OPTIMIZATION-BASED MECHANISM SYNTHESIS USING MULTI-OBJECTIVE PARALLEL ASYNCHRONOUS PARTICLE SWARM OPTIMIZATION

by

Robin McDougall

Bachelor of Engineering, McGill University, 2001

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Masters of Applied Science

In the Graduate Academic Unit of The Faculty of Engineering and Applied Science

Supervisor(s):	Scott Nokleby, Faculty of Engineering and Applied Science
Examining Board:	Remon Pop-Iliev, Faculty of Engineering and Applied Science
	Ghaus Rizvi, Faculty of Engineering and Applied Science
External Examiner:	Ed Waller, Faculty of Energy Systems and Nuclear Science

This thesis is accepted

Dean of Graduate Studies

THE UNIVERSITY OF ONTARIO INSTITUTE OF TECHNOLOGY

December, 2008

Dedication

To my family.

Abstract

A distributed variant of multi-objective particle swarm optimization (MOPSO) called multi-objective parallel asynchronous particle swarm optimization (MOPAPSO) is presented, and the effects of distribution of objective function calculations to slave processors on the results and performance are investigated and employed for the synthesis of Grashof mechanisms.

By using a formal multi-objective handling scheme based on Pareto dominance criteria, the need to pre-weight competing systemic objective functions is removed and the optimal solution for a design problem can be selected from a front of candidates after the parameter optimization has been completed.

MOPAPSO's ability to match MOPSO's results using parallelization for improved performance is presented. Results for both four and five bar mechanism synthesis examples are shown.

Acknowledgements

Without the technical support of my advisor Dr. Scott Nokleby and my colleague Conrad Housand this work would have been impossible. Both provided immeasurable encouragement and technical guidance that benefited this work greatly.

Table of Contents

List of Tables

List of Figures

Nomenclature

(P_x, P_y) Location of Coupler Point

(x_2, y_2) Base Pin Location

α Angle of Base Link

θ_2 Input Angle

c_1 Cognitive Velocity Constant

c_2 Social Velocity Constant

f_{obj} Objective Function

P Coupler Point

p_{gb} Fittest Position Found by any Particle in the Swarm So Far

$P_{id,init}$ ith Particle's Initial Position

p_{pb_i} Fittest Position Found by the ith Particle So Far

r_n nth Link in the Mechanism

r_{long} Length of the Longest Link

r_{short} Length of the Shortest Link

$randn()$ Random Number Drawn from a Uniform Distribution

$V_{id,init}$ ith Particle's Initial Velocity

V_i ith Particle's Velocity

V_{max} Maximum Particle Velocity

V_{min} Minimum Particle Velocity

x_i ith Particles Current Position

DOF Degrees-of-Freedom

GA Genetic Algorithm

GE Global Evolutionary Algorithm

GR Gearing Ratio

MO Multi-objective Optimization

MOPSO Multi-objective Particle Swarm Optimization

NGSA-II Niched Genetic Simulated Anealling II Optimization Algorithm

OBMS Optimization-based Mechanism Synthesis

PSO Particle Swarm Optimization

Chapter 1

Introduction

With the growth in availability of increasingly powerful personal computers, mechanism designers are becoming more ambitious in the degree of task specification and the complexity of objectives considered during mechanism synthesis. In mechanisms with tasks encapsulated in a single objective function with a small number of specified path points analytical or graphical solutions may be available [1]. However, for more sophisticated tasks with well defined behavioural specifications, Optimization-Based Mechanism Synthesis (OBMS) techniques are commonly employed.

An important consideration in the development of any OBMS routine is the choice of optimization algorithm to use. The trend toward highly parameterized models with objective functions of escalating nonlinearity has now begun to expose some deficiencies in the traditional deterministic derivative driven optimization routines that have in the past dominated engineering design. Specifically, as the number of parameters increases or the nonlinearity of the objective function surface grows,

there is an increased sensitivity to the arbitrary starting point used by the optimization algorithm and similarly an elevated likelihood to converge on local phenomena instead of the global minimum that is desired.

