2008

Improving solution characteristics of particle swarm optimization through the use of digital pheromones, parallelization, and graphical processing units (GPUs)

Vijay Kiran Kalivarapu
Iowa State University

Improving solution characteristics of particle swarm optimization through the use of digital pheromones, parallelization, and graphical processing units (GPUs)

by

Vijay Kiran Kalivarapu

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Co-majors: Mechanical Engineering; Human Computer Interaction

Program of Study Committee:
Eliot Winer (Co-major Professor)
Adin Mann
James Oliver
Judy Vance
Julie Dickerson

Iowa State University

Ames, Iowa

2008

To mom, sis
and
my wife Kavita

TABLE OF CONTENTS

List of Figures

List of Tables

Acknowledgements

This dissertation could not have been possible without the support and guidance of numerous people. Firstly, I wish to thank and express my deepest gratitude to my advisor Dr. Eliot Winer for providing me a tremendous graduate education experience. I owe a major share of my success to his constant encouragement, continuous support and above all his belief in my abilities as a researcher. He relentlessly fueled my analytical thinking and greatly assisted me with scientific writing.

I am also very grateful for having a wonderful doctoral committee and wish to thank Drs. Julie Dickerson, Adin Mann, Jim Oliver and Judy Vance for providing me invaluable input to this research.

This acknowledgement is not merely half complete if I did not thank the Virtual Reality Applications Center and the staff. The congenial work atmosphere and sense of pride they provide me is unparalleled. I would also like to thank everyone in my research group Alex, Andy, Brandon, Brett, Catherine, Christian, Eric, Kenny, Levi, Marisol, and Ruqin for their time and patience in offering me their precious pointers in shaping my dissertation and presentation. I cherished the times I spent with them sharing the woes of graduate students.

Special thanks are in order to Eric for cheering my spirits when things did not work the way I wanted. My research would have been half hearted if he did not show both sides of the coin by playing a devil's advocate.

Thanks goes out to my friends Prathibha, Vikram, Suman, Goutham, Krishnaveni, Kishore, Sreekanth, Shashank and Deepti for standing by me and providing encouragement and support.

Finally, I would like to thank the most important people in my life – mom, sis and my wife Kavita. They are the ones to whom I owe my existence. Mom's unending faith and confidence in me is what shaped me to be the person I am today. I would never forget mom's words on how nothing can go wrong with having a good education. Sis' unfaltering affection inculcated the sense of responsibility towards my family. Nothing comes close to the encouragement and support Kavita provides me. She rejoiced with me when I had happy moments at work and empathized with me during rough times. Kavita, you have taught me the value of love in life.

Abstract

Optimization has its foundations dating back to the days of Newton, Lagrange, Cauchy, and Leibnitz when differential calculus methods were developed to minimize and maximize analytical functions. Substantial progress in optimization became more prominent in the mid to late twentieth century when digital computers showed promise in offloading analytical problem solving into numerical methods through computer code for faster evaluations of designs.

Deterministic optimization methods such as steepest descent, conjugate gradient and Newton's methods are known for their robustness in iteratively reducing the objective function value for minimization problems. However, they are primarily suitable for solving single objective function problems that are unimodal and continuous. With increased sophistication in engineering problems, multimodal and multi-objective problems have become more prevalent drastically reducing the effectiveness of deterministic methods. This led to the development of heuristic methods, particularly evolutionary methods such as Genetic Algorithms, Ant Colony Optimization, and Particle Swarm Optimization. These methods have multiple design points exploring the design space over iterations as opposed to a single design point as in the case of deterministic methods. Evolutionary methods come with the capability to solve multimodal discontinuous design spaces with increased reliability and efficiency, but at considerable computational expense.

Particle Swarm Optimization (PSO) is one of the very recent population based heuristic methods similar in characteristics to other evolutionary search methods. In a basic PSO, an initial randomly generated population swarm propagates towards the global optimum over a series of iterations. The direction of the swarm movement in the design space is based on an individual particle's best position in its history trail (*pBest*) through exploration, and the best particle in the entire swarm (*gBest*) through exploitation. This information is used to generate a velocity vector indicating a search direction towards a promising location in the design space. The primary advantage of this method is its ease in implementation with a very small number of user-defined parameters. Although a relatively young method as it was developed in 1995, it has been added to the list of global search methods due to its reliability in finding the global optimum for a variety of problems.

There are a few disadvantages of the method that suppress its efficiency and accuracy and is the premise for the research presented in this thesis. Only two candidates - *pBest* and *gBest* dictate the search direction for each swarm member. Much more information is available if characteristics of additional swarm members could be utilized. Additionally, poor move sets specified by *pBest* and *gBest* in the initial stages of optimization can trap the swarm in a local minimum or cause slow convergence. To address this issue, a new approach to PSO using digital pheromones to coordinate swarms within n-dimensional design space has been developed. Digital pheromones are mathematical representations of real pheromones in that they dissipate in time and do not move in addition to the fact that a stronger pheromone field indicates a greater possibility for finding an optimum in the design space. The methods developed using digital pheromones with PSO have substantially improved the accuracy,

efficiency, and reliability characteristics when compared to a basic PSO. The implementation of this concept within a PSO is the first component in the development section of this thesis, where the challenges and method development are outlined. Statistical hypothesis testing is additionally performed to evaluate the efficacy of the developed method.

The second component of the research explores the possibility of multiple swarms searching the design space in a parallel computing environment. Two methods have been developed: 1) a synchronous coarse grain approach and 2) an asynchronous shared pheromone approach. These schemes leverage the computational capabilities offered by present day processor and network technologies in increasing the efficiency of particle swarms in reaching the global optimum in multimodal design spaces.

The third component of the research is to investigate hardware acceleration of PSO with digital pheromones using commodity graphics processing units (GPU). Methods have been developed to offload repetitive computations on to GPUs where they are computed in parallel and logical operations are carried out on the CPU that hosts the GPU. This computational outsourcing dramatically reduced the overall solution times without any significant compromise in the solution accuracy and reliability.

Realistic optimization problems are characterized by numerous inequality and equality constraints. To test the viability of digital pheromones within a PSO for solving constrained optimization problems, a sequential unconstrained minimization technique – Augmented Lagrange Multiplier (ALM) method has been implemented. This final research component

was to examine the usability of digital pheromones within PSO to solve constrained optimization problems.

The performance of each developed method was evaluated through a series of relevant multi-dimensional multimodal test problems, and the results from digital pheromone PSO were benchmarked against basic PSO implementations. Unconstrained problems were tested on serial, distributed parallel computing environments and workstations with GPUs. Constrained optimization problems were tested on serial computing environments and results are presented. The testing of the developed methods showed promising results and provided encouraging motivation for future development in addressing a wide variety of problems (discrete optimization problems, multi-objective problems, etc).

1 Introduction

In a most generic sense, optimization is the process of attaining a best output from a given set of inputs. Design engineers typically have to take into account many technological and managerial decisions during a design process. The eventual purpose of such decisions is to either minimize costs or maximize benefits or both. Design optimization provides necessary tools required to achieve these targets.

Engineering problems, when formulated appropriately can extensively be solved using design optimization techniques. Some such typical applications, but not limited to, are listed below:

1. Aircraft design
2. Design of structures such as frames, foundations, bridges, etc for minimum costs
3. Optimal design of mechanical components such as linkages, gears and machine tools
4. Design of material handling equipment such as conveyors, trucks and cranes for minimum costs
5. Traveling salesman problems
6. Optimal production planning, control, and scheduling
7. Optimal design of control systems

1.1 Formulation of Optimization Problems

Typically, a design optimization problem consists of an objective to be achieved, through satisfying certain conditions. This objective is termed the objective function, cost

function, or fitness value. The conditions that need to be satisfied while solving the problem are called the design constraints. A general optimization problem can mathematically be stated as follows:

Minimize:	$F_1(\mathbf{X}, \mathbf{Y}), F_2(\mathbf{X}, \mathbf{Y}), \ldots, F_p(\mathbf{X}, \mathbf{Y})$		Objective function
Subject to:	$g_j(\mathbf{X}) \leq 0,$	$j = 1, m$	Inequality constraints
	$h_k(\mathbf{X}) = 0,$	$k = 1, l$	Equality constraints
	$X^l_i \leq X_i \leq X^u_i$	$i = 1, n$	Side constraints

$\mathbf{X} = [X_1, X_2, \ldots, X_n]$, $\mathbf{Y} = [Y_1, Y_2, \ldots, Y_n]$
$\mathbf{X} \rightarrow$ Independent Design Variables (DVs)
$\mathbf{Y} \rightarrow$ Dependent Design Variables

A general optimization problem consists of one or multiple objectives to be minimized represented by $F_1(\mathbf{X}, \mathbf{Y})$ through $F_p(\mathbf{X}, \mathbf{Y})$. **X** is a vector of independent design variables, which are the foundational parameters that all other functions are built upon. **Y** represents a vector of dependent design variables that are linear or non-linear functions of **X**. Inequality constraints are typically denoted by g(**X, Y**), and equality constraints are represented as h(**X, Y**). The objective function F(**X, Y**), inequality constraints g(**X, Y**), and equality constraints h(**X, Y**) can each be linear or non-linear functions depending upon the problem to be solved. The side constraints provide lower and upper bounds for the design variables. If design vector **X** is plotted on an n-dimensional Cartesian coordinate system with each coordinate axis representing a design variable (X_1, X_2, X_3 … X_n), the space occupied by the coordinate system is called the design variable space or the design space. An objective function F(**X**) refers to the location in the design space for

a specific set of values assigned to the design vector **X**. Figure 1 represents a simple single dimensional objective function with four minimums.

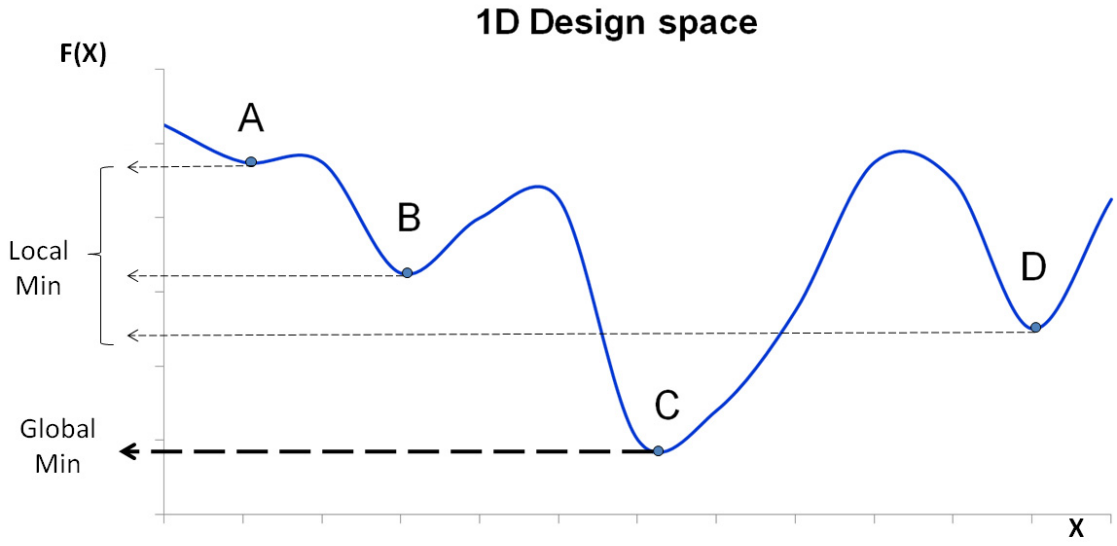


Figure 1 Simple One dimensional design space

In this design space the design variable X is plotted on the X-axis and the objective function is plotted on the Y-axis. Points A, B and D are local minimums and point C is the global minimum. Figure 2 represents a two dimensional objective function with one minimum. The design variables X_1 and X_2 are plotted on the coordinate axes and the objective function is represented as contours that are obtained for different combinations of X_1 and X_2.

$$c_1 < c_2 < c_3 \ldots < c_7$$

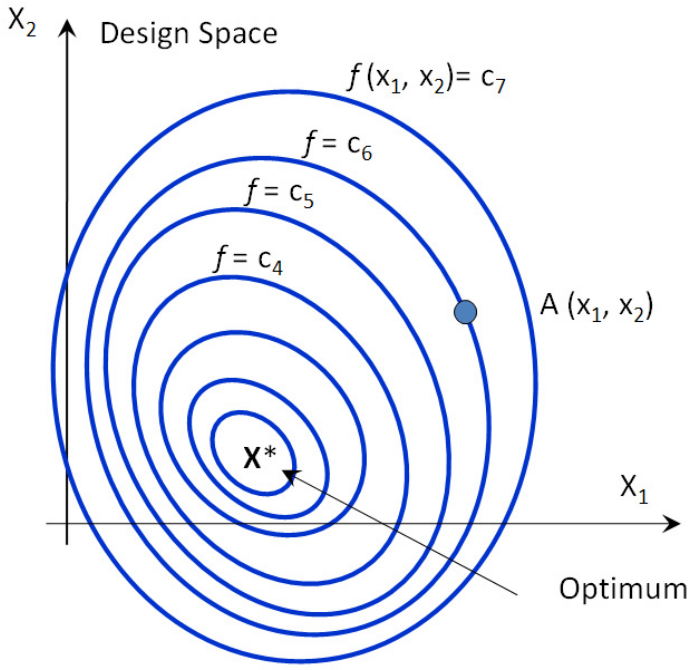


Figure 2 Contour plot of a 2D objective function

The smallest oval in the objective function contour represents the optimum and its value increases as the size of the oval increases. A design point A (X_1, X_2) encapsulates the design variable information. For a 10 dimensional objective function the design point A will have variable values X_1, X_2, X_3, …, X_{10}.

Realistic design problems are characterized by numerous inequality and equality constraints. Figure 3 represents a 2D objective function with one linear and three non-linear inequality constraints.

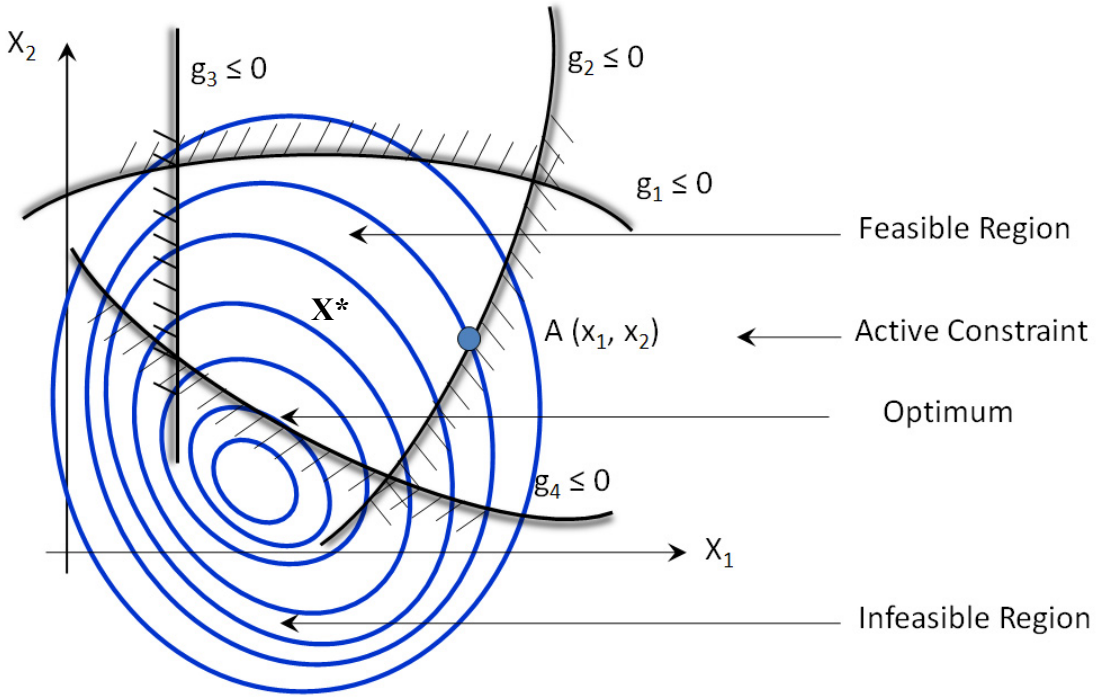


Figure 3 Contour plot of a 2-D objective function with inequality constraints

In figure 3, the smallest oval does not represent the optimum because it violates the constraints represented by $g_j(\mathbf{X}) \leq 0$, where j = 1 ... 4. The area enclosed within the constraints is the feasible region and a design point outside of the feasible region is infeasible. When a design point A (X_1, X_2) resides on a constraint boundary, the constraint is considered "active". The optimum value for this objective function is shown by **X*** that renders the inequality constraint $g_4(\mathbf{X})$ as active.

1.2 Classification of Optimization Problems

Optimization problems are classified into various categories as shown in Figure 4. If the objective function and all the constraints are linear functions of the design variables, the optimization problem is considered a Linear Programming (LP) problem. If the objective

function and/or the constraints are non-linear functions of the design variables it is termed a Non-linear Programming (NLP) problem.

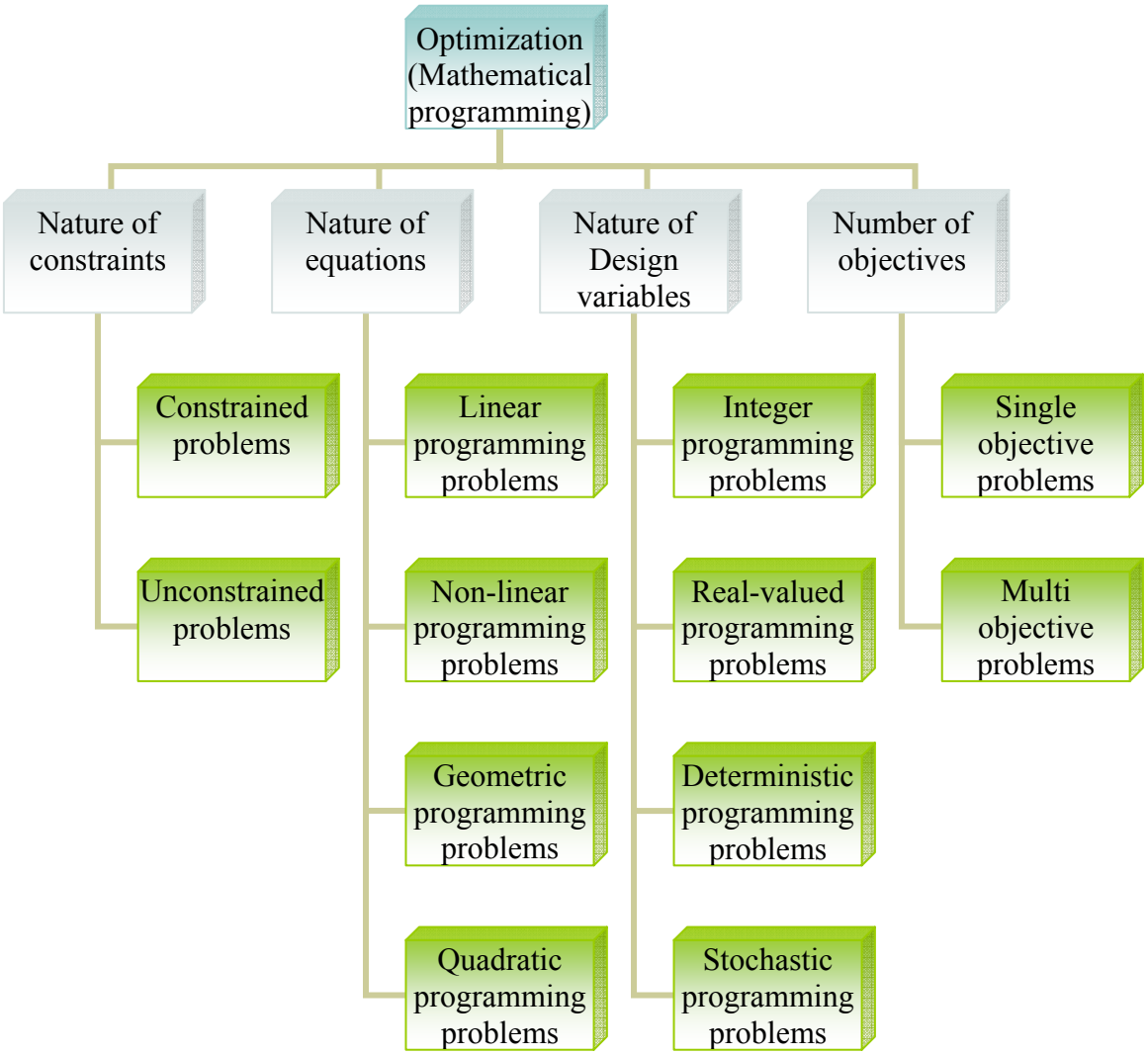


Figure 4 General classification of optimization problems

A Geometric Programming problem (GP) is one in which the objective function and constraints are expressed as posynomials[⊗] in **X**. A Quadratic Programming (QP) problem is a non-linear programming problem with a quadratic objective function and linear constraints. If some or all of the design variables in the design vector are restricted to take

[⊗] A function p(X) is called a posynomial if p can be expressed as the sum of power terms each of the form: $C_i x_1^{ai1} x_2^{ai1} x_3^{ai1} \ldots x_n^{ai1}$, where C_i and a_{ij} are constants with $C_i>0$ and $x_j>0$.

on only integer (or discrete) values, the problem is called an integer-programming problem. Real-valued programming problems are those where the design variables are permitted to take any real value. If the parameters (design variables and/or various pre-assigned values) are probabilistic, then the problems are considered stochastic (non-deterministic). Single objective and multi-objective problems are classified based on the number of objective functions to be minimized. In addition to these classifications, an objective function can be unimodal or multimodal. Unimodal objective functions are those that contain a single optimum while multimodal objective functions contain multiple optimums. A real design situation more often as a rule than exception, encompasses more than one of the above features into the design objective(s). For example, an aircraft wing design could have two objectives (multi-objective problem), one being simple linear and the other being highly non-linear, multimodal and multi-dimensional. Such problems are more difficult to solve than single unimodal objective problems.

1.3 Numerical and Evolutionary Methods

Methods to solve design optimization problems in various categories require different approaches and techniques [1] [2] [3]. Analytical methods use classical differential calculus theory and calculus of variations where the extremes of a function *f(x)* are obtained by finding the values of *x* that cause the derivatives of *f(x)* to vanish. These methods can be used to find unconstrained maximums and minimums of an objective function with several design variables, with the assumption that the design space is

continuous and functions are twice differentiable. Some such deterministic optimization methods include:

1) Simplex methods for linear programming problems
2) 1-D search methods for non-linear problems – exhaustive search, interval halving, golden section, Quadratic and Cubic interpolation methods, and Newton's method.
3) Unconstrained optimization methods – Random walk, Powell's method, Steepest descent (Cauchy's) method, Newton's method, and the Conjugate gradient (Fletcher-Reeves) method.
4) Constrained optimization methods – Sequential linear and quadratic programming, Penalty function methods, Augmented Lagrange Multiplier method, Method of Feasible Directions, Modified Method of Feasible Directions, and the Generalized Reduced Gradient Method.

The ubiquitous availability of cheap computational hardware resources made it possible to automate much of these design optimization processes thereby paving the way for numerical optimization. Through numerical methods, this computational hardware can perform number crunching quickly and achieve an optimal combination of design variables in a design iteratively. For example, consider a 2D design problem that we wish to investigate with 10 different values for each of the design variables. Therefore, a total of 100 (10x10) combinations of design variables exist. Let us assume that it would take $1/10^{th}$ of a CPU second for a computer to compute the objective function for each combination of the design variable. For the 100 combinations, it would take 10 seconds of computer time. Realistic design situations where objective functions are composed of

hundreds or thousands of design variables will require a substantial amount of computational resources. Advances in computational hardware (e.g., processor power) allows for increased clock cycles per second and hence faster evaluation of designs. For example, a 3.0 GHz processor is capable of performing 3.0 x 10^6 floating-point operations per second.

For years, many numerical methods (i.e. those guaranteeing a reduction in the objective function iteratively) were developed to solve different types of optimization problems. Most of these problems were single objective and unimodal in nature. For problems containing multiple extrema (multimodal), methods were devised to run from a number of initial points to determine the global solution. With increased sophistication in process and product design, these problems also grew larger and became increasingly multimodal and multi-objective in nature. The use of numerical methods alone was no longer sufficient, giving way to heuristic methods, particularly evolutionary methods. In an evolutionary method, a population of design points is generated and made to traverse and explore the design space to find the optimal objective function value (usually a maximum or minimum) and its corresponding design parameter values, over a series of iterations, while agreeing with the design constraints. Examples of such evolutionary algorithms are Genetic Algorithms (GA), Simulated Annealing (SA), Ant Colony Optimization, and Particle Swarm Optimization (PSO). These evolutionary methods have some natural advantages over traditional deterministic methods:

1) They can handle mixed continuous-discrete variables, and discontinuous and non-convex design spaces. Use of numerical methods can either be computationally very expensive or return incorrect values (i.e. get trapped in local minimums).
2) Evolutionary methods do not require derivative information to attain a solution.
3) A population of design points is used instead of a single design point to search for the optimum. Therefore, there is a higher probability of reaching the global optimum.
4) Due to random initialization of the population, the chances of getting trapped in local minimums are dramatically reduced.

The primary strength of evolutionary methods lie in that population members arrive at a global optimum through communication with each other. It can be thought that the performance of numerical methods can be equivalent to evolutionary methods when executed with multiple initial points searching the design space simultaneously. However, numerical methods are not designed to provide communication between multiple design points in the design space. Therefore, each design point will be subjected to a higher computational intensity (e.g., first and second derivative information) than a typical evolutionary algorithm. As such, evolutionary methods have proven themselves somewhat more efficient for reaching the global optimum than numerical methods.

The following three sections describe the salient characteristics of the most widely used evolutionary methods including their advantages and disadvantages. It is then followed by the motivation for the research presented in this thesis.

1.4 Genetic Algorithms

Genetic Algorithms (GA) are based on the principles of genetics and natural selection inspired from Darwin's theory of evolution – survival of the fittest. Holland [4] was the first to present it systematically and was later explained in the context of biological evolution by Rechenberg [5]. Due to its robustness and insensitivity to whether design spaces are continuous or discrete, they are one of the most widely used heuristic evolutionary optimization methods. They have been in existence for approximately 35 years and are still an active area of research [6] [7]. Although the implementation could be different and problem specific, the genetic search typically consists of three main components: (a) selection (reproduction), (b) crossover, and (c) mutation.

A population of design points is used instead of a single design point. The size of the population can range anywhere from $2n$ to $4n$ and sometimes up to $10n$, with n being the number of design variables. The design variables are typically represented as binary encoded strings corresponding to chromosomes in genetics. For example, a design variable vector <x1, x2, x3, x4> = <18, 3, 1, 4> can be represented as a binary string as shown in Figure 5.

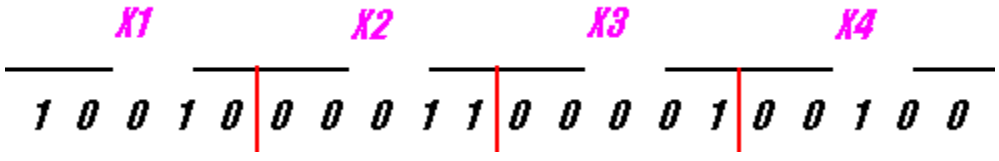


Figure 5 Binary representation of variables in Genetic Algorithms

In general, if a binary number is given by B_qB_{q-1} ... $B_2B_1B_0$ then its equivalent decimal representation is given by $\sum_{i=0}^{q} 2^i B_i$, where 'i' indicates the current position in the binary string. Therefore, x_1 (= 10010) as shown in Figure 5, is represented as $1x2^4 + 0x2^3 + 0x2^2 + 1x2^1 + 0x2^0$, a binary equivalent for the number 18. This flexibility in representing the design variables makes GAs naturally suitable for use in both discrete and continuous problems. Moreover, they do not require derivative information. The objective function value is termed as 'fitness value', which is analogous to the role of fitness in natural genetics. A new set of strings is produced in each generation (iteration) through selection, crossover and mutation from old generation.

In the selection process, the best genes would be retained while copies of the fittest genes would replace the weakest genes. Different methods are used to perform the selection process, tournament and roulette wheel selection being the most popular. Survival of the fittest theory makes highly fit individuals survive and reproduce in each generation. The algorithm automatically gets rid of least fit individuals through replacement by children from the highly fit individuals [4-8].

For crossover, two individual strings (chromosomes) are selected at random from the currently fit design vector. A crossover site is selected at random along the string length, and the binary digits are swapped between the two strings following the crossover site. Thus, a new string of design points are obtained, which is placed in the new population pool. There are various types of crossover implementations, the most common of them

being single point crossover, two point crossover, and cut-and-splice crossover methods [9-12].

A mutation process is then followed based on an assigned mutation probability. Mutation is an occasional random swapping of binary digits in a design variable from 0 to 1 and 1 to 0. This procedure intends to prevent any bias in the individuals proceeding toward the solution and controls trapped local optimums. When used sparingly with selection and crossovers, mutation serves as a safeguard that prevents premature loss of important genetic material during the course of the algorithm. Figure 6 represents typical crossover and mutation operations. In the figure, the crossover is performed on parents 'A' and 'B', where a portion of the binary string (shown in bold) in 'A' is swapped with the binary string in 'B'. Therefore, two parents of the form **11001**011 and 11011**111** combine to form **11001111** after crossover. Mutation is also explained in the figure where a selected bit in a string is swapped from 0 to 1 or 1 to 0.

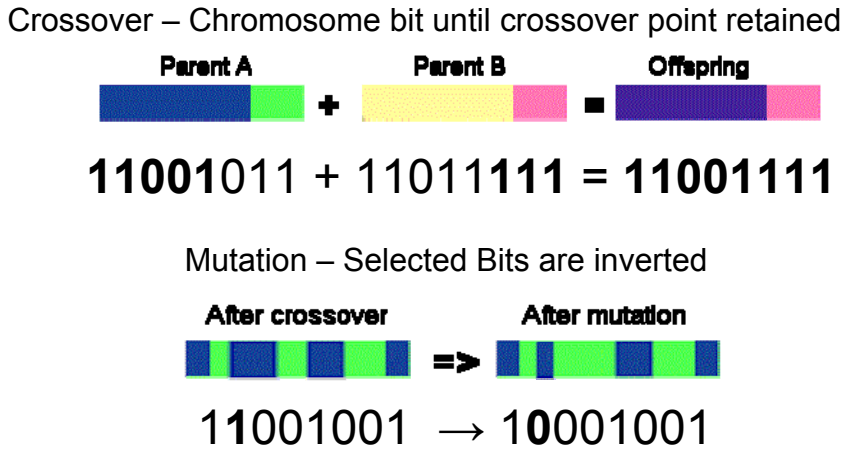


Figure 6 Crossover and Mutation operations in Genetic Algorithms

With new individuals obtained from each generation through these operations, fitness values for each individual are calculated. The algorithm stops with success when an appropriate convergence criterion is satisfied. For example, a problem is said to be converged if the difference in solution values are within 0.001 for 20 consecutive generations.

GAs are primarily designed to handle unconstrained optimization problems that have single objective functions. Creating an unconstrained pseudo objective function [1] [2] [3], which contains a representation of the original objective function and constraints, is used to solve constrained optimization problems. Similar methods [1] [2] [3] can also be used to address multi-objective problems.

The advantages of GAs are that they efficiently search the design space and are more likely to converge toward global minima when compared to direct methods [2]. Since the design variables are typically binary in nature, design spaces with discrete and integer design variables are well handled. Just as any other heuristic method, GAs do not require derivative information thereby avoiding the requirement for continuous design spaces. Additionally, since the method's parameters do not interfere with the population size, it can be easily parallelized to realize gains in performance and efficiency.

GAs have certain disadvantages that make it unsuitable for certain types of problems. For example, they show a very fast initial convergence, but improvements in fitness value slow as more generations are created. Based on the complexity of some GA

implementations, there could be a large number of user-controlled parameters that need to be carefully selected. Also, GA implementations are computationally intensive. Although mutation can sometimes help, population members getting trapped in local minimums are not uncommon.

1.5 Simulated Annealing

Simulated Annealing (SA) is a probabilistic algorithm to locate global optimum in multi-dimensional design spaces. This method was described by two researchers Kirkpatrick, et al [13] in 1983, and Černý [14] in 1985. It is an adaptation of the Metropolis-Hastings [15] algorithm, a Monte Carlo method to generate sample states of a thermodynamic system. The SA procedure described in this section is adopted from two sources [16] [17].

SA is similar to the process of thermal annealing involved in metal forming. Typical annealing process involves slow and controlled cooling of a metal to ensure proper solidification for a highly ordered crystalline state corresponding to the lowest energy state. Rapid cooling potentially causes defects and does not provide preferred material properties. In SA, the design space is considered to be the state of a physical system, and the objective function is analogous to the internal energy of the system in that state. The primary aim is to bring the system from an arbitrary initial state to one with least possible energy (i.e. the objective function is to be minimized). One of the advantages of SA lies in that entrapment in a local minimum is probabilistically avoided causing it to either stay

at the current position or only propagate to a position with lower energy state (i.e., minimum).

The algorithm starts from an initial vector $\mathbf{X_1}$, iteratively generates improved design points $\mathbf{X_2}$, $\mathbf{X_3}$, …, while moving towards the global minimum. A current design point X_i is randomly made to move along each coordinate direction. The values for the new coordinates are uniformly distributed around X_i, and are ensured to be within the design variable's lower and upper bounds. A design vector **X** is accepted or rejected based on a metropolis criterion. According to this criterion, a design is accepted if the objective function resulting from the new design point $\mathbf{X_{i+1}}$ is improved (typically less than) over the one resulting from the older point, $\mathbf{X_i}$. In such a case, $\mathbf{X_{i+1}}$ is set to be **X**. Otherwise, the design point is accepted with a metropolis acceptance probability, P, as shown in equation (1).

$$P(\Delta f) = e^{\frac{-\Delta f}{kT}} \qquad (1)$$

Where, $\Delta f = f(\mathbf{X_{i+1}}) - f(\mathbf{X_i})$ and k is Boltzmann's constant[1]. The value of 'k' influences convergence characteristics. The acceptance probability function P(e,T) defines the probability of making the transition from a current state s to a new state s' on a time-varying parameter, temperature (T). One of the requirements of the method is that 'P' should be non-zero when $f(\mathbf{X_{i+1}}) > f(\mathbf{X_i})$, allowing the system to move to a new state even when the energy is higher than the current state. This feature in SA prevents the method

[1] Boltzmann's constant = 1.3806503 x 10^{-23} m^2 kg/s^2K

from getting trapped in a local minimum – a state worse than global minimum but better than its neighbors.

SA starts with a high value for temperature, T_0. Design vectors are generated iteratively until equilibrium is reached. Then, the temperature is further reduced and a new sequence of design vectors is generated. This procedure is continued until a sufficiently low temperature is reached, at which stage no further improvement in the objective function value can be realized. SA by nature does not handle constraints just as in a GA. Methods should be incorporated within an SA to be able to handle inequality or equality constraints [1-3].

The advantage of SA lies in that it is relatively insensitive to the type of design space and can deal with arbitrary systems and cost functions. It statistically guarantees finding an optimal solution, i.e., it either improves the solution or stays put at the same solution from the previous iteration. Just as GA, SA does not need derivative information hence making it suitable to search discontinuous design spaces. Finally, SA can be very easily parallelized to realize better efficiencies.

There are a few disadvantages that make SA ill suited for certain optimization problem types. For example, iterative annealing is very slow and the problem is especially apparent with increasing complexity in the objective function. Although the characteristics of the design space are typically unknown, SA is computationally expensive especially if the design space is smooth or unimodal in nature. Direct methods

or other heuristic methods that can take advantage of additional information about the design space provide better performance characteristics when compared to SA. Additionally, a good cooling schedule is problem specific and is generally difficult to define thereby increasing the possibility of premature crystallization (entrapment in a local minimum).

1.6 Particle Swarm Optimization

PSO was developed by a psychologist, James Kennedy and an electrical engineer, Russell Eberhart in 1995 [18] [19] based on experiments derived from mathematical modeling of the flocking behavior of birds. The flocking models were originally developed by a biologist named Frank Heppner [20]. Heppner's model was different from other flocking models in that it imparts attraction characteristics to roosting areas. According to this model, birds begin by flying around with no set destination and form flocks with the rest of the birds. However, when the criteria of 'desire to roost' is set higher than 'desire to stay in the flock' for a bird, it would pull away from the flock and land. This behavior in one bird resulted in the remaining birds following until the entire flock had landed.

This concept was improvised by Kennedy and Eberhart to search multi-dimensional design spaces. Finding the roost is analogous to finding the global optimum, and the process in which a bird finds a roost making the remaining birds to follow the lead provides socio-cognitive characteristics in finding the global optimum. In this implementation, particles (mathematical models for birds) fly in the design space and

propagate towards the best solution. However, there are no rules in this model to avoid particles from propagating towards a local solution instead of a global solution. Therefore, Kennedy and Eberhart proposed a method to utilize the social and cognitive information gained by particles traversing through the design space. The social aspect gathers information from the remaining particles (exploitation), while the cognitive aspect takes advantage of information from a particle's own history (exploration). If there is too little exploration, particles tend to converge on the first good solution. On the other hand, particles will never converge if there is very little exploitation. Therefore, a balance of socio-cognitive information is required, which is what Kennedy and Eberhart achieved in their formulation for PSO.

PSO is a population based zero-order optimization method that exhibits several evolutionary characteristics similar to GAs. These are: 1) initialization with a population of random solutions, 2) design space search for an optimum through updating generations of design points, and 3) update based on previous generations [21]. In this method, each particle in the swarm denotes a location (i.e. design point) in the design space whose position is updated iteratively. Therefore, each particle moves from one position to another each iteration. A velocity vector, a function that captures the combined effects of each swarm member's exploration and exploitation characteristics, provides the direction and the magnitude of this movement. The algorithm iteratively updates the search direction of the swarm propagating towards the optimum. Although there were many preliminary implementations, equations (2) and (3) are the most popularly used definitions for the mathematical simulation of this behavior.

$$V_{iter+1,i}[] = w_{iter} \times V_{iter,i}[] + c_1 \times rand() \times (pBest_i[] - X_i[]) + c_2 * rand() \times (gBest[] - X_i[]) \quad (2)$$

$$X_{iter+1}[] = X_{iter}[] + V_{i+1}[] \quad (3)$$

$$w_{iter+1} = w_{iter} \times \lambda_w \quad (4)$$

Equation (2) represents the velocity vector update of a basic PSO method in iteration '*iter'*, for each design variable represented by square braces and for each swarm member, *i*. $rand_p()$ and $rand_g()$ are random numbers generated each for *pBest* and *gBest* between 0 and 1. c_1 and c_2 are user definable confidence parameters. Typically, these are set to values of 2.0. '$pBest_i[]$' represents the best position of the i^{th} particle in its history trail, and '*gBest[]*' represents the best particle location in the entire swarm. w_{iter} is termed "inertia" weight, and is used to control the impact of a particle's previous velocity on the calculation of the current velocity vector. A large value for w_{iter} facilitates global exploration, which is particularly useful in the initial stages of an optimization. A small value allows for more localized searching, which is useful as the swarm moves toward the neighborhood of the optimum [22] [23]. These characteristics are attributed to the swarm by implementing a decay factor, λ_w for the inertia weight, as shown in equation (4). Equation (3) denotes the updated swarm location in the design space.

One of the primary advantages of PSO is its ease in implementation with a small number of user defined parameters. The core of PSO requires very few lines of code when compared to GA and SA. PSO has been added to global search methods due to its reliability in finding global optimums for a wide range of problems [24] [25]. PSO is a population based method and hence can easily be implemented in parallel to gain

performance benefits. Moreover, it works with objective function evaluations alone and does not need derivative information. Therefore, discontinuities in design spaces can easily be handled.

