markov processes simultaneously in this mode. Since workers have different computational capacities, the faster worker has to wait for the slower ones before sending the input files. The worker tasks include:

Selection of initial points

Development of intermediate solutions

(idle time as they wait for the other workers)

Storage and management of solutions

Since the faster workers have to wait for the slower ones, benefits in speeding up convergence are limited.

(ii) Asynchronous mode

Workers execute their own independent Markov processes and communicate information with the master independently. Worker tasks include:

Selection of initial points

Development of intermediate solutions

Storage and management of solutions

In this mode, computation resources are more efficiently used and the computation time can be greatly reduced.

8.4 Demonstration

The distributed computing environment involves 1 master and 10 workers. The configurations of these computers are illustrated in Table 8.1.

Table 8. 1 Configurations of the master and the workers

Master	**CPU**	**Ram Memory**
Prise-sql	Intel(R) Core(TM) 2.13GHz	2Gb
Worker	**CPU**	**Ram Memory**
Bscgrid01-Bscgrid06	Intel(R) Pentium(R) 3GHz	2Gb
Khafre01-Khafre04	Pentium III (Cascades) 7000MHz	512MB

Bscgrid01-Bscgrid06 are 6 faster workers, while Khafre01-Khafre04 are 4 slower workers. The master and workers are connected via Internet. Information is transferred between the master and the workers or between the workers by FTP protocol.

8.4 Parallel and distributed application

(i) Performance and settings

Following the studies described in Chapter 7, the performance is reflected on the solution quality and the computation time required to converge. The solution quality is the best objective in the pools ($F^{\max}$). With workers on the Internet, it is difficult to find their

CPU time. The computation time is reflected on the clock time of the slowest worker. which is illustrated as:

$$T^{CL} = \max\{[T_{1,j}^{CL}]_{j=1}^{N_{PC}}\}$$

T^{CL} : computation time

j : index of workers

N_{pc} : the number of workers

$T_{1,j}^{CL}$: the clock time of worker j

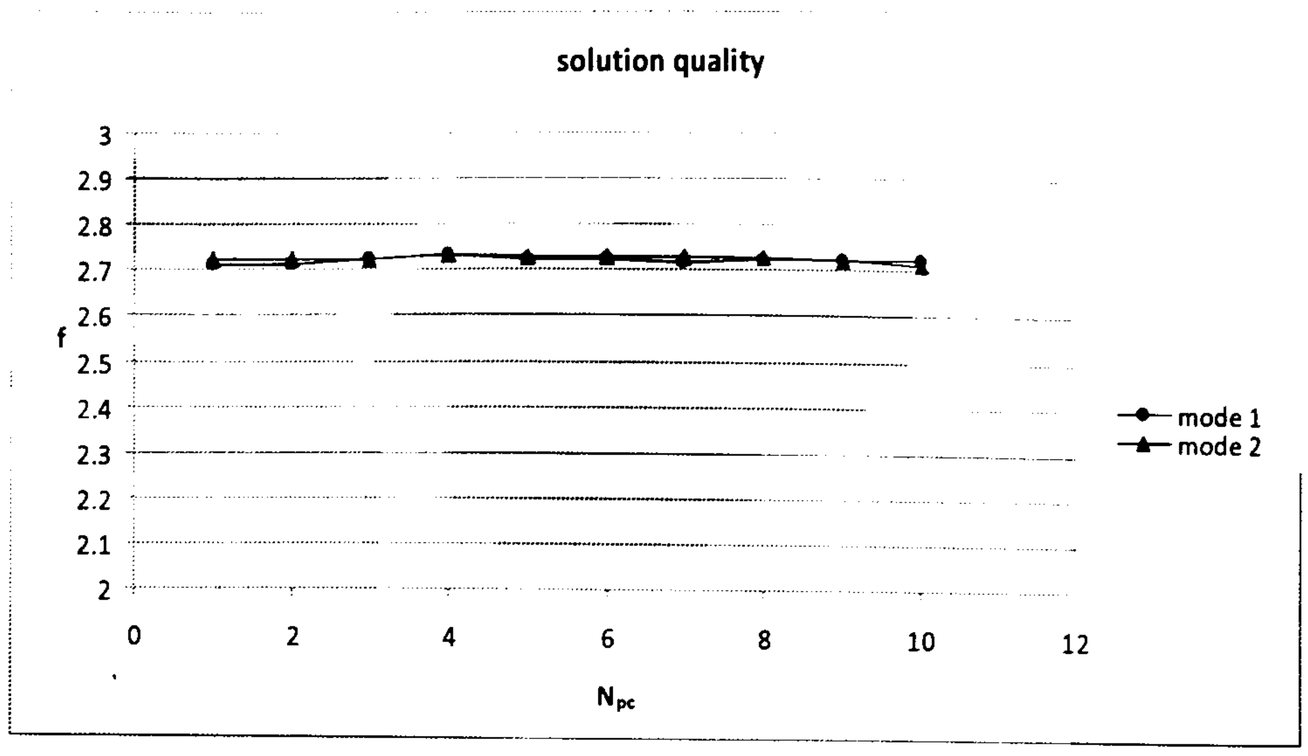
Besides, parameters (T_1, T_{n_p}, r, n_p,and L) and the termination criteria are set the same as in the studies of section 7.5.

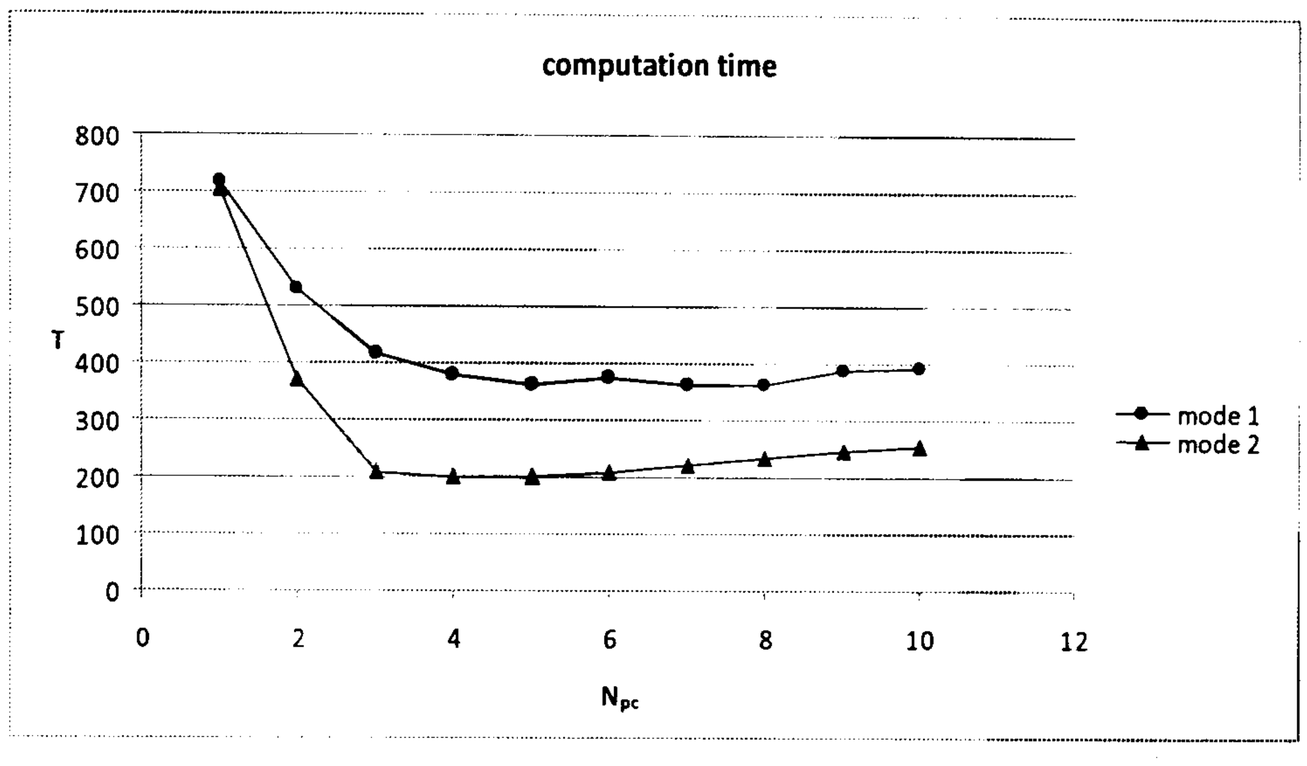
(ii) Benefit in speeding up by increasing number of workers