This deficiency has led to increasingly frequent use of a new class of stochastic optimization algorithms in engineering design. Genetic Algorithms (GA) or Global Evolutionary (GE) optimization algorithms [2] have been used in an increasing number of published optimization studies in a widening breadth of engineering and scientific domains. The motivation for the use of both are quite understandable. Members of these two classes often overcome the shortcomings of traditional deterministic optimization algorithms applied to complex systems - typically through the introduction of a stochastic influence. One particular member of the evolutionary optimization algorithm class is the derivative-free Particle Swarm Optimization (PSO) algorithm which has been used in various forms in an impressive range of engineering applications [3–5].

The use of stochastic optimization algorithms in the synthesis of planar four-bar and five-bar mechanisms with weighted tasks combined into singular objective functions is not new. Genetic algorithms [6], the stochastic Tabu-Gradient search algorithm [7], and more recently PSO [8, 9] have been successfully employed in OBMS.

The application of formal multi-objective optimization (MO) [10–12] techniques to mechanism synthesis is also quite encouraging [13], using this class of formal multi-objective handling techniques removes the requirement for pre-weighting objectives by solving for a series of optimum solutions for each value of each objective that cannot be improved without negatively affecting another objective.

In this work a new OBMS routine is presented. By using a variation of the general multi-objective particle swarm optimization (MOPSO) routine to handle multiple task objectives and a parallel implementation of the PSO algorithm [14, 15], the benefits of a formal multi-objective handling capability can hopefully be realized in a reduced amount of processing time - particularly in design problems with significant objective calculation times like OBMS. This routine aims to provide superior feasible search space coverage in a reasonable time with the potential to employ dynamic exploratory algorithmic settings.

1.1 Mechanisms

Mechanisms are devices that receive input forces and motions and transforms them to corresponding output forces and motions. From the near infinite set of class and configurations of mechanisms possible, one particularly popular class in both research and engineering design is the planar mechanism. Planar rigid-body mechanisms are typically closed-loop kinematic chains consisting of an arbitrary number of rigid-bodies connected by revolute pin-joints. Two classes of these mechanisms are studied in this work, four-bar and five-bar planar mechanisms.

1.1.1 Four-Bar Mechanisms

The general form of the four-bar mechanism is shown in Figure 1.1. The four links are the base link r_1, the input link (or crank) r_2, the coupler r_3, and the output link r_4. The total degrees-of-freedom (DOF) of the planar four-bar mechanism which cor-

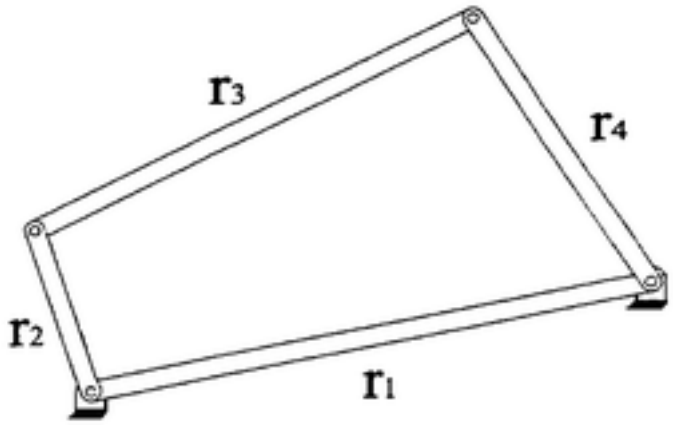

Figure 1.1: Four-bar Mechanism - General Form

responds to the number of independent variables required to define the instantaneous state of the system is one by Grubler's equation [16]:

$$DOF = 3(n-1) - 2f_1 - f_2 \tag{1.1}$$

where n is the number of links, f_1 is the number of joints permitting one DOF of relative motion between two links, and f_2 is the number of joints permitting two DOF of relative motion between two links.

Four-bar mechanisms are the most abundantly used planar mechanism and the most studied [17]. Compared to other planar mechanisms, four-bar mechanisms are often easier to design and cheaper to manufacture.