PSO is relatively young compared to other heuristic methods. Although it is intuitive and can solve various types of problems, there are a few disadvantages that suppress its efficiency and accuracy. At any instance, each particle is influenced by only *pBest* and *gBest*. This impedes the desired exploratory characteristics in the design space, and is not always sufficient to propagate toward the global optimum, especially in multimodal problems. Secondly, the method is initial condition dependent. Any poor location specified by *pBest* and *gBest* in the initial stages can offset the swarm from reaching the neighborhood of the optimum or delay convergence.

No single heuristic method is ideal and can guarantee global optimum for all types of optimization problems. GA and SA have a history of more than 25 years in development and research is still being done to enhance their performance characteristics. Particle Swarm Optimization (PSO) on the other hand is a more recent method and has tremendous potential for further improvement. The research presented in this thesis addresses the drawbacks listed above to realize performance gains and contribute towards improving PSO in solution efficiency, accuracy and reliability. The second chapter provides a comprehensive background on the past and present developments in PSO. It is then followed by various resources that modern computational infrastructure can offer for further development of PSO. Finally, the research issues are identified and defined.

2 Background

2.1 Particle Swarm Optimization

A significant number of modifications have been made to the basic PSO algorithm for realizing performance improvements after it originated in 1995. Natsuki and Iba [26], and Hu, et al. [27] have explored the possibilities of performance improvement through introducing mutation factors in PSO, similar to the ones used in GAs. Gao et al. have obtained improvements in PSO, through the use of a virus operator that propagates partial genetic information in the swarm by infection operators for enhanced design space search [28]. Ray and Saini [29] developed a method to improve swarm movement within the design space through information sharing between individual particle members. They have successfully implemented this strategy in solving both constrained and unconstrained problems as well.

Additionally, research has been done on utilizing PSO for constraint handling. Venter and Sobieski [30] implemented a quadratic exterior penalty function method to solve non-linear constrained optimization problems. Hu and Eberhart [31] modified the basic PSO method so that the swarm is repeatedly initialized until all constraints are satisfied, while also forcing *pBest* and *gBest* to be feasible in every iteration. Sedlaczek and Eberhard [32] implemented the augmented Lagrangian method for solving small, constrained non-linear optimization problems. Discrete PSO methods have been known to solve

constrained optimization problems as well, and Yang et al. have demonstrated it through converting by satisfaction problems into discrete optimization problems [33].

PSO was used and modified for multi-objective problems as well [34] [35]. Some of the recent advancements include solving traveling salesman problems using discrete PSO methods [36-39]. Penalty function approaches have been used to solve mixed discrete non-linear problems using PSO [40]. Other areas include developments in the areas of integer programming [41] and continuous variable problems [42]. Parsopoulos and Vrahatis [43] demonstrated the use of PSO for solving a wide range of problems including multi-objective, mini-max, and integer programming problems. The same authors have developed methods to compute all global minimizers of an objective function using PSO [44]. Similarly, He, et al. presented methods that tackled mixed variable types – integer, discrete and continuous variables [45]. A 'fly-back' constraint handling mechanism was also introduced in this research to maintain a feasible population. A substantial amount of success has been achieved in utilizing PSO for applications such as aircraft design [46] [47], topology and shape optimization [48] [49], structural optimization [50] [51], wireless network routing problems [52], optimization in manufacturing and production operations [53] [54], collision detection problems [55], and detection of optimal paths for unmanned aerial vehicles (UAVs) [56] [57].

2.2 Digital Pheromones

Pheromones are chemical scents produced by insects essentially as a means of communication in finding suitable food and nesting locations. The more insects that

travel a path, the stronger the pheromone trail. A digital pheromone works on the same principle and is analogous to a natural pheromone in that it is a marker to determine whether or not a region in the design space is promising for further investigation. Digital pheromones have been used in applications such as the automatic adaptive swarm management of Unmanned Aerial Vehicles (UAVs) [58] [59]. In this research, the implementation of digital pheromones causes swarms of UAVs to automatically adapt and navigate in potentially hazardous environments dramatically reducing the requirement of human operators at the ground control stations. Ant Colony Optimization [60-62] models the behavior of ants that release pheromones to find optimal paths to food from their nesting location. In this method, pheromones act as attractors released by members (ants) causing other members to be attracted to stronger pheromone trails. Digital pheromones are also used for solving network communication problems [63].

The concept of digital pheromones is relatively new [64], and has not been applied to investigate n-dimensional design spaces. The benefits of digital pheromones from swarm intelligence and the adaptive applications described above can be merged into PSO to improve design space exploration, particularly for a multimodal optimization problem where swarm communication is essential to locating the global optimum accurately, efficiently, and reliably.

2.3 Parallelization

Parallelization provides a very convenient alternative to improve solution times when single workstation environments are not sufficient. Processor technology advancements

in addition to low costs make scientific computation on massively parallel computer clusters viable and affordable in academics and industries. However, certain requirements are crucial for an algorithm to be implemented in parallel. The primary requirement for parallelization is the ability of the method to decompose into segments for multi-processor operation. In addition, the two highly desirable characteristics for parallelization are: a) scalability – the ability to adapt to any number of processors with no/minimal changes and b) processor load balancing – use of the available number of processors to the full extent without any processor substantially running idle.

Parallelization can be synchronous or asynchronous. Synchronous parallelization facilitates a step wise parallel execution of tasks. Coarse decomposition schemes are examples of synchronous parallelization where each processor has its own swarm exploring the design space. Solutions obtained from different processors are synchronized and gathered on a common processor (usually, the root processor) to evaluate the final global optimum. This is achieved through the use of a barrier function in the Message Passing Interface (MPI) [65], the most commonly used interface for parallel programming. Asynchronous parallelization is the dividing of a sequential algorithm into autonomous tasks each of which can be carried out on different processors. Dependencies among the tasks are modeled by message passing or through shared memory [66], depending upon the hardware configuration.

Population based optimization methods such as GA and PSO are computationally intensive and are a natural fit for parallelization because the method parameters do not limit the number of processors that can be used for solving the problem. Three different

types of parallelization are seen as most common in the literature for population based methods: 1) global (master-slave) model, 2) migration (distribution) model, and 3) diffusion model. In the master-slave model, the objective function is evaluated in parallel on slave processors, and the remaining operations are performed on the master, (e.g., selection and crossover in GA). This means that while the objective function values are evaluated on different processors by a subset of the population, the selection and crossover operations are performed for the total population. Therefore, no information about the population is lost due to information transition between processors, but the algorithm proceeds much faster [67] [68]. In the migration model, the population is divided into a number of sub-populations, with each propagating independently on a different processor and computing its own optimum. Depending upon the implementation, the best fitness value in all the processors is communicated to each other periodically. This model is known to produce better parallel efficiencies compared to the master-slave model but network communication causes considerable overhead [68-70]. The diffusion model depends on the locality concept where each population member is considered a separate breeding unit. Each population member is unaware of the best value in each iteration, and it moves only toward the best value in the neighborhood. The best value attained by each processor is broadcasted so that each member adjusts its own best location accordingly. Since each processor broadcasts its best value in the current iteration, entrapment in local minima is avoided. The effectiveness of this model depends upon the type of connection topology such as ring (two links per node) or fully connected [68].

Schutte, et al [71] developed a synchronous parallelization scheme for PSO where the participating processors synchronize after objective function evaluations prior to computing the velocity vector. This synchronization caused significant performance issues, which were addressed by Koh et al [72] in their adaptive concurrent asynchronous parallelization scheme. In this research, the particle order in a swarm is permitted to change continuously depending upon the speed at which each processor performs objective function evaluations. This approach allows for the elimination of an iteration counter altogether and hence the need for synchronization between processors – a major bottleneck in parallel algorithms. Similarly, Venter and Sobieski [73] developed another asynchronous parallelization scheme in a master-slave implementation. The master processor is primarily used for controlling communication between processors and the slaves perform PSO computations while maintaining load balancing between processors. Belal and El-Ghazawi [74] explained various parallel models in PSO including master-slave, migration, and diffusion PSO. Shi et al [75] developed a hybrid parallel method where PSO and GA interact, execute simultaneously and exchange design space information after a set number of iterations. Another approach was devised by the same authors where PSO and GA interact with each other in series. The benefits of PSO parallelization has been successfully applied in various fields such as the design of electromagnetic absorbers [76], power flow applications [77], and antenna designs [78] [79].

2.4 Computations Using Graphics Hardware

Recently, technologies such as hyper threading and multi-core processing [80] have been the main drivers increasing CPU performance as opposed to the addition of more transistors onto a CPU chip. While hyper threading requires an additional burden on the programmer to develop thread-enabled code to realize performance improvements, multi-core processor improvement is only linearly related to the number of cores used on the processor chip. For example, a dual core processor can only increase the CPU performance by approximately a factor of two. However, commodity Graphics Processing Units (GPUs) or more commonly graphics cards, another proven and developing technology, is capable of improving computational performance more than ten times that of a modern CPU [81]. For their price and ubiquitous availability, GPUs have a superior processing architecture when compared to modern CPUs. For example, a dual core processor has essentially two CPUs on one chip, but depending upon the type, GPUs can have greater than 24 processors (24 fragment shading pipelines). In addition, GPUs are capable of supporting hundreds of hardware threads as opposed to one or two on a CPU. Early GPUs had fixed functionality that made them ideal for supporting visualization and gaming. Modern GPUs include improved programmable processing units and support vectorized floating point operations. The advent of programmable graphics hardware in recent years has unlocked the use of GPUs for purposes other than visualization to enable CPU type operations to be performed. GPUs offer distinct advantages to any process involving large amounts of computation as they are now: 1) programmable, 2) priced significantly less than a high performance CPU, 3) data parallel in architecture, 4) highly threaded, and 5) good at reducing main memory access costs.

The programming component of GPUs primarily consists of vertex shaders and fragment shaders (also called pixel shaders). In graphics programming, vertex shaders handle transformation of vertices of an object and fragment shaders handle computing the pixel color values that fill the screen. Initially, graphics programmers created low-level (fine control) vertex and fragment shaders to achieve these tasks. However, due to the tediousness involved in programming with these shaders and limited flexibility in terms of debugging and code re-use, low-level shader programming is not a preferred method for graphics programming. High-level shading languages, which incorporate several low-level function calls into easier to use functions, are now available, which solve the rigid low-level programming issues. The function of a shading language is to compile a shader program into individual vertex and/or fragment components and perform required computations before rendering images on the screen. Even though these operations were designed to create realistic computer graphics, they are still mathematical. If it is understood what mathematics are being performed, the data placed in a texture can be multiplied, divided, or subjected to other complex mathematical operations.

While CPU programming has a large number of well-established programming languages to choose from, there are only few GPU programming languages such as Cg [82], GLSL [83], HLSL [84], Sh [85], and Ashli [86]. These languages are quite graphics specific, so the terminology used in programming follow the mapping constructs to CPU programming given in table 1.

Table 1 Terminology used for mapping CPU algorithms to the GPU

CPU	GPU
Arrays or streams	Textures
Parallel loops	Quads
Loop body	Vertex + fragment program
Output arrays	Render targets
Memory read	Texture fetch (gather)
Memory write	Framebuffer write (scatter)

These shader languages adopt a C/C++ style of programming syntax. While Cg abstracts the graphics hardware quite closely, GLSL has some data types defined outside of the scope of current day graphics cards such as integers and matrices. As graphics hardware begins to support these data types, GLSL will be a powerful language. Sh on the other hand provides stream-programming capabilities particularly suitable for general purpose GPU (GPGPU) programming. Ashli is a layer above the other shader languages that internally supports reading shaders written in GLSL and HLSL, thereby providing a higher level of flexibility in GPU programming.

Other high-level programming languages have emerged in recent years that focus more on the GPGPU functionality as opposed to graphics specific constructs. Some such languages are Brook [87], Scout [88], Microsoft Accelerator [89], CGiS [90], and the Glift template library [91]. Performance and other comparison characteristics for these languages have been studied [94] to provide a guideline for use in specific applications. CUDA [92] is one of the latest development tools from NVIDIA aimed at GPGPU computing. This promises to eliminate stream shader programming and GPUs can be programmed through multi-threaded C programming for exponential information flow.

Studies have shown that GPUs exceed the number of floating point operations per second and memory bandwidth on comparable CPUs. For example, a 3GHz Intel Pentium 4 processor peaks at 12 GFLOPS (Giga-Floating Point Operations) with ~6 GB/sec of memory bandwidth as opposed to an ATI Radeon X1800 XT GPU that peaks at 83 GFLOPS with 42 GB/sec of memory bandwidth. This is an improvement of almost 600% in floating point operations. The number of transistors that a GPU can hold is up to 222 million compared to 50 million on an Intel Pentium 4 CPU, an increase of over 400%. Clearly, it can be seen that GPUs promise a tremendous amount of computing power than their CPU counterparts [93-95]. The technological advancements in GPU hardware have been predicted to follow a pace equal to three-times that of Moore's law. In addition, most computers and workstations currently have a GPU. These performance gains could be instantly realized without the need to purchase additional hardware. If a computer is lacking a GPU, a robust graphics card can be purchased for as little as $100-$400 to acquire tremendous processing power. Figure 7 compares the performance curves of GPUs (NVIDIA and ATI) versus CPUs (Intel) in recent years.

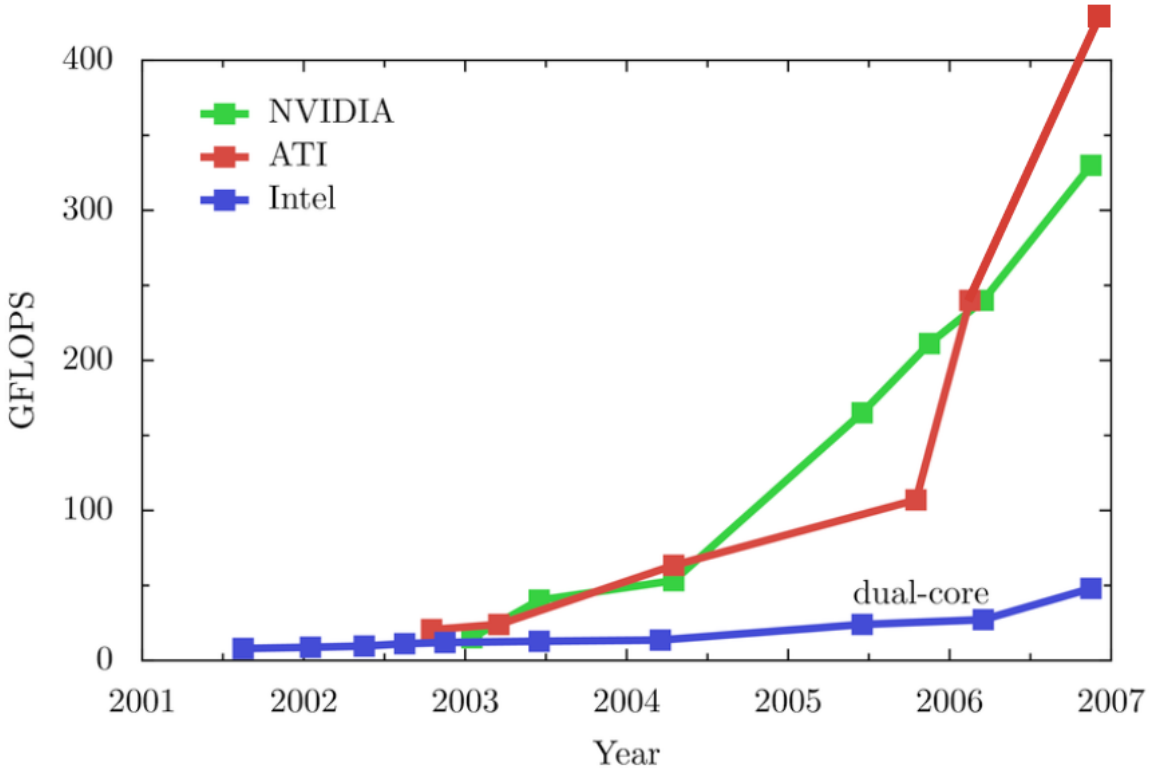


Figure 7 Floating point operation increase of GPUs and CPUs in the past 6 years (Figure Courtesy: www.gpgpu.org)

If these performance gains could be harnessed either on a single computer, a cluster, or a network of workstations (common in many companies and academic institutions), problems currently requiring enormous computational resources could be solved on commodity hardware. As identified in the introduction, large-scale, multi-objective optimization offers tremendous benefits to companies and researchers, if they have access to immense computational resources. By taking advantage of the power of GPUs, a new source of resources, already available, can become practically usable.

2.5 Research Issues

Although substantial research has been done in optimization over the past 50 years, industry has not been able to adopt it into their design processes to realize its benefits to the fullest extent. Below is a partial list of reasons for this slow adoption rate:

- Time: Without tangible and documented return on investment, a company will often incorporate optimization into their design processes
- Computing costs: Procuring computing technologies to solve optimization problems in an industry's design processes may not be cost effective
- Trust: A designer's optimization model may not incorporate various aspects of the problem and its accuracy is not guaranteed. Therefore, an industry trusts on a designer's experience better than a formal mathematical optimization model
- Awareness: Either an industry is unaware of the available tools to solve their optimization problems or their design problems are too complex to be solved by established optimization methods
- Competition: In a monopolized market, an industry does not have the incentive to incorporate optimization to improve on their design processes further

When successfully implemented in a design process, optimization has the potential to have a large impact on the quality, cost, and time for a product or process design. Increasing demand and competition will drive the use of optimization in industry, but it requires making optimization tools more practical and viable. The research presented in this dissertation attempts to address these possibilities and provide designers with robust tools to help improve their design processes.

Based on the needs defined and background material reviewed, three research issues have been identified. They are:

1) How can the solution quality, accuracy, and reliability of PSO be improved in identifying global optimum in multimodal n-dimensional design spaces?

Two drawbacks currently inhibit the performance of basic PSO in converging to a global optimum in an n-dimensional design space. The first drawback is that the particle updates are influenced by a limited number of factors. At any instance, each swarm member is directed by only two current or past swarm locations – *pBest* and *gBest*. Having just these two candidates impedes the desirable exploratory characteristics. In an n-dimensional design space, information from these two candidates alone will not always suffice to propagate a swarm toward the global optimum efficiently. A second drawback is that the method is initial condition dependent. Poor locations specified by *pBest* and *gBest* in the initial stages of optimization can potentially offset the swarm from attaining the neighborhood of the solution in the design space. This results in the swarm either being trapped in a local minimum or taking substantial time to recover from a bad location and reach the global optimum. Through the use of digital pheromones, PSO variations can be developed that can robustly explore and exploit design spaces for both unconstrained and constrained optimization problems. It is theorized that these methods will offer significant improvements in terms of solution quality and accuracy.

2) How can the solution efficiency for PSO be improved for a faster global convergence in multimodal n-dimensional design spaces?

Although simple to implement, PSO is computationally quite intensive, particularly due to a substantial number of function evaluations by the swarm members. Additionally, the involvement of digital pheromones to improve the search efficiency adds additional computations per iteration. Coarse and fine grain parallelization strategies will be developed as a part of the second research issue to significantly increase solution efficiency.

3) How can commodity graphics hardware be utilized to accelerate the optimization process in PSO?

Section 2.4 explains the ubiquitous availability of commodity graphics hardware and its potential for large-scale mathematical computations in less time and cost than comparable CPUs. These features will be exploited to investigate the feasibility of solving multimodal n-dimensional optimization problems in a CPU-GPU architecture.

3 Digital Pheromones in PSO

3.1 Rationale

In a basic PSO algorithm, the swarm movement is governed by the velocity vector computed in equation (2). Each swarm member is therefore, essentially presented with information obtained from two specific locations from the design space at any iteration. However, multiple pheromones released by the swarm members could provide much more information on promising locations within the design space when the information obtained from *pBest* and *gBest* are insufficient or inefficient. This is the primary thrust and premise for the research presented in this thesis.

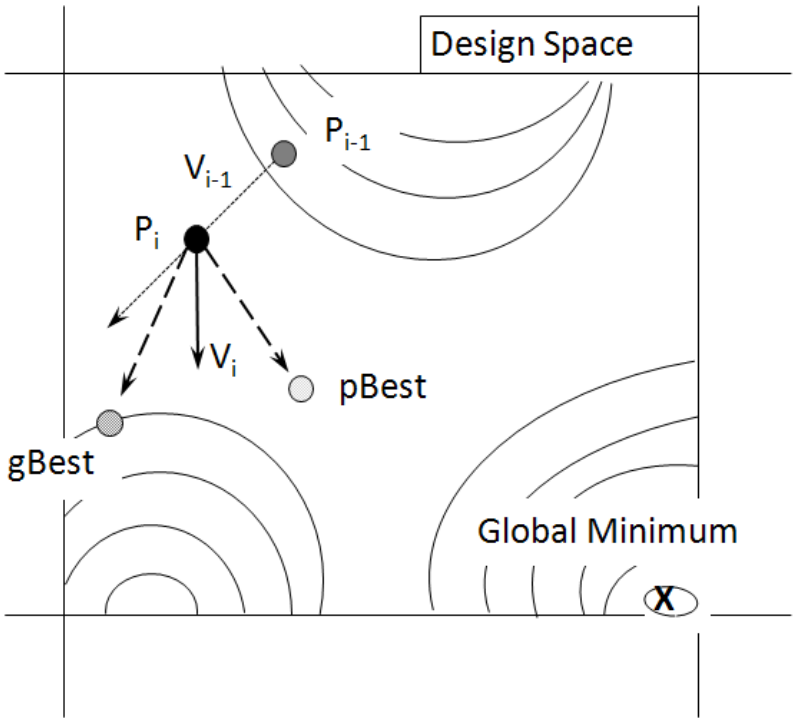


Figure 8 Particle movement in a basic PSO

Figure 8 displays a scenario of a swarm member's (P_i) movement whose direction is guided by *pBest* and *gBest* alone. The previous position of the particle is denoted by P_{i-1} and the previous velocity component is indicated by V_{i-1}. If $c_1 >> c_2$, the particle is attracted primarily towards its personal best position. On the other hand, if $c_2 >> c_1$, the particle is strongly attracted to the *gBest* position. In the scenario dominated by c_2 as presented in Figure 8, neither *pBest* nor *gBest* leads the swarm member to the global optimum, at the very least, not in this iteration adding additional computation to find the optimum.

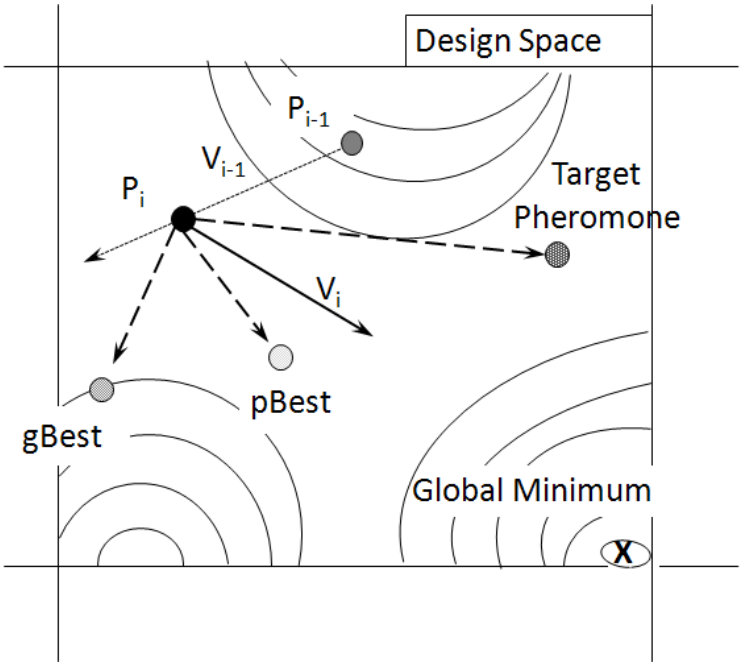


Figure 9 Particle movement with digital pheromones

Figure 9 shows the effect of implementing digital pheromones into the velocity vector. An additional pheromone component potentially causes the swarm member to result in a direction different from the combined influence of *pBest* and *gBest* thereby increasing the probability of finding the global optimum.

3.2 Method Overview

Figure 10 summarizes the procedure for PSO with steps involving digital pheromones highlighted.

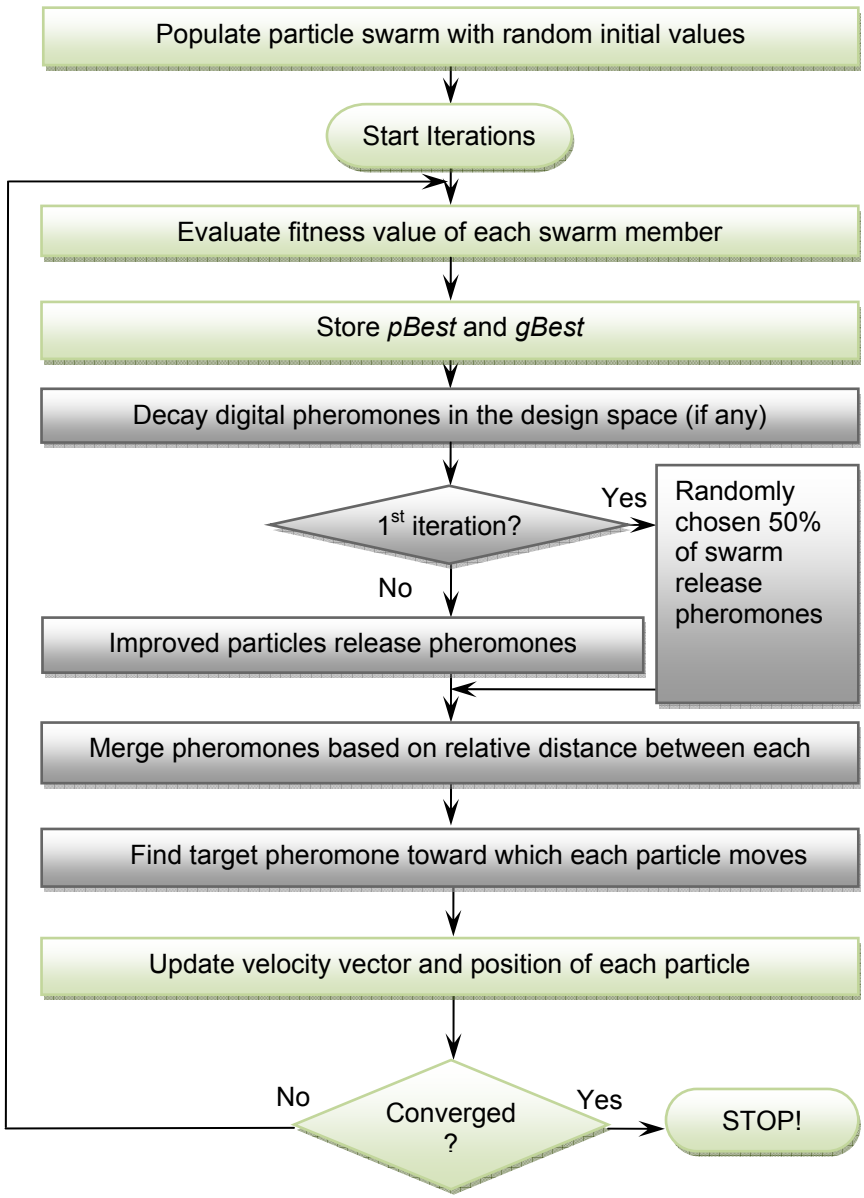


Figure 10 Overview of PSO with Digital Pheromones

The method initialization is similar to a basic PSO except that 50% of the swarm within the design space is randomly selected to release pheromones in the first iteration. This parameter is user-defined, but experimentation has shown 50% to be a good default value. For subsequent iterations, each swarm member that finds a better location releases a pheromone. Pheromones from the current as well as the past iterations that are close to each other in terms of the design variable value are merged into a new pheromone location. In addition, the digital pheromones are decayed in each iteration just as natural pheromones. This effectively creates a pheromone pattern across the design space while still keeping the number of pheromones manageable. Each distinct pheromone is then given a probability based on its pheromone level and its position relative to a particle. This probability is then used in a ranking process to select a target pheromone for each particle in the swarm. The target position for each particle will be an additional component of the velocity vector update in addition to *pBest* and *gBest*. Following this, the objective value for each particle is recalculated and the entire process is continued until a prescribed convergence criteria is satisfied.

3.3 Digital Pheromone Initialization and Merging Process

In order to populate the design space with an initial set of digital pheromones, 50% of the population is randomly selected to release pheromones, regardless of the objective function value. This is done to ensure a good spread of digital pheromones across the design space thus leading to effective swarm exploration. For subsequent iterations, the objective function value for each particle in the population is evaluated and only particles

finding an improvement in the objective function value when compared to the current *gBest* value will release a pheromone. Any newly released pheromone is assigned a level *P*, with a value of 1.0. The pheromone levels are normalized between 0.0 and 1.0. Just as natural pheromones produced by insects decay in time, a user defined decay rate, λ_P, (defaulting to 0.95), is assigned to the pheromones released by the particle swarm. Digital pheromones are decayed as the iterations progress forward to allow a swarm member to propagate toward a better design point by increasing the chances of attraction to a newer pheromone location with a better objective function value.

Every particle that finds a solution improvement releases a pheromone potentially making the number of pheromones unmanageably large as iterations progress. Therefore, an additional step to reduce them to a manageable number, yet retaining the functionality, was implemented. Pheromones that are closely packed within a small region of the design space are merged together. Figure 11 shows an example merging process in a 2D design space.

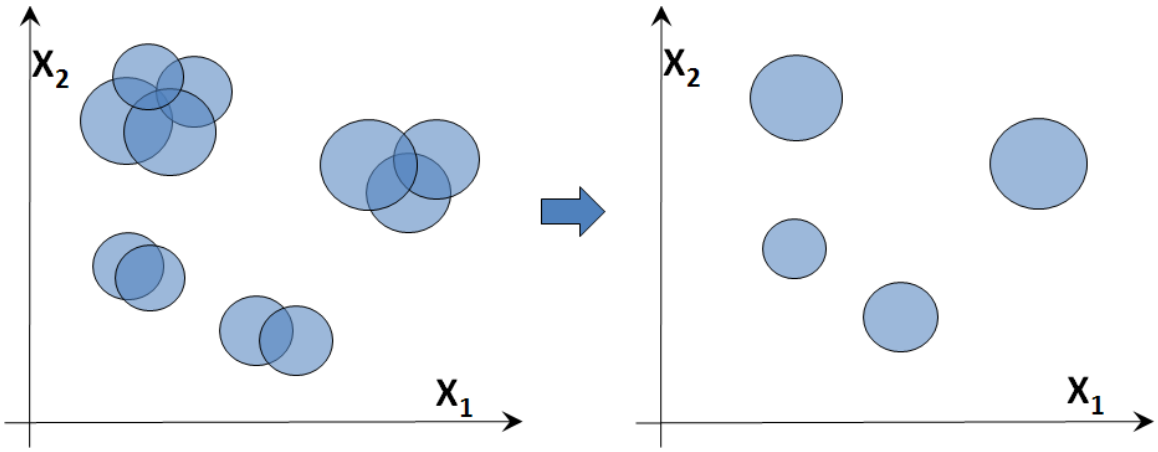


Figure 11 Merging of Digital Pheromones

To check for merging, each pheromone is associated with an additional, 'Radius of Influence' (*ROI*). For each design variable of a pheromone, an *ROI* is computed and stored. The value of this *ROI* is a product of the pheromone level and the range of the design variables. Any two pheromones for a design variable less than the sum of the ROIs are merged into one. This is analogous to two overlapping spheres merging into one. The average strength of the two merging pheromones is retained in the resulting pheromone. The location of the resultant pheromone is biased towards the stronger of the two merging pheromones. Through this approach, regions of the design space with stronger resultant pheromone levels will attract more particles and therefore, pheromones that are closely packed would indicate a high chance of optimality. Also similar to the pheromone level decay, the *ROI* also has its own decay factor, λ_{ROI}, whose value is set equal to λ_P as a default. This is to ensure that both the pheromone levels and the radius of influence decay at the same rate. Figure 12 is a flow chart illustrating the pheromone merging process.

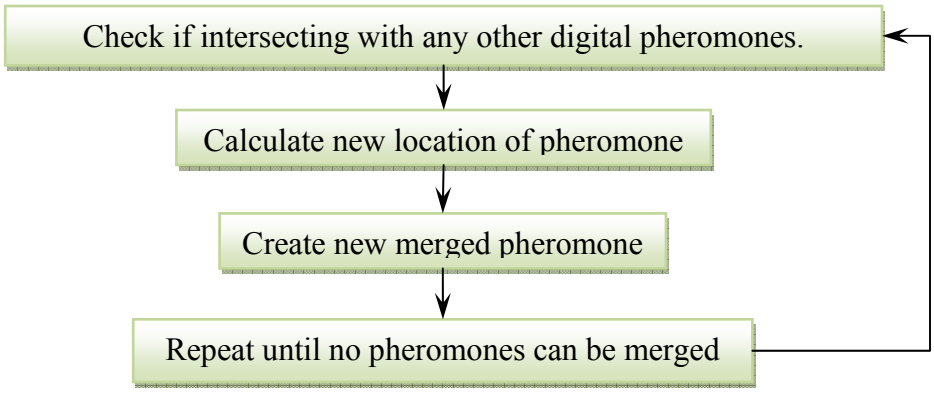


Figure 12 Flowchart of pheromone merging process

3.4 Proximity Analysis to Determine Target Pheromone

With numerous digital pheromones generated within the design space, a target pheromone needs to be identified for each swarm member. A criterion that is a function of both the pheromone level and its proximity from each particle needs to be considered in selecting the target pheromone. This is based on: a) the distance between a particle and pheromone, and b) the pheromone level. For each particle, a target pheromone attraction factor *P'* is computed to this effect, which is a product of the pheromone level and the normalized distance between the particle and the pheromone. Equation (5) shows how the attraction factor *P'* is computed, and equation (6) computes the normalized distance between the pheromone and each particle in the swarm. The variable $range_k$ is the difference in the upper and lower limits of k^{th} design variable.

$$P' = (1-d)P \tag{5}$$

$$d = \sqrt{\sum_{1}^{k}\left(\frac{Xp_k - X_k}{range_k}\right)^2}, k = 1:n \quad \#\text{ of design variables} \tag{6}$$

Xp – Location of pheromone

X – Location of particle

Figure 13 shows an example scenario of a particle being attracted to a target pheromone from a pool of four merged pheromones, each having a pheromone level and are at variable distance from the particle in the design space. The target attraction factor, *P'* is computed for each of these pheromones and rank ordered. The particle in the figure is attracted to pheromone number that has the highest *P'* value (in this case, pheromone 4) based on its proximity to other pheromones and their pheromone levels.

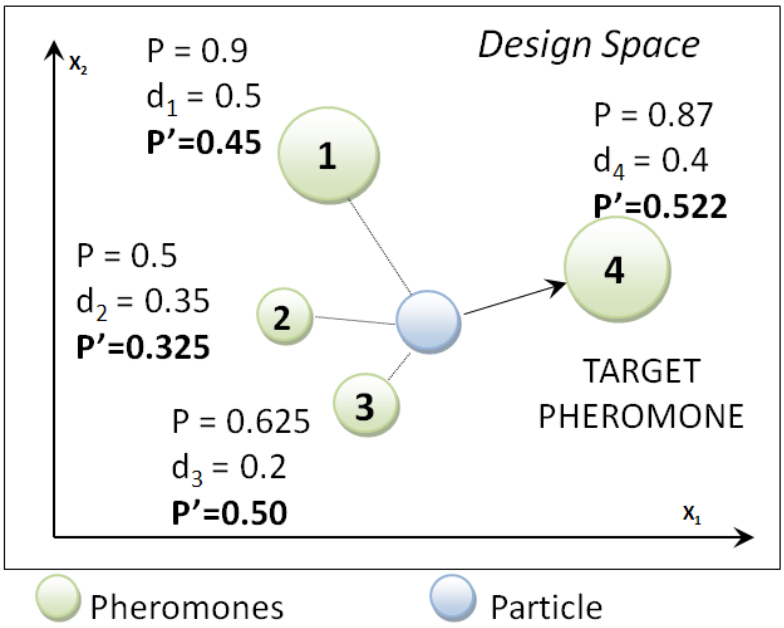


Figure 13 Illustration of target pheromone selection

3.5 Velocity Vector Update

Upon determining the target pheromone, the velocity vector from basic PSO is updated with a new component called the target pheromone component as shown in equation (7).

$$\begin{aligned} V_{iter+1,i}[] = {} & w_{iter} \times V_{iter,i}[] \\ & + c_1 \times rand_p() \times (pBest_i[] - X_i[]) \\ & + c_2 \times rand_g() \times (gBest[] - X_i[]) \\ & + c_3 \times rand_T \times (Target Pheromone_i[] - X_i[]) \end{aligned} \tag{7}$$

In equation (7), c_3 is a user defined confidence parameter for the pheromone component of the velocity vector similar to c_1 and c_2 in a basic PSO. c_3 combines the knowledge from the cognitive and social components of the velocity of a particle, and complements their deficiencies. The confidence parameter c_3 determines the extent of influence a target

pheromone can have on the swarm when the information from *pBest* and *gBest* alone are not sufficient or efficient to determine a particle's next move.

$rand_T$ is a random number generated between 0 and 1. Random numbers generated by computers are of two types: (a) Pseudo-Random Number Generators (PRNGs), and (b) True Random Number Generators (TRNGs). PRNGs are algorithms that use mathematical formulae or pre-calculated tables to produce sequence of numbers that appear random and hence are typically efficient. PRNGs are generally used for modeling and simulation. TRNGs on the other hand extract randomness from physical phenomena (roll of a die, atmospheric noise in thunderstorms, etc) and induce better random characteristics. TRNGs are typically preferred over PRNGs in applications where unpredictability is very important (data encryption, etc). They are lesser efficient than PRNGs [96]. In PSO, the influence of the swarm movement is weighted primarily by *pBest* and *gBest*. Therefore, the use of random numbers within the velocity vector equation does not considerably affect the outcome if PRNGs are used. Since efficiency is a significant concern in PSO, TRNGs are typically not used.

3.6 Geometric Interpretation of Target Pheromone and Confidence Parameter, c_3

In a basic PSO, the particle swarm does not have a memory of the entire path traversed in the design space apart from the best position of an individual particle (*pBest*) and the best member's position in the entire swarm (*gBest*). The target pheromone component addresses this issue. It is a container that functionally stores the trail path of the swarm and utilizes the

best features of it in steering towards a promising location in the design space. The use of the target pheromone relies heavily on *pBest* and *gBest*. If $c_3 = 0$, there is no influence of pheromones and the swarm behaves as if in a basic PSO. If either of c_1 or c_2 is 0 and $c_3 > 0$, then the target pheromone location is essentially determined only by the non-zero component of *pBest* or *gBest* and propagated into the velocity vector. This creates a bias thereby doubling the influence of non-zero *pBest* or *gBest* components on the swarm. This means that the swarm either explores or exploits the design space with double the intensity, either of which can prevent the swarm from converging. It is therefore essential that the influence of *pBest* and *gBest* be balanced (i.e. equal) for the pheromone component to provide accurate assistance in reaching the optimum.

Although analytical determination of a value for c_3 is out of the scope of this research, an empirical value has been determined through experimentation. A value between 2.0 and 5.0 has shown good performance characteristics and solved a variety of problems. The results chapter (chapter 7) will provide insight on why the chosen values were found to be favorable for a variety of problems.

A higher value of c_3 causes the velocity vector's magnitude to increase and places the swarm in a more general exploratory mode. However, it is desirable to make the swarm perform a tighter, local search as the swarm approaches the optimum. In this case, a lower value of c_3 is desirable. Therefore, decreasing c_3 can potentially help the swarm to move from an exploratory mode to an exploitation mode. To achieve this effect, a decay of c_3 has been investigated in this research in addition to a constant c_3, to adapt to the swarm movement as

required. Automatic adaptation of the confidence parameters is not new. Literature shows the use of such approaches in basic PSO as well [97] [98]. The results chapter (chapter 7) provides an explanation of how useful was c_3 decay in various test problems.