In theory, increasing the number of workers increases the computation capacity and decreases the computation time required to converge. Problems 1 and 2 described in section 7.2 are used to study the benefits when increasing the number of workers. Figure 8.9 illustrates the performance

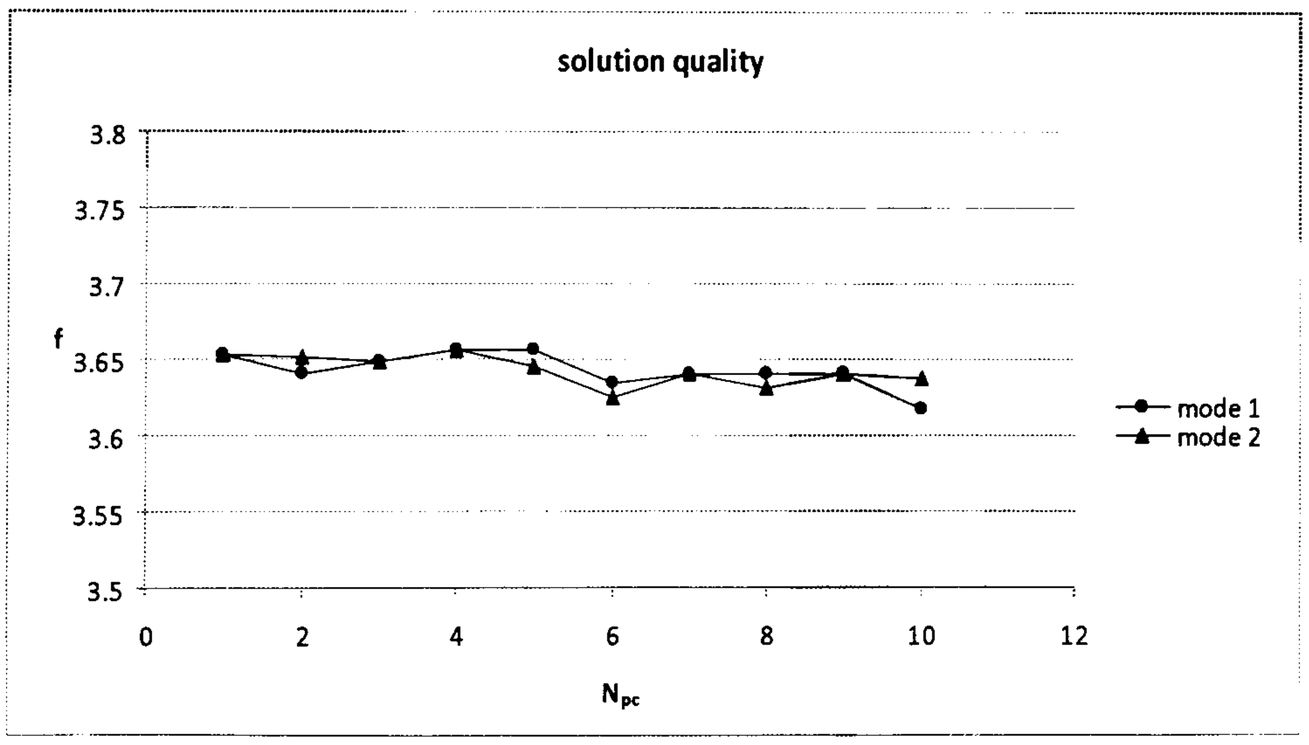
mode 1: the synchronous mode

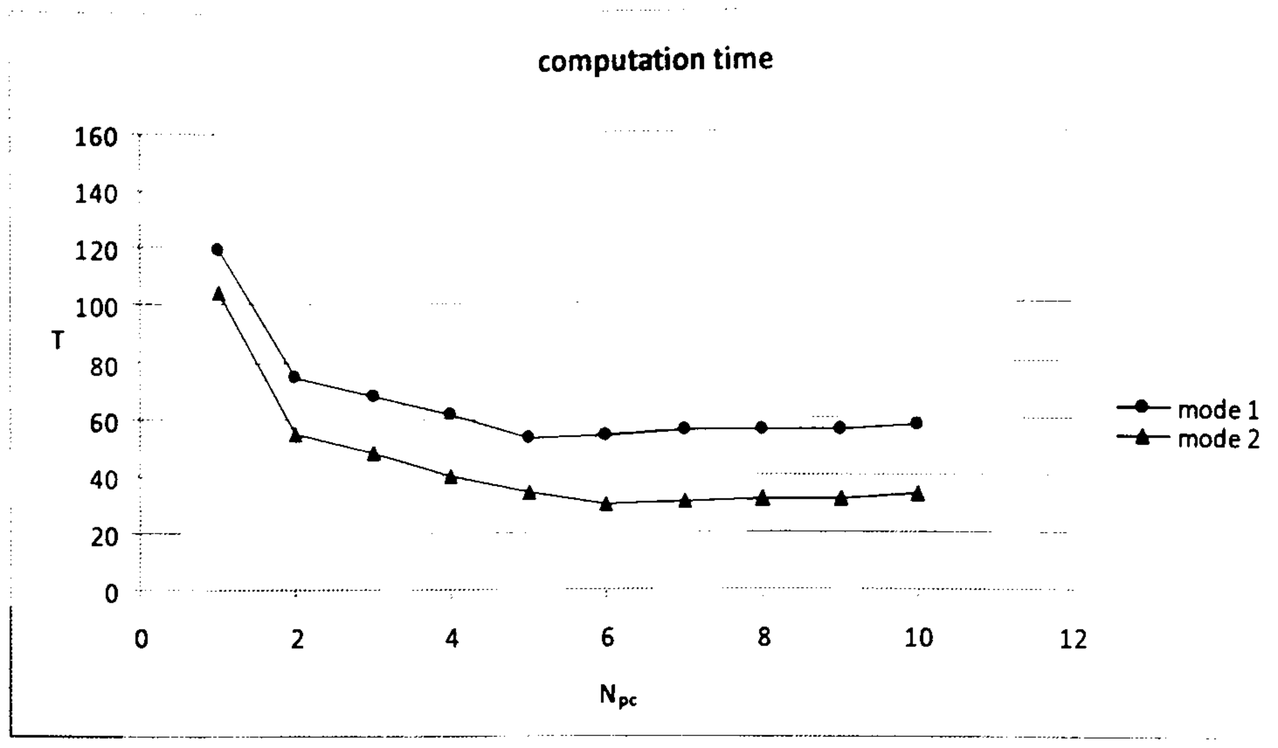
mode 2: the asynchronous mode


solution quality
f
N_pc
mode 1
mode 2


computation time
T
N_pc
mode 1
mode 2

(Problem 1)


solution quality
f
N_pc
mode 1
mode 2



(Problem 2)

Figure 8. 9 Performance with different numbers of workers

The results show that N_{pc} does not have a significant impact on the solution quality. The computation time quickly decreases and then increases slightly as N_{pc} increases. This is caused by the increasing communications overheads. The computation time now involves the communication time. With more and more workers, the communication time increases, resulting in the increased clock time. In addition, the asynchronous mode can converge more quickly than the synchronous mode. Let us take the results of problem 1 for example. The computation time for the synchronous mode is twice as long as that for the asynchronous mode when using three workers.

(iii) Comparison with other methods

Comparison between a parallel TS and COPT is performed to illustrate the advantage of COPT on distributed computing environments. Problem 3 described in section 7.2 is selected in this study. COPT is implemented as the asynchronous mode. The two methods run on the environment illustrated in Table 8.1. Referring to the studies by Ashley (2004), parameters and termination criteria of TS are fixed as:

- Tabu list that has a single entry is selected.

- Termination criteria are set at un-improving 50 iterations or maximum of 500 iterations.

Figure 8.10 outlines parallel TS.