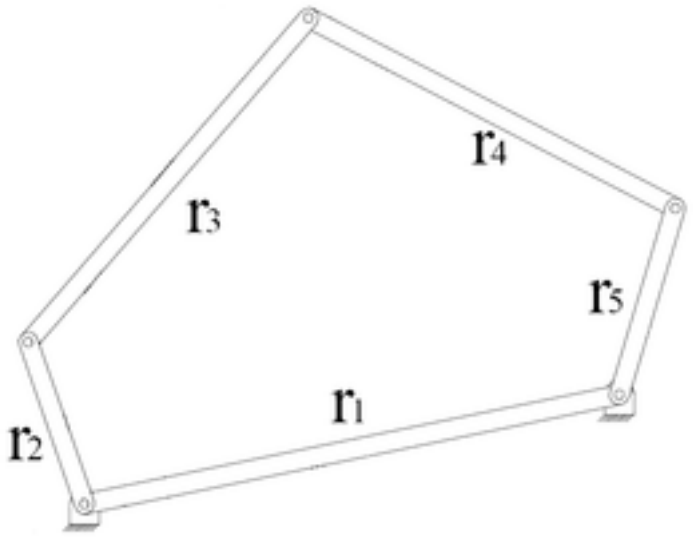

Figure 1.2: Five-bar Mechanism - General Form

1.1.2 Five-Bar Mechanisms

The general form of the five-bar mechanism is shown in Figure 1.2. The five links are the base link r_1, two input links r_2 and r_5, and two coupler links r_3 and r_4. The total DOF of the planar five-bar mechanism is two [16], meaning that two inputs are required to define the state of the system. In this study, as is often the case in industry, it is assumed that the two input links are geared together, thus forming a one degree-of-freedom device.

Five-bar mechanisms are used less frequently than four-bar mechanisms due to the cost of gearing though they can achieve more complex motions than four-bar mechanisms [17]. In addition, the dimensional synthesis of five-bar mechanisms is more complex than that of four-bar mechanisms.

1.1.3 Grashof Mechanisms

One prominent class of planar mechanisms are Grahsof mechanisms [16], which can receive a continuous rotary input and deliver a periodic regular output. These mechanisms, named for the 19th century engineer who first identified the geometrical requirements for four-bar mechanisms, satisfy the physical requirement that the geometry of a planar mechanism allows at least one pair of adjacent links to complete a full rotation relative to each other.

The determination of sufficient Grashof criteria for more sophisticated mechanism classes has been a topic of significant research over the last century, in this work the necessary conditions determined by Ting [18] were employed in the mechanism synthesis of geared five-bar Grashof mechanisms.

Mechanisms that satisfy the corresponding Grashof constraints for their corresponding class can be driven by a continuous rotary input, and can be seen in practice among many other places in the piston assemblies of automobiles and in the geometry of digging machines.

1.2 Mechanism Synthesis

The choice of mechanism synthesis routine is a procedure governed by three competing objectives: the desire to specify a mechanism tasks as explicitly as required, the capability to find the best mechanism possible to match those tasks, and completion of the mechanism synthesis exercise in a reasonable amount of time. For design problems with a limited number of specified path or transmission require-

ments, closed-form analytical solution techniques may be available which give exact solutions [16]. However, as design criteria mandate an increasing number of specifications, and closed-form analytical solutions no longer exist, optimization-based mechanism synthesis often becomes the design tool of choice.

Optimization is the mathematical process of solving the problem:

$$minimize\, f(x),\ x\ \in\ \Re^n \tag{1.2}$$

subject to constraints:

$$h(x) \leq 0 \tag{1.3}$$

$$g(x) = 0 \tag{1.4}$$

where $f(x)$ is the objective function(s), h(x) is one or more inequality constraints, g(x) is one or more equality constraints, and the search vector x contains the design variable parameters [19]. The objective function in this work is quantification for how well the proposed mechanism completes the mechanism tasks, when the design requirements demand that only Grashof mechanisms with reasonable geometries should be considered. This mechanism synthesis problem becomes a progressively nonlinear constrained optimization problem as the number of mechanism task specifications is increased.