An inertia weight, w_i of value 1.0 is initially chosen to preserve the influence of the velocity vector from previous iterations, and gradually decreased using an inertia weight decay factor similar to the one used in a basic PSO.

3.7 Move Limits

The additional pheromone term in the velocity vector update can considerably increase the computed velocity. This increase can potentially cause the solution to diverge if left unchecked. To address this, a move limit was imposed on the maximum value of the velocity vector's magnitude. To ensure a fair amount of freedom in exploring the design space, the swarm is allowed to digress up to 10% of the range of the design variables initially. A move limit decay factor of $\lambda_{ML} = 0.95$ is applied in subsequent iterations. While the move limit alone imposes a bound on the velocity vector's magnitude, the move limit decay factor further fine tunes the swarm towards a local search. This means that the swarm is free to explore the design space in the beginning and confines it for a local search towards the end. The magnitude of the velocity vector is multiplied by λ_{ML} in each iteration.

3.8 Statistical Significance of Digital Pheromones

The implementation of digital pheromones caters for improved performance of PSO in terms of accuracy, efficiency and reliability. Section 7.3 in the results (chapter 7) demonstrates this capability. However, a quantitative assessment of the developed method needs to be made to prove this claim. Therefore, it is necessary to perform statistical hypothesis testing to prove that particle swarms with digital pheromones perform better than without pheromones (basic PSO). This section explains the procedure involved and the results are discussed in section 7.4 of chapter 7.

3.8.1 Statistical Hypothesis Testing

In statistical terms, a population is a group or individual that represents all members of a certain category of interest. A sample is a subset drawn from the population. Descriptive statistics apply only to the members of a sample of data collected from the population. Inferential statistics, on the other hand refer to the use of sample data to reach conclusions about the characteristics of the population that the sample represents. A hypothesis is typically a statement about the parameters in a population distribution. It is called as hypothesis because it is not known whether the statement is true or not. The primary objective of hypothesis testing is to test whether or not the values of a random sample from the population is consistent with the claimed hypothesis or not. The hypothesis is considered 'accepted' if the random sample is consistent with the hypothesis under consideration. Otherwise, the hypothesis is 'rejected' [99] [100] [101] [102].

Within the context of this research, it is necessary to claim that digital pheromones when implemented in PSO perform better when compared to basic PSO in terms of solution accuracy and solution times.

The hypothesis that specifies a particular value for the parameter being studied is called the null hypothesis and is denoted by H_0. It represents the standard operating procedure of a system or a known procedure. The hypothesis that specifies those values of the parameter that represent an important change from standard operating procedure or known procedure is called the 'alternative hypothesis' or 'research hypothesis', and is denoted by H_a. Evidence from a result sample inconsistent with the stated hypothesis leads to rejection of the hypothesis, whereas evidence supporting the hypothesis leads to its acceptance. In statistical hypothesis testing, it is a norm that the acceptance of a proposed hypothesis is the result of insufficient evidence to reject it.

There are two ways that errors can be committed in the decision process using hypothesis testing. A type I error is committed if the null hypothesis is rejected when it is actually true. A type II error is committed if the null hypothesis is not rejected when it is actually false. Table 2 shows the truth table of decision making while performing hypothesis testing. The probability of committing a type I error is called the level of significance of the test and is denoted by α, and the probability of committing a type II error is denoted by β.

Table 2 Decisions and Errors in Hypothesis Testing

	Decision	
	Reject H_0	**Accept H_0**
H_0 True	Type I Error	Correct decision taken
H_0 False	Correct decision taken	Type II error

Hypothesis testing can be one-tailed or two-tailed. For example, H_0: $\mu = \mu_0$ and H_a: $\mu \neq \mu_0$ is called a two-tailed hypothesis where the equality of μ and μ_0 are tested. On the other hand, H_0: $\mu < \mu_0$ and H_a: $\mu \geq \mu_0$ (or) H_0: $\mu > \mu_0$ and H_a: $\mu \leq \mu_0$ is called a one-tailed test, where μ represents the population mean and μ_0 represents the sample mean. A t-test assesses whether the mean of two groups are statistically different from each other, and is an especially appropriate tool when comparison of the means of two different group of parameters is desired. The t-distributions are affected by the sample size, and they approach normal distributions with large sample sizes.

3.8.2 Hypothesis Testing Procedure

The following is a five-step procedure adopted for performing hypothesis testing of PSO with and without digital pheromones:

1. The null and alternate hypotheses are to be defined.

Hypothesis testing can be single-sample based or multi-sample based. In a single sample, the null and alternate hypothesis will have parameters only from the problem under consideration. A two-sample test on the other hand allows for comparison of means of two different methods (e.g., with and without digital pheromones). Since the objective is

to investigate the performance characteristics of PSO with and without digital pheromones, a one-tailed hypothesis test is performed. The null and the alternate hypotheses are defined as shown in equation (8):

$$H_0: \mu_1 - \mu_2 \leq 0 \text{ (null hypothesis)}$$
$$H_a: \mu_1 - \mu_2 > 0 \text{ (research or alternate hypothesis)} \quad (8)$$

Where, μ_1 and μ_2 corresponds to the means obtained from basic PSO and digital pheromone respectively. Therefore from equation (8), the null hypothesis H_0 signifies that the mean objective function values and solution times for basic PSO are statistically smaller than those obtained using digital pheromone PSO. Conversely, the research hypothesis H_a from equation (8) signifies that the mean objective function values and solution times for basic PSO are statistically larger than those obtained using digital pheromone PSO. Within the context of this research, the research hypothesis H_a is desired to be accepted, a possibility that can happen only when H_0 is rejected.

2. A level of significance equal to α needs to be chosen.

A 95% confidence level is chosen for hypothesis testing in this research. This means the hypothesis test is performed with a 0.05 probability for type I error. This is the most commonly used confidence level for statistical testing in general.

3. An appropriate test statistic (i.e., t) is to be selected and its corresponding critical value ($t_{critical}$) is to be obtained from t distribution tables.

Depending upon whether there is any dependency between the data samples obtained for PSO with and without pheromones, the test can be either independent or paired. Since the

test runs for basic PSO and digital pheromone PSO are performed independent of each other and have different random seed values for each during trial runs, independent two-sample hypothesis testing is performed. Therefore, the test statistic (or the t-value) is calculated using equation (9) (a standard t-value estimator whose description can be looked up in any standard statistics textbook), where $\bar{x}_1$ and $\bar{x}_2$ represents the means of basic and pheromone PSO respectively.

$$t = \frac{(\bar{x}_1 - \bar{x}_2)}{S_p \sqrt{\frac{1}{n_1} + \frac{1}{n_2}}} \tag{9}$$

$$S_p{}^2 = \frac{(n_1 - 1)s_1{}^2 + (n_2 - 1)s_2{}^2}{n_1 + n_2 - 2} \tag{10}$$

Where equation (10) represents the square of the standard deviation or the variance of the sample data from basic and pheromone PSO, with $(n_1 + n_2 - 2)$ degrees of freedom.

4. The value of the statistic (t) is to be computed from the random sample of size, n.

Most t-distribution tables consider degrees of freedom greater than 30 as an accurate approximation of a normal distribution. For statistical analysis in this research, 35 trial runs will be performed each for basic and pheromone PSO. The number of degrees of freedom for this hypothesis test is $(n_1 + n_2 - 2) = 68$, where n_1 represents the sample size of results from basic PSO, and n_2 represents the sample size of results obtained from digital pheromone PSO. This means that the data can be considered as normally distributed for all statistical testing purposes.

5. *H_0 is to be rejected if the statistic has a value in the critical region; otherwise H_a is to be rejected.*

A hypothesis is accepted if there is no evidence to reject it. If the value of '*t*' calculated from equation (9) is greater than $t_{critical}$, H_0 is rejected. If the value of '*t*' is less than $t_{critical}$, H_a needs to be rejected. The value of $t_{critical}$ is obtained from t-distribution tables corresponding to the probability of error chosen in step *2*.

3.9 Further Improvements

The use of digital pheromones provides substantial information about the design space, thereby increasing the solution accuracy, efficiency and reliability of particle swarms. Although the computational expense due to pheromone operations increase per iteration, the benefits due to additional design space information offsets this drawback by allowing the solution to converge in substantially less iterations. Chapter 5 demonstrates the capability of particle swarms augmented by information from digital pheromones.

In addition to the use of digital pheromones for improving solution characteristics of PSO, parallelization can further enhance the efficiency in searching multi-dimensional design spaces. This research takes advantage of PSO's inherent capability for parallelization. Chapter 4 describes the rationale followed by various parallelization strategies implemented for improving the performance of PSO.

4 Parallelization on Computer Clusters

4.1 Rationale for Parallelization

PSO is generally computationally intense. Additionally, pheromone operations further increase the number of computations per iteration. This is particularly apparent with larger swarm sizes on highly multimodal problems. One way to reduce the computational overhead is to strategically distribute independent tasks in the method into different processes. On a computer workstation, each of these processes can be handled by independent tasks called threads. Threading is effective for single processor workstation operations but is not capable of providing the computational horsepower for highly multimodal problems with a large number of design variables requiring large swarm sizes. Therefore, alternate computational techniques become necessary.

Fortunately, computing technologies have sufficiently advanced to provide affordable access to high performance cluster computing, which when used appropriately can prove to be a suitable alternative for improving particle swarm efficiencies. Parallelization is one such means where tasks can be distributed on multiple processors in a cluster of computers instead of multiple threads on a single workstation. Cluster computing increases solution efficiencies and not only reduces the computational burden on a single processor but also caters for additional computations if needed (e.g., multiple swarms). Communication between processors can be achieved through parallel application Programming Interfaces (APIs) such as MPI [65] or PVM [103] layers. These are

industry standard APIs that are simple to implement and effectively distribute information between processors thereby easing the computational intensity on a single processor. The communication network in a computer cluster is typically managed through Infiniband [104] [105] or Myrinet switches [106] [107]. PSO is a natural fit for parallelization, primarily due to the fact that it is population based and each swarm member is independently capable of traversing in the design space regardless of the whereabouts of the remaining swarm members.

This chapter explains various parallelization schemes explored in this research to determine if solution efficiencies can substantially be improved with digital pheromones when compared to a basic PSO algorithm.

4.2 Synchronous Coarse Grain Parallelization

Multiple independent swarms traversing the design space independently can garner significantly more information on the design space than a single swarm with a larger population size. This is because: (a) each deployed swarm is considerably smaller in size and the communication costs (computational overhead) are smaller and (b) each swarm independently explores the design space increasing the diversity in search and thereby eliminates the pitfalls of following a single leader (*gBest*).

On a single processor workstation, an equivalence of '*n*' independent swarms can be achieved by a serial execution of the code with each swarm deployed one after another.

In this case, it potentially takes at least '*n*' times the number of seconds for each swarm to report results. Another approach would be to deploy '*n*' independent swarms simultaneously on a computer workstation through a threaded code. In this approach, a processor can spawn multiple processes, each handling an independent swarm. However, the processor load increases substantially thereby resulting in degraded performance and increased solution times.

Simultaneous deployment of multiple swarms on different processors can dramatically reduce solution times. In this scenario, '*n*' swarms are deployed simultaneously on '*n*' different processors, where one design space is explored by groups of independent swarms, each reporting their solutions when converged. The solutions resulting from each processor can then be sorted to determine the actual best solution. Where a serial code potentially takes '*np*' seconds to solve a problem using '*p*' swarms, an ideally formulated parallel code would only take '*n*' seconds when '*p*' swarms are delegated each to a processor, dramatically reducing the solution time. However, parallelization comes at a considerable expense – network latencies. Communication between processors is currently limited by the available network technologies and every instance of a data transfer between processors is as fast as the slowest network connection. Therefore, benefits can be reaped only when the communication between the processors is kept to a minimum and synchronous coarse grain parallelization scheme is designed to do just that. Figure 14 shows a schematic of the developed synchronous parallel coarse grain decomposition of PSO with digital pheromones.

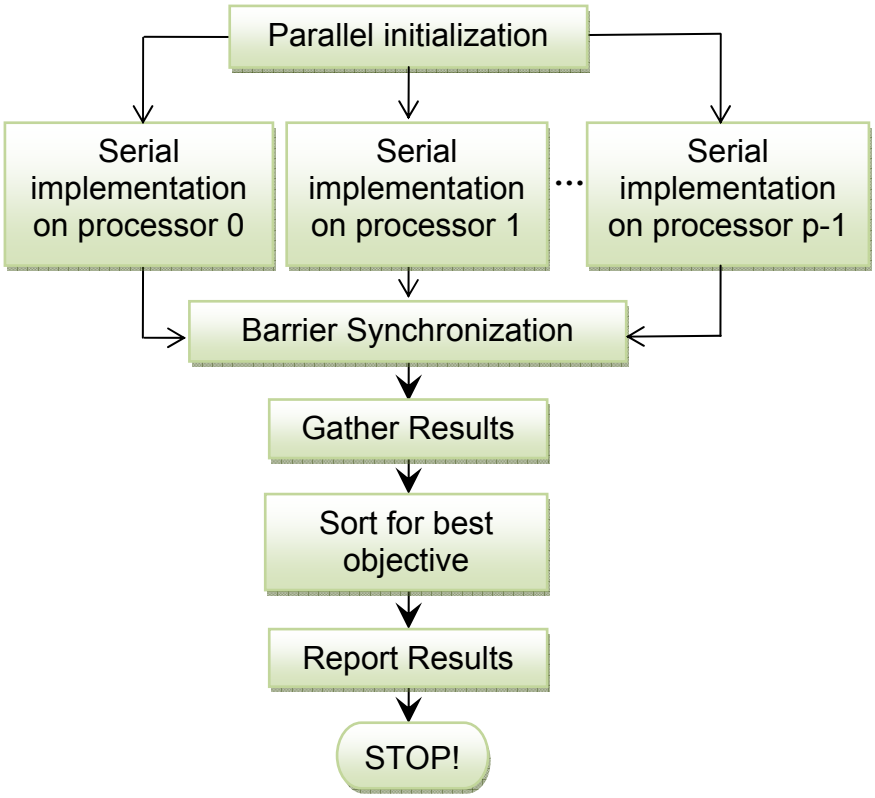


Figure 14 Schematic of synchronous coarse grain parallelization

In this approach, each participating processor runs an identical copy of the serial PSO code with digital pheromones with its own randomly initiated population swarm. Upon determining the velocity direction and updating the particle positions, each processor performs a convergence check and arrives at the participating processor's optimal point. This means that each processor containing a swarm determines its own *gBest*. Using barrier synchronization, optimal points from all the processors are synchronously gathered on the root processor and sorted for the best combination of the objective function and its corresponding design variables. Data communication between the processors takes place only at the end to gather each processor's optimal point and sort for the global optimum point. Until this point, there is no exchange of information

between participating processors. This approach potentially avoids the primary parallelization bottleneck – network latency. Therefore, the chances of locating the global optimum increase with the number of processors.

While it is true that coarse grain parallelization offers substantial advantages when compared to serial execution, each processor is unaware of the progress of the solution status of every other processor. Communication between the swarms on multiple processors substantially improves the chances of finding an optimum. However, the network latency costs due to exchange of information between swarm members across processors typically defeats the purpose of communication by substantially increasing the solution times. Each instance of data transfer between processors is only as fast as the slowest network connection. Moreover, the use of barrier synchronization causes the participating processors to wait until solutions are obtained from each processor. This means that the swarms on processors that find a solution stand idle until solutions are obtained from all other processors. This is an inefficient use of computational resources that could be more efficiently used with a suitable parallelization procedure. Therefore, a parallelization method that fosters the communication between swarms yet that retains latency costs to a manageable level is desired. This idea is explored through the idea of pheromone sharing across processors, a second parallel scheme.

4.3 Shared Pheromone Parallelization

A parallelization strategy has been developed where a swarm is deployed across multiple processors, similar to a coarse grain approach. However, the available processors are

divided into two categories: 1) optimization processor(s) and 2) a pheromone processor. The optimization processors are a function of the desired number of swarm members on each processor. Each optimization processor performs: a) random swarm generation, b) fitness value evaluation and pheromone release, c) calculation and storage of *pBest*, d) target pheromone calculation, e) velocity vector calculation, and f) particle position update. Figure 15 shows a flow chart of the developed method. The pheromone processor gathers a list of pheromones released by all participating optimization processors. They are then merged and decayed (for iteration numbers greater than one) appropriately as well. Therefore, the pheromone processor is a dedicated processor that exclusively performs pheromone operations and maintains a finalized repository of pheromones shared by multiple swarms spread across the design space on various optimization processors. Additionally, the pheromone processor also ranks from *gBest* candidates, called *processor-gBest*, sent by each optimization processor to find the actual *gBest* of all the swarms. Since the final pheromone list and actual *gBest* information takes up a tiny amount of memory, their broadcast to all optimization processors does not use a significant amount of network bandwidth. The pheromone processor performs the convergence check since it contains the most updated *gBest* information. If a specified convergence check is evaluated to true, this message is broadcast to all optimization processors upon which the code execution stops on all processors, and results are reported.

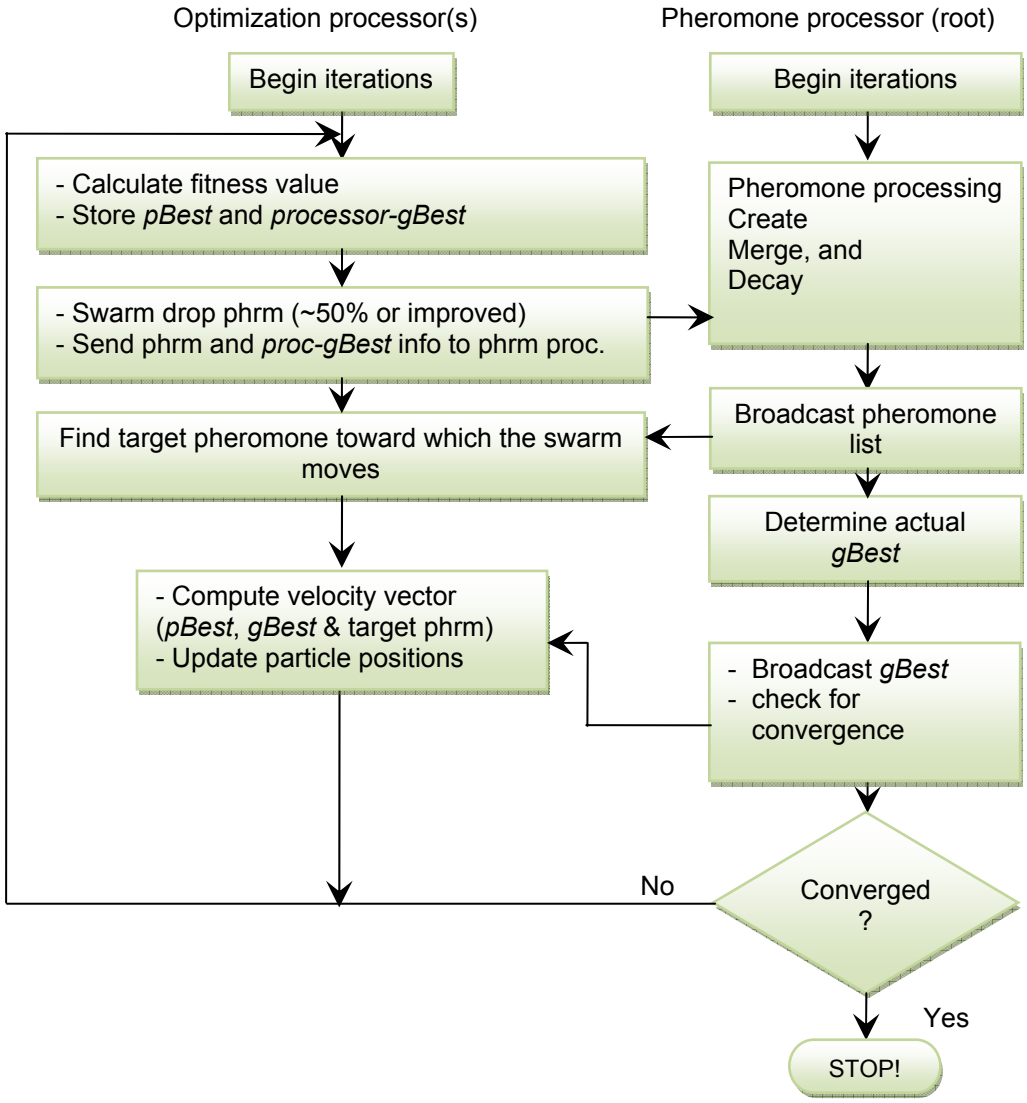


Figure 15 Shared pheromone parallel implementation flowchart

It is to be noted that the target pheromone calculations are performed on the optimization processors, and not on the pheromone processor. This is because a target pheromone is unique to each swarm member. Communicating this information from each optimization processor to the pheromone processor would cause significant network latencies slowing down the solution progress. In addition, the pheromone processor will not likely be able

to handle target pheromone computations for each swarm member of swarms from all processors efficiently.

There exists a two-fold advantage of the developed approach: 1) network latency costs between optimization processors are curtailed since they do not communicate with each other, and 2) the pheromone processor computes, stores and broadcasts the global pheromone list and the actual *gBest* to all optimization processors fostering communication between processors. Moreover, the participating processors do not have to synchronize at any point meaning that the method does not idle during any part of the code execution. Therefore, this approach combines the elements of information exchange between multiple swarms for improved search efficiency as well as reduced communication overhead across participating processors for better solution times.

5 Parallelization on Commodity Graphics Hardware

5.1 GPU Parallelization

Commodity Graphical Processing Units (GPUs), commonly known as graphics cards or video cards were traditionally used for visualization purposes until recently. A user could control various parameters in a graphics code, but the underlying functionality and sequence of operations were fixed. In recent years, this fixed functionality has been replaced with the capability to perform not only graphical operations but also general purpose computing. In 2004, the industry open standard OpenGL 2.0 API was released providing a formal channel for programmability of vertex and fragment shading operations under core OpenGL specifications [108]. Along with a hardware programmable component, hardware advancements has made GPUs capable general purpose processors capable of very high computational speeds for a variety of scientific applications. Their speed is attributed to their highly data parallel architecture. GPUs take advantage of their hardware parallelism, meaning that computations can be performed on multiple data simultaneously based on the Single Instruction Multiple Data (SIMD) technique.

Although the programmable functions in GPU are graphical in context, the underlying operations are mathematical. Since these operations can be performed dramatically faster than on a traditional CPU, GPUs are increasingly becoming the mainstream for scientific and computation intense operations. Figure 16 is a very simplified view of a fixed

function graphics pipeline containing relevant information on data traversal from within the graphics application to the frame buffer. A frame buffer is the region of the graphics memory that is modified as a result of OpenGL rendering. In a general sense, the frame buffer corresponds to an OpenGL rendering in a window.

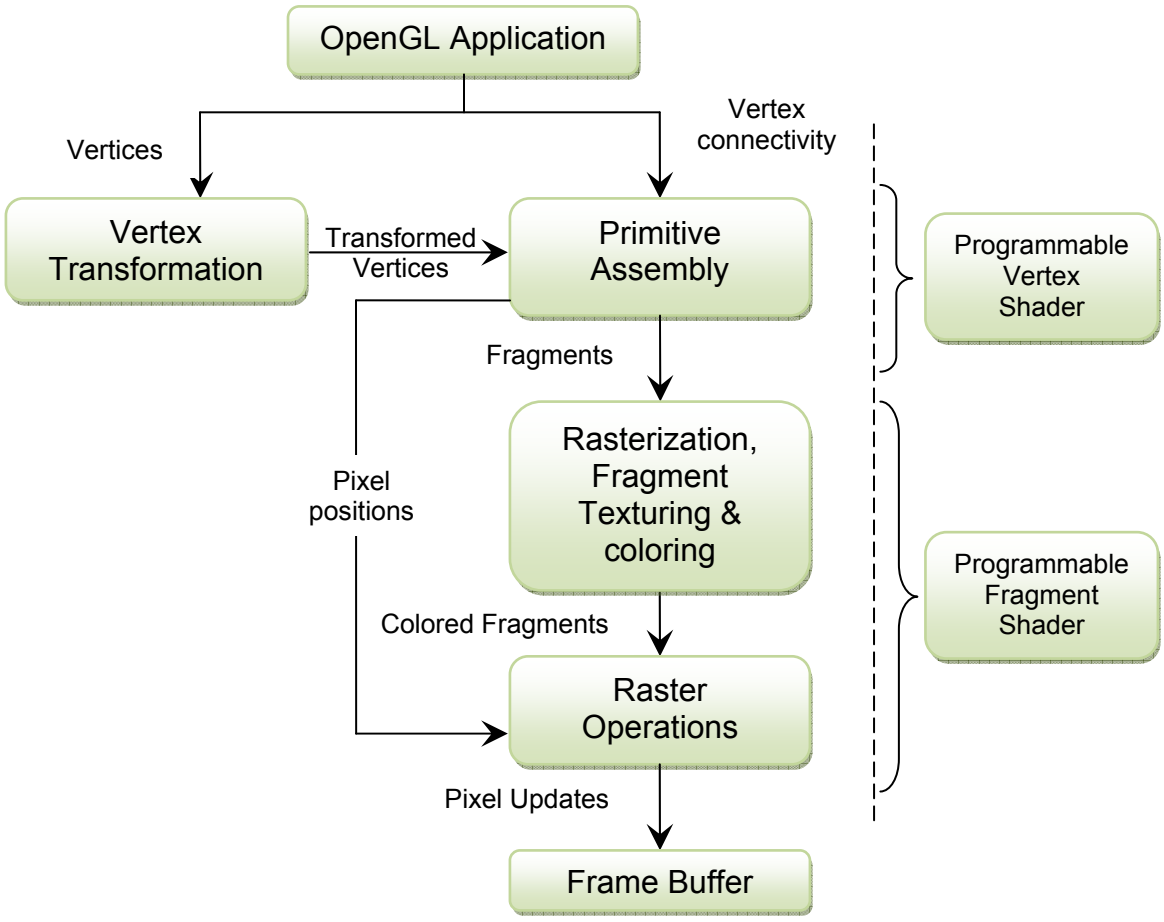


Figure 16 Simplified Graphics Pipeline (programmable components indicated)

In the vertex transformation component, the input vertices are appropriately transformed and passed to the assembly component where the vertices are assembled into a geometric primitive. Also, per vertex operations such as lighting, texture coordinates, clipping

against view frustum are computed in these components. Geometric primitives that passed through the primitive assembly component in the pipeline are decomposed into smaller units corresponding to pixels in the destination frame buffer in a process termed rasterization. Each decomposed small unit is called a fragment. For example, if a line covers 10 pixels on the screen, rasterization converts the line geometry information obtained from vertex primitive assembly component into 10 fragments. Each of these fragments is then subjected to various fragment processing operations such as texture mapping, fog, and coloring. The last stage of the graphics pipeline includes performing various per-fragment operations such as pixel ownership test, scissor test, alpha test, stencil test, and the depth test. The underlying operations for vertex and fragment processing are essentially mathematical and can be replaced by programmable vertex and fragment shaders as indicated on the right side of the Figure 16. Figure 17 is a visual summary of the various stages involved in vertex and fragment processing as explained above.

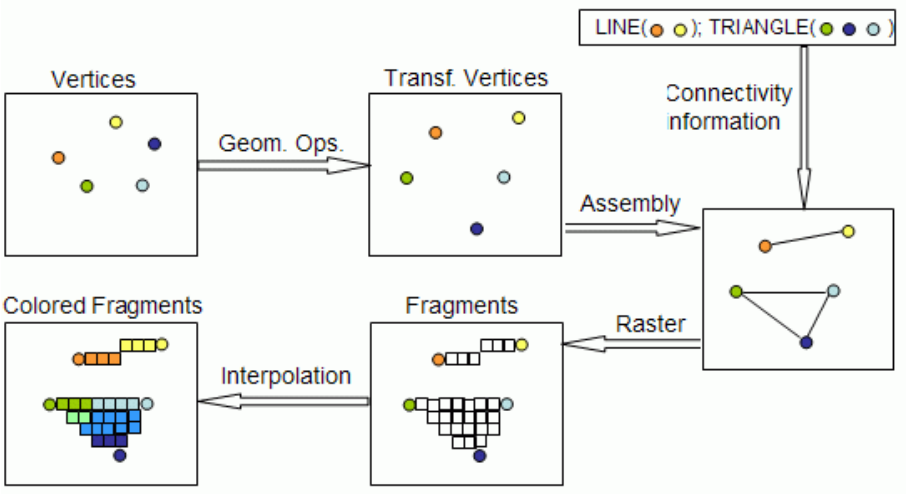


Figure 17 Visual Summary of a Fixed Functionality Graphics Pipeline
(Figure Courtesy: www.lighthouse3d.com)

5.2 Choice of GLSL as Shading Language

As outlined in section 2.4 of chapter 2, there are a handful of shading languages available to interface with graphics hardware. From the available choice of shading languages, GLSL was chosen for this research for the following reasons:

1. It is a high-level shading language that integrates directly with the OpenGL standard.
2. It is designed with intent for expansion and increased usability in the future. For example, current day graphics cards do not support double precision real valued data types but the pace of their advancements potentially support them in the near future. GLSL specifications support for such future developments and hence adaptation can be made with minimal alterations to vertex or fragment shaders.
3. It is cross platform compatible. Therefore, the shader can be re-used on workstations running different operating systems without any change in the code.
4. It supports most GPU chip makers (e.g. NVIDIA, ATI). With minor hardware alterations, GLSL can be used on a wide variety of GPUs.
5. It closely resembles C/C++ in its programming syntax.
6. It has in-built functions and reserved data types that are graphics in context and are derived from OpenGL. This means a non-graphical developer might have a considerable learning curve before realizing the full potential of GLSL. However, when compared to operating system specific (e.g., Microsoft Accelerator, HLSL, etc) or GPU hardware specific (e.g., CUDA) shaders, GLSL provides the flexibility of working with various operating systems and graphics hardware.

5.3 Vertex and Fragment Shaders

Both vertex and fragment shaders can provide hardware acceleration for execution of specific portions of a PSO code. However, marked differences between the two necessitate careful consideration of how to proceed. Output from a vertex shader is sent as input to the fragment shader (as seen in the graphics pipeline, Figure 16 and Figure 17), which in turn produces usable output to the main application. In other words, using a vertex-shader is a two-step process. Output from the fragment shader can directly be passed into the main application. Additionally, the fragment shader computes interpolated pixel values for the data provided from the vertex shader causing a possible loss of data or precision. Therefore, a logical choice is to use a fragment shader for this research.

5.4 Formulation for GPU Computations

Shaders typically work very well with two dimensional textures (analogous to 2D arrays on CPUs). Although 1D and 3D arrays are supported by GPUs, it is typically faster to compute and operate on 2D textures. Since the primary data holders in PSO are swarm members and their locations in the design space, it is a logical first step to create a 2D texture that can hold the design variable values for all swarm members. Older OpenGL releases (pre 2.0) are compatible only with square textures (i.e. of size 2^n – 32, 64, 128, etc). Therefore, a 2D texture of size 40 x 55 previously required creation of a texture of size 64 x 64 where unused texture coordinates would be filled with zeroes. Although this approach is not a very efficient procedure, it previously served as a good work around to

deal with operations on non-square textures. The latest release of OpenGL however addresses this issue and can handle arbitrary rectangular textures, where texture memory can be fully utilized, and hence used for implementation in this research.

The first step in transferring data to the GPU is to prepare OpenGL for off-screen rendering through a Frame Buffer Object (FBO). Graphical objects typically are represented by 8-bit precision each for red, green, blue and alpha channels on a graphics window (computer screen). The purpose of a frame buffer object is to set up off-screen computations in a 32-bit floating-point precision manner and eliminate 8-bit precision for the red, green, blue and alpha channels. The next step is to define appropriate arrays and textures for facilitating inputs and outputs between CPUs and GPUs. The format of the textures created is GPU hardware specific. For example, the texture format on an NVIDIA GPU is denoted by '`GL_FLOAT_R32_NV`' and a texture format on ATI GPU is denoted by '`GL_RGBA_FLOAT32_ATI`'. Additionally, an orthogonal projection and a viewport are needed to provide a one-to-one correspondence between geometry coordinates (used in rendering) and texture coordinates (data input) and pixel coordinates (data output). All these parameters can be set while initializing the FBO.

Design variables for each swarm member are stored in an array and uploaded into the GPU memory as a rectangular texture. The design variable values for each swarm member are filled into each column of the rectangular texture. Figure 18 shows an example 'design variable texture' of size *n*x*m* with the data entry and storage sequence indicated by dashed arrows within the cells.

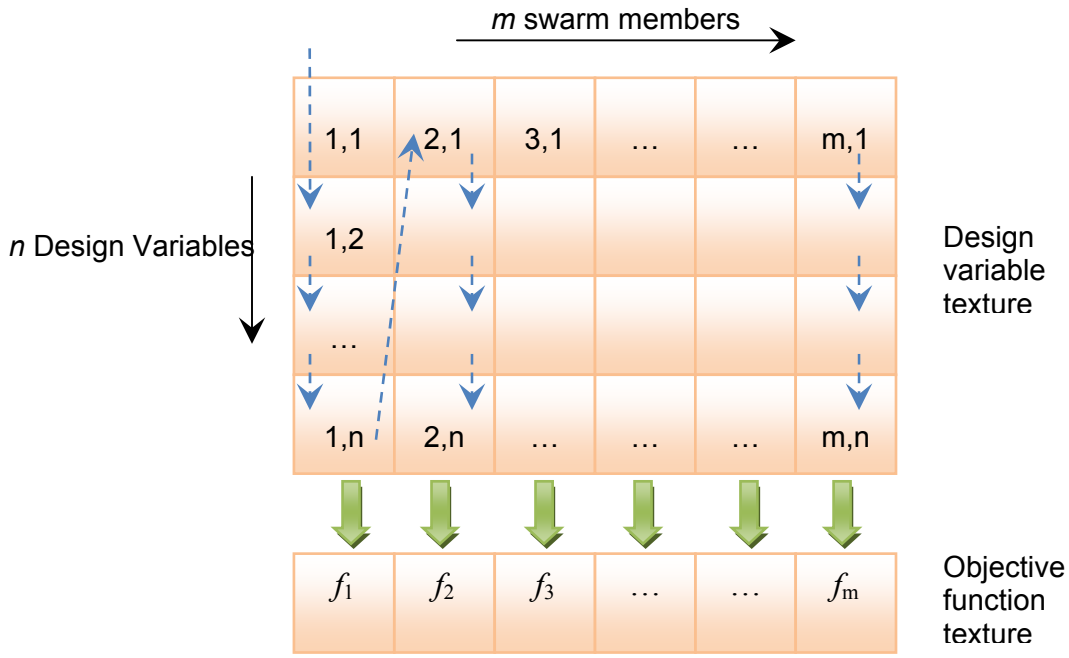


Figure 18 Data Entry Sequence in a Texture and its Use for Objective Function Evaluation

In the design variable texture, '*m*' is the number of swarm members and '*n*' is the number of design variables. The lower rectangular 'objective function texture' of size *1*x*m* holds the objective function values computed from each column of swarm members *1* through *m* from the design variable texture (Multiple Data). Each objective function texture entry requires a column of information (*1* through *n*) from the design variable texture.

5.5 GPU Implementation

In a PSO optimization routine, the bulk of the computational work comes from objective function evaluations. Thus, it was theorized that if objective function evaluations were delegated to the GPU, the efficiency of PSO would increase due to its data parallel

architecture. Although the costs of accessing the main memory on a CPU for input/output of data into the GPU are high, the benefits of data parallelization would outweigh these CPU-GPU network latencies. An overview of the GPU implementation of PSO with digital pheromones is outlined in Figure 19.

The GLSL initialization phase includes preparing the GPU for computations within the framework explained in section 5.4. Therefore, this stage involves defining and creating textures for off-screen computations. Design variables for each swarm member are stored into an array that automatically fills the design variable 2D texture as explained through Figure 18. The fragment shader is then invoked to perform per-pixel objective function evaluations. The fragment shader program consists of instructions to compute the objective function and is executed via rendering a quadrilateral to an off-screen buffer initialized in FBO. Therefore, with a single instruction, computations are performed on multiple data (swarm members) at once to compute the objective function.

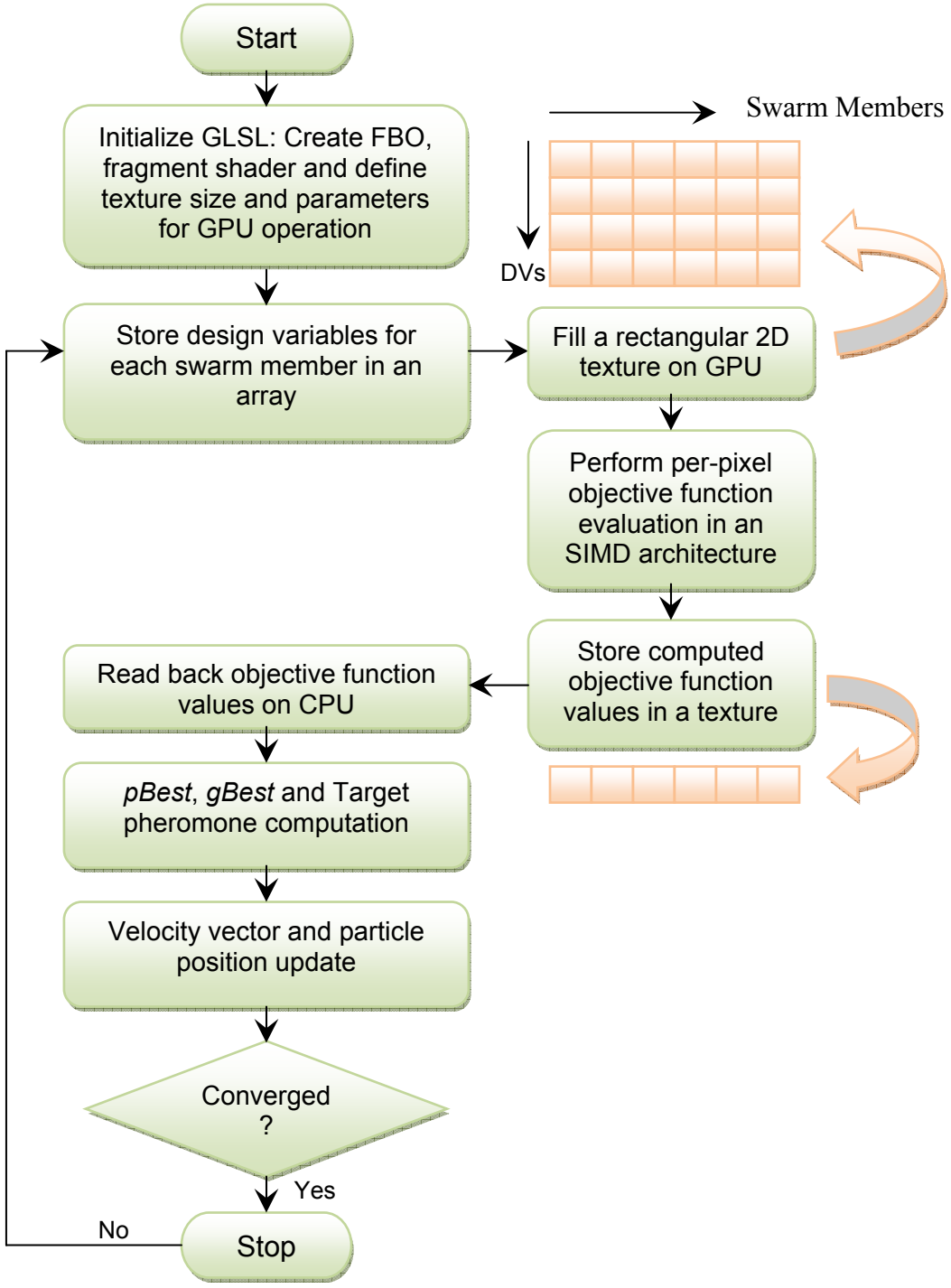


Figure 19 Flowchart for GPU Hardware Acceleration of PSO with Digital Pheromones

5.6 Percentage of GPU Vs CPU Usage

Current generation GPUs are not equipped to perform double precision floating point operations. Hence, the GPU implementation for this research is limited to single precision floating precisions. Therefore, depending upon the objective function sensitivity, there may be loss in precision. To account for this, the developed code is designed to compute objective function values on CPU and GPU on a percentage basis. This means that a user can specify the percentage of objective function evaluations that can take place on the CPU and GPU. For example, if a high precision is desired, a 30% GPU-CPU percentage can be specified where objective function are evaluated three out of 10 iterations on a GPU and seven out of 10 iterations on a CPU. Conversely, a user could specify a 90% GPU-CPU percentage where objective function evaluations on nine out of 10 iterations are carried out on a GPU and one iteration is carried out on the CPU, if efficiency gains are a more important goal.