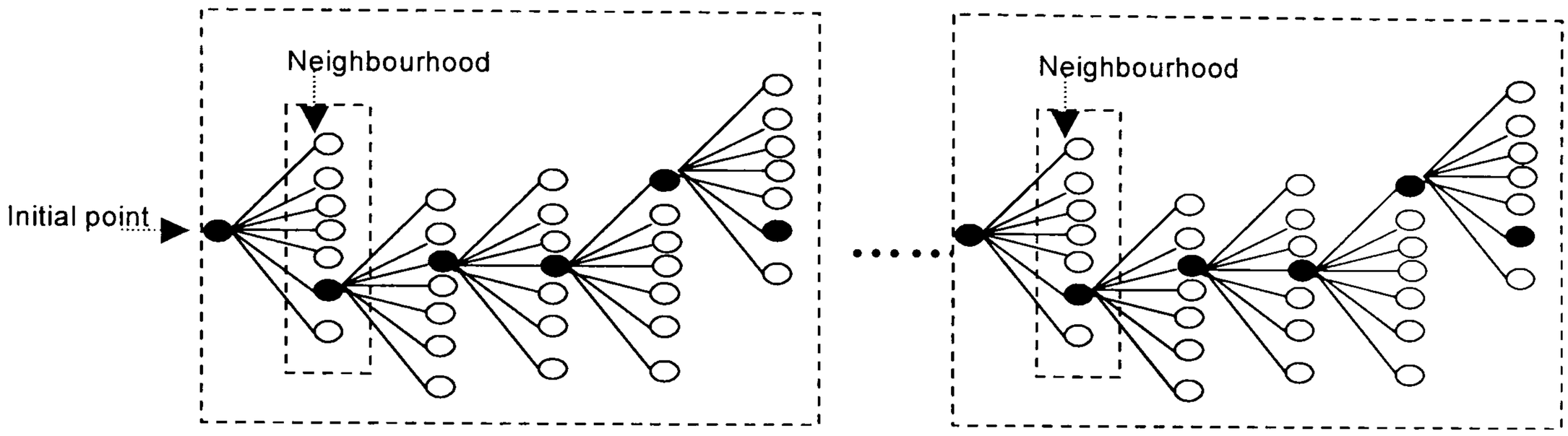


Figure 8. 10 Parallel TS

The parallel TS is implemented as a master-worker paradigm. It starts from an initial point (starting point). The master searches neighbours of this starting point and send them to workers for evaluation. Results are sent back to the master and the best neighbour is selected as a new starting point. The master searches neighbours of this starting point again and then sends them to workers for evaluations. The convergence of the algorithm is determined by the termination criteria. The performance of the algorithm is measured by the solution quality and computation time. The solution quality is the best objective ($F^{\max}$), whilst the computation time is represented by the clock time of the slowest worker. Figure 8.11 illustrates the comparison between parallel TS and COPT on a distributed computing environment.

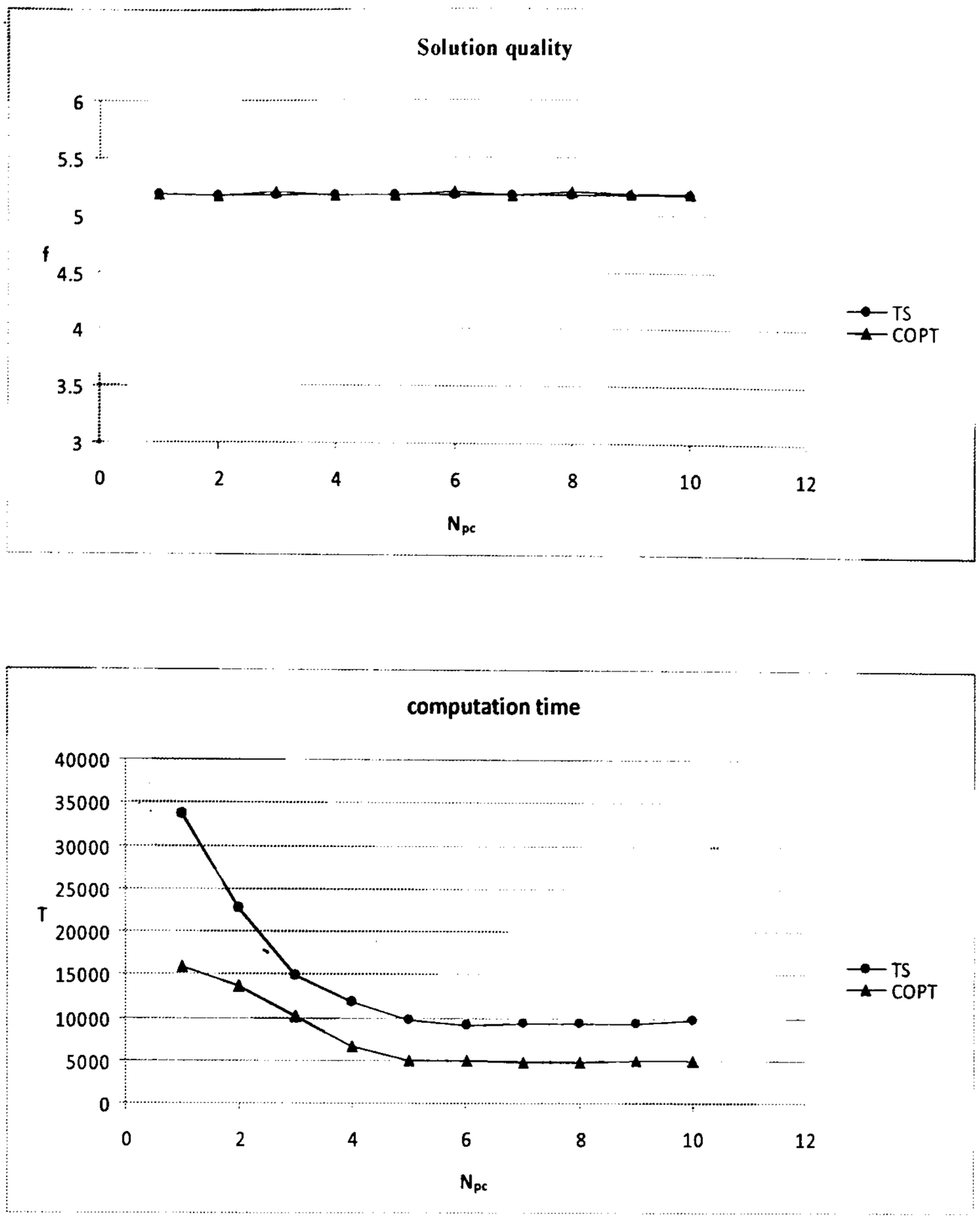


Figure 8. 11 Comparison between parallel TS and COPT

Both the parallel TS and COPT can converge to the solutions with similar qualities. Their objectives are within 5.196 ± 0.0132, but the computation time for COPT is much smaller than that for the parallel TS. This is mainly caused by the synchronous worker processes in the parallel TS, where the faster workers have to wait for the slower one in the algorithm. In contrast, workers are parallel and independent of each other in the asynchronous mode of COPT.

8.5 Remarks and discussions

Stochastic methods have been suffering from long computation time. Current research focuses on applying parallel and distributed computing techniques to speed up convergence. However, the applications are limited to small-scale computing

environments that have a small number of computers. With the parallel and independent tasks, COPT can be applied on the computing environments composed of a large number of distributed computers.

COPT has been implemented in synchronous and asynchronous modes. The asynchronous mode has workers running in parallel and independent of each others so that this mode can obtain the most benefit in speeding up convergence. COPT in both modes on a distributed computing environment was studied on problems 1 and 2. Results illustrate that both modes converge to similar solutions and that the computation time decreases as the number of workers increases. However, decrease in computation time is greater with asynchronous mode.

COPT on distributed computing environments was also studied on problem 3. Here results were compared with that of parallel TS and illustrate that COPT obtains more benefit in speeding up convergence when it is applied on parallel and distributed computing environments and therefore more fully exploits the computing potential of these environments.

Chapter 9 Conclusions and recommendation

9.1 Conclusion

Optimisation has been broadly applied to many fields. Classification of optimisation problems is based on their different characteristics. Based on the classification, many optimisation techniques have been developed, consisting primarily of deterministic optimisation techniques and stochastic optimisation techniques. The deterministic methods take advantage of mathematical and geometric techniques to solve the application problems. These methods can quickly converge to highly precise optimal solutions. However, the deterministic methods suffer from computationally intensive use of derivative transformations, inexistence of mechanisms to highly non-convex domains and difficulties in the problem initialisation. Compared to the deterministic methods, the stochastic methods follow a statistical random probabilistic driven search to find the optimal solutions. This randomness can help the stochastic methods converge to the global optimal solutions from the local optimal solutions. It also can speed up the convergence and make the algorithm less sensitive to the modelling errors. However due to the inherent randomness in search, the stochastic methods cannot quickly converge to the global optimal solutions. Advanced parallel and distributed computing techniques have been used to reduce the computational time. The previous research on the parallel and distributed applications of stochastic optimisation methods has been reviewed and it was found that the existing parallel stochastic optimisation methods cannot fully exploit computational resources in the computing environment because of the synchrony they require within computers.