Unlike analytical or graphical methods which aim to determine the best configuration for a particular task directly, OBMS is an iterative process where candidate configurations are proposed by an optimization algorithm with an aim to minimize an objective function which corresponds to the desired mechanism task.

1.3 Objectives

The objectives of this work are to develop an OBMS routine which uses a formal multi-objective handling scheme based on Pareto dominance in concert with a distributed PSO implementation. Continuing the work of Nokleby [17], the specific objectives of this research are to:

- Investigate the utility of using PSO as the optimization engine in OBMS.
- Explore existing PSO variations, and consider new PSO implementations, that can be used in highly-parameterized non-linear optimization problems.
- Any new work should be presented in an extensible form, easily applied to any general optimization problem.

1.4 Summary of Contents

In Chapter 2, the derivations of the governing equations required to define the motion of four-bar and geared five-bar mechanisms as a function of input angle(s) are shown. Chapter 3 provides an overview of the basic form of PSO, and reviews techniques to allow for parallelization of the algorithm's operations on a cluster to improve performance.

The development of a PSO routine that employs Pareto dominance, constraint handling, and parallelization is covered in Chapter 4.

In Chapter 5, the performance of the new multi-objective, distributed particle swarm optimization algorithm applied to OBMS is investigated.

Finally, Chapter 6 reviews the work, draws conclusions, and outlines areas for further investigation.

Chapter 2

Mechanism Models and Task Formulations

The first requirement to develop an OBMS routine is a parameterized model of the mechanism class under consideration. Once expressions for the instantaneous state of the mechanism as a function of the input link angle is obtained, the mechanism's cyclical behaviour can be determined.

2.1 Four-Bar Mechanism Model

Figure 2.1 shows a general four-bar mechanism. Ten parameters are required to sufficiently describe the state of a given four-bar mechanism. In this work the ten parameters used are: the input angle θ_2, the angle of the base link α, the link lengths r_1 to r_6, and the base pin location (x_2, y_2). From this parameter set, expressions for the position of P and the link angles θ_3 and θ_4 can be derived [20].

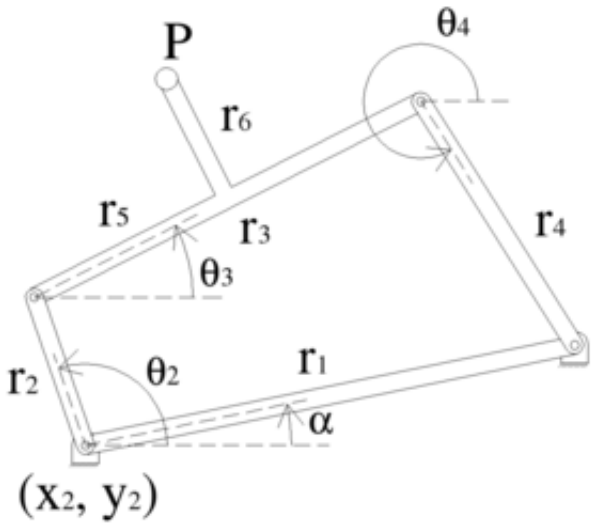

Figure 2.1: Four-Bar Mechanism - Link Lengths and Angles

Figure 2.2 is a vector representation of a four-bar mechanism. From Figure 2.2, two vector equations defining the mechanism and its coupler location P are:

$$\mathbf{z}_2 + \mathbf{z}_3 + \mathbf{z}_4 - \mathbf{z}_1 = \mathbf{0} \tag{2.1}$$

$$\mathbf{P} = \mathbf{z}_2 + \mathbf{z}_5 + \mathbf{z}_6 \tag{2.2}$$

Equation (2.1) can be written as two scalar equations:

$$r_2 cos\theta_2 + r_3 cos\theta_3 - r_1 cos\alpha = -r_4 cos\theta_4 \tag{2.3}$$

$$r_2 sin\theta_2 + r_3 sin\theta_3 - r_1 sin\alpha = -r_4 sin\theta_4 \tag{2.4}$$