5.7 Implementation Specifics

During the initial implementation stages, temporary array variables were defined to store design variable values and then used in computing per-pixel objective function values. Therefore, a temporary array of size 10 was defined to compute the objective function value of dimensionality 10. Though this approach did not pose a problem when solving lower dimensional problems, the GPU ran short of temporary internal registers as the dimensionality of the objective function increased. Registers are place holders for

converting GLSL code into a machine specific list of instructions. The available number of these registers is typically limited by the graphics hardware type. For example, the GPU used for the test problems supported only 32 internal registers but the Griewank function required more since there were 50 design variables and hence returned an error. To solve this problem, the temporary array variable definition was replaced by defining a single variable that performed a run-time texture look-up, which avoided redundant use of internal registers. This procedure turned out to be faster and efficient means to perform GPU computations.

When comparing solutions from CPU and GPU in the initial stages of GLSL implementation, it was observed that a number of trial runs on the GPU resulted in identical solution values (including design variable values and solution times). However, this was not observed on the CPU implementation. The solution values resulting from a CPU are independent in each trial run. Apparently, each GPU trial run did not have enough information to generate a distinct random seed for random number generation of design variables on the CPU. To avoid this problem, the seed for random initialization of design variable values was made a function of the current trial run, thereby ensuring a different seed in each run. This forced a different seed value in each run resulting in a distinct solution in each GPU trial run.

6 Constrained Optimization

The methods developed and explained in chapters three through five provide a promising potential for digital pheromones to solve n-dimensional multimodal unconstrained optimization problems. However, realistic design problems are usually characterized by numerous inequality and equality constraints. To be considered as a practical optimization tool, it is imperative to prove that digital pheromones can effectively assist PSO in solving constrained optimization problems as well. This chapter is dedicated to applying digital pheromones within PSO to solve constrained optimization problems.

6.1 Methods to Solve Constrained Problems

A general constrained problem with a single objective function F(**X**) is given by equation (11) as shown below, where g(**X**) represents '*m*' inequality constraints and h(**X**) represents '*l*' equality constraints:

Minimize,

$$F(\boldsymbol{X})$$

Subject to:

$$g_j(\boldsymbol{X}) \leq 0,\ j=1,\ m \qquad (11)$$

$$h_k(\boldsymbol{X}) = 0,\ k=1,l$$

A solution to a constrained optimization problem should necessarily satisfy three optimality conditions laid out by Kuhn and Tucker in 1950s [109] as shown in equations (12) (13) and (14). These conditions have been the guiding principles to solving any

constrained optimization problem. Figure 20 shows a 2-D objective function with four inequality constraints.

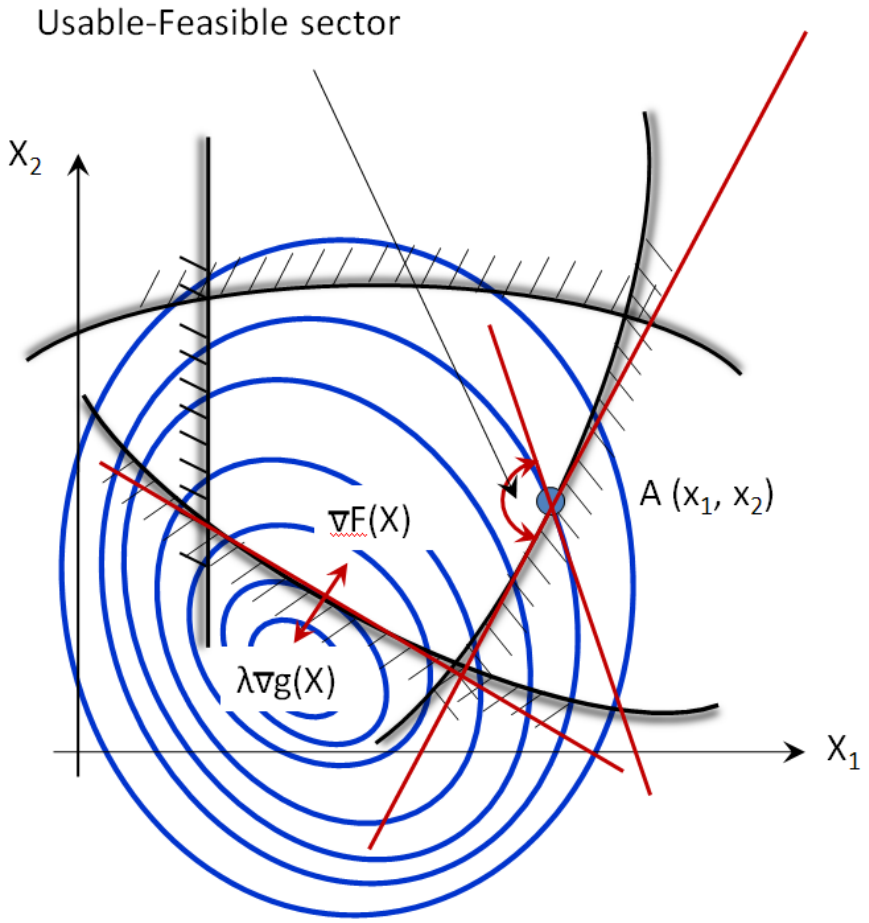


Figure 20 Optimality conditions for a constrained optimization problem

The first criterion that a design point should satisfy for being a solution is that it should be within the feasible region. This is given by Kuhn-Tucker's first condition of optimality as shown in equation (12).

$$\mathbf{X}^* \text{ (design vector that minimizes } F(\mathbf{X}) \text{) is feasible} \qquad (12)$$

Also, the product of λ_j and $g_j(\mathbf{X})$ should be zero as denoted by equation (13). λ_j is called the Lagrange multiplier for the j^{th} constraint. A Lagrange multiplier indicates the rate at

which the objective function value changes with a corresponding rate of change in the constraint value. When a constraint is active, the value of $g_j(X)$ becomes zero and the corresponding λ_j becomes non-zero. If a constraint is satisfied but not active, the Lagrange multiplier reduces to zero while the corresponding $g_j(\mathbf{X})$ is non-zero.

$$\lambda_j g_j(X^*) = 0, \quad j = 1:m, \lambda_j \geq 0 \tag{13}$$

To improve in the design, the point A (X_1, X_2) should move in the direction of decreasing objective function while being in the feasible region. This region is shown by the usable-feasible sector in figure 19. If a design point is not within this usable-feasible region, it either violates a constraint or increases the objective function or both. Let a direction 'S' denote a direction that the point 'A' takes to improve in the design. At the point of optimality, the direction of 'S' is perpendicular to the tangent made by the objective function contour and the constraint boundary so that $\nabla F(X)$ and $\lambda \nabla g(X)$ are exactly equal and opposite to each other. This is given by Kuhn-Tucker's third condition of optimality as shown in equation (14). This third condition (14) governs that no further move is available that will decrease the objective function while maintaining constraint feasibility.

$$\nabla F(X^*) + \sum_{j=1}^{m} \lambda_j \nabla g_j(X^*) + \sum_{k=1}^{l} \lambda_{m+k} \nabla h_k(X^*) = 0$$
$$\lambda_j \geq 0$$
$$\lambda_{m+k} \text{ unrestricted in sign} \tag{14}$$

A number of methods have been developed in the past to solve n-dimensional constrained optimization problems including sequential linear programming/cutting plane method [110], the method of feasible directions [111], and generalized reduced gradient method

[112] [113]. Another more popular approach for solving constrained optimization problems are through employing Sequential Unconstrained Minimization Techniques (SUMT).

As the name SUMT indicates, solving constrained optimization problems requires the solution of several unconstrained minimization problems, where the original constrained problem is typically substituted by a sequence of unconstrained sub-problems, called pseudo objective functions. The general strategy to solve constrained optimization problems would be to minimize the pseudo objective function as an unconstrained problem but impose penalties for constraint violations.

A pseudo objective function is shown in equation (15), where F(X) is the original objective function and P(X) is the penalty function whose form depends on the SUMT technique used. 'r_p' is a scalar that determines the magnitude of the penalty imposed on constraint violations.

$$\Phi(X) = F(X) + r_p P(X) \tag{15}$$

6.1.1 Exterior Penalty Function Method (EPF)

The EPF methods typically yield feasible optimum values for extremely large r_p values but potentially yields numerically ill-conditioned formulations, and hence are generally avoided in numerical methods, especially population based heuristic methods. On the

other hand, interior penalty function methods have the potential to reach discontinuous spaces, especially at constraint boundaries.

The EPF method is the simplest to implement, which penalizes the objective function when constraints are violated. A typically used penalty function P(X) in an exterior penalty function method is given by equation (16) below [1] [2] [3].

$$P(X) = \sum_{i=1}^{m} \left\{\max\left[0, g_j(X)\right]\right\}^2 + \sum_{k=1}^{l} \left[h_k(X)\right]^2 \qquad (16)$$

It can be seen from the equation that no penalty is imposed when all constraints are satisfied, but the square of the constraint is included when one or more constraints are violated. With a smaller value for 'r_p', the pseudo objective function $\Phi(\mathbf{X})$ can easily be minimized but potentially yields large constraint violations. On the other hand, a large value for 'r_p' can ensure near satisfaction of all constraints but can potentially be a numerically ill-conditioned problem. Therefore, 'r_p' is started small and increased by a small factor and $\Phi(\mathbf{X})$ is minimized each time beginning the optimization from the previous solution.

In addition to possible numerical ill-conditioning, another important disadvantage with the EPF method is that any optimization routine that is stopped prematurely could be unusable because the design points move from infeasible to feasible regions, and a design point used before convergence is not guaranteed to satisfy the constraints.

6.1.2 Interior Penalty Function Method (IPF)

The IPF method, as opposed to EPF method can provide a series of improving designs with each pseudo function optimization. IPF method penalizes the objective function as the design points approach constraints, and violations are not allowed. Thus, all design points during a solution run are feasible. The penalty function typically used in the IPF method is shown in equation (17) below [1-3]:

$$P(X) = \sum_{i=1}^{m} \left\{ \frac{-1}{g_j(X)} \right\} \tag{17}$$

$$\Phi(X) = F(X) + r_p' \times P(X) + r_p \times \sum_{k=1}^{l} [h_k(X)]^2 \tag{18}$$

Equation (18) shows the pseudo objective function of IPF. The second term on the right hand side in equation 15 is used to penalize inequality constraints and the third term is used to penalize equality constraints. The inequality penalty term introduces a new penalty parameter, r_p'. This term goes from a larger to smaller value (e.g., 20 to 1 during a solution run), while the r_p penalty parameter increases from small to large, exactly as it does in the EPF method.

Although the design points in this method are always in the feasible region and improving in objective function value every iteration, this comes at the cost of creating more complex minimization problems. Also, care must be taken to avoid function discontinuities at the boundaries of $g_j(\mathbf{X}) = 0$ in the pseudo objective function.

6.2 Augmented Lagrange Multiplier (ALM) Method

Extended interior penalty function methods (linear extended penalty function [114] [115], quadratic extended penalty function [116], variable penalty function methods [117]) incorporate the best features of interior and exterior penalty function methods, but still suffer from many of the same drawbacks as EPF and IPF. The Augmented Lagrange Multiplier (ALM) method is another SUMT method with distinct advantages over other constrained minimization techniques and is explained in this section.

The ALM method was originally developed for addressing equality constrained problems and later extended to solve inequality constraints. For a Lagrangian developed for an equality constrained problem, as shown in equation (19), the Kuhn-Tucker conditions require that the stationary conditions of L(X, λ) and feasibility requirements are the necessary conditions for optimality.

$$L(X,\lambda) = F(X) + \sum_{k=1}^{l} \lambda_k h_k(X) \tag{19}$$

This also means that the minimum of the Lagrangian subject to the equality constraints defined in the problem provides the solution to the original objective function. Thus, a pseudo objective function can be built from equation (19) that can be solved using an exterior penalty function approach. For an equality constrained problem, the pseudo objective function is given by equation (20), where $A(X, \lambda, r_p)$ is referred as the augmented Lagrangian.

$$A(X,\lambda,r_p) = F(X) + \sum_{k=1}^{l} \{\lambda_k h_k(X) + r_p [h(X)]^2\} \tag{20}$$

With a similar explanation for inequality constraints as well, the general augmented Lagrangian for a constrained (inequality and equality) problem is given by equation (21) [1-3].

$$A(X,\lambda,r_p) = F(X) + \sum_{j=1}^{m}[\lambda_j \Psi_j + r_p \Psi^2{}_j] + \sum_{k=1}^{l}\{\lambda_{k+m} h_k(X) + r_p [h_k(X)]^2\} \quad (21)$$

$$\Psi_j = \max\left[g_j, \frac{-\lambda_j^p}{2r_p}\right] \quad (22)$$

Therefore, minimizing the augmented Lagrangian is equivalent to minimizing the original objective function when the Kuhn-Tucker's necessary conditions for optimality are imposed. The first summation term after F(X) in equation (21) correspond to inequality constraints where, ψ [118] is given by equation (22). The second summation term in equation (21) corresponds to the penalty function term for equality constraints.

To minimize the augmented Lagrangian, a penalty factor of 'r_p', on each constraint, is imposed, just as any other penalty function method. With appropriate Lagrange multipliers (λ) known, one unconstrained minimization of the pseudo objective function is sufficient. Since these multipliers and penalty factors 'r_p' are typically unknown before hand, a series of unconstrained minimizations are carried out to arrive at the appropriate Lagrange multipliers and hence the solution of the actual objective function.

The update relations for the Lagrange multipliers, λ are shown in equation (23) for inequality constraints and equation (24) for equality constraints.

$$\lambda_j^{p+1} = \lambda_j^p + 2r_p \max\left[g_j, \frac{-\lambda_j^p}{2r_p} \right], j = 1, m \tag{23}$$

$$\lambda_j^{p+1} = \lambda_j^p + 2r_p h_k(X), k = 1, l \tag{24}$$

Some of the distinct advantages of using ALM are: a) the inclusion of Lagrange multipliers to speed convergence, b) the penalty parameters play far less important role in determining convergence and therefore the method is insensitive to the value of r_p, c) the starting design vector need not necessarily be feasible, and d) the non-zero values of Lagrange multipliers identify active constraints automatically. Together with the geometric significance of ALM for equality and inequality constraints, the above advantages over other penalty function methods make it a very favorable candidate for implementation in PSO with digital pheromones.

A series of unconstrained problems are required to be formulated and PSO implemented on each to arrive at an appropriate combination of Lagrange multipliers that minimizes the original objective function. The first approach would be is to arrive at a combination of Lagrange Multipliers at the end of solving a pseudo objective function, and use it for the subsequent pseudo objective function. On the other hand, if a reasonable approximation in solving the pseudo objective function is acceptable, it is possible for more frequent Lagrange multiplier updates through limiting the number of pseudo objective function iterations or imposing a loose convergence criterion on the pseudo objective function. Since there are more Lagrange multiplier updates, the probability of finding the optimum combination of the multipliers that can solve the actual objective

function increase and hence a faster solution time. In this research, both approaches are investigated, and the results are presented.

The description for ALM is well explained in Gill et al [3], and has been used as the basis for the development of the method. Figure 21 shows a flow chart of the ALM implementation in PSO with digital pheromones. At the beginning, a population swarm with an initial selection of design variables **X**, Lagrange multipliers 'λ's, and penalty parameter 'r_p' for each of the design constraints, a positive integer 'a_{max}', serving as an upper bound of the total number of unconstrained minimizations and 'p_{max}' to limit the number of iterations during pseudo objective function minimization are initialized. Upon evaluating the actual objective function value, a feasibility check followed by a convergence check is performed to determine if it attained the optimum while satisfying all constraints. If either of feasibility or convergence is not attained, values of the 'λ's and 'r_p' are updated. The algorithm stops if convergence is achieved or if the number of unconstrained minimizations exceeds the maximum limit (a_{max}). A problem is converged when the difference in solutions is within a tolerance for certain number of consecutive iterations. With updated 'λ's and 'r_p', a new unconstrained pseudo objective function is constructed and solved using PSO with digital pheromones.

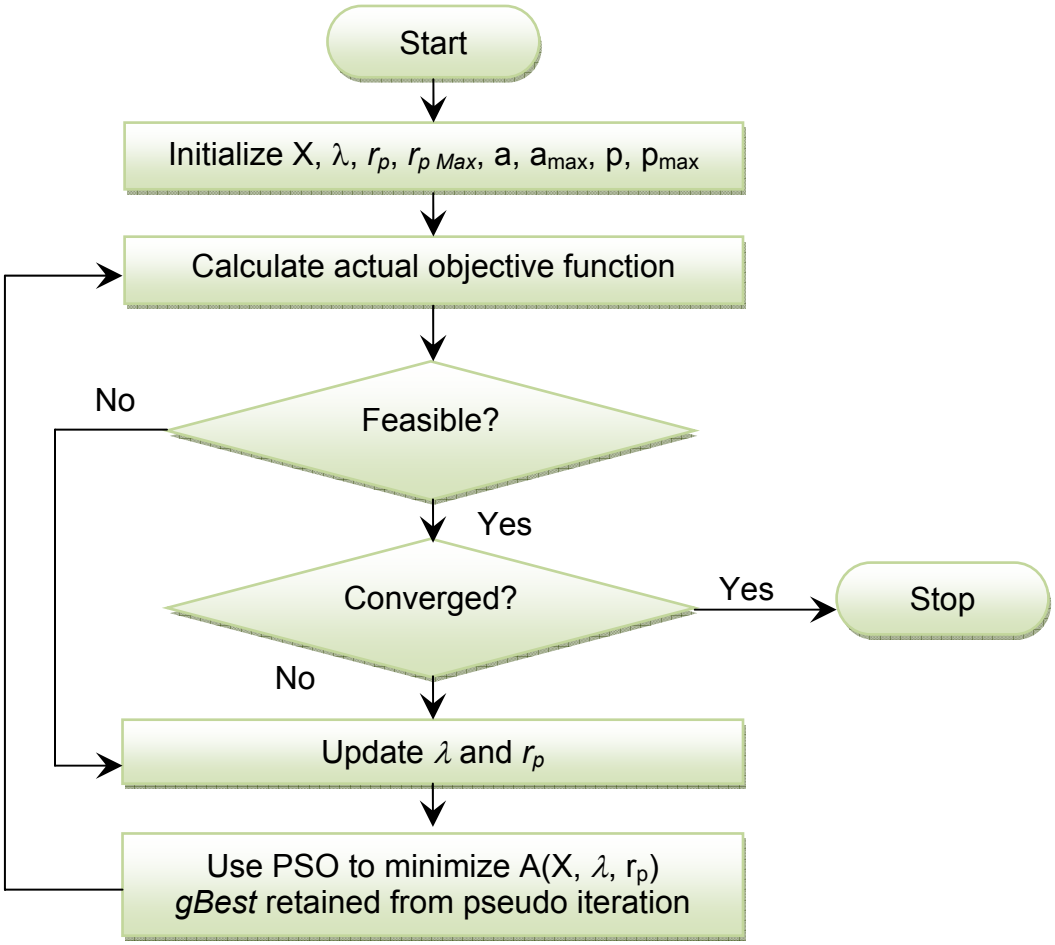


Figure 21 Flowchart for ALM implementation in PSO with digital Pheromones

Two approaches were implemented to handle solution of the pseudo objective functions: a) Pseudo objective function was solved to convergence before proceeding to the next pseudo formulation and b) a limit was put on the number of pseudo iterations to facilitate more frequent Lagrange multiplier updates. Apart from this change, the remainder of the pseudo objective function solution adheres to the process outlined in Figure 10.

With a single design point searching the design space, as in the case of most deterministic methods, the solution from a current pseudo objective function minimization is used as an

input for the next pseudo minimization. Since there is a population of points in PSO, the *gBest* from the current pseudo objective minimization is used in the subsequent pseudo objective minimization and the remaining swarm members are randomly initialized. Therefore, out of 'n' swarm members, (n-1) are randomly initialized and one swarm member is retained (the *gBest* from previous pseudo minimization). The solution is converged if the difference in actual objective function values over a set number of iterations were within a specified tolerance limit, provided the constraints were satisfied.

When the constraints are satisfied during the convergence check, the penalty values are decreased by a factor of 0.5 and increased by 2.0 when they are violated. This formulation is applied to both inequality and equality constraints and was adopted from Sedlaczek and Eberhard [119]. Equations (25) and (26) portray these update schemes with the lowest values for 'r_p' bounded at 1.0.

$$r_{p,j}^{a+1} = \begin{cases} 2 \times r_{p,j}^{a} & if: g_j(x^a) > g_j(x^{a-1}) \quad \text{OR} \quad g_j(x^a) > \varepsilon_g \\ \frac{1}{2} \times r_{p,j}^{a} & if: g_j(x^a) \le \varepsilon_g \end{cases} \tag{25}$$

$$r_{p,k}^{a+1} = \begin{cases} 2 \times r_{p,j}^{a} & if: \ |h_j(x^a)| > |h_j(x^{a-1})| \ \text{OR} \ |h_j(x^a)| > \varepsilon_h \\ \frac{1}{2} \times r_{p,k}^{a} & if: |h_j(x^a)| \le \varepsilon_h \end{cases} \tag{26}$$

7 Results and Discussion

7.1 Overview

The use of digital pheromones within PSO is theorized to improve the accuracy, efficiency and reliability of particle swarms. This chapter demonstrates this capability through testing the developed methods on different test problems with varied dimensionality and modality (unimodal and multimodal). Although it is possible to compare results of digital pheromone PSO against published results from another evolutionary method such as a GA, differences in computational environments (processing speeds, memory capacities, processor loads) can be an unfair comparison measure. Since this research is aimed at improving the performance of PSO, results from digital pheromone implementation are therefore benchmarked against a basic PSO, as outlined in section 1.6.

The algorithmic implementation was made in C/C++ on a RedHat Linux computing environment. User defined parameters (c_1, c_2, c_3, inertia weight decay, move limit decay, inertia weight decay, radius of influence, etc) were provided as an input to the algorithm run-time using xml configuration files (see xml specifications [120]). Also, other information such as the maximum number of runs and iterations, convergence tolerances, test problem specifications that include number of design variables, number of constraints, lower and upper limits for the test problems was also provided in the xml configuration file.

7.2 Test Problem Description

Table 3 is a broad overview of test problem numbers used for testing a specific method's performance. Section 7.2 describes these test problems with their published solution values in detail.

Table 3: List of problem numbers used for testing the developed methods

Method	Description in Chapter #	Test Problems used
Digital pheromone implementation of PSO	3	7.2.1 – 7.2.10
Statistical Analysis	7	7.2.1 – 7.2.4
Coarse Grain Parallelization	4	7.2.5 – 7.2.10
Shared Pheromone Parallelization	4	7.2.5 – 7.2.10
GPU Parallelization	5	7.2.5 – 7.2.10
Constrained Optimization	6	7.2.11 – 7.2.16

The following are the test problems used for evaluating the performance of digital pheromones within PSO. Full mathematical descriptions for these test problems can be found in [121-123].

7.2.1 Six-hump Camelback 2D function

This is a multimodal optimization problem with six local minima, two of which are global minima. Figure 22 shows the contours of the function.

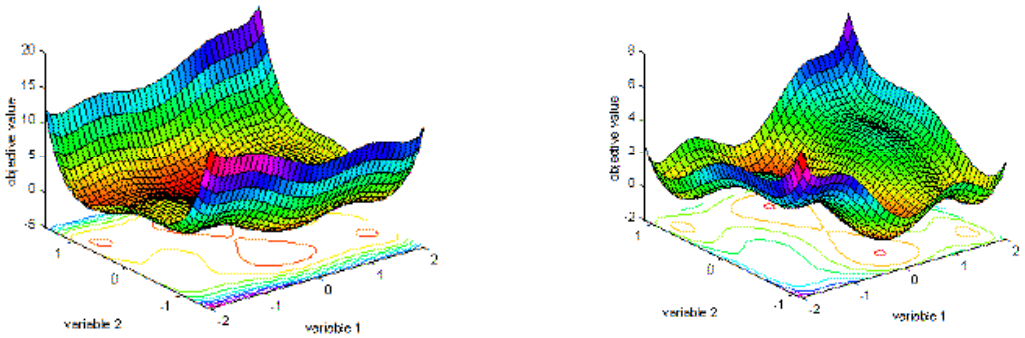


Figure 22 Six-hump Camelback Function
(Figure Courtesy: http://www.geatbx.com)

The optimization problem statement is:

Minimize:

$$F(x_1, x_2) = \left(4 - 2.1x_1 + \frac{x_1^4}{3}\right)x_1^2 + x_1x_2 + \left(-4 + 4x_2^2\right)x_2^2$$

$$-3 \le x_1 \le 3 \quad \text{and} \quad -2 \le x_2 \le 2$$

Published solution:

$$F_{\min}(x_1, x_2) = -1.031628$$

$$(x_1, x_2) = (0.0898, -0.7126), \quad (-0.0898, 0.7126)$$

7.2.2 Himmelblau 2D function

This is a multimodal optimization problem with four local minima. Figure 23 shows the contours of the function.

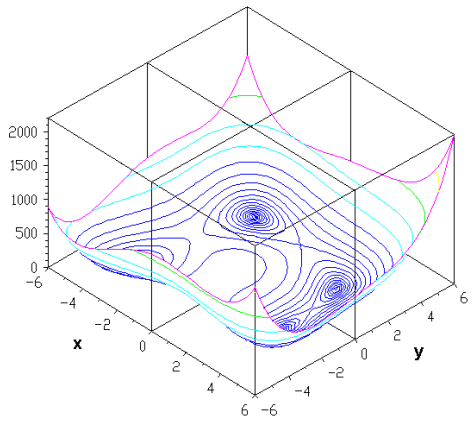


Figure 23 Himmelblau function
(Figure Courtesy: http://en.wikipedia.org/wiki/Himmelblau's_function)

The optimization problem statement is:

$$F(x_1, x_2) = (X_1^2 + X_2 - 11)^2 + (X_1 + X_2^2 - 7)^2$$

$$-5 \le X_1 \le 5 \quad \text{and} \quad -5 \le X_2 \le 5$$

Published solution:

$$F_{\min}(x_1, x_2) = 0.0$$
$$(x_1, x_2) = (3,2)$$

7.2.3 Rosenbrock 5D function

Rosenbrock's valley is also known as the Banana function. The global optimum is inside a long, narrow and parabolic shaped flat valley. Arriving at the neighborhood of the valley is trivial, but converging to the global optimum is difficult. This function is scalable to any number of dimensions. In this research, a five dimensional Rosenbrock function was used as a test case. Figure 19 shows a two dimensional Rosenbrock's function to understand how the function behaves.

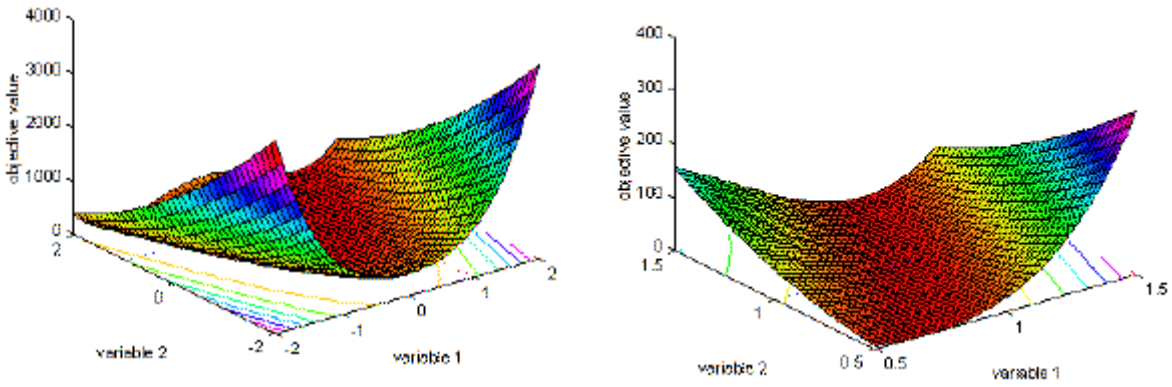


Figure 24 Rosenbrock's Valley Function
(Figure Courtesy: http://www.geatbx.com)

The optimization problem statement is:

$$F(X) = \sum_{i=1}^{n-1} 100 \times \left(X_{i+1} - X_i^2\right)^2 + \left(1 - X_i\right)^2$$
$$-2.048 \le X_i \le 2.048$$

Published Solution:

$$F_{\min}(x_i) = 0.0$$
$$(x_i) = 1.0, i = 1:n$$

7.2.4 Ackley's 10D Path Function

Ackley's path function is a highly multimodal problem and is widely used as a test problem for unconstrained optimization methods. The problem seems to look like a unimodal problem with its bounds (-32.768 to 32.768), but the multimodal nature of the function becomes apparent when the bounds are decreased to (-2, 2). This function is scalable to any number of dimensions. Problem 7.2.4 will represent a 10 dimensional Ackley's path function. Figure 25 shows the two dimensional Ackley's path function.

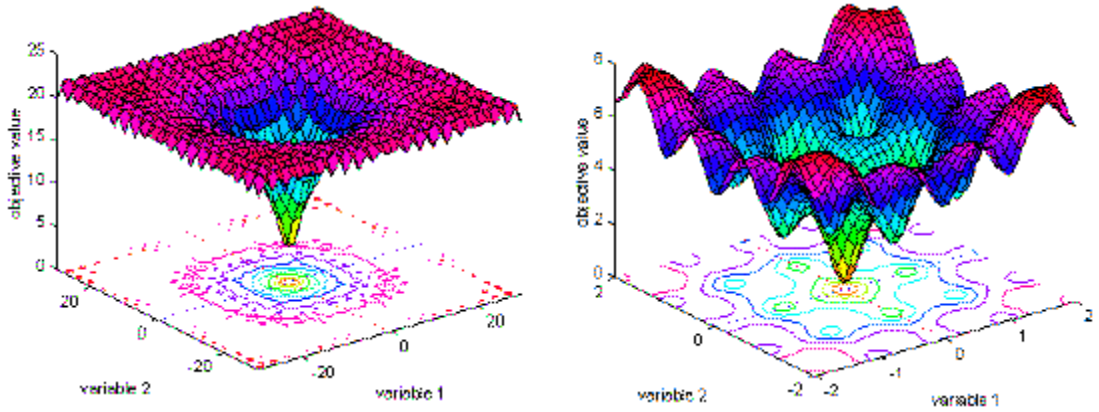


Figure 25 Ackley's Path Function
(Figure Courtesy: http://www.geatbx.com)

The optimization problem statement is:

$$F(x) = -a \cdot e^{-b\cdot\sqrt{\frac{\sum_i^n x_i^2}{n}}} - e^{\frac{\sum_i^n \cos(c\cdot x_i)}{n}} + a + e^1$$
$$a = 20; \quad b = 0.2; \quad c = 2 \cdot PI; \quad i = 1:n;$$
$$-32.768 \le x_i \le 32.768$$

The published solution is:

$$F_{\min}(x) = 0.0, \quad x_i = 0.0$$

7.2.5 Dixon and Price 15D function

This function is scalable to any number of dimensions and a 15 dimensional version was used in this research. Figure 26 shows the two dimensional Dixon and Price function.

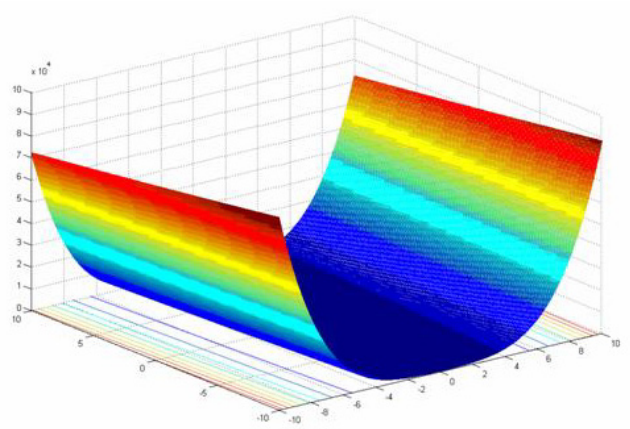


Figure 26 Dixon and Price Function
(Figure Courtesy:
http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar_files/TestGO.htm)

The optimization statement for the problem is as follows:

$$F(X) = (X_0 - 1.0)^2 + \sum_{i=2}^{n} (i) \times \left(2 \times X_i^{\,2} - X_{i-1}\right)^2$$

$$-10.0 \le X_i \le 10.0, \quad i = 0 : n$$

Published Solution:

$$F_{\min}(x) = 0.0, \qquad x_i = 0.0$$

7.2.6 Ackley's 20D Path Function

This test problem is the same as described in section 7.2.4 except that the number of design variables in this case is 20. Therefore, the published solution is 0.0000 and the design variable values that minimizes the path function are also 0.0000 as well. It was treated as a separate test case in this research and provided its own problem number for easy reference.

7.2.7 Levy 25D Function

This function is scalable to any dimensions and a 25 dimensional Levy function [124] was used as one of the test cases in this research.

The optimization problem statement is:

$$f(x) = \frac{\pi}{n}\left\{k\sin^2(\pi y_1) + \sum_{i=1}^{n-1}\left[(y_i - a)^2\left(1 + k\sin^2(\pi y_{i+1})\right)\right] + (y_n - a)^2\right\}$$

$$y_i = 1 + 0.25(x_i - 1)$$

$$-10 \le x_i \le 10,\ i = 1:n$$

Published solution:

$$F_{\min}(x) = 0.0, \quad x_i = 0.0$$

7.2.8 Sum of Squares 30D Function

This function is scalable to any number of dimensions and a 30 design variable version was used for this research. Figure 27 shows a two dimensional sum of squares function.

The optimization problem statement is:

$$f(x) = \sum_{i=0}^{n}(i+1)\times(x_l^2)$$

$$-10 \le x_i \le 10,\ \ i = 1:n$$

Published solution:

$$F_{\min}(x) = 0.0, \quad x_i = 0.0$$

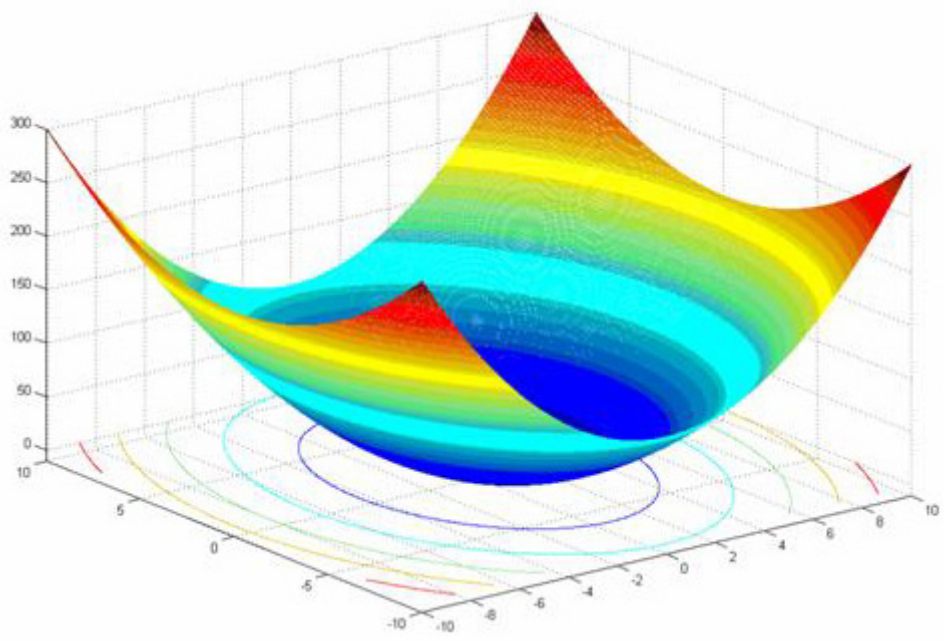


Figure 27 Sum of Squares Function
(Figure Courtesy:
http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar_files/TestGO.htm)

7.2.9 Sphere 40D Function

This is similar to the sum of squares function except that there is no multiplication factor (i+1) in the summation term. It is also known as De-Jong's function. In this research, a 40 dimensional sphere function was used. Figure 28 shows a two dimensional sphere function.

The minimization problem is given by

$$f(x) = \sum_{i=0}^{n} (x_i^2)$$

$$-10 \le x_i \le 10, \quad i = 1 : n$$

Published Solution:

$$F_{\min}(x) = 0.0, \quad x_i = 0.0$$

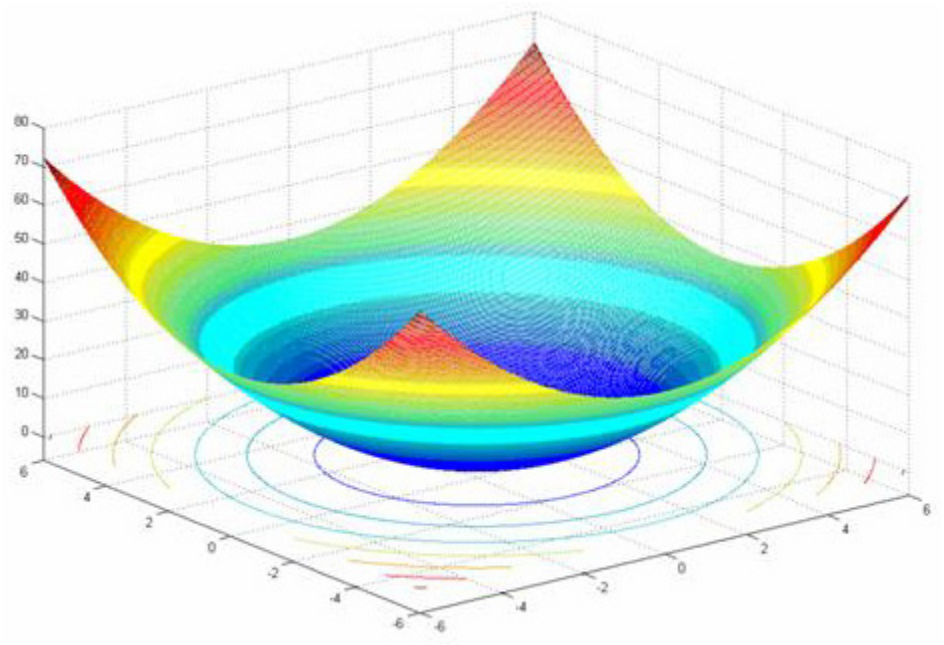


Figure 28 Sphere (De Jong's) Function
(Figure Courtesy:
http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar_files/TestGO.htm)

7.2.10 Griewank's 50D Function

Griewank's function is a highly multimodal problem and many optimization methods frequently get trapped in local minima. A 50 dimensional Griewank's function was used as one of the test cases in this research. Figure 29 shows a two dimensional Griewank's function.