A new stochastic optimisation method (COPT) was developed for the large-scale distributed computing environment. Such algorithm takes advantage of the Markov process, similar to that in SA, but eliminates its inherent sequential nature. COPT

introduces partitions and pools to store intermediate solutions and corresponding objective values. Populations in pools are inflected periodically to keep pools follow a specified order. Partitions and pools grow as the Markov process generates new intermediate solutions. With the partitions and pools, multi Markov processes can be launched simultaneously and COPT can be applied on parallel and distributed computing environments. Each pool is associated with a temperature decreasing from the top pool to the bottom pool. The Markov process started from a higher pool can accept more solutions than the one with the same length started from a lower pool. This temperature also helps COPT to inflect population of pools. COPT was implemented as a master-worker paradigm. Partitions and pools are implemented as the tables in database. In the course of the study, two different implementations were employed.

Several important features were studied on both the small-scale problems and the large-scale problems. The features involved the management of stochastic search, optimisation structure size, the depth of search, and the selection and coordination of the termination criteria. The termination criteria were selected related to controls in population size, optimisation progress, and distribution features. The performance is reflected on the solution quality and the computation time. The management of stochastic search is reflected on the distribution function. Based on the results, COPT with the concave distributions can achieve a better solution quality but at the expense of more computation time than that achieved with the convex distributions. The optimisation structure size is determined by the number of pools. Based on the results, increasing the number of pools increases the solution quality but meanwhile increases the computation time. The search depth is determined by the length of the Markov process. Based on these studies, the length of the Markov process does not have an apparent impact on the solution quality. On the other hand, increasing the length of the Markov process increases the computation time. Termination criteria related to the control in population size defines a minimum population and a maximum population. COPT does not terminate if the population of pools is smaller than the minimum population and terminate if the population of pools approaches the maximum population. Another termination criterion

related to the control in optimisation progress tests the best objective value in pools at each iteration. COPT terminates if the best objective value stay same for a number of iterations. The last termination criterion related to the controls in population features which involve pool level and cascade level. The pool level and cascade level are measured by the standard deviation of population of pools and the density function of domain respectively. With control in pool level, COPT terminates if the standard deviation stays same for a number of iterations. With control in cascade level, COPT terminates if the density function stays same for a number of iterations.

As the pools are associated with different temperatures, the Markov process follows different acceptance probabilities. The form of distributing acceptance probabilities in search can affect the performance of COPT. Search policies were proposed to determine the form and implemented by clustering the pools to the floating range, the middle range, and the settling range. Each of these ranges is associated with a probability of selecting pools within these ranges. Based on the results, COPT can converge to the good solutions but require a long computation time if the middle range or the settling range has the larger probability than the floating range.

Comparison with the stochastic methods was performed on a complex engineering problem. Results illustrated that COPT can converge to similar optimal solutions more quickly than the stochastic method. Finally, COPT was applied on the parallel and distributed computing environment. Some adjustments were needed on the implementation to observe the requirements of the parallel and distributed applications. In order to prove COPT's applicability on such computing environment, comparison with the performance of the parallel TS is selected. Results illustrated that in terms of convergence time COPT can outperform the parallel TS.

9.2 Originality

The major contribution of this project to optimisation is the development of COPT that is suitable for a large-scale distributed computing environment and meanwhile storing system knowledge in the additional memory. COPT offers the major improvements over existing optimisation techniques as follows:

(i) Performs a robust optimisation for complex problems. Achieves quicker convergence than the stochastic optimisation method

(ii) Able to be applied on the parallel and distributed computing environments and to fully exploit computational resources in these environments.

(iii) Records system knowledge in the additional memory and has the potential to incorporate knowledge acquisition in future work.

9.3 Recommendation of future work

COPT has been developed in this thesis to address the limitations in the applications of existing stochastic methods when applied upon the distributed computing environment. The method is still in the early stages of development and is expected to improve through future work. The attention of future work should focus on two views as follows:

(a) Distribute tasks to workers according to their capabilities

In COPT, the workers run the Markov processes of the same lengths. The slower workers would generate much less points than the faster workers and the computation resources are not effectively utilized. In order to solve this problem, we must :

(i) Distribute Markov processes of different lengths optimally amongst workers.

The Markov processes of different lengths should be employed and distributed to

workers according to their computation capabilities. The faster workers should take the longer Markov processes, while the slower workers should take the shorter Markov processes.

(ii) Distribute Markov processes of different partitions optimally amongst workers

Because of the usage of Metropolis criterion, the Markov processes launched from the higher pools are more easily generate points than those launched from the lower pools. To use the computation resources efficiently, the slower workers should take the Markov processes that are launched from the higher pools and the faster workers in contrast should take the Markov processes that are often launched from the lower pools.

a) Apply knowledge acquisition techniques

As system knowledge is stored in memory, the knowledge acquisition techniques, such as Data Mining and Ontology, can be utilized to assess the knowledge at run time and guide the search so that COPT can converge to the global optimal solution more rapidly. Currently, Du (2008) is developing the knowledge based system for COPT using Ontology software. The research has already produced some impressive results.

Chapter 10 Reference

Aarts, E. H. L., and van Laarhoven, P. G. M., (1985) Statistical cooling: a general approach to combinatorial optimisation problems. *Philips Journal of Research* , 40, 193.

Achenie, L. K. E., Biegler L. T., (1990) Superstructure based approach to chemical reactor networks synthesis, *Computer and Chemical Engineering, 22*: 1159-1179.

Androulakis, I. P., Maranas, C. D., Floudas, C. A., (1995) aBB: A global optimisation method for general constrained nonconvex problems, *Journal of Global Optimisation*, 7: 337-363.

Antonopoulos, N., Linke, P., and Kokossis, A. C., (2004) A prototype grid framework for the chemical process industries, *Chemical Engineering Communications*, 28 2391

Arsham, H. (1998) Techniques for Monte Carlo Optimising, Monte Carlo Methods and Applications, vol. 4, pp. 181-229.

Ashley, V. M., and Linke, P., (2004) A novel approach for reactor network synthesis using knowledge discovery and optimisation techniques. *Trans IchemE, Part A, Chemical Engineering Research and Design,* 82(A8): 952-960.

Ashley, V. M., (2004) On the development of knowledge driven optimisation methods – application to complex reactor network synthesis (Ph. D. thesis), UK, University of Surrey.

Badia R.M., Labarta J., Sirvent R., Perez J.M., Cela J.M. and Grima R., (2003) Programming Grid Applications with GRID Superscalar, *Journal of Grid Computing 1*: 151-170, CEPBA-IBM Research Institute, UPC, Spain.

Bailey, J.E., Ollis, D. F., (1986) Biochemical Engineering Fundamentals 2nd Ed, McGraw Hill, London.

Bakken, A. P., Hill Jr, C. G., Admundson, C. H.,(1989) Hydrolysis of lactose in skim milk by immobilised b-galactosidase in a spiral flow reactor, *Biotechnology and Bioengineering* 33: 1249-1275.

Baluja, S., (1994) Population-Based Incremental Learning: A Method for Integrating Genetic Search Based Function Optimization and Competitive Learning. *Carnegie Mellon University.*

Technical Report. CMU-CS-94-163.

Bender, J.F., (1962) Partitioning procedures for solving mixed-variable programming problems, *Numerische Mathmatik,* 4: 238-252.

Berman, F., Fox, G., Hey, T., (2003) The Grid: past, present future, In: *Grid Computing Making the Global Infrastructure a Reality,* pp. 9-50, *Wiley and Sons*. ISBN 0470853190.

Biegler, L. T., Grossman, I. E., Westerberg, A. W., (1997) Systematic Methods of Chemical Process Design, Prentice Hall, London.

Black P.E., (2005) greedy algorithm in Dictionary of Algorithms and Data Structures [online], *U.S. National Institute of Standards and Technology*, webpage: NIST-greedyalgo.

Bonabeau, E., Dorigo, M., and Theraulaz, G., (1999) Swarm Intelligence from Natural to Artificial Systems, *Oxford University Press*, New York.

Brooks, R.J., and Tobias, A.M., (1996) Choosing the best model: level of detail, complexity and model performance. *Mathematical and Computer Modelling*, 24(4), Pp 1-14.