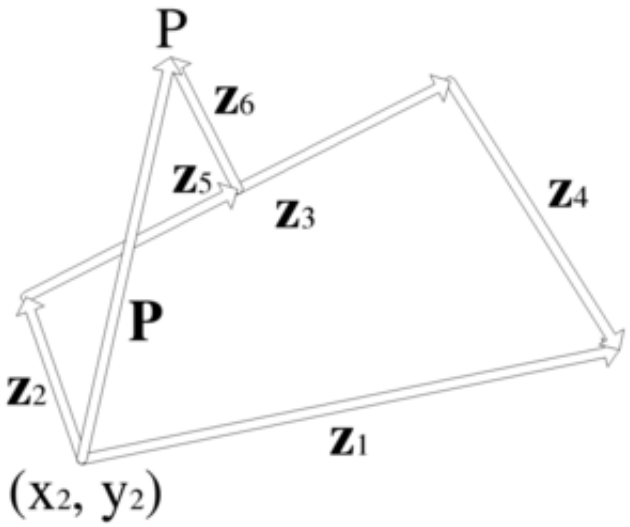

Figure 2.2: Four-bar Mechanism - Vector Representation

Squaring and adding equations (2.3) and (2.4) allows the θ_3 solution:

$$\theta_3 = arctan2(A2, A1) \pm arctan2(\sqrt{A_1^2 + A_2^2 - A_3^2}, A_3) \tag{2.5}$$

where $A_1 = r_2 cos\theta_2 - r_1 cos\alpha$, $A_2 = r_2 sin\theta_2 - r_1 sin\alpha$, and $A_3 = -(A_1^2 + A_2^2 + r_3^2 - r_4^2)/(2r_3)$, and $arctan2$ calculates $arctan(y/x)$ and returns an angle in the correct quadrant. The plus/minus sign in equation (2.5) reflects the two possible assembly modes of the mechanism. Solving for $cos\theta_4$ and $sin\theta_4$ from equations (2.3) and (2.4) allows the θ_4 solution:

$$\theta_4 = arctan2(A_4, A_5) \tag{2.6}$$

where

$$A_4 = r_4 sin\theta_4 = r_1 sin\alpha - r_2 sin\theta_2 - r_3 sin\theta_3 \tag{2.7}$$

$$A_5 = r_4 cos\theta_4 = r_1 cos\alpha - r_2 cos\theta_2 - r_3 cos\theta_3 \tag{2.8}$$

With θ_3 known, the coupler point location P defined in equation (2.2) can be written in terms of its two scalar components:

$$P_x = x_2 + r_2 cos\theta_2 + r_5 cos\theta_3 + r_6 cos(\theta_3 + \frac{\pi}{2}) \tag{2.9}$$

$$P_y = y_2 + r_2 sin\theta_2 + r_5 sin\theta_3 + r_6 sin(\theta_3 + \frac{\pi}{2}) \tag{2.10}$$

2.1.1 Four-Bar Grashof Criteria

In many cases it is desirable to apply a continuous rotary input, these mechanisms must satisfy the Grashof criteria for four-bar mechanisms:

$$r_{long} + r_{short} \leq r_a + r_b \tag{2.11}$$

where r_{long} is the length of the longest link, r_{short} is the length of the shortest link, and r_a and r_b are the lengths of the other two links [16]. A mechanism satisfying this criteria will contain at least one pair of links capable of completing a full rotation relative to one another.