The optimization statement is given by:

$$f(x) = \sum_{i=1}^{n}\left(\frac{x_i^2}{4000}\right) - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right)$$
$$-600 \le x_i \le 600, \quad i = 1:n$$

Published solution:

$$F_{\min}(x) = 0.0, \quad x_i = 0.0$$

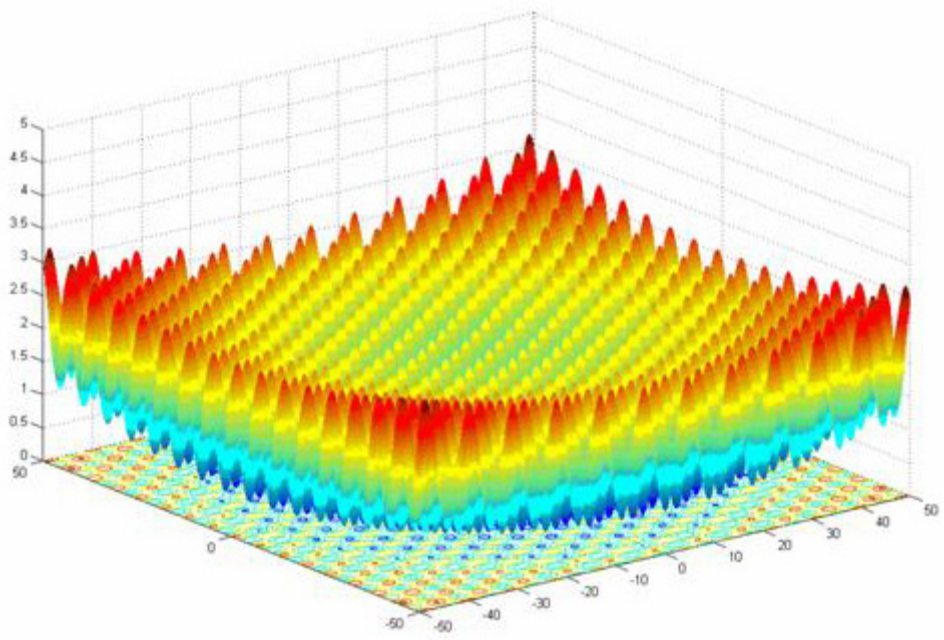


Figure 29 Griewank's Function
(Figure Courtesy:
http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar_files/TestGO.htm)

7.2.11 One Dimensional Two Inequality Constrained Problem

This is a simple one dimensional two inequality constrained test problem. The optimization statement is given by:

$$f(x) = \frac{(x+2)^2}{20}$$
$$g_1(x) = \frac{(1-x)}{2} \le 0$$
$$g_2(x) = \frac{(x-2)}{2} \le 0$$
$$1 \le x_i \le 2, \quad i = 1, 2$$

Published solution:

$$F_{\min}(x) = 0.45, \quad x = 1.0$$

7.2.12 Two Dimensional Single Inequality Problem

This is a two dimensional design variable problem subject to a non-linear inequality constraint. The optimization statement is given by:

$$f(x_1, x_2) = x_1^2 + 2x_2^2 - 2x_1x_2 - 14x_1 - 14x_2 + 10$$
$$g(x_1, x_2) = 4x_1^2 + x_2^2 - 25 \le 0$$
$$-5 \le x_i \le 5, \quad i = 1, 2$$

Published solution:

$$F_{\min}(x) = -50, \quad x_1, x_2 = \{-2.0,\ 3.0\}$$

7.2.13 Two Dimensional Two Inequality Problem

This is a two dimensional design variable problem subject to a linear and a non-linear inequality constraint. The optimization statement is given by:

$$f(x_1, x_2) = x_1^2 + x_2^2 - 6x_1 - 8x_2 + 10$$
$$g_1(x_1, x_2) = 4x_1^2 + x_2^2 - 16 \le 0$$
$$g_2(x_1, x_2) = 3x_1 + 5x_2 - 15 \le 0$$
$$1 \le x_i \le 10, \quad i = 1, 2$$

Published solution:

$$F_{\min}(x) = -9.2347, \quad x_1, x_2 = \{1.7456,\ 1.9527\}$$

7.2.14 Four Dimensional Eight Inequality Constrained Weld Beam Problem

This welded beam function is a standard constrained minimization test problem with 8 inequality constraints [2]. Figure 30 shows the physical system of the weld beam problem:

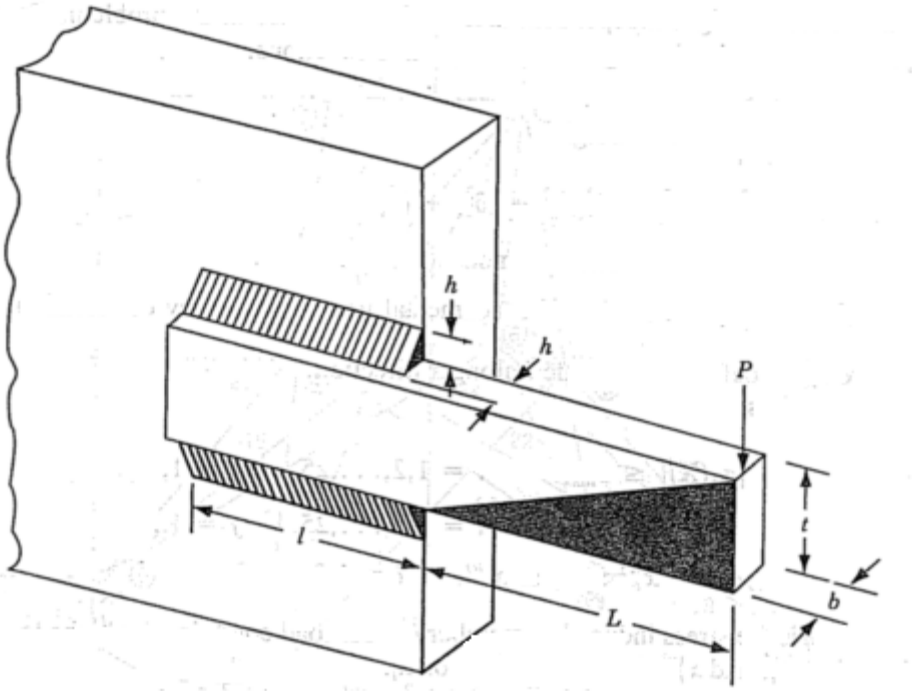


Figure 30 Illustration of Weld Beam Problem
(Figure Courtesy: [2])

The optimization statement is:

$$F(X) = 1.1047 X_1^{\,2} X_2 + 0.04811 X_3 X_4 (14.0 + X_2)$$

Subjected to :

$$g_1(X) : \tau(X) - \tau_{\max} \le 0$$

$$g_2(X) : \sigma(X) - \sigma_{\max} \le 0$$

$$g_3(X) : X_1 - X_4 \le 0$$

$$g_4(X) : 0.10471 X_1^{\,2} + 0.04811 X_3 X_4 (14.0 + X_2) - 5.0 \le 0$$

$$g_5(X) : 0.125 - X_1 \le 0$$

$$g_6(X) : \delta(X) - X_1 \le 0$$

$$g_7(X) : P - P_c(X) \le 0$$

$$0.1 \le X_i \le 2.0, \quad i = 1, 4$$

$$0.1 \le X_i \le 10.0, \quad i = 2, 3$$

Where,

$$\tau(X) = \sqrt{(\tau')^2 + 2\tau'\tau''\frac{X_2}{2R} + (\tau'')^2}$$

$$\tau' = \frac{p}{\sqrt{2}X_1X_2}, \tau'' = \frac{MR}{J}, M = P\left(L + \frac{X_2}{2}\right)$$

$$R = \sqrt{\frac{X_2^2}{4} + \left(\frac{X_1 + X_3}{2}\right)^2}$$

$$J = 2\left\{\sqrt{2}X_1X_2\left[\frac{X_2^2}{12} + \left(\frac{X_1 + X_3}{2}\right)^2\right]\right\}$$

$$\sigma(X) = \frac{6PL}{X_4X_3^2}$$

$$\delta(X) = \frac{4PL^3}{EX_3^3X_4}$$

$$P_C(X) = \frac{4.013\sqrt{E(X_3^2X_4^6/36)}}{L^2}\left(1 - \frac{X_3}{2L}\sqrt{\frac{E}{4G}}\right)$$

$$P = 6000lb, L = 14in., E = 30\times10^6\ psi, G = 12\times10^6\ psi$$

$$\tau_{\max} = 13{,}600\,psi, \sigma_{\max} = 30{,}000\,psi, \delta_{\max} = 0.25in.$$

The published solution for this problem is: 2.386 and the solution set X* is: {0.2455, 6.1960, 8.2730, 0.2455}.

7.2.15 Golinski's Speed Reducer Problem

This is a seven dimensional eleven inequality constrained test problem, and is typically used as one of the standard test cases to evaluate the performance of a constrained optimization method. The speed reducer, as shown in Figure 31, represents a simple gear box that is typically utilized in airplane applications.

The reducer consists of a gear-pinion mounted on two shafts. Each shaft is supported by one bearing at each end. A typical system includes gear, pinion, shafts and bearings enclosed in a common housing.

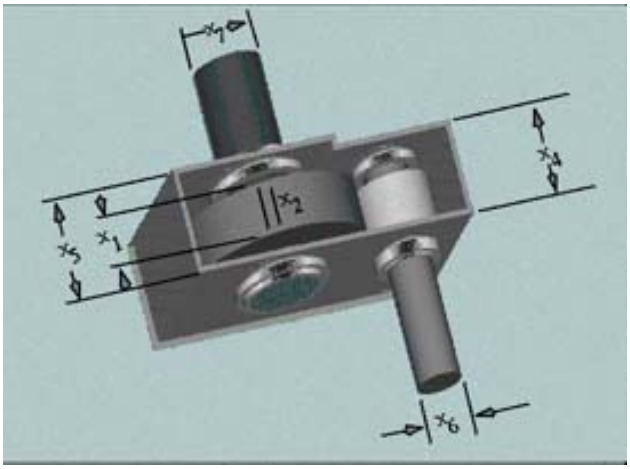


Figure 31 Golinski's Speed Reducer

The objective is to minimize the speed reducer weight while satisfying a number of constraints imposed by gear and shaft design practices. Table 4 shows a description of the design variables with its lower and upper bounds.

Table 4 Description of design variables for Golinski's speed reducer problem

Design Variable	Description	Lower Limit	Upper Limit
x1	face width of gear, cm	2.6	3.6
x2	teeth module, cm	0.7	0.8
x3	# of pinion teeth	17	28
x4	shaft length #1, between bearings, cm	7.3	8.3
x5	shaft length #2, between bearings, cm	7.3	8.3
x6	shaft diameter #1, cm	2.9	3.9
x7	shaft diameter #2, cm	5	5.5

The optimization statement is given by:

$$f = C_{f1}\, x_1\, x_2{}^2 \left(C_{f2}\, x_3{}^2 + C_{f3}\, x_3 - C_{f4} \right) - C_{f5} \left(x_6{}^2 + x_7{}^2 \right) x_1 + C_{f6} \left(x_6{}^3 + x_7{}^3 \right) + C_{f1} \left(x_4\, x_6{}^2 + x_5\, x_7{}^2 \right)$$

Where the coefficients are given by:

$$C_{f1} = 0.7854 \qquad C_{f4} = 43.0934$$

$$C_{f2} = 3.3333 \qquad C_{f5} = 1.5079$$

$$C_{f3} = 14.9334 \qquad C_{f6} = 7.477$$

Subjected to the following constraints (although 25 constraints are listed, 14 of them represent the lower and upper bounds for the design variables thereby reducing the total number of constraints to 11):

$$g_1 = \frac{C_{g1}}{x_1 {x_2}^2 x_3} \leq 1.0$$

$$g_2 = \frac{C_{g2}}{x_1 {x_2}^2 {x_3}^2} \leq 1.0$$

$$g_3 = \frac{C_{g3} {x_4}^3}{x_2 x_3 {x_6}^4} \leq 1.0$$

$$g_4 = \frac{C_{g4} {x_5}^3}{x_2 x_3 {x_7}^4} \leq 1.0$$

$$g_5 = \frac{\sqrt{\frac{{C_{A12}}^2 {x_4}^2}{{x_2}^2 {x_3}^2} + C_{A1}}}{C_{g5} C_B {x_6}^3} \leq 1.0$$

$$g_6 = \frac{\sqrt{\frac{{C_{A12}}^2 {x_5}^2}{{x_2}^2 {x_3}^2} + C_{A2}}}{C_{g6} C_B {x_7}^3} \leq 1.0$$

$$g_7 = \frac{x_2 x_3}{C_{g7}} \leq 1.0$$

$$g_8 = \frac{C_{g8} x_2}{x_1} \leq 1.0$$

$$g_9 = \frac{x_1}{C_{g9} x_2} \leq 1.0$$

$$g_{10} = \frac{C_{g10}}{x_1} \leq 1.0$$

$$g_{11} = \frac{x_1}{C_{g11}} \leq 1.0$$

$$g_{12} = \frac{C_{g12}}{x_2} \leq 1.0$$

$$g_{13} = \frac{x_2}{C_{g13}} \leq 1.0$$

$$g_{14} = \frac{C_{g14}}{x_3} \leq 1.0$$

$$g_{15} = \frac{x_3}{C_{g15}} \leq 1.0$$

$$g_{16} = \frac{C_{g16}}{x_4} \leq 1.0$$

$$g_{17} = \frac{x_4}{C_{g17}} \leq 1.0$$

$$g_{18} = \frac{C_{g18}}{x_5} \leq 1.0$$

$$g_{19} = \frac{x_5}{C_{g19}} \leq 1.0$$

$$g_{20} = \frac{C_{g20}}{x_6} \leq 1.0$$

$$g_{21} = \frac{x_6}{C_{g21}} \leq 1.0$$

$$g_{22} = \frac{C_{g22}}{x_7} \leq 1.0$$

$$g_{23} = \frac{x_7}{C_{g23}} \leq 1.0$$

$$g_{24} = \frac{C_{g24} x_6 + C_{g245}}{x_4} \leq 1.0$$

$$g_{25} = \frac{C_{g25} x_7 + C_{g245}}{x_5} \leq 1.0$$

Where the coefficients for the constraints are given by:

$$C_{g1} = 27.0 \qquad C_{g12} = 0.7$$

$$C_{g2} = 397.5 \qquad C_{g13} = 0.8$$

$$C_{g3} = 1.93 \qquad C_{g14} = 17$$

$$C_{g4} = 1.93 \qquad C_{g15} = 28$$

$$C_{g5} = 1100.0 \qquad C_{g16} = 7.3$$

$$C_{A12} = 745.0 \qquad C_{g17} = 8.3$$

$$C_{A1} = 0.169 \times 10^{8} \qquad C_{g18} = 7.3$$

$$C_{B} = 0.1 \qquad C_{g19} = 8.3$$

$$C_{A2} = 0.1575 \times 10^{9} \qquad C_{g20} = 2.9$$

$$C_{g6} = 850.0 \qquad C_{g21} = 3.9$$

$$C_{g7} = 40.0 \qquad C_{g22} = 5.0$$

$$C_{g8} = 5.0 \qquad C_{g23} = 5.5$$

$$C_{g9} = 12.0 \qquad C_{g24} = 1.5$$

$$C_{g10} = 2.6 \qquad C_{g25} = 1.1$$

$$C_{g11} = 3.6 \qquad C_{g245} = 1.9$$

Published solution:

$$F_{\min}(x) = 2985.22$$

$$\{x_1,\ x_2,\ x_3,\ x_4,\ x_5,\ x_6,\ x_7\} = \{3.5,\ 0.7,\ 17,\ 7.3,\ 7.3,\ 3.35,\ 5.29\}$$

7.2.16 Himmelblau 5D Constrained Problem

This is a five dimensional constrained version of the Himmelblau function [21]. It is a minimization problem with 7 inequality constraints and the optimization statement is:

$$F(X) = 5.3578547X_1^{\ 2} + 0.8356891X_1X_5 + 37.2932239X_1 - 40792.141)$$

Subjected to :

$$g_1(X) : 85.334407 + 0.0056858X_2X_5 + 0.00026X_1X_4 - 0.0022053X_3X_5 - 92.0 \le 0$$

$$g_2(X) : -1 \times (85.334407 + 0.0056858X_2X_5 + 0.00026X_1X_4 - 0.0022053X_3X_5) \le 0$$

$$g_3(X) : 80.51249 + 0.0071317X_2X_5 + 0.0029955X_1X_2 + 0.0021813X_3^2 - 110.0 \le 0$$

$$g_4(X) : -1 \times (80.51249 + 0.0071317X_2X_5 + 0.0029955X_1X_2 + 0.0021813X_3^2) \le 0$$

$$g_5(X) : 9.300961 + 0.0047026X_3X_5 + 0.0012547X_1X_3 + 0.0019085X_3X_4 - 25.0 \le 0$$

$$g_6(X) : -1 \times (9.300961 + 0.0047026X_3X_5 + 0.0012547X_1X_3 + 0.0019085X_3X_4) \le 0$$

$$78 \le X_1 \le 102,\ 33 \le X_2 \le 45,\ 27 \le X_3 \le 45,\ 27 \le X_4 \le 45,\ 27 \le X_4 \le 45$$

The published solution for this problem is: -31025.56142 and the solution set X* is: {78.0, 27.0, 27.070997, 45.0, 44.96924255}.

7.3 Results from Digital Pheromone Implementation in PSO

7.3.1 Test Problem Settings

Problems 7.2.1 – 7.2.10 as shown in Table 5 were used as test cases to evaluate the digital pheromone implementation within PSO. The dimensionalities (i.e. number of design variables) of the test problems are also shown in the table. Of these, problems 7.2.1 – 7.2.4 were used for determining the default values for the new user defined parameters introduced by the implementation of digital pheromones.

Table 5 Test Problem Matrix for serial implementation of PSO with digital pheromones

Problem	Test Problem	# of Design Variables
7.2.1	Camelback function	2
7.2.2	Himmelblau function	2
7.2.3	Rosenbrock function	5
7.2.4	Ackley's path function	10
7.2.5	Dixon and Price function	15
7.2.6	Ackley's path function	20
7.2.7	Levy function	25
7.2.8	Sum of Squares function	30
7.2.9	Spherical function	40
7.2.10	Griewank function	50

Several values were for the user defined parameters as shown in Table 6. There are 128 unique combinations of parameters and each was used to solve a problem 20 times, to test repeatability of the method. This was performed for problems 7.2.1 - 7.2.4, yielding a total of 10,240 independent solution runs. The remaining test problems (7.2.5 – 7.2.10) were then solved with the pheromone parameters that consistently provided the best answers and performance measures.

Table 6 Digital Pheromone Parameters

Pheromone parameters	Combination of values tested	# Combinations
c_3 decay (0.5%)	No, Yes (decayed once every 10 iterations)	2
c_3	10.0, 5.0, 2.0, 1.0	4
Pheromone level decay, λ_P	0.995, 0.95, 0.9, 0.85	4
Move limit decay, λ_{ML}	0.995, 0.95, 0.9, 0.85	4

The values for c_3 ranged from 1.0 through 10.0 to investigate if the pheromone influence needed to be large or small compared to *pBest* and *gBest*. Also, it was useful for studying the effect of the pheromone decay factor. The influence of pheromone levels and move limits were tested with decay rates ranging from 0.995 (0.5%) to 0.85 (15%) of their values in the previous iteration. The lower limit of the pheromone level decay was capped at 15% since a significant drop in the pheromone level could cause the influence of pheromones to be counter-productive for the swarm in reaching the global optimum. A similar reason was attributed for setting a lower bound on the move limit decay at 0.85.

The solutions obtained for test problems 7.2.1 – 7.2.4 were ranked in order of smallest average objective function value. Conclusions were made based on the results (in terms of solution accuracy compared to published solutions) and suitable values for pheromone

parameters were determined. These parameters were then used into the developed method to solve problems 7.2.5 – 7.2.10.

The solutions (accuracy and solution times) from solving these test cases with experimentally determined digital pheromone parameters were compared against those obtained from a basic PSO. The swarm size was defined as 10 times the number of design variables, and was capped at a maximum of 500. The test problems were considered converged when the difference in solutions was within 0.001 for 10 consecutive iterations. All test cases were performed on a PC running the RedHat Enterprise Linux Operating System with an Intel Xeon processor and 2GB of system memory.

7.3.2 Results and Discussion

Table 7 provides a summary of the results obtained from solving problems 7.2.1 – 7.2.4. Problems solved using a basic PSO are designated with a "B" in parentheses next to the problem number and those using pheromones have a "P". It tabulates the best ranked solutions from 128 combinations of pheromone parameter values. For example, the pheromone parameter values determined for the camelback 2D problem (problem 7.2.1) were $c_3 = 1$, c_3Decay = NO, pheromone decay factor = 0.995 and maximum velocity decay = 0.85, which resulted in an optimum solution of 1.031618.

It can be seen from Table 7 that all pheromone parameters consistently produced an average objective function value less than that from the basic PSO. Since averages are not a true

measure for performance, two other columns – the smallest objective function value achieved, and the standard deviation are also noted in the table. The smallest objective function is the lowest value obtained in 20 trial runs for each test problem. The results demonstrate that the use of digital pheromones to search the design space provided substantially more information for the swarm to investigate the design space and attain the global optimum at a higher accuracy than a basic PSO, even when different parameter values were used. Although the solution value changed, it was still substantially better than using a basic PSO.

Table 7 Solution averages obtained from solving preliminary test problems

Prob. No.	Average Objective Function	Smallest Objective Function	Standard Deviation	Average Duration per run (sec)	Average No. of Iterations	c_3	c_3 decay	λ_P	λ_{ML}
7.2.1 (B)	-1.018154	-1.030545	0.01049	0.40	19.65	-	-	-	-
7.2.1 (P)	-1.031618	-1.031628	0.00001	0.43	21.25	1	NO	0.995	0.85
7.2.2 (B)	0.473549	0.000192	0.687921	0.43	21.25	-	-	-	-
7.2.2 (P)	0.000168	0.000001	0.000234	0.61	30.55	1	NO	0.995	0.85
7.2.3 (B)	0.117105	0.000186	0.16704	1.06	21.1	-	-	-	-
7.2.3 (P)	0.000371	0.000000	0.00042	1.57	30.8	2	YES	0.95	0.95
7.2.4 (B)	3.542873	0.002991	3.22811	10.26	102.5	-	-	-	-
7.2.4 (P)	0.001433	0.000661	0.00086	8.21	79.5	2	NO	0.85	0.9

Legend: 7.2.1 – Camelback 2D, 7.2.2 – Himmelblau 2D, 7.2.3 – Rosenbrock 5D, 7.2.4 – Ackley's path 10D. (B) – Results from basic PSO, (P) – Results from PSO with Digital Pheromones implemented

It should also be noted that PSO with digital pheromones required longer times to solve the Camelback 2D, Himmelblau 2D, and Rosenbrock 5D problems. This behavior was found true in all 128 test runs. The reason is attributed to the additional number of pheromone operations needed. However, as the complexity of the objective function in terms of the number of design variables increased, the pheromones provided more information about the

design space to the swarm thereby converging in significantly less iterations. This is evident from the average solution times of the Ackley 10D problem where the solution duration with digital pheromones was smaller compared to that of the basic PSO. This suggests that decreased solution times become more prominent as the complexity of the optimization problem increases.

It can be seen from the table that the camelback 2D and Himmelblau 2D problems performed best with a c_3 value of 1.0, but Rosenbrock 5D and Ackley 10D performed best with $c_3 = 2.0$. Although it was inconclusive, it provided some evidence for requiring a higher value for c_3 as the dimensionality of the problem increased. Also, Rosenbrock 5D required a decay in the c_3 value with the progress in iterations, while other test problems did not. In addition, the average number of iterations for Rosenbrock 5D was 30.8, which means that the value of c_3 was decayed only three times by a factor of 5% (since decay is performed once every 10 iterations). Although solution sets from all pheromone parameter combination possibilities are not shown in this paper, it was observed that the performance of the method was influenced by the value of c_3 but relatively insensitive to decay in c_3. For example, in the Ackley 10D (problem 7.2.4), the pheromone parameters that ranked second, in terms of average objective function value, required a c_3 value of 10.0 to produce a solution of 0.001496, as opposed to the parameter values that produced a solution of 0.001433. The solution accuracy between the two parameter sets was significant only in the fifth decimal place. However, it was noticed that in the first 10 ranked solution sets for Ackley 10D, four out of 10 cases required a c_3 value of 10 and three out of 10 cases required a c_3 value of 5.0. This suggests that higher dimensional problems might require a higher value of c_3 so as to

increase the influence of pheromones over the swarm. From these observations a value of c_3 ranging from 2.0 to 5.0 is suggested.

The results also suggest that a value of 0.9 to 0.95 is an appropriate choice of value for the pheromone decay factor. The test cases revealed that a value greater than 0.95 or less than 0.9 allowed the pheromone component to become too large or small compared to the *pBest* and *gBest* components in the velocity vector. To achieve the maximum benefit from digital pheromones, a balance needs to be in effect for all of these components of the velocity vector.

A pattern was also observed in the value of the move limit decay for the top 10 ranked solutions for all the test problems. Camelback 2D and Himmelblau 2D required a move limit decay value of 0.85, Rosenbrock 5D required 0.95, and Ackley 10D required 0.9 for attaining global optimum solutions. Although the scales of objective function values for each test problems are different, a range of values between 0.85 and 0.95 seemed to be appropriate. It is to be noted that although pheromone parameters are suggested in this section, they are user-defined parameters that can be altered to suit to a specific optimization problem.

Based on the knowledge gained about the pheromone parameters, the following values were used for solving problems 7.2.5 – 7.2.10:

- c_3 = 5.0 with no decay,
- Pheromone decay = 0.95, and
- Move limit decay = 0.95

Table 8 provides the summary of results from these test runs. The table shows that digital pheromones when used in PSO consistently displayed superior performance when compared with solutions from a basic PSO.

Table 8 Summary of results from solving problems 7.2.5 – 7.2.10

Problem No.	Solution Accuracy	Objective Function			Average No. of iterations	Duration (secs/run)
		Average	Smallest	Std Dev		
7.2.5 (B)	65%	48.366	0.0007	143.556	166.3	25.29
7.2.5 (P)	85%	0.148	0.0003	0.466	92.0	14.72
7.2.6 (B)	0%	4.658	2.659	2.740	166.7	16.899
7.2.6 (P)	85%	0.171	0.003	0.418	143.3	15.095
7.2.7 (B)	100%	0.143	0.131	0.041	96.5	24.819
7.2.7 (P)	100%	0.132	0.130	0.001	40.8	11.289
7.2.8 (B)	0%	16.301	0.521	48.692	212.45	66.10
7.2.8 (P)	85%	0.084	0.006	0.128	138.25	46.46
7.2.9 (B)	100%	0.033	0.0174	0.0078	162.65	68.35
7.2.9 (P)	100%	0.002	0.0007	0.0007	85.15	39.38
7.2.10 (B)	0%	1.189	1.056	0.133	186.1	99.014
7.2.10 (P)	100%	0.008	0.003	0.005	158.1	93.801

Legend: 7.2.5 – Dixon and Price function15D, 7.2.6 – Ackley's path function 20D, 7.2.7 – Levy function 25D, 7.2.8 – Sum of squares function 30D, 7.2.9 – Spherical function 40D, 7.2.10 – Griewank function 50D. (B) – Results from Basic PSO, (P) – Results from PSO with Digital Pheromones implemented.

Since the published solutions for most of the problems in Table 8 are 0.000, there was no measure for percentage accuracy. Therefore, a tolerance was given and accuracy was measured based on the number of times the obtained solution was within the tolerance limits. For example, a tolerance limit of +/-0.5 was assigned for a 20 design variable problem. If the solution was within this tolerance limit 85 times in 100 runs of the problem, the solution accuracy was 85%.

The solution accuracies noted in the table were within a tolerance limit of +/-0.5. The choice of 0.5 was not arbitrary. If the value was smaller than 0.5 basic PSO did not solve most of the problems. Thus, for some sort of comparison a slightly larger, but still sufficient tolerance was used. In all test cases, the solution accuracy of PSO with digital pheromones was either equal or superior when compared to basic PSO. For example, problem 7.2.6 (Ackley's 20D), the basic PSO was not able to solve the problem whereas the pheromone PSO attained the solution within the specified tolerance limits 85 out of 100 runs.

For the Dixon and Price function (problem 7.2.5) a solution accuracy of 65% was achieved by basic PSO. However, the average objective function value was 48.366, meaning that when the swarm did not locate the optimum within the tolerance limits, it was a very bad design point. The pheromone PSO method resulted in 85% accuracy with an average objective function value of 0.148 and a standard deviation of 0.466. So, even when the optimum was not located within the tolerance, the solution was still in the neighborhood of the optimum. A benefit, if restarting the method is an option. Also, the average duration per run was significantly lower for pheromone PSO when compared to basic PSO.

The solution accuracy measure for the Levy function (problem 7.2.7) was essentially equal between basic and pheromone PSO. Both the methods solved the problem with 100% accuracy. Although the average, smallest and standard deviation between the methods was very close, there was almost a 50% decrease in the solution time for pheromone PSO.

The basic PSO failed to solve the 30 dimensional sum of squares function (problem 7.2.8) within the specified tolerance limit of +/- 0.5. The smallest objective function value returned was 0.521. However, the pheromone PSO solved the problem within the tolerance limits on all 20 trial runs along with improved average objective function value, standard deviation and duration. Given the unimodal nature of the test problem, the failure to solve the problem by basic PSO may be attributed to the high swarm size (300) causing substantial swarm activity negatively impacting convergence in the design space. The pheromone PSO on the other hand was relatively unaffected by the swarm size and converged faster when compared to basic PSO.

Both basic PSO and pheromone PSO were able to solve the 40D spherical function (problem 4.9) with 100% accuracy. However, as seen from the table, the average objective function evaluated by the pheromone PSO (0.002) was about 16 times better than the average objective function returned by basic PSO (0.033). Moreover, the solution time of pheromone PSO (39.38 seconds) was 42% faster when compared to basic PSO (68.35 seconds). Although the variation of the results in basic PSO was small, the pheromone PSO showed superior consistency as evident from the standard deviation of the objective function values.

Pheromone PSO was able to solve the 50D highly multimodal Griewank problem (problem 7.2.10) with 100% solution accuracy whereas the basic PSO could not reach the global minimum in any of the 20 runs. There was a 5% improvement in the solution duration as well and the standard deviation was significantly better than a basic PSO.

To emphasize that that the suggested pheromone parameters determined from problem 7.2.1 - 7.2.4 were good default values, they were used to solve problems 7.2.5 – 7.2.10. However, fine tuning of the pheromone parameters could potentially produce superior solutions than those in Table 8. For example in problem 7.2.5, a non decaying c_3 value of 5.0, a pheromone decay factor of 0.995 and a move limit decay of 0.95 produced a solution of 0.00534 when compared to 0.148 with the suggested pheromone parameters. This means that although the suggested values perform well, additional performance improvement can be realized through refinements in the pheromone parameters.

7.3.3 Simulating Realistic Objective Functions

The test cases presented thus far are academic in nature with easily computed analytical objective functions. They are not a true representative of the type of problems solved in industrial settings, where function evaluations can take a considerable amount of computational time. To test if longer function evaluation times have any impact on the performance of the developed method, a sleep time was added when evaluating the objective function. This was done to simulate an objective function with a longer evaluation time. Table 9 shows the solution times for solving Ackley's path function of 20 design variables when sleep times of 0, 5, 10, and 20 milliseconds were added. The other parameter values used were c_3=5.0, λ_P=0.95 and λ_{ML}=0.95.

Table 9 Summary of results for Ackley 20D with variable function evaluation time

Sleep time (milli-secs)	Basic PSO		Pheromone PSO		solution time % improved
	Avg Obj func	Duration (sec)	Avg Obj func	Duration (sec)	
0.0	4.659	16.898	0.171	15.095	10.67 %
5.0	4.622	115.671	0.146	103.148	10.83 %
10.0	5.016	185.863	0.065	177.323	4.59%
20.0	4.258	374.963	0.051	333.056	11.18%

The table summarizes average objective function values, solution times and the improvement in solution times between the basic and pheromone PSO from 20 solution runs. The results indicate that whereas the basic PSO attained a local minimum in all four sleep time scenarios, PSO with digital pheromones solved the problem with superior accuracy levels, very close to the global solution. The time improvement for a 10 millisecond sleep time was only 4.59% but basic PSO converged prematurely. Also, about two times out of 20 runs, PSO with digital pheromones converged to a local minimum at ~1.5 which increased the average objective function value in the case of 0 and 5 millisecond sleep time. Overall, when compared to the performance of basic PSO in all sleep time scenarios, PSO with digital pheromones displayed substantial improvement.

7.4 Statistical Analysis

7.4.1 Test Problem Settings

Four unconstrained test problems (problems 7.2.1 – 7.2.4) of varying dimensionality were used for performing hypothesis testing on their solution accuracy and solution times. These problems were solved using PSO with and without digital pheromones 35 times each to ensure an acceptable normal distribution. Two hypotheses were tested for each test problem with and without pheromones: a) whether the solution accuracy of PSO with digital pheromones compare better against basic PSO, and b) whether the solution times of PSO with digital pheromones compare better against basic PSO. The hypothesis tests were performed at a 95% confidence level, or a 0.05 probability for type I error.

$$H_0: \mu_1 - \mu_2 \leq 0 \text{ (null hypothesis)}$$
$$H_a: \mu_1 - \mu_2 > 0 \text{ (research or alternate hypothesis)} \quad (27)$$

As shown in equation (27) (same as equation (8)), the null hypothesis (H_0) states that basic PSO fares better in comparison to PSO with digital pheromones and the research hypothesis (H_a) states that PSO with digital pheromones has better performance characteristics than basic PSO. The five-step procedure for performing the hypothesis test was outlined in section 3.8.2, and was used for all four test cases. The pheromone parameter combinations are shown in equation (28). The $t_{critical}$ value for 0.05 probability in error (95% confidence level) for 68 degrees of freedom obtained from t-distribution tables is -1.645.

$$\begin{aligned} &\text{Combination 1: } c_3=2.0,\ \lambda_p = 0.85,\ \lambda_{ML} = 0.85 \\ &\text{Combination 2: } c_3=5.0,\ \lambda_p = 0.95,\ \lambda_{ML} = 0.95 \\ &\text{Combination 3: } c_3=5.0,\ \lambda_p = 0.85,\ \lambda_{ML} = 0.95 \\ &\text{Combination 4: } c_3=5.0,\ \lambda_p = 0.85,\ \lambda_{ML} = 0.85 \end{aligned} \quad (28)$$

The swarm size used for each test problem was chosen as 10 times the number of design variables with a maximum set to 500. The test problems were considered converged when the difference in objective function value was within 0.001 for 10 consecutive iterations. The computing platform for the trial runs was workstation running the Red Hat Enterprise Linux Operating System, with a processor speed of 3.2GHz and 2GB of system memory.

7.4.2 Results and Discussion

Table 10 explains in detail the results obtained from performing hypothesis testing on the 2D camelback function (problem 7.2.1) at a 95% confidence level. The first pheromone

parameter combination from equation (28) was used. From the table, n_1 and n_2 are the number of samples (trial runs) drawn from PSO with and without digital pheromones respectively.

Table 10 Hypothesis test results for Camelback 2D function

$\bar{x}_1 = -1.0166$	(solution mean of trial runs of basic PSO)
$\bar{x}_2 = -1.03152$	(solution mean of trial runs of PSO with digital pheromones)
$S_p = 0.000167087$	(from equation (10))
$df = 68$	(Degrees of freedom = $n_1 + n_2 - 2$)
$t_{calculated} = -1.145509941$	(using equation (9))
$t_{critical}$ (α=0.05) = -1.645	(from t-distribution tables with probability of error = α)

It can be seen from table that $t_{calculated}$ is greater than $t_{critical}$, which leads to the conclusion that the null hypothesis, H_0 can be rejected. This means that the solution quality of basic PSO is not better than PSO when implemented with digital pheromones. Since there is no evidence to prove that basic PSO fares better than PSO with digital pheromones, the research hypothesis (H_a) that the solution quality of PSO with digital pheromones is better than a basic PSO is considered as 'accepted'. The solution quality and solution timings for all test problems are estimated using the procedure laid out in Table 10. Table 11 below summarizes the hypothesis testing of the camelback problem (problem 7.2.1) for solution accuracy and solution timings for all stated combinations of pheromone parameters.

Table 11 Summary of hypothesis testing for Camelback 2D problem

Test Problem	**Combination 1**		**Combination 2**		**Combination 3**		**Combination 4**	
7.2.1	t_{calc}	H_0	t_{calc}	H_0	t_{calc}	H_0	t_{calc}	H_0
Solution accuracy	-1.145 ($> t_{critical}$)	R	-1.567 ($> t_{critical}$)	R	-1.685 ($< t_{critical}$)	A	-2.288 ($< t_{critical}$)	A
Solution Times	-0.488 ($> t_{critical}$)	R	0.210 ($> t_{critical}$)	R	-0.495 ($> t_{critical}$)	R	-0.322 ($> t_{critical}$)	R

Legend: R – Rejected, A - Accepted

This table shows that the null hypothesis, H_0 is rejected in two different instances for solution times – when using pheromone parameter combination one and two. This means that the solution quality of basic PSO is not better than PSO with digital pheromones at a 95% confidence level. The results from hypothesis testing of solution times also concur with the fact that digital pheromones has a positive influence in reducing the solution times when compared to basic PSO. However, with the pheromone parameter combinations three and four, the null hypothesis was accepted at 95% confidence level. That means basic PSO performed better when compared to PSO with digital pheromones in terms of solution quality. However, the null hypothesis is rejected for pheromone combinations three and four for solution times. While this hypothesis test demonstrates that not all suggested pheromone parameter combinations can be beneficial, it points to the fact that slight changes in the values for pheromone parameters substantially affects the performance of PSO when implemented with digital pheromones, especially in two-dimensional optimization problems.

Table 12 summarizes the hypothesis testing for the Himmelblau 2D function (problem 7.2.2) for solution accuracy and timings for all stated pheromone combinations.

Table 12 Summary of hypothesis testing for Himmelblau 2D problem

Test Problem	**Combination 1**		**Combination 2**		**Combination 3**		**Combination 4**	
7.2.2	t_{calc}	H_0	t_{calc}	H_0	t_{calc}	H_0	t_{calc}	H_0
Solution accuracy	-0.195 ($> t_{critical}$)	R	-0.551 ($> t_{critical}$)	R	-0.587 ($> t_{critical}$)	R	-0.675 ($> t_{critical}$)	R
Solution Times	2.390 ($> t_{critical}$)	R	0.365 ($> t_{critical}$)	R	0.339 ($> t_{critical}$)	R	1.296 ($> t_{critical}$)	R

Legend: R – Rejected, A - Accepted

The results from Table 12 shows that the calculated *t* value ($t_{calculated}$) is greater than $t_{critical}$ for all pheromone parameter combinations. This means that the null hypothesis stating that basic PSO is better than PSO with digital pheromones can be rejected at an error probability of 0.05. Both the solution quality and solution times suggest that H_0 can be rejected. Since there is no other evidence showing that basic PSO performs better, the research hypothesis that pheromone PSO performs better is accepted.

Table 13 summarizes the hypothesis testing for the Rosenbrock 5D function (problem 7.2.3) for solution accuracy and solution timings.

Table 13 Summary of hypothesis testing for Rosenbrock 5D problem

Test Problem	**Combination 1**		**Combination 2**		**Combination 3**		**Combination 4**	
7.2.3	t_{calc}	H_0	t_{calc}	H_0	t_{calc}	H_0	t_{calc}	H_0
Solution accuracy	-0.721 ($> t_{critical}$)	R	-1.128 ($> t_{critical}$)	R	-1.184 ($> t_{critical}$)	R	0.640 ($> t_{critical}$)	R
Solution Times	1.089 ($> t_{critical}$)	R	1.017 ($> t_{critical}$)	R	0.797 ($> t_{critical}$)	R	0.552 ($> t_{critical}$)	R

Legend: R – Rejected, A - Accepted

Hypothesis testing for solution quality and solution times of Rosenbrock 5D problem showed that the null hypothesis can be rejected. The table shows that the $t_{calculated}$ value was greater than $t_{critical}$ for all suggested pheromone parameter combinations. Therefore, the null hypothesis stating that basic PSO performs better when compared to PSO with digital pheromones is rejected. Since there is no other evidence to prove that basic PSO can perform better, the research hypothesis stating that PSO with digital pheromones has better performance characteristics in terms of solution quality and solution timings.