Cantú-Paz, E., (1997) Designing Efficient Master-Slave Parallel Genetic Algorithms, *IlliGAL Technical Report No. 97004.*

Cantú-Paz, E., (1997) A survey of parallel GAs. IlliGAL R. 97003, A revised version of 1995. A summary of research on parallel genetic algorithms, TR#95007.

Cardoso, Salcedo, R. L., Feyo de Azevedo, S., and Barbosa, D., (2000) Optimisation of reactive distillation processes with simulated annealing, *Chemical and Engineering Science* 55, 5059.

Carroll, D L. (1996) Genetic algorithms and optimizing chemical oxygen-iodine lasers, *Developments in Theoretical and Applied Mechanics*, 18(3): 411–424

Cavin, L., Fischer, U., Glover, F., Hungerbuhler, K., (2004) Multi-objective process design in multi-purpose batch plants using a Tabu search algorithm, *Computer and Chemical Engineering 28*: 459-478.

Cerny, V. (1985) Thermodynamical approach to the traveling salesman problem: An efficient simulation approach. *Journal of Antonopoulos Optimisation Theory and Applications*, 45-1: 41-51.

Chakrapani, J., and Skorin-Kapov, J., (1993) Massively Parallel Tabu Search for the Quadratic Assignment problem, *Annals of Operations Research* V41, 327-341.

Chaudhuri, P. D., and Diwekar, U. M., (1996) Process synthesis under uncertainty: a penalty function

approach. *American Institute of Chemical Engineering Journal 42* (3), 742.

Chaudhuri, P. D., and Diwekar, U. M., (1997) Synthesis under uncertainty with simulators, *Computers and Chemical Engineering* 21 (7), 733.

Chen, H., and Flann, S. N., (1994) Parallel Simulated Annealing and Genetic Algorithms: a Space of Hybrid Methods. PPSN 1994: 428-438.

Chwif, L., Barretto, M. R. P., Santoro, M. C., (1998) Model reduction: Some results, *Proceedings of the The 31st Annual Simulation Symposium,* Washington, DC, USA.

Cordero, J. C., Davin, A., Floquet, P., Pibouleau, L., Domenech, S., (1997) Synthesis of optimal reactor networks using mathematical programming and simulated annealing, *Computer and Chemical Engineering, 21*: S47.

Cox, A., and Rajamony, R., Parallel Programming Tools, *Encyclopedia of Electrical and Electronics Engineering*, pp 111-123.

Crainic, T. G., Toulouse, M., and Gendreau, M., (1995a) Synchronous Tabu Search Parallelization Strategies for Multicommodity Location-Allocation with Balancing Requirements. *OR Spektrum* 17(2/3), 113-123.

Crainic, T. G., Toulouse, M., and Gendreau, M., (1995b). Parallel Asynchronous Tabu Search for Multicommodity Location-Allocation with Balancing Requirements. *Annals of Operations Research* 63, 277-299.

Crainic, T. G., Toulouse, M., and Gendreau, M., (1997) Towards a Taxonomy of Parallel Tabu Search Algorithms, *INFORMS Journal on Computing* 9(1), 61-72.

Das, H., Cummings, P. T., and LeVan, M. D., (1990) Scheduling of serial multiproduct batch processes via simulated annealing. *Computers and Chemical Engineering* 14 (12), 1351.

Dixon, L. C. W., Szeco, G. P. (1978) (EDs.) Toward Global Optimization 2, North-Holland, Amsterdam.

Dixon, L. C. W., Szeco, G. P., (1975) (EDs.) Toward Global Optimization. North-Holland, Amsterdam.

Dolan, W. B., Cumming, P. T., LeVan, M. D., (1990) Algorithmic efficiency of simulated annealing for heat exchanger networks. *Computer and Chemical Engineering, 14*: 1039.

Dolan, W. V., Cummings, P. T., and LeVan, M. D., (1989) Process optimisation via simulated annealing: application to network design. *American Institute of Chemical Engineering*

Journal 35 (5), 725.

Dolan, W. V., Cummings, P. T., and LeVan, M. D., (1990) Algorithmic efficiency of simulated annealing for heat exchanger networks. *Computer and Chemical Engineering* 14 (10), 1039.

Dorigo, M. (1992) Optimization, Learning and Natural Algorithms (PhD thesis), DEI, Politechnico di Milano, Italy.

Dorigo, M., Di Caro G., and Gambardella, L. M., (1999) Ant Algorithms for Discrete Optimization. *Artificial Life,* 5 (2): 137–172.

Duran, M. A., Grossman, I. E., (1986) An outer-approximation algorithm for a class of mixed-integer nonlinear programs, *Mathematical Programming,* 36: 307-339.

Du, D., Yang, S., Kokossis. A. C., and Linke, P., (2007) Experience on gridification and hyperinfrastructure experiments in optimization and process synthesis, *17th European Symposium on Computer Aided Process Engineering*, 27-30 May Bucharest, Romania: Elsevier, 7-9

Dzeroski, S., Grbovic, J., Walley, W., Kompare, B., (1997) Using machine learning techniques in the construction of models. II. Data analysis with rule induction, *Ecological Modelling*, 95: 95-111.

Cantú-Paz, Erick. 1998. A survey of parallel genetic algorithms. *Calculateurs Paralleles.* Vol. 10, No. 2. Paris: Hermes.

Fayyad, U. M., Piatetsky-Shapiro, G., and Smyth, P., (1996) From Data Mining to Knowledge Discovery: An Overview, In *Advances in Knowledge Discovery and Data Mining*, eds. U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy,1–30. Menlo Park, Calif.: AAAI Press.

Floquet, P., Pibouleau, L., Domenech, S., (1994) Separation sequence synthesis: how to use simulated annealing procedure, *Computer and Chemical Engineering 18*: 1141.

Floquet, P., Pibouleau, L and Domenech, S., (1994). Separation sequence synthesis-how to use simulated annealing procedure, *Computers and Chemical Engineering 18*, 1141.

Foster, I., and Kesselman, C., (1997) Globus: a metacomputing infrastructure toolkit, *International Journal of Supercomputer Applications, 11*: 115-128.

Foster, T., Gillespie, K., McClelland, R., *et al* (1999) Risk factors for suicide independent of DSM—III—R Axis I disorder. Case—control psychological autopsy study in Northern Ireland. *British Journal of Psychiatry*, 175, 175-179.

Foster, I., Kesselman, C., Tuecke, S., (2001) The anatomy of the Grid: enabling scalable virtual organizations, *International Journal of Supercomputer Applications and High Performance Computing, 15*: 200-222.

Fouskakis, D. and Draper, D. (2002) Stochastic Optimization: A Review, *International Statistical Review*, vol. 70, pp. 315-349.

Frantz, F. K., (1995). A taxonomy of model abstraction techniques, In: *Proceedings of the1995 Winter Simulation Conference,* ed. Institute of Electrical and Electronics Engineers Piscataway, New Jersey.

Fu, M.C. (2002), Optimization for Simulation: Theory vs. Practice (with discussion by S. Andradóttir, P. Glynn, and J.P. Kelly), *INFORMS Journal on Computing*, vol. 14, pp. 192-227.

Gambardella, L. M., Taillard, E. D., and Agazzi, G., (1999) MACS-VRPTW: A multiple ant colony system for vehicle routing problems with time windows. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 63–76. McGraw Hill, London, UK.

Geoffrion, A. M., (1972) Generalised Bender decomposition, *Journal of Optimisation Theory and Applications,* 10: 237-260.

Giorno, L., Drioli, E., (2000) Biocatalytic membrane reactors: applications and perspectives, *Trends in Biotechnology, 18*:339-349.

Glover, F., (1989) Tabu search- Part I, *ORSA Journal of Computing*, 1: 190.

Glover, F., (1990) Tabu search- Part II, *ORSA Journal of Computing*, 2: 4.

Glover, F., (1993) A user's guide to Tabu search, *Annals in Operational Research*, 41: 3.

Glover, F., Taillard, E., Laguna, M., de Werra, D., (1993) Tabu search, *the Annals of Operations Research,* vol. 41.

Goldberg, D. E., Deb, K., and Clark, J. H., (1992) Genetic Algorithms, Noise, and the Sizing of Populations, in: *Complex Systems*, *Complex Systems Pub., Inc.,* vol. 6 pp. 333-362.