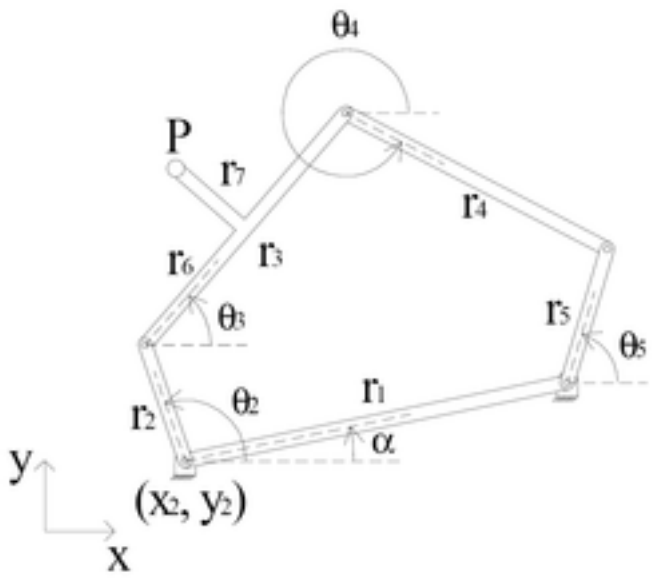

Figure 2.3: Five-bar Mechanism - Link Lengths and Angles

2.2 Five-Bar Mechanism Model

Figure 2.3 shows that a five-bar mechanism consists of five links pinned at the ends forming a closed loop. Twelve parameters are required to sufficiently describe a given five-bar mechanism with coupler point P. In this work the twelve parameters used are the input angles θ_2 and θ_5, the angle of the base link α, the link lengths r_1 to r_5, two lengths r_6 and r_7 defining the coupler point location, and the base pin location (x_2, y_2). From these parameters, expressions for the corresponding position of point P and the link angles θ_3 and θ_4 can be derived [17].

Figure 2.4 is a vector representation of a five-bar mechanism. From Figure 2.4, two vector equations defining the mechanism and its coupler location are:

$$\mathbf{r}_2 + \mathbf{r}_3 + \mathbf{r}_4 - \mathbf{r}_5 - \mathbf{r}_1 = \mathbf{0} \tag{2.12}$$

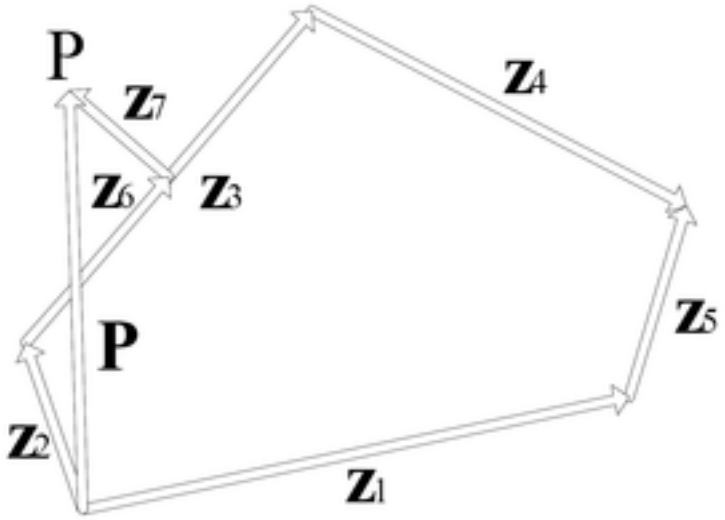

Figure 2.4: Five-bar Mechanism - Vector Representation

$$\mathbf{P} = \mathbf{r}_2 + \mathbf{r}_6 + \mathbf{r}_7 \tag{2.13}$$

Equation (2.12) can be written as two scalar equations:

$$r_2 cos\theta_2 + r_3 cos\theta_3 - r_5 cos\theta_5 - r_1 cos\alpha = -r_4 cos\theta_4 \tag{2.14}$$

$$r_2 sin\theta_2 + r_3 sin\theta_3 - r_5 sin\theta_5 - r_1 sin\alpha = -r_4 sin\theta_4 \tag{2.15}$$

Squaring and adding equations (2.14) and (2.15) allows the θ_3 solution:

$$\theta_3 = arctan2(B2, B1) \pm arctan2(\sqrt{B_1^2 + B_2^2 - B_3^2}, B_3) \tag{2.16}$$

where $B_1 = r_2 cos\theta_2$-$r_5 cos\theta_5$-$r_1 cos\alpha$, $B_2 = r_2 sin\theta_2 - r_5 sin\theta_5 - r_1 sin\alpha$, and $B_3 = -(B_1^2 + B_2^2 + r_3^2 - r_4^2)/(2r_3)$. The plus/minus sign of equation (2.16) reflects the

two possible assembly modes of the mechanism. Solving for $cos\theta_4$ and $sin\theta_4$ from equations (2.14) and (2.15) allows the θ_4 solution:

$$\theta_4 = arctan2(B_4, B_5) \tag{2.17}$$

where

$$B_4 = r_5 sin\theta_5 + r_1 sin\alpha - r_2 sin\theta_2 - r_3 sin\theta_3 \tag{2.18}$$

$$B_5 = r_5 cos\theta_5 + r_1 cos\alpha - r_2 cos\theta_2 - r_3 cos\theta_3 \tag{2.19}$$