Table 14 summarizes the hypothesis testing for the Ackley's path 10D function (problem 7.2.4)

Table 14 Summary of hypothesis testing for Ackley 10D problem

Test Problem	**Combination 1**		**Combination 2**		**Combination 3**		**Combination 4**	
7.2.4	t_{calc}	H_0	t_{calc}	H_0	t_{calc}	H_0	t_{calc}	H_0
Solution accuracy	-0.418 ($> t_{critical}$)	R	-0.443 ($> t_{critical}$)	R	-0.429 ($> t_{critical}$)	R	-0.423 ($> t_{critical}$)	R
Solution Times	-0.219 ($> t_{critical}$)	R	0.040 ($> t_{critical}$)	R	0.187 ($> t_{critical}$)	R	-0.233 ($> t_{critical}$)	R

Legend: R – Rejected, A - Accepted

This table shows that the null hypothesis, H_0 is rejected for all suggested combinations of pheromone parameters. This means that the hypothesis testing demonstrates that basic PSO is not better than PSO with digital pheromones at a 0.05 probability for error. The fact that $t_{calculated}$ value exceeds $t_{critical}$ value for both solution quality and solution times suggest that the research hypothesis H_a can be accepted due to the lack of evidence to prove superior performance of basic PSO. That means that PSO with digital pheromone PSO compares better against basic PSO in terms of both solution quality and solution timings for this test problem.

It can be seen that all combinations of pheromone parameters (except combinations three and four for Camelback 2D, i.e. problem 7.2.1) that the null hypothesis (H_0) was rejected. That means that there is no evidence to prove that basic PSO fares better than PSO with digital pheromones. Therefore, it can be inferred that the research hypothesis (H_a) is accepted. The pheromone parameter combinations three and four for the Camelback 2D function suggest that they are not suitable values for lower dimensional problems. This is

understandable because the confidence parameter c_3 was set to a high value of 5.0, which potentially increased pheromone activity causing longer times needed to converge.

To check if a lower value for c3 was suitable for lower dimensional problems, an additional test was performed. In this case, hypothesis testing was performed on a 100D Ackley's path function using the same pheromone parameter combinations. Table 15 summarizes the results.

Table 15 Summary of hypothesis testing for Ackley 100D problem

Test Problem	**Combination 1**		**Combination 2**		**Combination 3**		**Combination 4**	
Ackley 100D	t_{calc}	H_0	t_{calc}	H_0	t_{calc}	H_0	t_{calc}	H_0
Solution accuracy	0.233 ($> t_{critical}$)	R	-1.247 ($> t_{critical}$)	R	-1.038 ($> t_{critical}$)	R	0.314 ($> t_{critical}$)	R
Solution Times	-4.817 ($< t_{critical}$)	A	-1.054 ($> t_{critical}$)	R	-0.130 ($> t_{critical}$)	R	-4.305 ($< t_{critical}$)	A

Legend: R – Rejected, A - Accepted

Table 15 is the result of performing hypothesis testing on the 100 design variable Ackley's path function. The table shows that the $t_{calculated}$ values for solution quality was greater than $t_{critical}$ for all combinations of suggested pheromone parameters. This means that a lower value for c_3 is acceptable for a higher dimensional problem (combination one). However, a high value of c_3 may not be suitable for a lower dimensional problem (camelback 2D). Also for this problem, the null hypothesis stating that basic PSO fares better in comparison to pheromone PSO can be rejected.

It can be noted that the hypothesis testing of solution times shows that the null hypothesis can be accepted for pheromone combinations one and four. This means that the solution times for basic PSO were better when compared to PSO with digital pheromones. However, it comes at a cost of generally poor solution accuracy. This means that although basic PSO compares better against PSO with digital pheromones in terms of solution times, basic PSO was unable to solve the problem with a reasonable accuracy.

7.5 Coarse Grain Parallelization Results

In this section, results from solving problems 7.2.5 – 7.2.10 (shown in Table 16) in a synchronous coarse grain parallel computing environment are presented.

Table 16 Test problem matrix for synchronous coarse grain parallelization

Problem	Test Problem	Dimensions
7.2.5	Dixon and Price function	15
7.2.6	Ackley's path function	20
7.2.7	Levy function	25
7.2.8	Sum of Squares function	30
7.2.9	Spherical function	40
7.2.10	Griewank function	50

7.5.1 Test Problem Settings

The pheromone parameter values established by testing digital pheromones in PSO with 128 different settings as described in section 7.3 were used for the evaluation of the developed coarse grain parallelization method. Six unconstrained problems listed in table 15 were used for this purpose. Though customization of parameters could potentially

improve the solution characteristics, the following parameter values catered well for most problems:

- c_3 = 5.0 with no decay
- Pheromone decay = 0.95, and
- Move limit decay = 0.95

The swarm size was defined as 10 times the number of design variables, and was limited to a maximum of 500 as the dimensionality of the problems increased. A total of 20 trial runs were performed for each test case, with and without digital pheromones. The test problems were again considered converged when the difference in objective function values was within 0.001 for 10 consecutive iterations. All test cases were solved on 2, 4, and 8 Intel Xeon processors (3.06 GHz) of a RedHat Linux cluster that houses 2GB system memory per node and high bandwidth Myrinet network switches. The algorithm was implemented using the C++ programming language and MPI communication libraries (MPICH implementation) for data distribution between processors.

The results section is divided into two main categories:

a. Performance evaluation of PSO with and without digital pheromones: Accuracy, efficiency, and reliability of PSO with digital pheromones are compared against basic PSO in a parallel computing architecture. This is described in section 7.5.1.1
b. Parallel performance: Evaluation of the developed method for adaptability to the parallel computing architecture. This involves evaluating parallel speedup and efficiencies of PSO with digital pheromones. This is described in section 7.5.1.2

7.5.2 Results and Discussion: Evaluation With/Without Pheromones

Table 17 provides a summary of objective function values obtained from the test runs on 2, 4, and 8 processors. Upon convergence on all participating processors, the root processor gathered the solution information and sorted for best objective function values. The root processor retained these values and the others were discarded. Results displayed in table 2 indicated these sorted values. The table contains three markers that assess the performance of the developed method – average objective function value, smallest objective function value and the standard deviation. The smallest objective function is the lowest value obtained in 20 trial runs for each test problem.

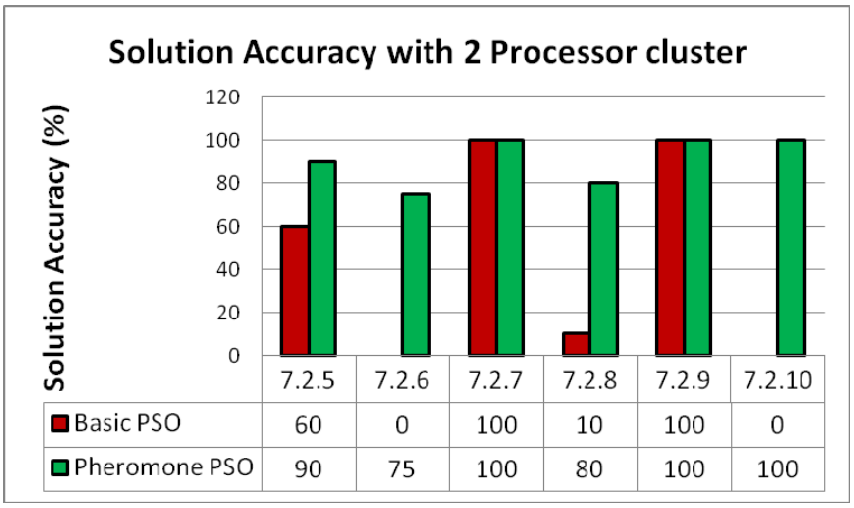
Table 17 Summary of solutions from coarse grain parallelization

	Objective Function (2 Processors)			**Objective Function (4 Processors)**			**Objective Function (8 Processors)**		
	Average	Smallest	Std Dev.	Average	Smallest	Std Dev.	Average	Smallest	Std Dev.
7.2.5 (B)	14.170	0.002	35.015	34.822	0.003	126.622	3.628	0.001	13.676
7.2.5 (P)	0.211	0.001	0.643	0.098	0.001	0.324	0.441	0.001	1.083
7.2.6 (B)	5.456	2.245	3.504	6.181	1.991	4.031	5.946	1.900	4.013
7.2.6 (P)	0.354	0.002	0.723	0.393	0.004	0.890	0.133	0.002	0.399
7.2.7 (B)	0.134	0.131	0.002	0.133	0.130	0.002	0.134	0.131	0.003
7.2.7 (P)	0.131	0.130	0.001	0.131	0.130	0.001	0.131	0.130	0.001
7.2.8 (B)	3.166	0.259	3.212	1.531	0.273	1.070	2.178	0.220	2.313
7.2.8 (P)	0.228	0.006	0.304	0.236	0.005	0.220	0.174	0.003	0.278
7.2.9 (B)	0.035	0.017	0.014	0.036	0.012	0.018	0.031	0.009	0.019
7.2.9 (P)	0.002	0.001	0.001	0.002	0.001	0.001	0.002	0.001	0.001
7.2.10 (B)	1.151	1.067	0.064	1.146	1.051	0.044	1.155	1.061	0.074
7.2.10 (P)	0.012	0.003	0.009	0.011	0.004	0.006	0.009	0.003	0.005

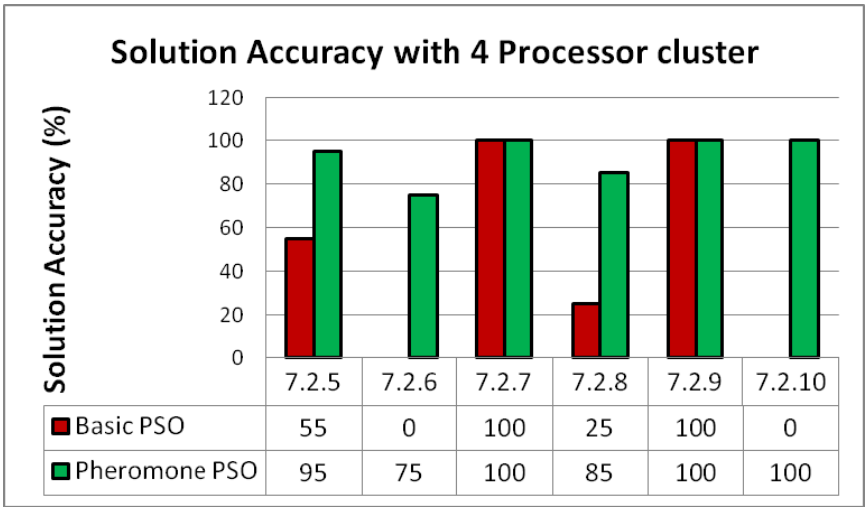
Legend: 7.2.5 – Dixon and Price function15D, 7.2.6 – Ackley's path function 20D, 7.2.7 – Levy function 25D, 7.2.8 – Sum of squares function 30D, 7.2.9 – Spherical function 40D, 7.2.10 – Griewank function 50D. (B) – Results from Basic PSO, (P) – Results from PSO with Digital Pheromones implemented.

The results in the table show that PSO with digital pheromones (designated with a P) consistently displayed superior performance when compared with solutions from a basic PSO (designated with a B).

Since the published solutions for the solved problems were 0.000, there was no measure to determine the percentage accuracy. Therefore, a tolerance was again given and accuracy was measured based on the number of times the obtained solution was within the tolerance limits.



	7.2.5	7.2.6	7.2.7	7.2.8	7.2.9	7.2.10
Basic PSO	60	0	100	10	100	0
Pheromone PSO	90	75	100	80	100	100



	7.2.5	7.2.6	7.2.7	7.2.8	7.2.9	7.2.10
Basic PSO	55	0	100	25	100	0
Pheromone PSO	95	75	100	85	100	100

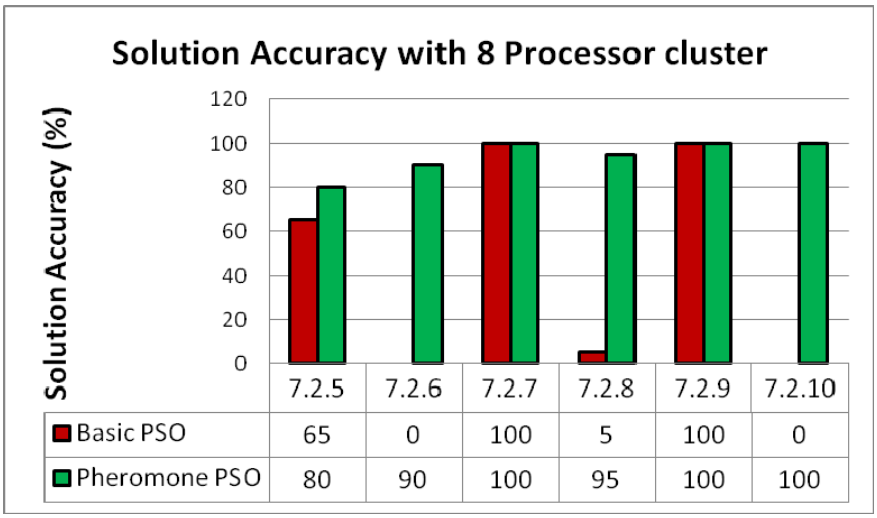


Figure 32 Solution accuracy measure across 2, 4, and 8 processors

Figure 32 shows the solution accuracy charts for the test problems across using the different number of processors. As evident from Figure 32, the solution accuracy of PSO with digital pheromones was either equal or superior when compared to basic PSO. For example, in problem 7.2.6 (Ackley's 20D), the basic PSO was not able to solve the problem whereas the pheromone PSO attained the solution within the specified tolerance limits 75 out of 100 runs on 2 and 4 processors. When tested with 8 processors, the accuracy of pheromone PSO increased to 90%.

The published solution for the 15 dimensional Dixon and Price function (problem 7.2.5) is 0.000 and the swarm in basic PSO was unable to locate the optimum on any of 2, 4 or 8 processor runs. The swarm reached closer to the optimum on the 8 processor cluster (3.628) but was still well out of the tolerance limits. On the other hand, PSO with digital pheromones solved the problem with a solution accuracy ranging between 80-95%. The standard deviation of pheromone PSO, as evident from Table 17, was substantially better than basic

PSO as well. Also, in all the participating processors, the average solution time per run was at least 23% shorter for pheromone PSO than for basic PSO.

Both basic and pheromone PSO were able to solve the 25 dimensional Levy function (problem 7.2.7) within the tolerance limits as evident from the 100% solution accuracy in Figure 32. Although the average, smallest and standard deviations between the methods (with and without digital pheromones) were quite close to each other on results from all processors, there was almost a 50% decrease in the solution time for pheromone PSO, showing that it was much more efficient than a basic PSO. Table 18 summarizes the average solution time and number of iterations for all test cases across 2, 4 and 8 processors.

Table 18 Summary of solution times and number of iterations from coarse grain parallelization

	2 Processors		**4 Processors**		**8 Processors**	
	Avg # iterations	Avg Duration (secs/run)	Avg # iterations	Avg Duration (secs/run)	Avg # iterations	Avg Duration (secs/run)
7.2.5 (B)	162.4	27.45	171.9	30.16	171.0	33.29
7.2.5 (P)	116.9	20.45	118.9	23.17	113.7	24.77
7.2.6 (B)	142.8	18.69	125.0	19.65	139.2	20.07
7.2.6 (P)	141.2	16.68	137.9	17.23	146.9	17.84
7.2.7 (B)	95.8	25.68	97.7	26.39	93.9	27.17
7.2.7 (P)	40.5	12.28	41.2	12.55	41.9	12.98
7.2.8 (B)	211.5	68.07	210.1	69.94	211.7	71.67
7.2.8 (P)	151.2	53.94	154.4	55.83	149.0	57.44
7.2.9 (B)	163.4	70.88	165.2	71.98	160.0	73.76
7.2.9 (P)	85.2	39.80	83.3	40.52	85.1	41.57
7.2.10 (B)	182.9	100.31	180.8	101.99	185.3	104.48
7.2.10 (P)	159.7	96.45	159.6	96.71	159.5	97.65

The lowest objective function value obtained by the 30 dimensional sum of squares function (problem 4.4) by basic PSO was 0.220. Compared to the published solution of 0.000, this value is within the tolerance limits. However, the solution accuracy resulting from basic PSO method was only within the range of 5-25% when tested across 2, 4 and 8 processors. On the other hand, pheromone PSO was able to solve the problem within tolerance 80-95% of the time. Moreover, the solution time was 20% better than the basic PSO.

Both basic PSO and pheromone PSO solved the 40 dimensional spherical function (problem 7.2.5) within the tolerance limits resulting in 100% accuracy. That means that both basic PSO and pheromone PSO were able to find a solution within specified tolerance limits in the 20 trial runs. However, it can be observed from Table 17 that the average objective function evaluated by pheromone PSO (0.002) is about 15 times better than the lowest average objective function returned by basic PSO (0.031). Moreover, the highest solution time for pheromone PSO (41.57 seconds per run on 8 processor cluster) is about 43% shorter when compared to that of basic PSO. Results from all the processors consistently showed a very small variation in the solution (standard deviation of 0.001), which proves the reliability of pheromone PSO.

The highly multimodal 50 dimensional Griewank function (problem 7.2.10) was also attempted to solve using basic and pheromone PSO. While the basic PSO could not reach the global optimum on any of the 20 trials across 2, 4 and 8 processors, pheromone PSO was able to obtain a solution within the tolerance limits in all the trials. In addition, the variation of the results as seen from the standard deviation values in Table 17 show that pheromone

PSO was significantly consistent in performance when compared to basic PSO. This also must be taken into account when considering iterations and solution times as basic PSO converged prematurely on every single solution run.

In all the test cases, the digital pheromone implementation of PSO displayed superior performance characteristics in terms of accuracy (closeness to published solution), efficiency (solution duration) and reliability (standard deviation) when implemented in a parallel architecture.

7.5.3 Results and Discussion: Parallel Efficiency and Speedup Characteristics

The output from parallelizing an algorithm is typically measured and compared in terms of speedup and parallel efficiency. The speedup defines how fast a code runs in parallel; it is a ratio of the amount of time the code spends in communication to the amount of time it spends on computing. If the time taken to run a code on one processor is t_1 seconds, and the time it takes to run the same code on 'p' processors is t_p seconds, then the parallel speedup is given by equation (29).

$$\text{Speedup} = \frac{t_1}{t_p} \tag{29}$$

Parallel efficiency is a percentage measure of how well the available processors are used. In other words, it provides information on how well the load balancing is maintained. Equation (30) shows the procedure for calculating parallel efficiency.

$$\text{Parallel Efficiency} = \frac{Speedup}{p} \tag{30}$$

The parallel performance characteristics of pheromone PSO can be measured through speedup and efficiency calculations. Ideally, the parallel speedup should be equal to the number of processors, and the parallel efficiency should be 100%. However, due to communication latencies, these values do not usually reach these ideal values. Therefore, the measure of parallel performance is based on how close they are to ideal values. In this section, the parallel performance characteristics were evaluated and presented for Pheromone PSO.

Figure 33 shows the speedup characteristics of pheromone PSO plotted for all participating processors. The plot was generated based on the speedup values evaluated for each test problem running on 2, 4 and 8 processors. The x-axis portrays various test problems with their dimensionality and the y-axis shows the parallel speedup.

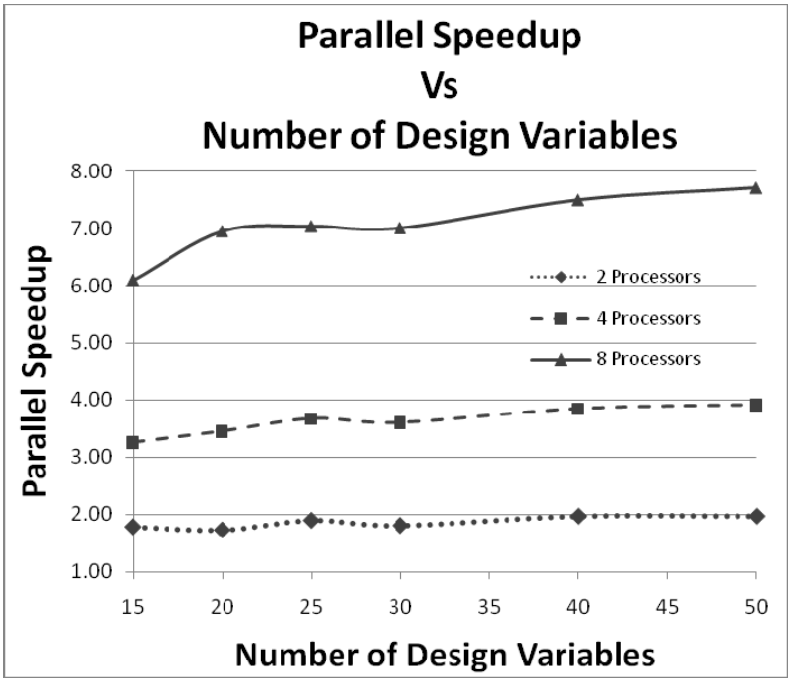


Figure 33 Parallel Speedup characteristics of PSO with digital pheromones

The plot shows that the parallel speedup was almost ideal when two processors were used alone. This means that the network latencies have a very small effect on the two swarms in a parallel architecture. When four swarms were deployed on a four processor system, the speedup did not reach the ideal as quickly. However, parallel speedup approached the ideal value of 4.00 as the dimensionality of the problem increased to 40 (spherical function – problem 7.2.9). With eight swarms simultaneously deployed, a plateau was noticed at a non-ideal speedup (i.e., 7.00) when the problem dimensionality was between 20 (Ackley's path – problem 7.2.6) and 30 (Sum of squares – problem 7.2.8). The speedup gradually increased towards ideal (i.e., 8.00) as the problem dimensionality increased to 50 (Griewank function – problem 7.2.10).

It can be inferred from these findings that the processor communication latencies had more influence and hence a lower parallel speed up was noticed on lower dimensional problems. As the problem dimensionality increases along with the number of processors, the network latencies are offset and near ideal parallel speedups are attained.

Figure 34 shows the parallel efficiency characteristics of pheromone PSO across the participating processors.

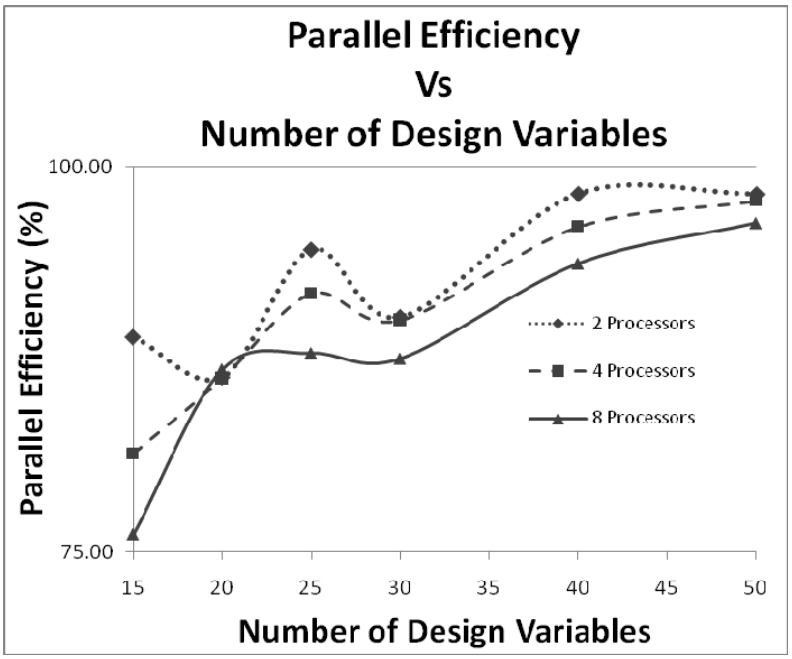


Figure 34 Parallel Efficiency characteristics of PSO with digital pheromones

Parallel efficiency provides load balancing information and is dependent upon the speedup and the number of participating processors. A 100% parallel efficiency means that the system is perfectly load balanced. Figure 34 shows that the parallel efficiency values ranged from 85% through 75% across 2, 4, and 8 processors for a 15 dimensional Dixon and Price function (problem 7.2.5). For this problem, this means that the load balancing worsened when the number of processors increased from 2 to 8. This trend more or less continued to the remaining problems as well. However, the decrease in parallel efficiency was negligible with higher dimensional problems (i.e., Griewank 50D). This means that the load balancing improved considerably as the dimensionality of the problems increased.

Figure 35 shows the relation between parallel efficiency and the number of processors in the context of the six test problems. Figure 35 is equivalent to Figure 34, but the relation

between parallel efficiency and number of processors in the context of test cases is better understood.

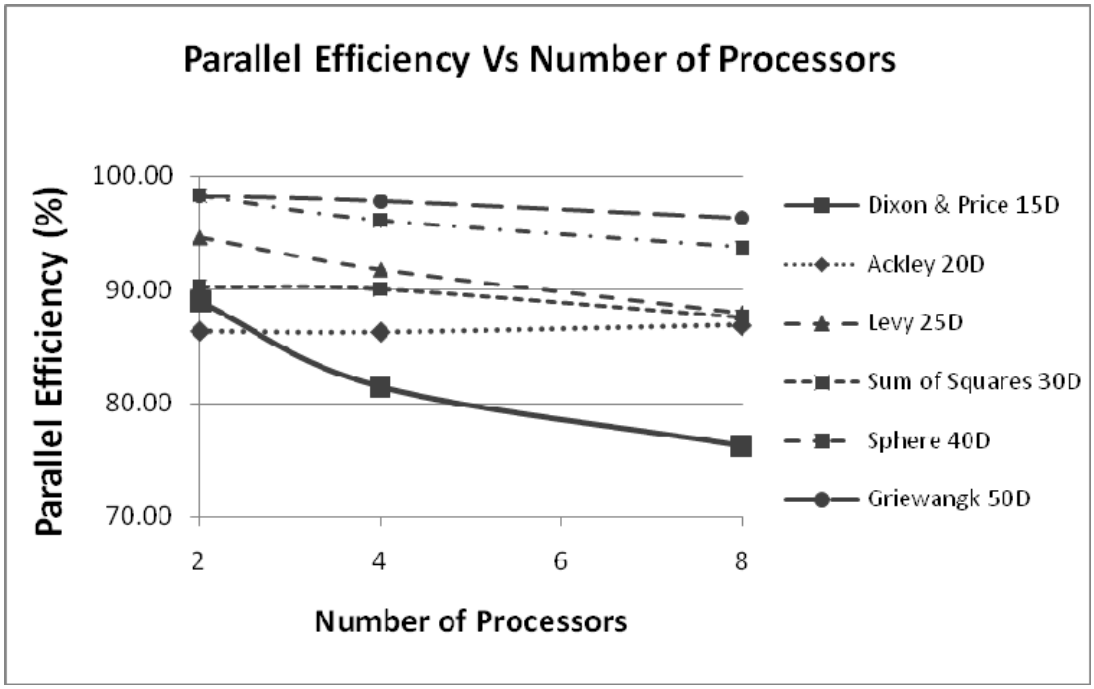
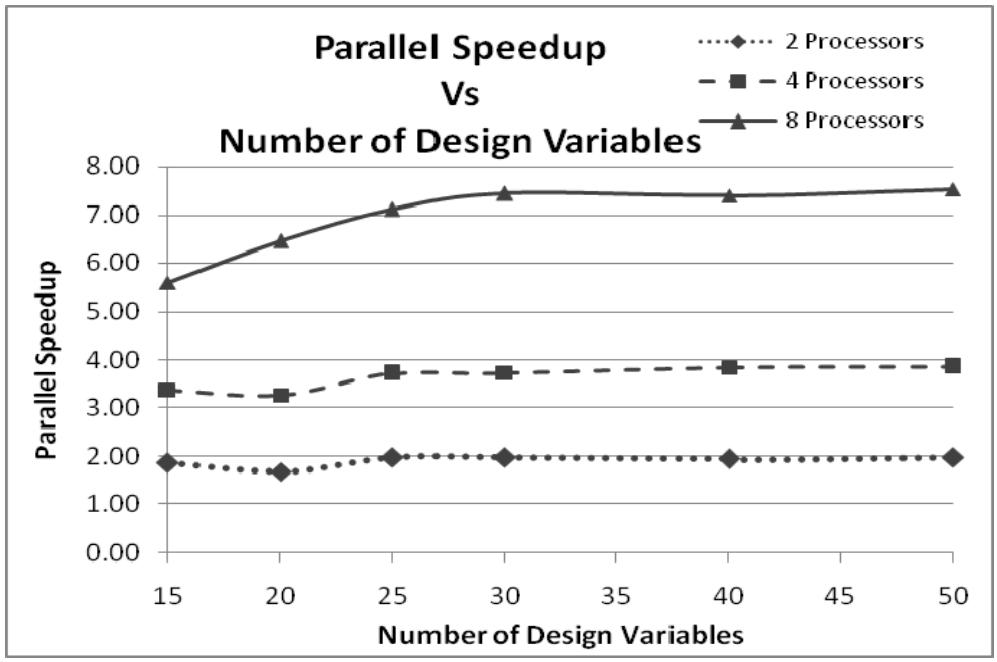


Figure 35 Effect of number of processors on parallel efficiency

With the increase in problem dimensionality, a pattern of increasing parallel efficiency can be observed in Figure 35. The 15 dimensional Dixon and Price function (problem 7.2.5) has the smallest parallel efficiency, while the 50 dimensional Griewank function (problem 7.2.10) shows the best efficiency characteristics. The parallel efficiencies for the test problems gradually increased as the dimensionality increased, which corroborates with the findings shown in Figure 34 as well.

A similar speedup and efficiency study was also performed on the basic PSO method with multiple swarms traversing the design space, and the results concur with the pattern of findings reported above. Figure 36 show the plots from testing basic PSO.



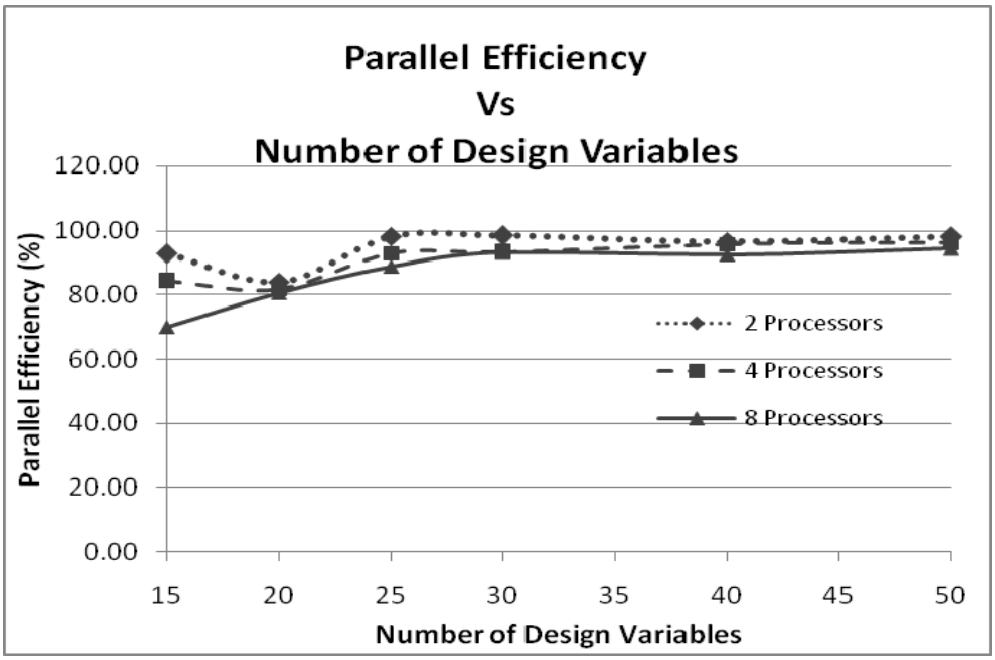


Figure 36 Charts for Basic PSO: Speedup (Left), Parallel Efficiency (Right)

The pheromone parameter settings used were the default ones determined. However, with refined settings, the solution characteristics could be further improved. For example, solving the 40D sphere problem with the suggested pheromone parameters resulted in an

average objective function value of 0.0019 that took an average of 41.57 seconds per run on an 8 processor computing environment. However, with tuned parameter settings, the best solution achieved was 0.0015 that took an average of 29.76 seconds per run. The altered parameter values showed a 20% improvement in the objective function value along with a 28% improvement in solution time. The changes in these parameters are problem dependent and currently do not have mathematical rules to ascertain the most optimal parameter settings. In spite of the additional computations per iteration due to pheromone operations, solution times nearly always decreased when compared to a basic PSO. This is attributed to the information provided by the digital pheromones thereby facilitating the swarms in propagating towards the global optimum faster.

7.6 Shared Pheromone Parallelization Results

In this section, results from implementing the shared pheromone parallelization scheme are presented. Problems 7.2.5 – 7.2.10 (shown in Table 19) were used as test cases.

Table 19 Test problem matrix for shared pheromone parallelization

Problem	Test Problem	Dimensions
7.2.5	Dixon and Price function	15
7.2.6	Ackley's path function	20
7.2.7	Levy function	25
7.2.8	Sum of Squares function	30
7.2.9	Spherical function	40
7.2.10	Griewank function	50

7.6.1 Test Problem Settings

The pheromone parameters used for shared pheromone parallelization were the same as in coarse grain parallelization. They are:

- $c_3 = 5.0$ with no decay,
- Pheromone decay, $\lambda_p = 0.95$, and
- Move limit decay, $\lambda_{ML} = 0.95$

Though customization of parameters for each problem would further improve solution characteristics, the default parameter values catered well for most problems. A total of 20 trial runs were performed for each test case. All test cases were solved on 2, 4, and 8 Intel Xeon processors (2.8 GHz) of a RedHat Linux distributed memory cluster that houses high bandwidth Myrinet network switches. The algorithm was implemented using the C++ programming language with the MPI communication libraries (MPICH implementation) for data distribution across processors. As a general rule of thumb, the swarm size was defined as 10 times the number of design variables, and was capped at 500 per processor as the dimensionality increased.

To evaluate the performance of the developed method, results were compared against: a) basic PSO, and b) pheromone PSO, both executed in a coarse grained parallel strategy. One of the comparison measures for coarse grained and shared pheromone parallelization was the swarm size. In coarse grained parallelization, swarms propagated independently of each other and no communication existed between processors. This meant that a coarse grained execution of a 15 dimensional problem with four processors had one swarm containing 150 particles per processor. However, shared pheromone parallelization with a similar problem set up had 150 swarm members on each processor, with communication between them

effectively putting the total swarm size at 150x4 = 600. Moreover, the swarm operations and hence swarm size only occurred on the optimization processors. The shared pheromone processor only performed pheromone operations.

To address this issue, two different swarm schemes were developed in the shared pheromone parallelization method: a) fixed swarm size per processor and b) fixed overall swarm size. For the fixed swarm size per processor a swarm size of 10 times the number of design variables per processor was mandated. For example, an 8 processor run for a 20 design variable problem would have 200 swarm members per optimization processor. For the fixed overall swarm size a swarm size of 10 times the number of design variables overall was used. For example, an 8 processor run (7 optimization processors + 1 pheromone processor) for a 20 design variable problem will have 200/7 = ~29 members per optimization processor. Any fractional value when distributing the swarm over the optimization processors in this scheme was rounded to the next nearest whole number. Using this method allowed the shared pheromone method to be more precisely compared to the basic and coarse grain PSO implementations.

7.6.2 Results and Discussion: Fixed Swarm Size per Processor

Table 20 provides a summary of objective function values obtained from test runs on 2, 4, and 8 processors with swarm sizes 10 times the number of design variables per processor. Although averages show a general trend, they are not a true indicator of the method's performance. Therefore, the smallest value achieved and the standard deviation is also

reported to denote the reach of multiple swarms in the design space and their reliability when communicating through digital pheromones.

Table 20 Summary of solutions from shared pheromone parallelization

	Objective Function (2 Processors)			**Objective Function (4 Processors)**			**Objective Function (8 Processors)**		
	Average	Smallest	Std Dev.	Average	Smallest	Std Dev.	Average	Smallest	Std Dev.
7.2.5 (B)	14.170	0.002	35.015	34.822	0.003	126.622	3.628	0.001	13.676
7.2.5 (P)	0.211	0.001	0.643	0.098	0.001	0.324	0.441	0.001	1.083
7.2.5 (SP)	0.635	0.355	0.238	0.386	0.018	0.237	0.206	0.032	0.183
7.2.6 (B)	5.456	2.245	3.504	6.181	1.991	4.031	5.946	1.900	4.013
7.2.6 (P)	0.354	0.002	0.723	0.393	0.004	0.890	0.133	0.002	0.399
7.2.6 (SP)	0.368	0.013	0.274	0.139	0.025	0.095	0.059	0.002	0.050
7.2.7 (B)	0.134	0.131	0.002	0.133	0.130	0.002	0.134	0.131	0.003
7.2.7 (P)	0.131	0.130	0.001	0.131	0.130	0.001	0.131	0.130	0.001
7.2.7 (SP)	0.130	0.130	0.000	0.130	0.130	0.000	0.130	0.129	0.000
7.2.8 (B)	3.166	0.259	3.212	1.531	0.273	1.070	2.178	0.220	2.313
7.2.8 (P)	0.228	0.006	0.304	0.236	0.005	0.220	0.174	0.003	0.278
7.2.8 (SP)	0.107	0.001	0.168	0.021	0.000	0.019	0.011	0.000	0.017
7.2.9 (B)	0.035	0.017	0.014	0.036	0.012	0.018	0.031	0.009	0.019
7.2.9 (P)	0.002	0.001	0.001	0.002	0.001	0.001	0.002	0.001	0.001
7.2.9 (SP)	0.003	0.000	0.005	0.001	0.000	0.000	0.001	0.000	0.001
7.2.10 (B)	1.151	1.067	0.064	1.146	1.051	0.044	1.155	1.061	0.074
7.2.10 (P)	0.012	0.003	0.009	0.011	0.004	0.006	0.009	0.003	0.005
7.2.10 (SP)	0.375	0.000	0.310	0.170	0.000	0.150	0.099	0.001	0.133

Legend: 7.2.5 – Dixon and Price function15D, 7.2.6 – Ackley's path function 20D, 7.2.7 – Levy function 25D, 7.2.8 – Sum of squares function 30D, 7.2.9 – Spherical function 40D, 7.2.10 – Griewank function 50D. (B) – Results from Basic PSO from coarse grain parallelization, (P) – Results from PSO with Digital Pheromones implemented from coarse grain parallelization, (SP) – Results from shared pheromone parallelization.