Gosavi, A., (2003) Simulation-Based optimisation: Parametric Optimisation techniques and Reinforcement Learning, *Kluwer Academic, Norwell, MA.*

Hanke. M., and Li, P. (2000). Simulated annealing for optimisation of bath distillation processes. *Computers and Chemical Engineering* 24 (1), 1.

Innis G. and Rexstad E. 1983 Simulation model simplification techniques, *Simulation*, 41(1), Pp7-15.

Iosifescu, M., Scutaru, H., On six-dimensional canonical realisations of the so (4, 2) algebra. *J. Math*

Phys. 21 (1980) 2033-2045.

Jayaraman, V. K., Kulkarni, B. D., Karale, S., Shelokar, P. (2000). Ant colony framework for optimal design and scheduling of batch plants, *Computers and Chemical Engineering, 24*(8). 1901.

Karush, W., (1939). *Minima of Functions of Several Variables with Inequalities as Side Constraints*. M.Sc. Dissertation. Dept. of Mathematics, Univ. of Chicago, Chicago, Illinois.

Karlin, S., and Taylor, H. M., (1975). A First Course in Stochastic Processes, 2nd ed., *Academic Press, New York*.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P., (1983) Optimisation by simulated annealing, *Science* 220, 671.

Klee V. and Minty G.J. (1972). How good is the simplex algorithm?. In: Shisha, O. (eds) Inequalities III, pp 159–175. Academic, New York

Kokossis, A. C., Floudas, C. A., (1990) Optimisation of complex reactor network –I. Isothermal operation, *Chemical Engineering Science,* 45(3): 595-614.

Kruger, F., Merkle, D., and Middendorf, M., (1998) Studies on a Parallel Ant System for the BSP Model; Unpub. Manuscript.

Lei, F., Jorgensen, S. B., (2001) Estimation of kinetic parameters in a structured yeast model using regularisation. *Journal of Biotechnology,* 88: 223-237.

Lei, F., Rotboll, M., Jorgensen, S. B., (2001) A biochemically structured model for Saccharomyces cerevisiae, *Journal of Biotechnology,* 88: 205-211.

Leite, J.P.B., and Topping, B. H. V., (1999) Parallel simulated annealing for Structural Optimisation, *Computers & Structures,* 73 (1-5):545-564.

Leitold, A., Hangos, K.M., Tuza, Zs.: Structure simplification of dynamic process Models. *Computers and Chemical Engineering,* 25: pp. 1633-1646 (2001)

Liang, Y., Gan, F., (2001) 'Chemical knowledge discovery from mass spectral database. I. sotope distribution and Beynon table', *Analytica Chimica Acta*, 446: 115-120.

Lin, B., Miller. D. C., (2004). Tabu search algorithm for chemical process optimisation, *Computer and Chemical Engineering 28*: 2287-2306.

Linke, P., and Kokossis. A. C., (2003a) On the robust application of stochastic optimisation technology for the synthesis of reaction/separation systems, *Computers and Chemical Engineering,* 27: pp 733-758.

Linke, P., and Kokossis, A. C., (2003b) Attainable reaction and separation processes from a superstructure-based method, *AIChE Journal,* 49: 1451-1470.

Luna, J. J., (1993) Hierarchical relation in simulation models, *Winter Simulation Conference 1993*: 132-137

Mahfoud, S. W., and Goldberg, D. E., (1992) A Genetic Algorithm for Parallel Simulated Annealing. *PPSN* 1992: 303-312.

Maia, L. O. A., Vidal de Carvalho, L. A., and Qassim, R. Y. (1995) Synthesis of utility systems by simulated annealing, *Computer and Chemical Engineering* 19 (4), 481.

Marcoulaki, E. C., and Kokossis, A. C., (1999) Screening and scooping complex reaction networks using stochastic optimisation, *American Institute of Chemical Engineering Journal* 45 (9), 1977.

Marcoulaki, E. C., and Kokossis, A. C., (2000) On the development of novel chemicals using a systematic synthesis approach. I. Optimisation framework, *Chemical and Engineering Science 55* (13), 2529.

Mariusz, N., and Riccardo, P., (1999) Dynamic demes parallel genetic algorithm Third International Conference on Knowledge-Based Intelligent Information Engineeing Systems, 31" Aug- I" Sept 1999, Adelaide, Australia.

Mehta, V. L., and Kokossis, A. C., (1997) Development of novel multiphase reactors using a systematic design framework, *Computer and Chemical Engineering* 21, S325.

Mehta, V. L., and Kokossis, A. C., (1998) New generation tools for multiphase reaction system: A validated, systematic methodology for novelty and design automation, *Computer and Chemical Engineering*, 22:S119.

Mehta, V. L., and Kokossis, A. C., (2000) Nonisothermal Synthesis of Homogeneous and Multiphase Reactor Networks, *AIChE Journal 46,* 2256.

Metropolis, N., Rosenbluth A., Rosenbluth M., Teller A., and Teller E., (1953) Equation of calculations by fast computing machines. *J. Chemical Physics*, 21, 1087.

Michalewicz, Z., and Attia, N., (1994) Evolutionary optimisation of constrained problems, *World Scientific Publishing*, *River Edge*, NJ, pp.98-108.

Nandi, S., Badhe, Y., Lonari, J., Sridevi, S., Rao, B. S., Tambe, S. S., Kulkarni, B. D.,(2004) Hybrid process modeling and optimization strategies integrating neural networks / support vector

regression and genetic algorithms: study of benzene isopropylation on Hbeta catalyst', *Chemical Engineering Journal*, 97: 115-129.

Painton, L. A., and Diwekar, U. M., (1995) Stochastic annealing for synthesis under uncertainty. *European Journal of Operations Research* 83, 489.

Papadopoulos, A. I., Linke, P., (2004) On the synthesis and optimisation of liquid-liquid extraction process using stochastic search method, *Computer and Chemical Engineering 28*: 2391-2406.

Papadopoulos, A. I., Ashley, V. M., Linke, P., (2005) Grid computing in integrated computing-aided solvent and process design, *In proceeding of 7th world congress in chemical engineering (WCCE 7), Glasgow, U.K.*

Patel, A. N., Mah, R. S. H., and Karimi, I. A., (1991) Preliminary design of multiproduct noncontinuous plants using simulated annealing, *Computer and Chemical Engineering 15* (7), 451.

Pham, D. T., and Wagner, M., (1998) A geostatistical model for linear prediction analysis of speech. *Pattern Recognition 31*(12): 1981-1991 (1998)

Piriyakumar, D.A.L., and Levi P., (2002) A New Approach to Exploiting Parallelism in Ant Colony Optimisation, *International Symposium on Micromechatronics and Human Science*(MHS), pp7.

Pronzato, L., Walter, E., Venot, A, Lebruchec, J. F. A., (1980) general purpose global optimizer: *Implementation and applications. Math. Comput. Simul.* 26, pp 412-422.

Rajesh, J., Gupta, K., Kusumakar, H. S., Jayaraman, V. K., Kulkarni, B. D. (2001) Dynamic optimization of chemical processes using ant colony framework. *Computers and Chemistry, 25*(6), 583.

Raman, R., Grossman, I. E., (1994) Modelling and computational techniques for logic based integer programming, *Computing and Chemical Engineering*, 18:563-578.

Randall, M., and Lewis, A., (2002) A parallel implementation of ant colony optimisation, *Journal of Parallel and Distributed Computing*, 62(9): 1421-1432

Robinson, S. (1994) Simulation Projects: Building the Right Conceptual Model. *Industrial Engineering*, 26 (9). 34--36.

Romeo, F., A. Sangiovanni Vincentelli , And Sechen C. (1984) Research on simulated annealing at Berkeley. *In Proceedings of the IEEE International Conference on Computer Design, ZCCD*

84,IEEE New York, pp 652-657.

Shor, N. Z. , (1970) Utilization of the operation of space dilatation in the minimization of convex functions, Kibernetica, v6, pp. 6-12.

Schittkowski, K., (1985) NLQPL: A FORTRAN-Subroutine Solving Constrained Nonlinear Programming Problems, *Annals of Operations Research*, Vol. 5, pp 485-500.

Simon, H. A. (1964) The architecture of complexity, *General Systems Yearbook*, 10: 63-76.

Spall, J.C. (2003) Introduction to Stochastic Search and Optimisation: Estimation. Simulation, and Control, *Wiley, Hoboken*, NJ. Pp 1024-1025.

Trivedi, K.S., (1982) Probability and Statistics with Reliability, Queueing, and Computer Science Applications. *Prentice-Hall, Englewood Cliffs*, NJ.