The results in Table 20 show that the shared pheromone parallelization (designated with an 'SP') consistently showed improvement in average objective function values returned when compared to the coarse grained pheromone PSO (designated with a 'P') and out-performed basic PSO (designated with a 'B') in all test cases. This trend is especially pronounced when

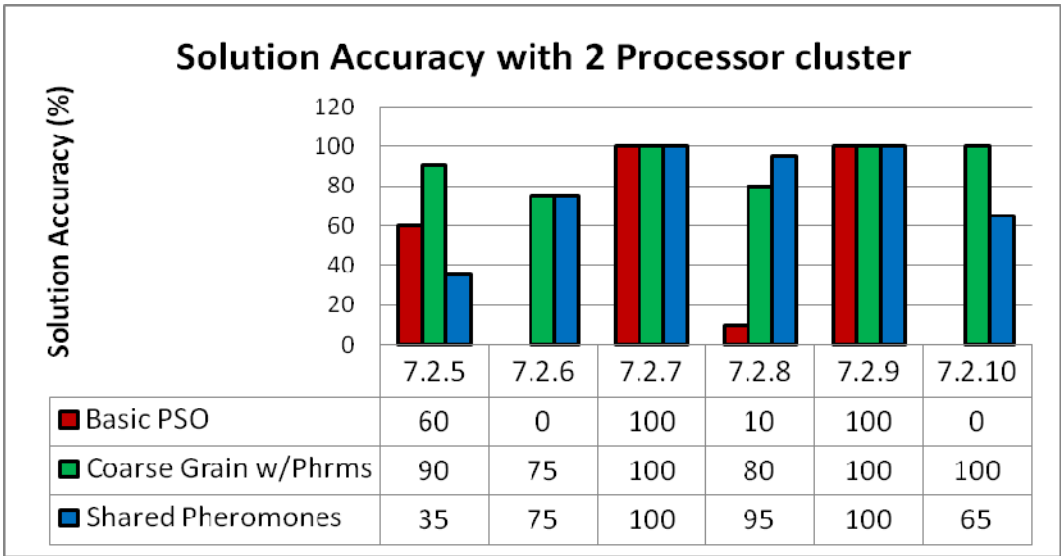
the number of processors increased suggesting that the method's performance improves as the number of processors grows and hence the swarm size grows. This evidence suggests that design space information is being better distributed throughout the swarm through digital pheromones and the parallelization method. For example on the 20 dimensional Ackley's path function (problem 7.2.6) whose published solution is 0.000, the average objective function obtained with 2 processors was 0.368, but the value improved to 0.139 using 4 processors and to 0.059 with 8 processors.

While the results showed a superior performance improvement over basic PSO, results from the coarse grain pheromone PSO outperformed the shared pheromone parallelization results in some of the test cases, especially on 2 processor runs. The reason for this behavior can be attributed to the number of swarms.

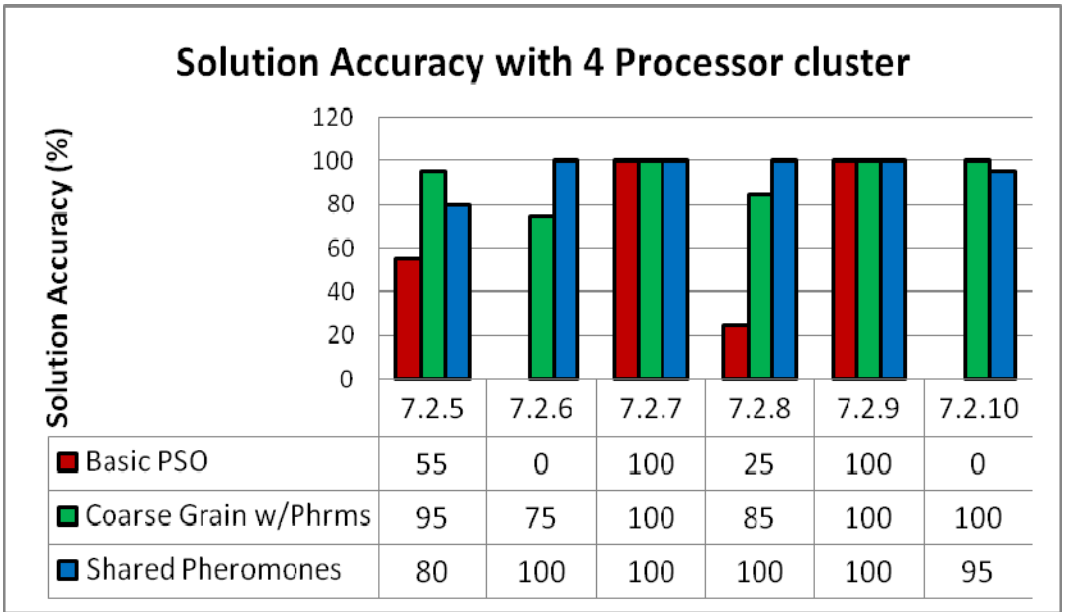
A 2 processor shared pheromone parallel execution has only one swarm (one swarm on the one optimization processor) while a 2 processor coarse granular execution has two swarms searching the design space. It is theorized that this difference in the number of swarms caused the coarse granular pheromone PSO to outperform the shared pheromone parallelization. However, as the number of participating optimization processors increased, the shared pheromone parallelization method performed better. For example, the average objective function value returned by the shared pheromone parallelization for the 15 dimensional Dixon & Price function (problem 7.2.5) was 0.635 when compared to 0.211 for coarse grained pheromone PSO on two processors. With 8 processors however, the shared

pheromone method returned 0.206 as opposed to 0.441 on coarse grained pheromone PSO. This is a 53% improvement in the objective function value.

Table 20 also shows that the standard deviations of the shared pheromone parallelization improved as the number of processors increased. On the 30 dimensional sum of squares function (problem 7.2.8), the standard deviation was 0.168 on 2 processors when compared to 0.017 on 8 processors. The standard deviation also improved when compared to coarse grained PSO and pheromone PSO as well. This suggests that the consistency and reliability in solving the problem increases with the developed method and improves as more processors were used. The particle swarms received more information from shared pheromones resulting in reliable solutions.



	7.2.5	7.2.6	7.2.7	7.2.8	7.2.9	7.2.10
Basic PSO	60	0	100	10	100	0
Coarse Grain w/Phrms	90	75	100	80	100	100
Shared Pheromones	35	75	100	95	100	65



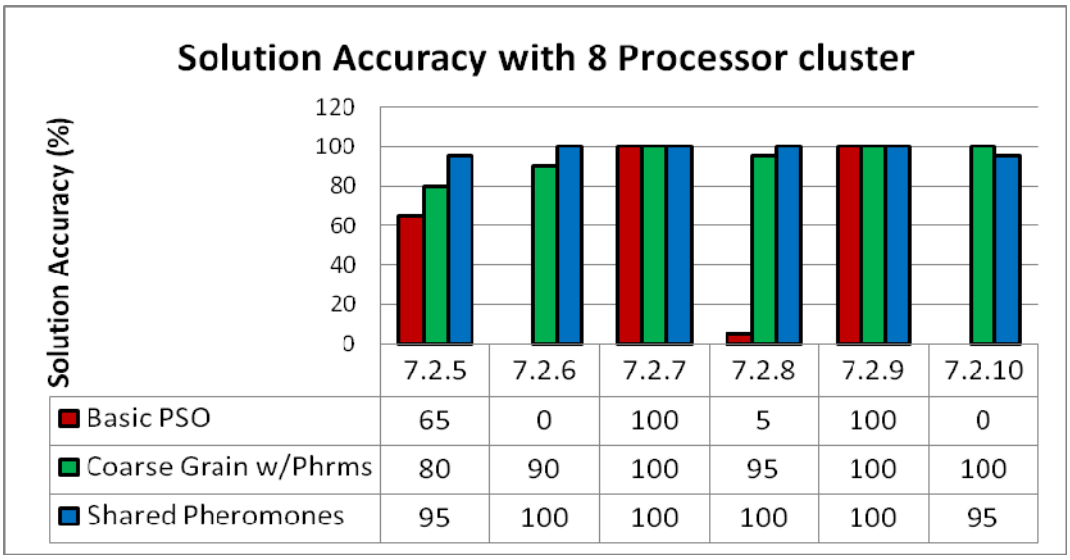


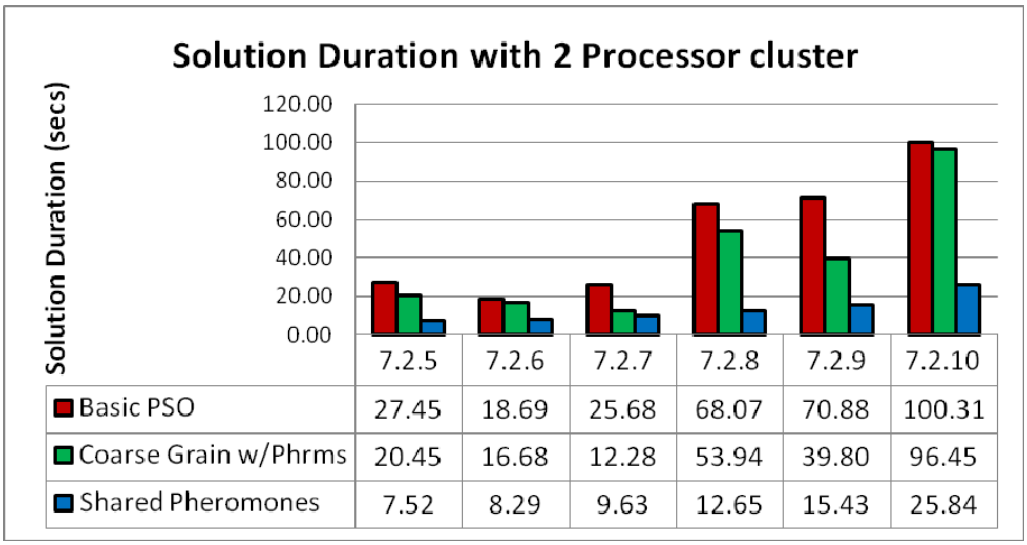
Figure 37 Solution accuracy charts for test problems with fixed swarm size per processor

Figure 37 shows the solution accuracy charts for test problems 7.2.5 – 7.2.10 on 2, 4, and 8 processors. The published solutions to these problems are 0.0, so a tolerance value close to the optimum was again used to determine solution accuracy.

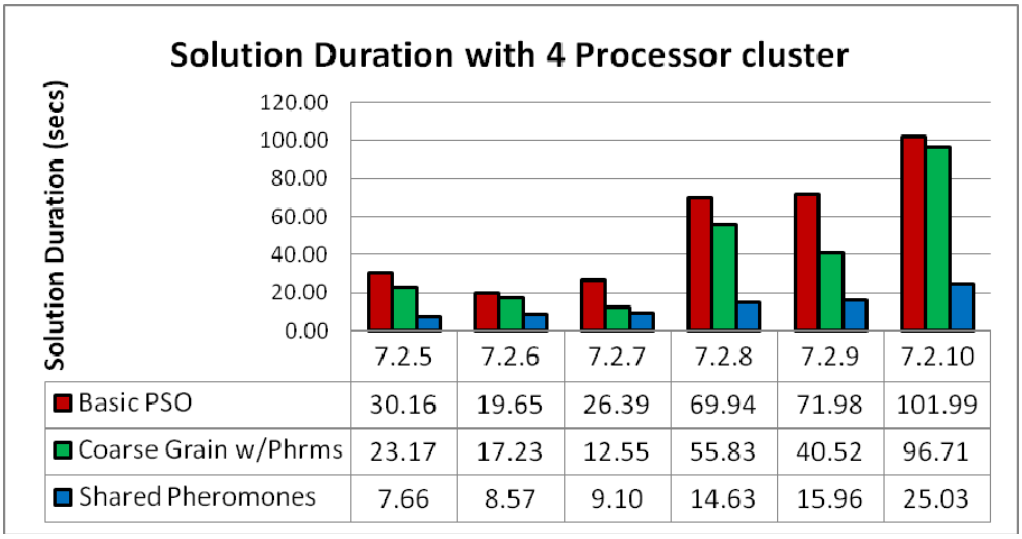
The solution accuracy was calculated based on 20 trial executions for each of: a) coarse grain parallelization on basic PSO, b) coarse grain pheromone PSO parallelization, and c) shared pheromone parallelization, represented by three vertical bar graphs as portrayed in the solution accuracy charts. The numbers 7.2.5 – 7.2.10 below the bar plots indicate the test problem number. As seen from these plots, the solution accuracies reported from shared pheromone parallelization was superior when compared to coarse grained basic PSO and pheromone PSO.

This trend is more evident with an increased number of processors. For example, the solution accuracy was only 75% for the 20 dimensional Ackley's path function (problem 7.2.6) on 2 processors whereas the solution accuracy was 100% with 4 and 8 processors. Similarly, the solution accuracy upon solving Griewank's function (problem 7.2.10) was 65% on 2 processors while it was 95% on 4 and 8 processors.

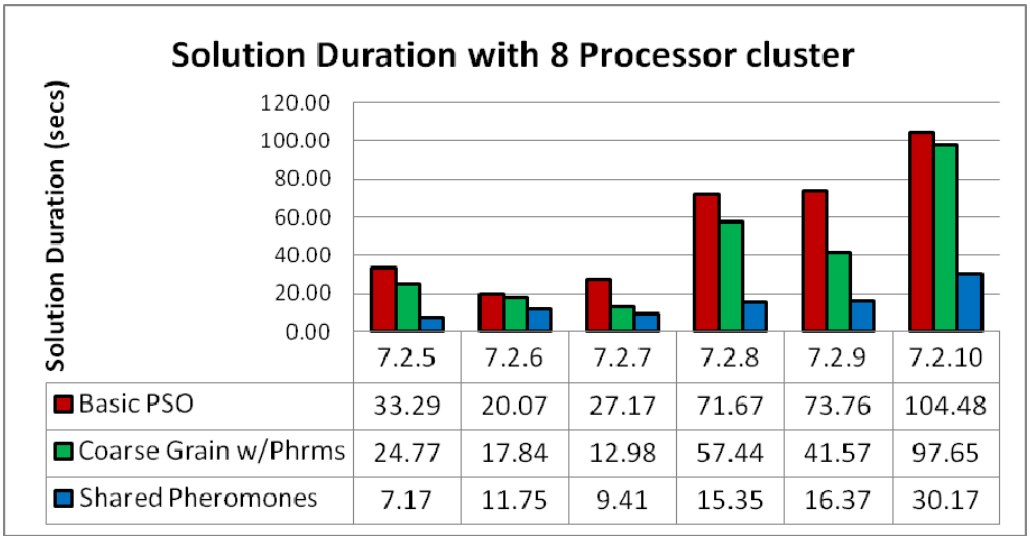
The solution accuracy is only one measure of the method's performance. Figure 37 show that coarse grain parallelization performed close to the shared pheromone method, albeit slightly worse in accuracy. Figure 38 shows the solution duration charts for test problems 7.2.5 – 7.2.10 on 2, 4, and 8 processors.



	7.2.5	7.2.6	7.2.7	7.2.8	7.2.9	7.2.10
Basic PSO	27.45	18.69	25.68	68.07	70.88	100.31
Coarse Grain w/Phrms	20.45	16.68	12.28	53.94	39.80	96.45
Shared Pheromones	7.52	8.29	9.63	12.65	15.43	25.84



	7.2.5	7.2.6	7.2.7	7.2.8	7.2.9	7.2.10
Basic PSO	30.16	19.65	26.39	69.94	71.98	101.99
Coarse Grain w/Phrms	23.17	17.23	12.55	55.83	40.52	96.71
Shared Pheromones	7.66	8.57	9.10	14.63	15.96	25.03



	7.2.5	7.2.6	7.2.7	7.2.8	7.2.9	7.2.10
Basic PSO	33.29	20.07	27.17	71.67	73.76	104.48
Coarse Grain w/Phrms	24.77	17.84	12.98	57.44	41.57	97.65
Shared Pheromones	7.17	11.75	9.41	15.35	16.37	30.17

Figure 38 Solution duration charts for test problems with fixed swarm size per processor

The shared pheromone parallelization showed a dramatic decrease in solution duration when compared to coarse grained basic PSO and coarse grained pheromone PSO on all test problems. As much as a 79% decrease in solution times was observed for the 30 dimensional sum of squares function (problem 7.2.8) on 4 processors when compared to coarse grained basic PSO. When compared to coarse grained pheromone PSO, the solution time decreased by about 74%. On the 20 dimensional Ackley's path function (problem 7.2.6), the solution duration decrease was about 41% when compared to basic PSO and was 34% when compared to coarse grain pheromone PSO on an 8 processor execution.

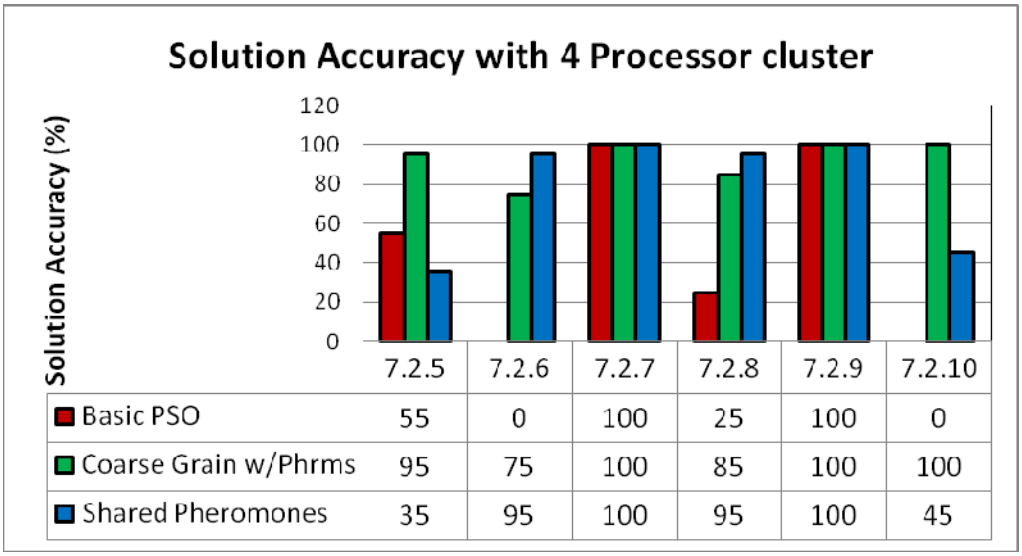
Although the solution times varied from problem type and number of design variables, they indicate that sharing of digital pheromones presents substantial information of the design space to multiple swarms. This resulted in faster solution times, when compared to coarse grained parallelization in basic PSO and pheromone PSO that have no communication between them until the end of a generation. Since the amount of information exchanged between processors was small, the network latency costs were insignificant.

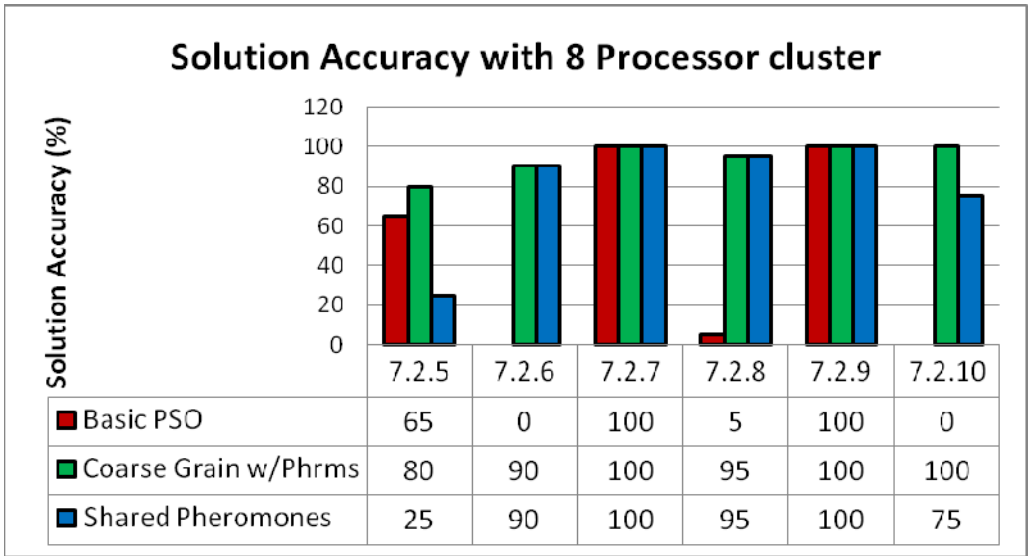
It was noticed that the solution accuracy for the 15 dimensional Dixon and Price function performed quite poorly with shared pheromones on two processors (35% accurate) when compared to 4 processors (80% accurate) and 8 processors (95% accurate). This meant that the method was able to find a solution within the tolerance limits only 35 times out of 100 on two processors. It is hypothesized that the shape of the function could be the reason for the behavior of the method. The function contours take the shape of a trough where the slopes are not steep enough to be noticed by minute changes in objective function while checking

for convergence. A tighter convergence criterion with more precision could improve the solution characteristics. When tested with increased swarm size (i.e., 300) per processor, the method was able to achieve the solution with substantial accuracy. Although the course of the developed method did not find the solution with reasonable solution accuracy, the swarm size could be customized to improve the performance.

7.6.3 Results and Discussion: Fixed Overall Swarm Size

Solution accuracies of coarse grained parallel PSO (basic and pheromones) and shared pheromone parallelization with fixed overall swarm size is shown in Figure 39. Results from two processor execution were eliminated because the swarm distribution with a fixed overall swarm is identical to that of fixed per-processors swarm size.

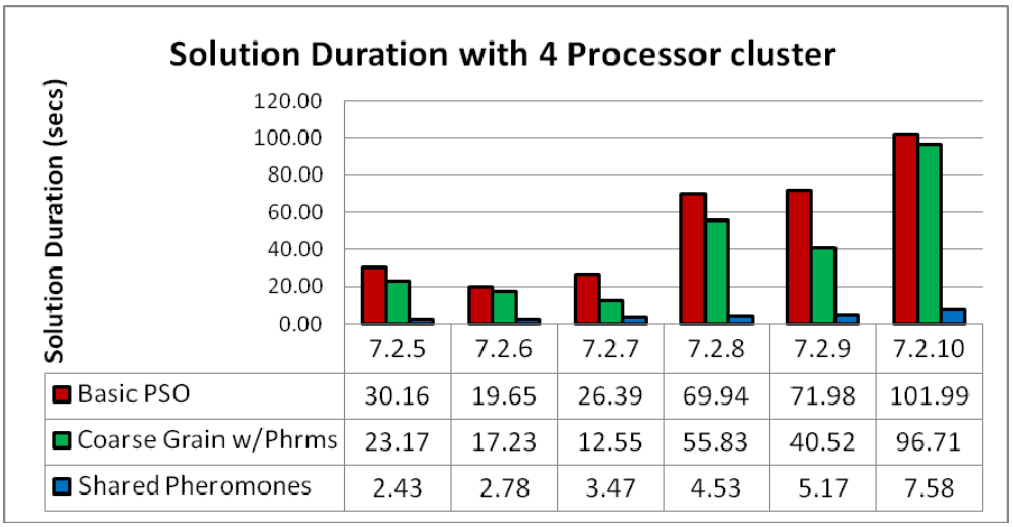




	7.2.5	7.2.6	7.2.7	7.2.8	7.2.9	7.2.10
Basic PSO	65	0	100	5	100	0
Coarse Grain w/Phrms	80	90	100	95	100	100
Shared Pheromones	25	90	100	95	100	75

Figure 39 Solution accuracy charts for test problems with fixed overall swarm size

It was observed that the solution accuracies either stayed the same or decreased with fixed overall swarm size when compared to fixed per-processor swarm size. While the solution accuracy decrease was most (95% to 25% on 8 processor executions) for the 15 dimensional Dixon and Price function (problem 7.2.5), the solution accuracy worsened only by 20% (95% to 75% on 8 processor executions) in the case of the 50 dimensional Griewank's function (problem 7.2.10). The reason for this behavior is theorized that even though communication was made possible between processors on shared pheromone parallelization, the overall swarm size was not sufficient to explore the design space and converge on the optimum. For problems 7.2.6 – 7.2.10, the solution accuracies remained very high (between 95% and 100%) in both fixed overall swarm size and fixed per-processor swarm size schemes.



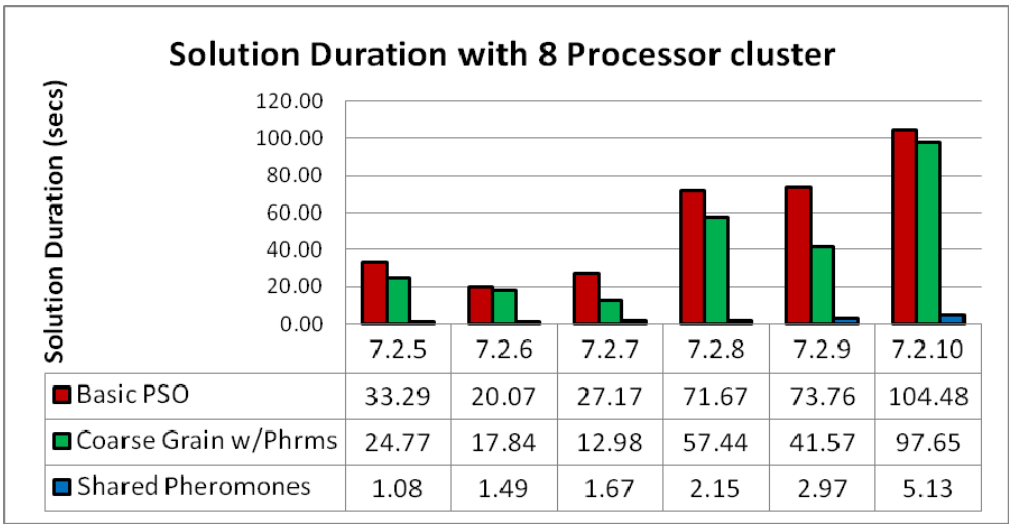


Figure 40 Solution duration charts for test problems with fixed overall swarm size

Figure 40 shows the solution duration characteristics of coarse granular PSO (basic and pheromones) and shared pheromone parallelization with fixed overall swarm size. It can be seen from the figures that there is a dramatic decrease in solution times for fixed overall swarm size (Figure 40) when compared to solution times for fixed per-processor swarm size (Figure 38). For example, the solution duration for a 4 processor execution of the 30 dimensional sum of squares function (problem 7.2.7) was 9.1 seconds for fixed per-processor swarm size but only 3.47 seconds for fixed overall swarm size. This is almost a 62% decrease

in solution time with the solution accuracy remaining at 100% for both the cases. For the 20 dimensional Ackley's path function (problem 7.2.6), the solution duration decrease was about 87% on an 8 processor execution although the solution accuracy decreased by a mere 10% (100% on fixed per-processor swarm size execution to 90% on fixed overall swarm size). Although the solution durations for the 15 dimensional Dixon and Price function (problem 7.2.5) had greater disparity, the comparison is not justified because the fixed overall swarm size solved the problem with 25% solution accuracy.

The difference in solution accuracies and duration between the two schemes of swarm sizes is a direct result of the number of processors. The fixed swarm size per processor scheme had more particles per processor than the fixed overall swarm size. This simply meant that more operations had to be computed thus resulting in increased solution times (as evident in Figure 38). The conclusion to be reached is that the shared pheromone parallelization method consistently found the global solution in considerably less time than other parallel PSO methods. Depending on the problem, a user may want to refine the number of particles per processor to improve the method's performance further.

To eliminate the possibility of any hardware errors in calculating solution durations, all test problems were executed and timed on a different Linux cluster with a comparable computational platform. Although the results varied slightly, the results appeared consistent with the ones presented in figures Figure 39 and Figure 40.

7.6.4 Note on parallel speedups and efficiencies

Since the shared pheromone method requires at least two processors, the time it takes to execute the code in serial (i.e. t_1) cannot be obtained from a single processor execution. Since the total number of operations performed on two processors with shared pheromone parallel method was equivalent to the number of operations on a serial execution of the code, it was this time that was used for parallel speedup and efficiency calculations. With this approach, parallel speed ups of magnitude ranging from 2.4 through 7.7 were observed for solutions with two processor runs using shared pheromones for fixed per-processor swarm size scheme. The parallel efficiencies on two processor execution for the above range were between 120% and 384%. On eight processors, speedups ranging from 9.7 through 26 were observed thereby placing parallel efficiencies within the range of 121% through 327%. These values are extremely high and thus merited further research. It was found that there are actually multiple ways to compute parallel speedups and efficiencies. Due to the nature of this parallelization method (e.g., containing no barrier synchronization) further work must be performed to determine the best manner to quantify the speedup and efficiency calculations. However, obtaining the "correct" number does not detract from the conclusion that this new approach offers significant speedup and efficiency over a serial or coarse grained approach.

7.7 GPU Parallelization Results

In this section, results from implementing PSO with digital pheromones on a GPU are presented. Problems 7.2.5 – 7.2.10 (shown in Table 21) were used as test cases.

Table 21 Test problem matrix for GPU parallelization

Problem	Test Problem	Dimensions
7.2.1	Camelback function	2
7.2.2	Himmelblau function	2
7.2.3	Rosenbrock function	5
7.2.4	Ackley's path function	10
7.2.5	Dixon and Price function	15
7.2.6	Ackley's path function	20
7.2.7	Levy function	25
7.2.8	Sum of Squares function	30
7.2.9	Spherical function	40
7.2.10	Griewank function	50

7.7.1 Test Problem Settings

The pheromone parameters used for the GPU implementation follows the values as established by the serial implementation of PSO with digital pheromones. Therefore, the value of c_3 for lower dimensional problems (2D through 5D) is different from that of higher dimensional problems (above 5D). The values are:

- $c_3 = \begin{cases} 2.0 \text{ for problems } 7.2.1 - 7.2.3, \text{ no decay} \\ 5.0 \text{ for problems } 7.2.4 - 7.2.10, \text{ no decay} \end{cases}$

- Pheromone decay, $\lambda_p = 0.95$, and

- Move limit decay, $\lambda_{ML} = 0.95$

Though customization of parameters for each problem would further improve solution characteristics, the default parameter values catered well for most problems. A total of 35 trial runs were performed for each test case using the GPU method, and were benchmarked against test runs from CPU. Since GPUs, as of the time the research was done, did not support double precision computations, test runs were executed using single precision. Since the serial implementation results discussed in section 7.3 were performed using double

precision and were not a true match to the GPU implementation, all test problems listed in Table 21 were executed both on CPU and GPU on a single workstation with single precision for a fair comparison. Also to emphasize the difference in performance between CPU and GPU, the test runs were performed only on the digital pheromone implementation of PSO. Basic PSO was not implemented.

The CPU used was an Intel Xeon processor (3.2 GHz) of a RedHat Enterprise Linux workstation with 2MB of cache memory. The system memory was 2GB DDR. The GPU used was an NVIDIA Quadro FX 4400 with 512MB of DDR memory. The NVIDIA driver version at the time of the code execution was 169.07. The algorithm was implemented using the C++ programming language, and the GPU implementation was made in GLSL, as described in section 5.2. As a general rule of thumb, the swarm size was defined as 10 times the number of design variables, and was capped at 500 per processor as the dimensionality increased.

7.7.2 Results and Discussion

Table 22 provides a summary of results obtained from solving problems 7.2.1 – 7.2.10. Values obtained from the CPU and GPU are indicated against each problem number in the table. The average, smallest and standard deviation of the objective function values were noted along with averages of solution duration and number of iterations as well.

It can be seen from the table that the objective function values returned by the GPU were extremely close to the values returned by the CPU, in almost all test cases. In seven out of 10 test cases, the average objective function values returned by GPU were equal to or improved when compared to CPU. For example, on the 25 dimensional Levy Function (problem 7.2.7), there is a ~97% improvement in the solution value for GPU (0.004) when compared to the CPU result (0.129). Also, the improvement in the solution value was very consistent on the GPU implementation as apparent from the standard deviation (0.000).

Test problems such as Ackley's path function (7.2.4, 7.2.6) and Levy function (problem 7.2.7) are prone to errors in accuracy due to having trigonometric relations in the objective functions. However, the solution accuracies were not compromised because of this reason. Figure 41 shows a visual comparison of solution accuracies between the CPU and GPU results. With the exception of the 30D Sum of squares function (problem 7.2.8), the solution accuracies on all other problems for the GPU implementation were either equal to or better than that of the CPU implementation. This suggests that GPUs can be capable co-processors for computations and not have a major effect in the outcome of the solution qualities.

Table 22 Results obtained from GPU implementation

CPU/GPU	Solution Accuracy (%)	Objective Function		
		Average	Smallest	Std Dev
7.2.1 (CPU)	100.00%	-1.032	-1.032	0.000
7.2.1 (GPU)	100.00%	-1.032	-1.032	0.000
7.2.2 (CPU)	100.00%	0.000	0.000	0.000
7.2.2 (GPU)	100.00%	0.000	0.000	0.000
7.2.3 (CPU)	100.00%	0.000	0.000	0.000
7.2.3 (GPU)	100.00%	0.000	0.000	0.000
7.2.4 (CPU)	100.00%	0.000	0.000	0.000
7.2.4 (GPU)	100.00%	0.000	0.000	0.000
7.2.5 (CPU)	82.86%	0.261	0.000	0.592
7.2.5 (GPU)	82.86%	0.382	0.000	0.977
7.2.6 (CPU)	91.43%	0.077	0.000	0.262
7.2.6 (GPU)	97.14%	0.053	0.000	0.163
7.2.7 (CPU)	100.00%	0.129	0.129	0.000
7.2.7 (GPU)	100.00%	0.004	0.004	0.000
7.2.8 (CPU)	82.86%	0.286	0.001	0.329
7.2.8 (GPU)	71.43%	0.298	0.003	0.321
7.2.9 (CPU)	100.00%	0.000	0.000	0.000
7.2.9 (GPU)	100.00%	0.000	0.000	0.000
7.2.10 (CPU)	100.00%	0.005	0.000	0.006
7.2.10 (GPU)	100.00%	0.008	0.000	0.008

Legend: 7.2.1 – Camelback 2D, 7.2.2 – Himmelblau 2D, 7.2.3 – Rosenbrock 5D, 7.2.4 – Ackley 10D, 7.2.5 – Dixon and Price function15D, 7.2.6 – Ackley's path function 20D, 7.2.7 – Levy function 25D, 7.2.8 – Sum of squares function 30D, 7.2.9 – Spherical function 40D, 7.2.10 – Griewank function 50D. (CPU) – Results from CPU implementation, (GPU) – Results from GPU implementation.

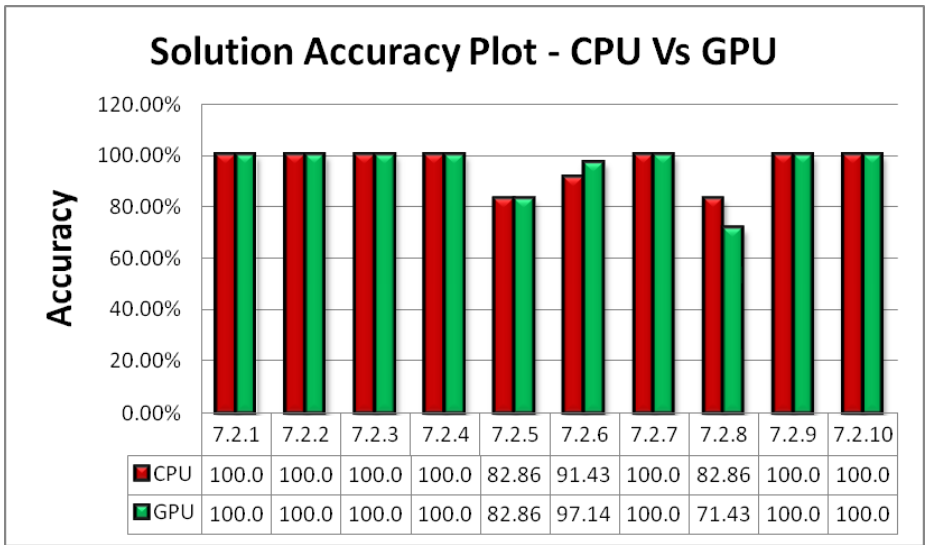


Figure 41 Solution accuracy plot for CPU and GPU implementation of PSO with digital pheromones

Figure 42 shows the solution duration charts for the GPU implementation as compared with the CPU implementation.

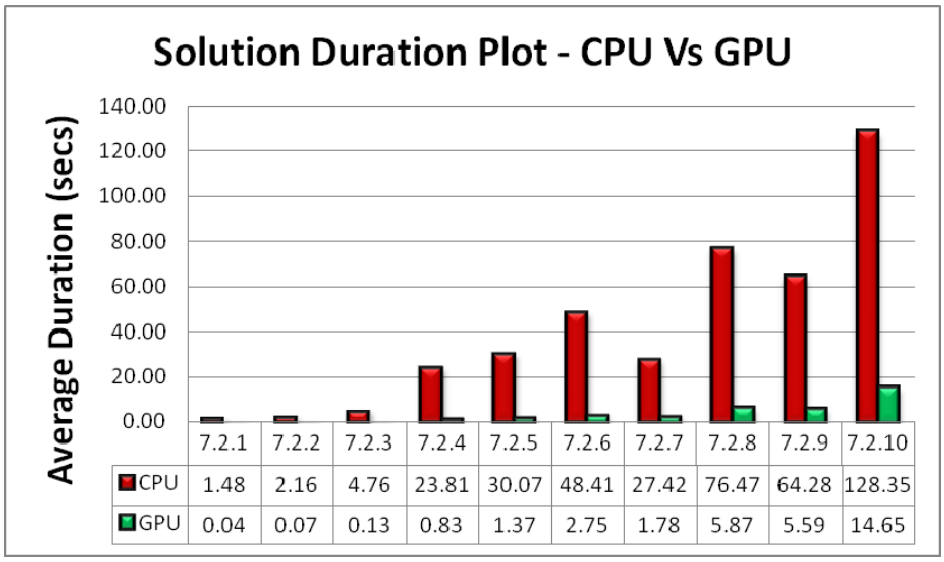


Figure 42 Solution Duration plot for CPU and GPU implementation of PSO with digital pheromones

It can be seen that the solution duration for all test problems dramatically reduced on the GPU implementation as opposed to the CPU counterpart. The reduction in the solution

duration is a clear indication that GPUs are not just comparable in performance to traditional CPUs, but they could exceed the throughput by a factor of about 10 or more in terms of solution times. For example, the average solution time for the 10 dimensional Ackley's path function (problem 7.2.4) on a CPU was 23.81 seconds where as the GPU took 0.83 seconds. This is approximately a 96.5% decrease. Similarly, the 50 dimensional Griewank function (problem 7.2.10) resulted in an 88% decrease in solution time on a GPU (14.65 seconds) when compared to the solution time on a CPU (128.35 seconds). The same trend was seen in all the test problems, with reduction in solution durations ranging from 88% - 97% were observed. Table 23 tabulates the average duration, average number of iterations and the percentage decrease in solution duration through using GPUs when compared to CPU usage alone. The data parallel architecture of a GPU is attributed to this dramatic reduction in solution times. Since GPUs are inherently hardware parallel in architecture, a single instruction can be performed on multiple data simultaneously resulting in enormous time savings as evident from figures Figure 41 and Figure 42 and tables Table 22 and Table 23. Although the amount of time savings can be hardware and problem specific, the results show that a GPUs has great potential to outperform CPUs with no marked difference in solution quality for optimization computations. There was also no notable difference in the number of iterations for each test problem when executed on a CPU or a GPU. This indicates that the data traversal between CPU and GPU did not significantly affect the overall algorithm's performance. This suggests that commodity graphics cards can potentially be a very viable option in optimization computations when time and cost are important factors.

Table 23 Comparison of solution duration and number of iterations on CPU Vs GPU

CPU/GPU	Average Duration (secs per run)	Average # iterations	% Decrease in duration
7.2.1 (CPU)	1.48	73.97	97.03%
7.2.1 (GPU)	0.04	73.86	
7.2.2 (CPU)	2.16	107.71	96.63%
7.2.2 (GPU)	0.07	115.49	
7.2.3 (CPU)	4.76	93.54	97.19%
7.2.3 (GPU)	0.13	94.89	
7.2.4 (CPU)	23.81	231.11	96.50%
7.2.4 (GPU)	0.83	234.11	
7.2.5 (CPU)	30.07	192.11	95.46%
7.2.5 (GPU)	1.37	189.40	
7.2.6 (CPU)	48.41	229.69	94.31%
7.2.6 (GPU)	2.75	235.77	
7.2.7 (CPU)	27.42	103.03	93.49%
7.2.7 (GPU)	1.78	103.80	
7.2.8 (CPU)	76.47	235.89	92.33%
7.2.8 (GPU)	5.87	235.00	
7.2.9 (CPU)	64.28	146.57	91.30%
7.2.9 (GPU)	5.59	140.29	
7.2.10 (CPU)	128.35	228.94	88.59%
7.2.10 (GPU)	14.65	231.46	

Legend: 7.2.1 – Camelback 2D, 7.2.2 – Himmelblau 2D, 7.2.3 – Rosenbrock 5D, 7.2.4 – Ackley 10D, 7.2.5 – Dixon and Price function15D, 7.2.6 – Ackley's path function 20D, 7.2.7 – Levy function 25D, 7.2.8 – Sum of squares function 30D, 7.2.9 – Spherical function 40D, 7.2.10 – Griewank function 50D. (CPU) – Results from CPU implementation, (GPU) – Results from GPU implementation.