Two Crows Corporation (1999) Introduction to Data Mining and Knowledge Discovery (Third Edition), Two Crows Corporation.

Venkatasubramanian, V., Rengaswamy, R., Kavuri, S.N., and Yin K., (2003) A review of process fault detection and diagnosis part II: qualitative models and search strategies. *Computers and Chemical Engineering* 27: 313-326

Wang, C., Quan, H., and Xu, X., (1999) Optimal design of multiproduct batch chemical processes using Tabu Search, *Computer and Chemical Engineering* 23, 427.

Webster et al., (1984) Determining the level of detail in a simulation model - A case study, *Computers and Industrial Engineering*, 8(3/4): 215-225.

Appendix A

The four small-scale optimisation problems are illustrated as follows:

Test problem 1

$$\min\left\{(0.00533\cdot \mathrm{x1}^2 + 11.669x1 + 0.00889\cdot \mathrm{x2}^2 + 10.333x2 + 0.00741\cdot \mathrm{x3}^2 + 10.833x3)\right\}$$

$$-\left(\begin{array}{l} 0.01\times\left(\begin{array}{l} 0.0676\mathrm{x1}^2 + 0.00953x1x2 - 0.00507x1x3 + 0.00953x2x1 + 0.0521x2^2 + 0.00901x2x3 \\ 0.00507x3x1 + 0.00901x3x2 + 0.0294x3^2 \end{array}\right) \\ -0.000766x1 - 3.42e-5*x3 \end{array}\right) + x4$$

$$= 0.040357$$

$$x1 + x2 + x3 + x4 \geq 210$$

Optimal objective is 3155.258

Test problem 2

$$\min\left\{\frac{(0.0039\times \mathrm{x7} + 0.0039\times \mathrm{x8})\times(495\times \mathrm{x4} + 385\times \mathrm{x5} + 315\times \mathrm{x6})}{\mathrm{x10}}\right\}$$

subject to

$$-0.5\times \mathrm{x9}\times \mathrm{x4}\times(0.8\times \mathrm{x7} + 0.333333333333333\times \mathrm{x8}) + \mathrm{x1} = 0$$

$$-0.5\times \mathrm{x9}\times \mathrm{x5}\times(0.8\times \mathrm{x7} + 0.333333333333333\times \mathrm{x8}) + \mathrm{x2} = 0$$

$$-0.5\times \mathrm{x9}\times \mathrm{x6}\times(0.8\times \mathrm{x7} + 0.333333333333333\times \mathrm{x8}) + \mathrm{x3} = 0$$

$$\sqrt{\mathrm{x10} - \mathrm{x7}} - (\sqrt{\mathrm{x8}} - \sqrt{\mathrm{x9}}) \geq 0$$

$$\mathrm{x1} - 8.465273475\times \mathrm{x10} \geq 0$$

$$\mathrm{x2} - 9.650065104166667\times \mathrm{x10} \geq 0$$

$$\mathrm{x3} - 8.871679687475\times \mathrm{x10} \geq 0$$

$$0.5\times \mathrm{x1}\times \mathrm{x9} - 2.2\times(8.4652734375\times \mathrm{x10})^{1.33333333333333} \geq 0$$

$$0.5\times \mathrm{x2}\times \mathrm{x9} - 2.2\times(9.650065104166667\times \mathrm{x10})^{1.33333333333333} \geq 0$$

$$0.5\times \mathrm{x3}\times \mathrm{x9} - 2.2\times(8.871679687475\times \mathrm{x10})^{1.33333333333333} \geq 0$$

$$\mathrm{x4} - 0.011177174788380 1\times \mathrm{x7} \geq 0.2$$

$$\mathrm{x5} - 0.013765536041142 7\times \mathrm{x7} \geq 0.2$$

$$\mathrm{x6} - 0.015566387225364 8\times \mathrm{x7} \geq 0.2$$

$$\mathrm{x4} - 0.011177174788380 1\times \mathrm{x8} \geq 0.2$$

$$\mathrm{x5} - 0.013765536041142 7\times \mathrm{x8} \geq 0.2$$

$$\mathrm{x6} - 0.015566387225364 8\times \mathrm{x8} \geq 0.2$$

Optimal objective is 5.541

Test problem 3

$$\min\left\{\begin{array}{l}(x1\times(\log(x1/x11)-6.05576803624071)+x2\times(\log(x2/x11)-17.1307680362407)\\+x3\times(\log(x3/x11)-34.0207680362407)+x4\times(\log(x4/x11)-5.8807680362407)\\+x5\times(\log(x5/x11)-24.6877680362407)+x6\times(\log(x6/x11)-14.9527680362407)\\+x7\times(\log(x7/x11)-24.0667680362407)+x8\times(\log(x8/x11)-10.6747680362407)\\+x9\times(\log(x9/x11)-26.6287680362407)+x10\times(\log(x10/x11)-22.1447680362407)\end{array}\right\}$$

$$x1+2\times x2+2\times x3+x6+x10=2$$

$$x4+2\times x5+x6+x7=1$$

$$x3+x7+x8+2\times x9+x10=1$$

$$-x1-x2-x3-x4-x5-x6-x7-x8-x9-x10+x11=0$$

$$x1,x2,x3,x4,x4,x6,x7,x8,x9\geq 0.001$$

Optimal objective is -0.675

Test problem 4 (TP4)

$$\min\{x13+x14+x15+x16+x17+x18\}$$

$$(x1-x2)^2+(x7-x8)^2\leq 1$$

$$(x1-x3)^2+(x7-x9)^2\leq 1$$

$$(x1-x4)^2+(x7-x10)^2\leq 1$$

$$(x1-x5)^2+(x7-x11)^2\leq 1$$

$$(x1-x6)^2+(x7-x12)^2\leq 1$$

$$(x2-x3)^2+(x8-x9)^2\leq 1$$

$$(x2-x4)^2+(x8-x10)^2\leq 1$$

$$(x2-x5)^2+(x8-x11)^2\leq 1$$

$$(x2-x6)^2+(x8-x12)^2\leq 1$$

$$(x3-x4)^2+(x9-x10)^2\leq 1$$

$$(x3-x5)^2+(x9-x11)^2\leq 1$$

$$(x3-x6)^2+(x9-x12)^2\leq 1$$

$$(x4-x5)^2+(x10-x11)^2\leq 1$$

$$(x4-x6)^2+(x10-x12)^2\leq 1$$

$$(x4-x6)^2+(x11-x12)^2\leq 1$$

$$-0.5\times(x1x8-x7x2)+x13=0$$

$$-0.5\times(x2x9-x8x3)+x14=0$$

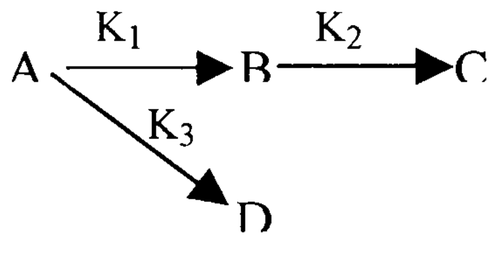
Optimal objective is -47.707

Appendix B

The brief descriptions of the Van de Vusse reaction, Lactose production, and the Biocatalytic (Saccharomyces Cerevisiae) schemes are illustrated as follows:

Van de Vusse

The Van de Vusse reaction scheme consists of a combination of parallel and serial reactions (Marcoulaki and Kokossis, 1999; Kokossis and Floudas, 1990). The general idea of Van de Vusse reaction is:



$$r_1 = k_1 C_A \qquad \mathrm{K}_1 = 10.0\,\mathrm{s}^{-1}$$

$$r_2 = k_2 C_B \qquad \mathrm{K}_2 = 1.0\,\mathrm{s}^{-1}$$

Where

A, B, and C are components,

C_A, C_B, C_C are the concentrations of these components,

r_1, r_2, r_3 are reaction rates of reaction 1, 2, and 3.

In Van de Vusse reaction, the feed flow is 100 L/s and contains pure component A, The objective of this reaction is to maximize the outlet concentration of B and meanwhile minimize that of C. The optimal structure for Van de Vusse consists of a CSTR (or an equally distributed DSSR) followed by serials of PFR, which has been proved by Kokossis and Floudas (1990) and Marcoulaki and Kokossis (1999).

Lactose Hydrolysis

Another reaction problem is the hydrolysis of lactose by β-galactosidase. In this reaction, immobilized enzyme is utilized to convert the disaccharide lactose via hydrolysis into its monosaccharide component, glucose and galactose. This procedure of hydrolysis can be approximated with following model (Marcoulaki and Kokossis,1999; Bakken et al., 1989; Bailey and Ollis, 1986).

$$R_1\begin{cases} E + S \leftrightarrow ES \rightarrow E + GL + \beta G \\ E + \alpha G \leftrightarrow E_{\alpha G} \\ E + \beta G \leftrightarrow E_{\beta G} \end{cases} \qquad r_1 = C_E C_S^* (1 + C_{\alpha G}/K_{\alpha G} + C_{\beta G}/K_{\beta G})^{-1}$$

$$K_{\varepsilon G} = 0.003, \quad K_{\beta G} = 0.079$$

$$R_2 \quad \beta G \leftrightarrow \alpha G \qquad r2 = 50^* (C^*_{\alpha G} - C_{\alpha G}),$$

$$C_{aG} = (C_S^0 + C_{\alpha G}^0 + C_{\beta G}^0 - C)^* K/(K+1)$$

$$K = 0.111$$

$$R_3 \quad GL \rightarrow GA \qquad r_3 = 0.083\, C_{GL}$$

$$R_4 \quad E \rightarrow Ed \qquad r_4 = 0.047\, C_E$$

Where

E: enzyme (β-galactosidase),

S: lactose,

GL: glucose,

G : galactose with α and β forms

GA: gluconic acid,

Ed: deactivated enzyme,

r_1, r_2, r_3, and r_4: the reaction rates for four reactions (R_1, R_2, R_3 , and R_4).

The feed flow is 100L/min that includes 0.65 mol L^{-1} enzyme, 0.001mol L^{-1} α galactose and 0.001 mol L^{-1} β galactose. The objective is to maximize the outlet concentration of GL. The optimal structure of reactor network for this reaction problem is a series of PFRs,

which is in the agreement with published results (Marcoulaki and Kokossis, 1999).

Biocatalytic problem

The usage of biocatalyst can date back to thousands years ago in the production of beer and wine as well as food such as bread and cheese. Recently biocatalyst has been broadly used to pharmaceutical and agrochemical industries. Biocatalytic process is advantageous of mild temperature condition, low energy requirement, safety, pollution prevention, high selectivity and often-high product quality (Giorno and Drioli, 2000). However, the sensitivity of organism and enzyme to conditions, contamination, and motion makes the Biocatalytic process difficult to control. Furthermore, the complexity of Biocatalytic process makes its model highly non-linear and difficult to be optimised. In the study of cascade optimisation algorithm, we select one of Biocatalytic problem (Saccharomyces Cerevisiae) to demonstrate the applicability of the cascade optimisation to complex problems. Saccharomyces cerevisiae is a yeast organism used throughout food and drink industries. A number of catabolic and anabolic reactions are used for cell metabolism and biomass production starting from a substrate of glucose. The key interest of these reactions is to optimise and explore the condition under which ethanol is produced. The kinetic mode of Saccharomyces Cerevisiae as follows:

$$S_{ght} \xrightarrow{1} S_{pyr} + 0.33NADH$$
$$S_{pyr} \xrightarrow{2} CO_2 + 1.67NADH$$
$$S_{pyr} \xrightarrow{3} 0.67S_{acetald} + 0.33CO_2$$
$$S_{acetald} \xrightarrow{4} S_{acetate} + 0.5NADH$$
$$S_{acetate} \xrightarrow{5} CO_2 + 2NADH$$
$$S_{acetald} + 0.5NADH \xrightarrow{6} S_{EtOH}$$
$$S_{ght} \xrightarrow{7} 0.913X_a + 0.087CO_2 + 0.119NADH$$
$$S_{acetate} \xrightarrow{8} 0.778X_a + 0.222CO_2 + 0.401NAD$$

$$X_a \xrightarrow{9} X_{Acdh}$$

$$X_a \xrightarrow{10} \deg radation$$

$$X_{Acdh} \xrightarrow{11} \deg radation$$

$$NADH + 0.5O_2 \xrightarrow{12} ATP$$

Where

S_{ght} : intracellular glucose concentration,

S_{pyr} : intracellular pyruvate concentration,

$S_{acetald}$: intracellular acetaldehyde concentration,

$S_{acetate}$: acetate concentration,

S_{EtOH} : intracellular ethanol concentration,

x: biomass concentration,

X_a : percentage of biomass that is active cell material,

X_{Acdh} : proportion of activity of the protein caused by the enzyme Acetaldehyde dehydrogenase,

$NADH$: nicotinamide adenine dinucleotide,

ATP: adenosine triphosphate.

Except for reaction 12 in which $NADH$ immediately generate ATP if sufficient oxygen is present, reaction rates for the remaining 11 reactions are:

$$r_1 = k_{1l}\frac{s_{glu}}{s_{glu}+K_{1l}}X_a + k_{1h}\frac{s_{glu}}{s_{glu}+K_{1h}}X_a + k_{1e}\frac{s_{glu}}{s_{glu}(K_{1i}s_{acetald}+1)+K_{1e}}s_{acetald}X_a$$

$$r_2 = k_2\frac{s_{pyr}}{s_{pyr}+K_2}\frac{1}{K_{2i}s_{glu}+1}X_a$$

$$r_3 = k_3\frac{{s_{glu}}^4}{{s_{glu}}^4+K_3}X_a$$

$$r_4 = k_4\frac{s_{acetald}}{s_{acetald}+K_4}X_aX_{Acdh}$$

$$r_5 = k_5\frac{s_{acetate}}{s_{acetate}+K_5}X_a + k_{5e}\frac{s_{acetate}}{s_{acetate}+K_{5e}}\frac{1}{1+K_{5i}s_{glu}}X_a$$

$$r_6 = k_6\frac{s_{acetald}-k_{6r}s_{EtOH}}{s_{acetald}+K_6+K_{6e}s_{EtOH}}X_a$$

$$r_7 = k_7\frac{s_{glu}}{s_{glu}+K_7}X_a$$

$$r_8 = k_8\frac{s_{acetate}}{s_{acetate}+K_{5e}}\frac{1}{1+K_{5i}s_{glu}}X_a$$

$$r_9 = \left(k_9\frac{s_{glu}}{s_{glu}+K_9} + k_{9e}\frac{s_{EtOH}}{s_{EtOH}+K_{9e}}\right)\frac{s_{acetate}}{K_{9i}s_{glu}+1}X_a + k_{9c}\frac{s_{glu}}{s_{glu}+K_9}X_a$$

$$r_{10} = k_{10}\frac{s_{glu}}{s_{glu}+K_{10}}X_a + k_{10e}\frac{s_{EtOH}}{s_{EtOH}+K_{10e}}X_a$$

$$r_{11} = k_{11}X_{Acdh}$$

where

r_i : the reaction rate of reaction i.

The kinetic constants are shown in the following table.

Kinetic constants for Saccharomyces cerevisiae

Constant	Value	Constant	Value	Constant	Value
k_{1h}	0.584	k_4	4.80	K_7	0.0101
K_{1h}	0.0116	K_4	0.000264	k_8	0.589

k_{1l}	1.43	k_5	0.0104	k_9	0.008
K_{1l}	0.94	K_5	0.0102	K_9	$1.0\text{x}10^{-6}$
k_{1e}	47.1	k_{5e}	0.775	k_{9e}	0.0751
K_{1e}	0.12	K_{5e}	0.10	K_{9e}	13
K_{1i}	14.2	K_{5i}	440	K_{9i}	25
k_2	0.501	k_6	2.82	k_{9c}	0.00399
K_2	0.002	K_6	0.034	k_{10}	0.392
K_{2i}	0.101	k_{6r}	0.0125	K_{10}	0.0023
k_3	5.81	K_{6e}	0.057	k_{10e}	0.00339
K_3	5.0x10-7	k_7	1.203	K_{10e}	0.0018
				k_{11}	0.02

The detailed explanation of this model is written in the literature (Lei *et al.* 2001). The feed flow involves: glucose 14 g/s, ethanol 0.13 g/s, biomass X 0.002 g/s, *Xa* 0.1 g, *XAcdh* 0.0075 g, and water 984.86 g/s (Lei and Jorgensen, 2001). The objective of Saccharomyces Cerevisiae reaction problem is the production of ethanol. Ashley (2002) studied this Biocatalytic system using the superstructure optimisation along with both a numerical optimisation algorithm and TS. Similar optimal structures are reported which are the combinations of plug flow and mixing.