7.8 Constrained Problems

In this section, results from solving problems 7.2.11 – 7.2.16 (shown in Table 24) is presented.

Table 24 Test problem matrix for constrained problem solving

Problem	Test Problem	Dimensions	# of Constraints
7.2.11	One Dimensional Two Inequality	1	2
7.2.12	Two Dimensional Single Inequality	2	1
7.2.13	Two Dimensional Two Inequality	2	2
7.2.14	Weld Beam	4	8
7.2.15	Golinski's Speed Reducer Problem	7	11
7.2.16	Himmelblau constrained	5	7

7.8.1 Test Problem Settings

The following PSO and digital pheromone parameters were used to run all constrained problems:

- $c_1=c_2=2.0$
- $c_3=2.0$ with no decay
- Pheromone decay = 0.95
- Move limit decay = 0.95
- Inertia weight initialized at 1.0 and decreased at 0.5% every iteration
- Particle swarm size of 10 times the number of design variables

Twenty test runs were performed on each problem using two approaches: a) pseudo objective function solved completely before updating Lagrange multipliers and b) pseudo iterations limited to five before updating Lagrange multipliers. Although the number of pseudo iterations might seem arbitrary, literature [32] shows that three to five pseudo iterations generally worked well in other ALM implemented methods. The algorithm was

made to report its current results at the end of 10,000 iterations if a feasible solution is not found. The constraint feasibility and solution tolerance were set to 0.01. The test problems were considered converged when the difference in solutions was within 0.01 for 10 consecutive iterations. In the case of problems where each pseudo objective function is completely solved, the convergence tolerance was set to 0.01 for three pseudo iterations. All Lagrange multipliers were initialized to zero, and the penalty values were initialized to one. Although ALM does not require any restriction on the penalty parameter values, an upper limit of 100,000 was imposed to avoid any numerical ill-conditioning resulting from penalizing infeasible solutions.

7.8.2 Results and Discussion

The results from solving constrained problems 7.2.11 – 7.2.16 are tabulated in Table 25 with the test problem descriptions given in section 7.2. The column entries with '*' indicates that none of the solutions in 20 trial runs were feasible. However, the lowest, average and highest values are reported to demonstrate the difference from published solutions. For column entries that are not marked with '*', the lowest and highest values are reported only for feasible solutions. This is done to indicate the range of solutions obtained when they are feasible. Averages are reported on all test problems regardless of whether they are feasible or infeasible.

Table 25 Solutions from complete solving of pseudo objective functions

Prob.	Published solution	Solution Obtained (Feasible solutions reported in Lowest & Highest columns)			Constraint Satisfaction consistency	# of constraints
		Lowest	Average	Highest		
7.2.11	0.45	0.45	0.45	0.45	20 of 20 runs	2
7.2.12*	-50.00	-50.48	-50.26	-50.05	0 of 20 runs	1
7.2.13	-9.24	-9.23	-9.36	-9.07	5 of 20 runs	8
7.2.14	2.39	1.79	2.19	2.92	17 of 20 runs	7
7.2.15*	2985.22	2634.16	2658.07	2731.65	0 of 20 runs	11
7.2.16*	-31025.57	-32217.43	-32217.43	-32217.43	0 of 20 runs	6

* indicates no feasible solutions were found in each of 20 trial runs

It can be seen from the table that the method was able to solve three out of six problems. The method found the solution in all 20 trial runs in the case of problem 7.2.11 while 17 of 20 trial runs solved the problem 7.2.14. It can also be seen that the lowest value attained for problem 7.2.14 (four dimensional weld beam problem) is lower (1.79) than the published solution (2.39). The reason is attributed to the information provided by digital pheromones to particle swarms in searching the design space. Although problem 7.2.13 was solved only five out of 20 runs, the lowest and highest feasible solution values are within a close neighborhood of the published solutions.

The algorithm was unable to solve problems 7.2.12, 7.2.15, and 7.2.16 within the tolerance limits specified in any of the 20 trial runs. It was observed that problem 7.2.15 (Golinski's speed reducer problem violated the most number of constraints when compared to other problems (~ 4 out of 11 constraints were violated in all 20 trial runs). However, it was observed that the average violations were between zero and one indicating that the swarm was very close to the feasible region. Similar behavior was observed in other test cases that failed to be feasible in all 20 trial runs. For example,

problem 7.2.12 having one inequality constraint violated only by 0.28 on average of all 20 trial runs. However, problem 7.2.16 was an exception where all except one of the six constraints hovered at 2.3.

The problems were also solved with a limited number of pseudo function iterations. These results are tabulated in Table 26. It can be seen from the table that four of six problems have feasible solutions of the 20 trial runs. The method found the solution in all 20 trial runs in the case of problems 7.2.11 and 7.2.14 (four dimensional weld beam problem with eight constraints).

Table 26 Solutions from limited pseudo iterations

Prob.	Published solution	Solution Obtained (Feasible solutions reported in Lowest & Highest columns)			Constraint Satisfaction consistency	# of constraints
		Lowest	Average	Highest		
7.2.11	0.45	0.45	0.45	0.45	20 of 20 runs	2
7.2.12	-50.00	-50.64	-50.24	-49.96	3 of 20 runs	1
7.2.13	-9.24	-9.89	-9.29	-7.37	5 of 20 runs	8
7.2.14	2.39	1.87	2.39	3.09	20 of 20 runs	7
7.2.15*	2985.22	2714.23	2828.83	2957.30	0 of 20 runs	11
7.2.16*	-31025.57	-32217.43	-32217.43	-32217.43	0 of 20 runs	6

* indicates no feasible solutions were found in each of 20 trial runs

It is worth noting that the solutions resulting from both these problems are equal to or better than the published solutions. For example, the weld beam problem resulted in the lowest value of 1.87, which is about 22% better than the published solution. The average solution value obtained over 20 trial runs was 2.39, which is equal to the published solution. For problem 7.2.11, the solution obtained was exactly equal to the published solution. A similar behavior is observed in problem 7.2.12 as well, where the lowest

feasible solution (-50.64) fared better than the published solution (-50.00). Although the difference is not very significant and only three of 20 trial runs were feasible, suggesting that the particle swarm gathered significant information about the design space and digital pheromones assisted this behavior. The average solution values of infeasible and feasible solutions returned -50.24, which is yet better than the published solution.

For problem 7.2.13, five trial runs of 20 resulted in a feasible solution, of which the lowest feasible solution was 7% better than the published solution. There are two potential reasons for 15 of the 20 being infeasible: a) the swarm is trapped in a local minimum resulting in a solution value better than the published solution but is in an infeasible space or b) the swarm is very close to the optimum but could not find the precise design variable values necessary to satisfy all constraints. To clarify this issue, the design variables that returned an infeasible solution were observed. For example, one of the 20 trial runs returned design variable values of {1.738, 1.978}, resulting in one of the constraint being violated by 0.105 and the second constraint active at -0.002. The published solution set for this problem is {1.746, 1.953}, which is very close to the obtained solution set. A similar behavior was observed for the rest of the trial runs as well suggesting that the swarm was trapped in a location very close to the optimal point, but with a small degree of infeasibility. The main reason for this is thought to be the formulation of the pseudo objective function. Since this method is not handling constraints directly, there is a mapping that has to occur from the actual constrained design space to the unconstrained pseudo design space. A pseudo design space must be different each time if a different, and improved, design point is to be found. Most likely,

for the test cases that could not converge, the penalty being applied was too small to influence the pseudo design space to meet all constraints. In other words, the pseudo representation reached a certain point and could not change further.

None of the trial runs returned a feasible solution for problems 7.2.15 (Golinski's speed reducer problem) and 7.2.16 (constrained Himmelblau problem). A feasible solution was not obtained even when the pseudo objective functions were completely solved, as noted in section 7.8.2. This suggests that these problems are very sensitive to the behavior of the swarm movement within the design space, and a different constraint handling approach might be required to address this issue.

Overall, the ALM implementation of PSO with digital pheromones produced promising results. Out of a total of 240 trial runs for all test cases (both complete pseudo solving and limited pseudo iterations), 90 trial runs resulted in feasible solutions. This can be observed from the 'constraint satisfaction consistency' columns in Table 25 and Table 26. Problems that resulted in feasible solutions were solved quite fast and within significantly less number of iterations. For example, the weld beam problem (7.2.14) took ~2470 seconds and 10,000 iterations to result in a solution of 1.907 that violated one constraint, while it took only 4.25 seconds and 15 iterations before converging to a feasible solution of 1.913.

Since only about 37% of the test cases resulted in feasible solutions, further research has to be done to improve the reliability of the method. Two fundamental issues that must be

dealt with are: a) continuous formulation of the pseudo objective function to ensure improvement until feasibility and convergence and b) an intelligent distribution of the penalty across the swarm so that members exploring “bad” regions of the design space do not exert undue influence on the remainder of the swarm. However, the method did perform well in accuracy as it found better solutions than currently published for some of the test cases. This alone is a significant contribution to the field for an emerging constrained optimization method.

8 Conclusions and Future Work

8.1 Conclusions

This research presents a novel use of digital pheromones to search n-dimensional multimodal design spaces in PSO. A basic PSO is known for its simplicity in implementation because of a small number of parameters to alter. The use of digital pheromones within PSO introduces three new parameters namely the confidence parameter (c_3), move limit decay (λ_{ML}) and pheromone decay (λ_P). Although these are user defined, default values have been empirically established through testing with 128 different combinations of pheromone parameters with three different test cases. These values were used as inputs for the remaining test cases to test the feasibility of digital pheromones to aid a particle swarms to search for the global optimum in n-dimensional design spaces. It was observed that the objective function values resulting from using digital pheromones were nearly always equal to published, if not better, ones for the test cases used. Although additional computations were added per iteration, solutions times still decreased when digital pheromones were implemented due to faster convergence. The viability of solving realistic multimodal optimization problems was simulated through imposing sleep-times on objective function evaluations. When a basic PSO was unable to solve a problem, the additional information about the design space through digital pheromones caused a faster convergence on the global minimum with increased accuracy, efficiency and reliability.

Statistical tests at a 95% confidence level were performed to test the significance of the results obtained when compared to a basic PSO. In all of the test cases the objective function values from digital pheromone implementation was significantly better than a basic PSO. With very few new pheromone parameters added to a basic PSO, the solution accuracy, efficiency and reliability characteristics of PSO substantially increased thereby improving the usability of PSO in practical design processes in an industry.

Additionally, the developed method was implemented in parallel to determine its feasibility in a cluster computing environment. Six different problems were tested in synchronous coarse grain parallelization and asynchronous shared pheromone parallelization schemes. Although the solution accuracies of coarse grain parallelization did not differ much from serial implementation, it was demonstrated that substantial savings were achieved in terms of solution times. The parallel efficiency and speedup studies showed that almost ideal parallel speedups were achieved in spite of network latencies. The parallel efficiencies and speedups improved as the dimensionality and number of processors increased.

In the shared pheromone parallelization method, multiple swarms deployed across available processors share a common repository of digital pheromones. These served as information communication sources for particle swarms resulting in substantial improvement in solution accuracy, efficiency and reliability. Additionally, these gains improved as the number of processors increased suggesting the method's scalability to a large number of processors. For a fair performance comparison, two modes of shared

pheromone parallelization were introduced – fixed swarm size per processor, and fixed overall swarm size. Both implementation schemes performed significantly better than coarse grain parallelization demonstrating that communication between swarm members is essential for improved solution efficiency.

The GPU implementation of PSO with digital pheromones was another important accomplishment of this research, where solution speedups of up to ~97% were realized compared to CPU computing with comparable solution qualities. The objective function values were computed using GPUs although the percentage of GPU involvement could be altered using configuration files. This is especially helpful when high precisions combined with computational efficiencies are primary requirements for a designer. GPUs are traditionally used for visualization purposes and typically not used for solving complex design optimization problems. Adoption of GPUs as a means to substantially improve solution efficiencies in highly multimodal design problems serves as a noteworthy contribution to the field of Human-Computer Interaction in this research.

On GPUs, objective function evaluations are currently computed on a Single Instruction Multiple Data scheme that makes large number of computations possible simultaneously – an inherent hardware property of GPUs that is different from a traditional CPU. Although the GPU implementation merits further research for realizing more performance benefits, the implemented method serves as a proof of concept that GPUs are a cost effective and faster means to perform scientific computations. This implementation is further a testimony for realizing enormous solution efficiencies

through parallelization with little (~$100 USD) or no changes in hardware infrastructure in an industry.

In addition, the developed method was tested for solving constrained optimization problems. The Augmented Lagrange Multiplier method was used to accomplish this task. Significant solution accuracies were observed for those problems that the method was able to solve. In some cases, the solutions exceeded published solution values. However, further investigation is warranted to improve the reliability of the method.

8.2 Future Work

There are many future directions for further implementation of digital pheromones in PSO. While refining the performance of digital pheromones to solve a wide range of problems (e.g., multi-objective, discrete optimization problems, etc) is an ongoing venture, the following are some of the future goals that are worth investigating and implementing:

1) Develop a graphical user interface to specify various problem parameters during run-time and visualization of solution progress in the design space using various n-dimensional visualization techniques.
2) Improve the reliability characteristics for solving constrained optimization problems.

3) Develop methods to off-load more independent computations on to GPUs, while also reduce data traffic between CPUs and GPUs.
4) Currently, shared pheromone operations in parallel are performed by only one processor (the root processor). It would be beneficial to mathematically, analytically, and statistically determine the appropriate number of pheromone processors for efficient utilization of computational resources as problem characteristics change.
5) Possible elimination of *pBest* and usage of *gBest* alone with digital pheromones.

9 References

[1] Vanderplaats, G., "Numerical Optimization Techniques for Engineering Design", 3rd Edition, VR&D Publications, ISBN: 094-495-6009, 1999

[2] Rao, S. S., "Engineering Optimization – Theory and Practice", 3rd Edition, New Age Publications, ISBN: 047-155-0345, 1996

[3] Gill, P., Murray, W., Wright, M. H., "Practical Optimization", Academic Press, ISBN: 012-283-9501, 1981

[4] Holland, J., H., "Adaptations in Natural and Artificial Systems", University of Michigan Press, Ann Arbor, Michigan, 1975

[5] Rechenberg, I., "Cybernetic Solution Path of an Experimental Problem", Library Translation 1122, Royal Aircraft Establishment, Farnborough, Hampshire, England, 1965

[6] Srinivas, M., Patnaik, L., "Genetic Algorithms: a Survey", IEEE Computer, Volume 27, Issue 6, ISSN:0018-9162, pp. 17-26, June 1994

[7] Haupt, R., Haupt, S., *Practical Genetic Algorithms*, Wiley Publications, ISBN: 0471455652, 2004

[8] Sivanandam, S. N., Deepa S. N., "Introduction to Genetic Algorithms", Springer-Verlag Publications, ISBN 978-354-073-1894, 2008

[9] Poon, P. W., Carter, J. N., "Genetic Algorithm Crossover Operators for Ordering Applications", Computers Operations Research, Elsevier Science Ltd., Vol. 22, No. 1, pp. 135-147, 1995

[10] Goldberg, D. E., Lingle, R. L., "Alleles, loci and the traveling salesman problem", Proceedings, First International Conference on Genetic Algorithms and their Applications, pp. 154 – 159, Erlbaum, 1985.

[11] Oliver, I. M., Smith, D. J., Holland, J. R. C., "A Study of Permutation Crossover Operators on the Traveling Salesman Problem", Proceedings of Second International Conference on Genetic Algorithms and their Applications, p. 224 – 230, Erlbaum, 1987

[12] Syswerda, G., "Schedule Optimization using Genetic Algorithms", Handbook of Genetic Algorithms (Edited by L. Davis), pp. 332 – 349, Van Nostrand Reinhold, Amsterdam, 1991

[13] Kirkpatrick, S., Gelatt, C., Vecchi, M., "Optimization by Simulated Annealing", Science, Vol. 220, Number 4598, pp. 671-680, 1983

[14] Cerny, V., "A Thermodynamical Approach to the Travelling Salesman Problem: an Efficient Simulation Algorithm", Journal of Optimization Theory and Applications, Vol. 45, pp. 41-51, 1985

[15] Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A., Teller, E., "Equations of State Calculations by Fast Computing Machines", Journal of Chemical Physics, 21(6): 1087 – 1092, 1953

[16] 'Simulated annealing – Wikipedia, the free encyclopedia', web reference: http://en.wikipedia.org/wiki/Simulated_annealing, retrieved, April 10, 2008

[17] Rao, S., "Engineering Optimization – Theory and Practice", 3rd Edition, Wiley-Interscience publications, ISBN: 047-155-0345, 1996

[18] Eberhart RC., Kennedy J., "A New Optimizer using Particle Swarm Theory", Proceedings of the sixth international symposium on micro machine and human science. Institute of Electrical and Electronics Engineers, Piscataway, NJ, pp. 39-43, 1995

[19] Kennedy, J., Eberhart, RC., "Particle Swarm Optimization", Proceedings of the 1995 IEEE international conference on neural networks, vol. 4, Institute of Electrical and Electronics Engineers, Piscataway, NJ, pp. 1942 – 1948, 1995

[20] Heppner, F., Grenander, U., "A Stochastic Non-linear Model for Coordinated Bird Flocks", In S., Krasner, Ed., The Ubiquity of Chaos, AAAS Publications, Washington DC, ISBN: 087-168-3504, 1990

[21] Hu, X., Eberhart, R., Shi Y., "Engineering Optimization with Particle Swarm", IEEE Swarm Intelligence Symposium, 2003, pp 53-57

[22] Shi, Y., Eberhart, R., "Parameter Selection in Particle Swarm Optimization", Proceedings of the 1998 Annual Conference on Evolutionary Computation, March 1998

[23] Shi, Y., Eberhart, R., "A Modified Particle Swarm Optimizer", Proceedings of the 1998 IEEE International Conference on Evolutionary Computation, pp 69-73, Piscataway, NJ, IEEE Press May 1998

[24] Eberhart R. C., Shi Y., "Particle Swarm Optimization: Developments, Applications, and Resources", In proceedings of the 2001 congress on evolutionary computation, pp. 81-86, 2001

[25] Kennedy J., Eberhart R. C. "Swarm Intelligence", Morgan Kauffman publishers, San Mateo, CA, 2001

[26] Natsuki, H., Iba, H., "Particle Swarm Optimization with Gaussian Mutation", Proceedings of IEEE Swarm Intelligence Symposium, Indianapolis, pp. 72-79, 2003

[27] Hu, X., Eberhart, R., Shi, Y., "Swarm Intelligence for Permutation Optimization: A Case Study of n-Queens Problem", IEEE Swarm Intelligence Symposium, Indianapolis, IN, 2003

[28] Gao, F., Liu, H., Zhao, Q., Cui, G., "Virus-Evolutionary Particle Swarm Optimization Algorithm", Lecture Notes in Computer Science, Vol. 4222/2006, Springer Berlin/Heidelberg Publications, pp. 156-165, September 2006

[29] Ray, T., Saini, P., "Engineering Design Optimization Using a Swarm with an Intelligent Information Sharing Among Individuals", Engineering Optimization, Vol. 33, pp. 735-748, 2001

[30] Venter, G., Sobieszczanski-Sobieski, J., "Particle Swarm Optimization", *AIAA Journal*, Vol.41, No.8, pp 1583-1589, 2003

[31] Hu, X., Eberhart, R., "Solving Constrained Nonlinear Optimization Problems with Particle Swarm Optimization", 6th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2002), Orlando, USA, 2002

[32] Sedlaczek, K., Eberhard, P., "Using Augmented Lagrangian Particle Swarm Optimization for Constrained Problems in Engineering", Structural Multidisciplinary Optimization Journal, Vol. 32, pp. 277-286, 2006

[33] Yang, Q., Sun, J., Zhang, J., Wang, C., "A Hybrid Particle Swarm Optimization for Binary CSPs", Lecture Notes in Computer Science, Springer Berlin/Heidelberg Publications, Vol. 4115/2006, pp. 39-49, September 2006

[34] Hu, X., Eberhart, R., Shi, Y., "Particle Swarm with Extended Memory for Multiobjective Optimization", Proceedings of 2003 IEEE Swarm Intelligence Symposium, pp 193-197, Indianapolis, IN, USA, IEEE Service Center, April 2003

[35] Coello C., Lechuga M., "A proposal for multiple objective particle swarm optimization", Technical report EVOCINV-01-2001, Evolutionary Computation Group at CINVESTAV-IPN, Mexico, 2001

[36] Clerc, M., "Discrete Particle Swarm Optimization", New Optimization Techniques in Engineering, Springer, Berlin Heidelberg, NY, 2004

[37] Wang, K, Huang, L., Zhou, C., Pang, W., "Particle Swarm Optimization for Traveling Salesman Problem", Proceedings of the Second International Conference on Machine Learning and Cybernetics, November 2003

[38] Li, X., Tian, P., Hua, J., Zhong, N., "A Hybrid Discrete Particle Swarm Optimization for the Traveling Salesman Problem", Lecture Notes in Computer Science, Vol. 4247/2006, Springer Berlin/Heidelberg Publications, pp. 181-188, October 2006

[39] Shen, B., Yao, M., Yi, W., "Heuristic Information Based Improved Fuzzy Discrete PSO Method for Solving TSP", Lecture Notes in Computer Science, Springer Berlin/Heidelberg Publications, Vol. 4099/2006, pp. 859-863, July 2006

[40] Kitayama, S., Arakawa, M., Yamazaki, K., "Penalty Function Approach for the Mixed Discrete Non-Linear Problems by Particle Swarm Optimization", Structural and Multidisciplinary Optimization, Vol. 32, No. 3, pp. 191-202, 2005

[41] Liu, J., Sun, J., Xu, W., "Quantum-Behaved Particle Swarm Optimization for Integer Programming", Lecture Notes in Computer Science, Springer Berlin/Heidelberg Publications, Vol. 4233/2006, pp. 1042-1050, October 2006

[42] Tayal, M., Wang, B., "Particle Swarm Optimization for Mixed Discrete, Integer and Continuous Variables", 10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, Albany, New York, Aug 30-1, 2004

[43] Parsopoulos, K. E., Vrahatis, M. N., "Recent Approaches to Global Optimization Problems Through Particle Swarm Optimization", Natural Computing, Vol. 1, pp. 235-306, 2002

[44] Parsopoulos, K. E., Vrahatis, M. N., "On the Computation of All Global Minimizers Through Particle Swarm Optimization", IEEE Transactions on Evolutionary Computation, Vol. 8, No. 3, pp. 211-224, 2004

[45] He. S., Prempain, E., Wu, Q. H., "An Improved Particle Swarm Optimizer for Mechanical Design Optimization Problems", Engineering Optimization, Vol. 36, No. 5, pp. 585-605, October 2004

[46] G. Venter and J. Sobieszczanski-Sobieski, "Multidisciplinary optimization of a transport aircraft wing using particle swarm Optimization", In 9th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization 2002, Atlanta, GA

[47] Pidaparti, R., Jayanti, S., "Corrosion Fatigue Through Particle Swarm Optimization", AIAA Journal, Vol. 41, No. 6, June 2003

[48] P.C. Fourie and A.A. Groenwold, "The particle swarm algorithm in topology optimization", In Proceedings of the Fourth World Congress of Structural and Multidisciplinary Optimization 2001, Dalian, China

[49] Fourie, P. C., Groenwold, A. A., "The Particle Swarm Optimization Algorithm in Size and Shape Optimization", Structural Multidiscplinary Optimization, Vol. 23-4, pp. 259-267, 2002

[50] Bochenk, P., Fory's P., "Structural Optimization for Post-Buckling Behavior Using Particle Swarm", Structural and Multidisciplinary Optimization, Vol. 32, No. 6, pp. 521-530, 2006

[51] Schutte, J., Groenwold, A., "Sizing Design of Truss Structures Using the Particle Swarms", Structural and Multidisciplinary Optimization, Vol. 25, pp. 261-269, 2003

[52] Yang, S., Huang, R., Shi, H., "Mobile Agent Routing Based on a Two-Stage Optimization Model and a Hybrid Evolutionary Algorithm in Wireless Sensor Networks", Lecture Notes in Computer Science, Springer Berlin/Heidelberg Publications, Vol. 4222/2006, pp. 938-947, September 2006

[53] Onwubolu, G., Clerc, M., "Optimal Path for Automated Drilling Operations by a New Heuristic Approach using Particle Swarm Optimization", International Journal of Production Research, Vol. 42, No. 3, pp 473-491, 2004

[54] Rameshkumar, K., Suresh, R., Mohanasundaram, K., "Discrete Particle Swarm Optimization (DPSO) Algorithm for Permutation Flowshop Scheduling to Minimize Makespan", Lecture Notes in Computer Science, Vol. 3612/2005, pp. 572-581, July 2005

[55] Tianzhu, W., Wenhui, L., Yi, W., Zihou, G., Dongfeng, H., "An Adaptive Stochastic Collision Detection Between Deformable Objects Using Particle Swarm Optimization", Lecture Notes in Computer Science, Springer Berlin/Heidelberg, Vol. 3907/2006, pp. 450-459, March 2006

[56] Batkiewicz, T., Dohse, K., Kalivarapu, V., Dohse, T., Walter, B., Knutzon, J., Parkhurst, D., Winer, E., Oliver, J., "Multimodal UAV Ground Control System", 11th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, AIAA 2006-6963, 6 – 8 September 2006, Portsmouth, VA

[57] Foo, J., Knutzon, J., Oliver, J., Winer, E., "Three-Dimensional Path Planning of Unmanned Aerial Vehicles Using Particle Swarm Optimization", 11th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, AIAA 2006-6995, 6 – 8 September 2006, Portsmouth, VA

[58] Walter, B., Sannier, A., Reiners, D., Oliver, J., "UAV Swarm Control: Calculating Digital Pheromone Fields with the GPU", The Interservice/Industry Training, Simulation & Education Conference (I/ITSEC), Volume 2005 (Conference Theme: One Team. One Fight. One Training Future), 2005

[59] Gaudiano, P, Shargel, B., Bonabeau, E., Clough, B., "Swarm Intelligence: a New C2 Paradigm with an Application to Control of Swarms of UAVs", In Proceedings of the 8th International Command and Control Research and Technology Symposium, 2003

[60] Colorni, A., Dorigo, M., Maniezzo, V., "Distributed Optimization by Ant Colonies", *In Proc. Europ. Conf. Artificial Life,* Editors: F. Varela and P. Bourgine, Elsevier, Amsterdam, 1991

[61] Dorigo, M., Maniezzo, Colorni, A., "Ant System: Optimization by a Colony of Cooperating Agents", *In IEEE Trans. Systems, Man and Cybernetics*, Part B, Vol. 26, Issue 1, pp 29-41, 1996

[62] Montgomery, J., "Towards a Systematic Problem Classification Scheme for Ant Colony Optimization", *Technical Report tr02-15*, School of Information Technology, Bond University, Australia, 2002

[63] White, T., Pagurek, B., "Towards Multi-Swarm Problem Solving in Networks", icmas, pp. 333, Third International Conference on Multi Agent Systems (ICMAS'98), 1998

[64] Parunak, H., Purcell M., O'Conell, R., "Digital Pheromones for Autonomous Coordination of Swarming UAVs", In Proceedings of First AIAA Unmanned Aerospace Vehicles, Systems, Technologies, and Operations Conference, Norfolk, VA, AIAA, 2002

[65] Gropp, W., Lusk, E., Skjellum, A., "Using MPI – Portable Parallel Programming with the Message Passing Interface", 2nd Edition, MIT Press, ISBN: 0-262-571-323, 1999

[66] Itzigehl, P., R., "A Method for Asynchronous Parallelization", International Conference on Software Engineering, Proceedings of the 10th International Conference on Software Engineering, Singapore, pp.4-9, ISBN: 0-89791-258-6, 1988

[67] Chalermwat, P., El-Ghazawi, T., LeMoegne, J., "2-phase GA-based Image Registration on Parallel Clusters", Future Generation Computer System 17, 2001

[68] Belal, M., Ghazawi, T., "Parallel Models for Particle Swarm Optimizers", International Journal on Intelligent Cooperative Information Systems 4, pp. 100-111, 2004

[69] Aguire, H., Tanaka, K., SUgimara, T., Oshita, S., "Improved Distributed Genetic Algorithm with Cooperative-Competitive Genetic Operators", IEEE 2000

[70] Vladimir, L., Burgher, J., Tkachuk, A., Gnatjiuk, V., "The Application of the Distributed Genetic Algorithm to the Decision of the Packing in Containers Problem", Proceedings of the 2002 IEEE International Conference on Artificial Intelligence Systems, 2002

[71] Schutte, J., Reinbolt, J., Fregly, B., Haftka, R., George, A., "Parallel Global Optimization with the Particle Swarm Algorithm", *Int. J. Numer. Meth. Engng*, 2003

[72] Koh, B, George A. D., Haftka, R. T., Fregly, B., "Parallel Asynchronous Particle Swarm Optimization", International Journal for Numerical Methods in Engineering, 67:578-595, 2006

[73] Venter, G., Sobieszczanski-Sobieski, J., "A Parallel Particle Swarm Optimization Accelerated by Asynchronous Evaluations", 6th World Congresses of Structural and Multidisciplinary Optimization, Rio de Janiero, Brazil, June 2005

[74] Belal, M., El-Ghazawi, T., "Parallel Models for Particle Swarm Optimizers", The International Journal of Intelligent Computing and Information Sciences, 4(1):100-111, 2004

[75] Shi, X., Lu, Y., Zhou, C., Lee, H., Lin, W., Liang, Y., "Hybrid Evolutionary Algorithms Based on PSO and GA", 2003 Congress on Evolutionary Computation, pp. 2393-2399, Vol. 4, 2003

[76] Cui, S., Weile, D., "Application of Parallel Particle Swarm Optimization Scheme to the Design of Electromagnetic Absorbers", IEEE Transactions on Antennas and Propagation, pp. 3616-3624, Vol. 53, issue 11, November 2005

[77] Kim, J., Jeong, H., Lee, H., Park, J., "PC Cluster Based Parallel PSO Algorithm for Optimal Power Flow", International Conference on Intelligent Systems Applications to Power Systems, pp. 1-6, November 2007

[78] Jin, N., Rahmat-Samii, Y., "Parallel Particle Swarm Optimization and Finite-Difference Time-Domain (PSO/FDTD) Algorithm for Multiband and Wide-band Patch Antenna Designs", IEEE Transactions on Antennas and Propagation, pp. 3459-3468, Vol. 53, Issue 11, November 2005

[79] Gies, D., Rahmat-Samii, Y., "Reconfigurable Array Design Using Parallel Particle Swarm Optimization", Internationsl Symposium of IEEE Antennas and Propagation Society, 2003, pp. 177-180, Vol. 1, June 2003

[80] Sutter, H., "The Free Lunch is Over: A Fundamental Turn Toward Concurrency in Software", Dr. Dob's Journal, 30(3), March 2005, website: http://www.gotw.ca/publications/concurrency-ddj.htm, accessed February 2007

[81] Owens, J., Luebke, D., Govindaraju, N., Harris, M., Krüger, J., Lefohn, A., and Purcell. T., "A Survey of General-Purpose Computation on Graphics Hardware" In *Eurographics 2005, State of the Art Reports*, August 2005, pp. 21-51

[82] Fernando, R., Kilgard, M., *The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics,* Addison-Wesley Publications, 2003, ISBN: 0321194969

[83] Rost, R., *OpenGL(R) Shading Language (2nd Edition) (OpenGL),* Addison-Wesley Publications, 2006, ISBN: 0321334892

[84] DirectX 9 High Level Shading Language (Microsoft DirectX 9 HLSL), http://msdn2.microsoft.com/en-us/library/ms810449.aspx, accessed February 2007

[85] McCool, M., D Toit, S., Popa T., Chan, B., Moule K., "Shader Algebra", ACM Transactions on Graphics 23,3 August 2004, pp. 787-795

[86] Bleiweiss, A., Preetham, A., "Ashli-Advanced Shading Language Interface", ACM Siggraph Course Notes, July 2003, http://ati.amd.com/developer/SIGGRAPH03/AshliNotes.pdf, accessed February 2007

[87] Buck, I., Foley, T., Horn, D., Sugerman J., Fatahalian K., Houston, M., Hanrahan, P., "Brook for GPUs: Stream Computing on Graphics Hardware", ACM Transactions on Graphics 23, 3, August 2004, pp. 777-786

[88] McCormick, P., Inman J., Ahrens, J., Hansen, C., Roth, G., "Scout: A Hardware-Accelerated System for Quantitatively Driven Visualization and Analysis", In IEEE Visualization 2004, October 2004, pp. 171-178

[89] Tarditi D., Puri, S., Oglesby, J., "Accelerator: Using Data-Parallelism to Program GPUs for General Purpose Uses", In Proceedings of the Twelfth International Conference on Architectural Support for Programming Languages and Operating Systems, October 2006

[90] Lucas P., Fritz, N., Wilhelm, R., "The CGiS Compiler", In proceedings of the 15th International Conference on Compiler Construction", vol. 3923 of Lecture Notes in Computer Science, Springer, March 2006, pp. 105-108

[91] Lefohn, A., Kniss, J., Strzodka, R., Sengupta, S., Owens, J., "Glift: An Abstraction for Generic, Efficient GPU Data Structures", ACM Transactions on Graphics 26, 1, January 2006, pp. 60-99

[92] "NVIDIA CUDA Homepage", http://developer.nvidia.com/object/cuda.html, accessed February 2007

[93] Wasson, S., "ATI Stakes Claims on Physics, GPGPU Ground", The Tech Report – Personal Computing Explored", Oct 11, 2005 - http://techreport.com/onearticle.x/8887, accessed February 2007

[94] Owens, J., Luebke, D., Govindaraju, N., Harris, M., Krüger, J., Lefohn, A., and Purcell. T.,. "A Survey of General-Purpose Computation on Graphics Hardware", Volume 26 (2007). Computer Graphics Forum, Accepted for publication in March 2007 or June 2007

[95] Kruger J., Schiwietz, T., Kipfer, P., Westermann, R., "Numerical Simulations on PC Graphics Hardware", EuroPVM/MPI 2004, LNCS 3241, pp. 442-449, Springer-Verlag Berlin Heidelberg, 2004

[96] Introduction to Randomness and Random Numbers, Web Ref: http://www.random.org/randomness/, accessed July 15, 2008

[97] Ratnaweera A. C., Halgamuge S. K., Watson H. C., "Particle Swarm Optimizer with Time Varying Acceleration Coefficients", In: proceedings of the International Conference on Soft Computing and Intelligent Systems, pp. 240-255, 2002

[98] Suganthan P. N., "Particle Swarm Optimiser with Neighborhood Operator", Proceedings of the IEEE Congress on Evolutionary Computation, IEEE, Piscataway, NJ, pp. 1958 – 1062, 1999

[99] Walpole, R., Myers, R., "Probability and Statistics for Engineers and Scientists", 2nd Ed. Macmillan Publishing Co., Inc., ISBN: 0-024-241105, 1978

[100] McClave, S., "Probability and Statistics for Engineers", PWS Publishers, ISBN: 0-534-06486, 1986

[101] Ross, S., "Introduction to Probability and Statistics for Engineers and Scientists', 2nd edition, Academic Press, ISBN: 0-125-984723, 2000

[102] Urdan, T., "Statistics in Plain English", 2nd edition, Lawrence Erlbaum Associates Publishers, ISBN: 0-805-852417, 2005

[103] Geist, A., Beguelin, A., Dongarra, J., Manchek, R., Jiang, W., Sunderam, V., "PVM 3 User's Guide and Reference Manual", Technical Report ORNL/TM-12187, Oak Ridge National Laboratory, Knoxville, TN, 1994

[104] "InfiniBand – Wikipedia, the free encyclopedia", http://en.wikipedia.org/wiki/Infiniband, accessed May 2008

[105] "InfiniBand® Trade Association: Home", http://www.infinibandta.org/home, accessed May 2008

[106] "Myrinet – Wikipedia", http://en.wikipedia.org/wiki/Myrinet, accessed May 2008

[107] "Myricom Home Page", http://www.myri.com/, accessed May 2008

[108] Rost, R., "OpenGL Shading Language", Second Edition, Addison Wesley Publications, ISBN: 032-133-4892, 2006

[109] Kuhn, H. W., Tucker, A., "Nonlinear Programming", Proceedings of the 2nd Berkeley symposium on Mathematical Statistics and Probability, University of California Press, Berkeley, 1951

[110] Kelly, J. E., "The Cutting Plane Method for Solving Convex Programs", J. SIAM, Vol. 8, pp. 702-712, 1960

[111] Zoutendijk, G.., "Methods of Feasible Directions", Elsevier publications, Amsterdam, 1960

[112] Abadie, J., Carpentier, J., "Generalization of the Wolfe Reduced Gradient Method to the Case of Nonlinear Constraints, in Optimization", R. Fletcher (ed.), Academic Press, New York, pp. 37-47, 1969

[113] Gabriele, G. A., and Ragsdell, K. M., "The Generalized Reduced Gradient Method: A Reliable Tool for Optimal Design", ASME J. Engin. Ind., series B, vol. 99, no. 2, pp. 394-400, May 1977

[114] Kavlie, D., Moe, J., "Automated Design of Frame Structures", ASCE Journal of Struct. Div., vol. 97, No. ST1, pp 33-62, January 1971

[115] Cassis, J. H., Schmit, L. A., "On Implementation of the Extended Interior Penalty Function", International Journal for Numerical Methods in Engineering, vol. 10, no. 1, pp. 3-23, 1976

[116] Haftka, R. T., Starnes, Jr., J. H., "Applications of a Quadratic Extended Interior Penalty Function for Structural Optimization", AIAA Journal, vol. 14, no. 6, pp. 718-724, June 1976

[117] Prasad, B., "A Class of Generalized Variable Penalty Methods for Nonlinear Programming", Journal of Optimization Theory and Applications, vol. 35, no. 2, pp. 159-182, October 1981

[118] Rockafellar, R. T., "The Multiplier Method of Hestenes and Powell Applied to Convex Programming", J. Optim. Theory Appl., vol. 12, no. 6, pp. 555 – 562, 1973

[119] Sedlaczek, K., Eberhard, P., "Using Augmented Lagrangian Particle Swarm Optimization for Constrained Problems in Engineering", Structural Multidisciplinary Optimization Journal, Vol. 32, pp. 277-286, 2006

[120] "Extensible Markup Language (XML) 1.0 (Fourth Edition)", Web reference: http://www.w3.org/TR/REC-xml/ retrieved: May 23, 2008

[121] Engelbrecht, A., "Fundamentals of Computational Swarm Intelligence, Wiley Publications, NY, ISBN: 047-009-1916, 2006

[122] "Test Problems in Global Optimization", Web Reference: http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar_files/TestGO_files/Page364.htm, cited May 23, 2008

[123] "GEATbx: Example Functions (Single and Multi-objective Functions) 2 Parametric Optimization", Web Reference: http://www.geatbx.com/docu/fcnindex-01.html, Cited May 23, 2008

[124] Levy, A, and Montalvo, A., "The Tunneling Algorithm for the Global Minimization of Functions", SIAM Journal of Scientific and Statistical Computing 6, pp. 15-29, 1985