With θ_3 known, the coupler point location P defined in equation (2.13) can be written in terms of its two scalar components:

$$P_x = x_2 + r_2 cos\theta_2 + r_6 cos\theta_3 + r_7 cos(\theta_3 + \frac{\pi}{2}) \tag{2.20}$$

$$P_y = y_2 + r_2 sin\theta_2 + r_6 sin\theta_3 + r_7 sin(\theta_3 + \frac{\pi}{2}) \tag{2.21}$$

2.2.1 Gear Ratio

The gear ratio, G_R, for a five-bar mechanism with links two and five (r_2 and r_5) geared together relates the change in θ_2 to the change in θ_5:

$$\Delta\theta_5 = G_R \Delta\theta_2 \tag{2.22}$$

The gear ratio must be an integer to ensure that for one complete rotation of r_2, r_5 will complete $G_R 2\pi$ radians of rotation. For the purpose of this work it is assumed

that an integer gear ratio is specified.

2.2.2 Geared Five-Bar Grashof Criteria

If it is assumed that the five-bar mechanism to be synthesized has its two input links as the two shortest links (r_2=$r_{input1} = r_{short1}$ and $r_5 = r_{input2} = r_{short2}$) and its base link as the longest link ($r_1 = r_{long}$), a modified Grashof criterion for geared five-bar mechanisms that have links two and five geared together can be defined as [18]:

$$d_{max} \leq r_3 + r_4 \tag{2.23}$$

where the vector $\mathbf{d}$ is $\mathbf{d} = \mathbf{r}_1 + \mathbf{r}_5 - \mathbf{r}_2$ and the length d is:

$$d = ||\mathbf{d}|| = ((r_1 cos\alpha + r_5 cos\theta_5 - r_2 cos\theta_2)^2 + (r_1 sin\alpha + r_5 sin\theta_5 - r_2 sin\theta_2)^2)^{1/2} \tag{2.24}$$

2.3 Model Execution

Equation pairs (2.9 & 2.10) and (2.20 & 2.21) give position of the coupler guidance point P for four and five-bar mechanisms, respectively, where the motion of each mechanism class can be expressed as a function of the input angle only:

$$(P_x, P_y) = function(\theta_2) \tag{2.25}$$

The kinetics of each mechanism then can be determined by sweeping the input crank θ_2 through one complete revolution (from 0 to 2π). The result will be a coupler curve

showing the path of point P.

Chapter 3

Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a relatively new global evolutionary optimization method for nonlinear systems inspired by the social behaviour of flocking birds or schooling fish [21]. In this scheme, particles analogous to members of the flock search for the most profitable area to "feed" which corresponds to the best solution for a optimization problem. Iteratively, these particles begin to converge on optimum areas of the feasible space through influences of each particle's search history and the collective progress of the entire swarm.

In the algorithm, system parameters are each mapped to a particle position dimension resulting in a Particle Swarm Optimizer exploring a multi-dimensional feasible space, the order of which corresponds to the number of parameters being varied. The fitness of each particle is the instantaneous objective function value of the candidate system corresponding to each particle's position.

The swarm of particles explores the n-dimensional feasible space, drawn to areas of in-

creased fitness discovered individually and globally. With each iteration, the updated position for each particle is determined by a random combination of these influences, which over time will cause the individual particle (and by extension the swarm) to survey the particle space while generally trending toward the global optimum value. Cognitive and social gains applied to each influence allow the exploratory and exploitationary nature of the algorithm to be varied before the search or dynamically during the run itself.

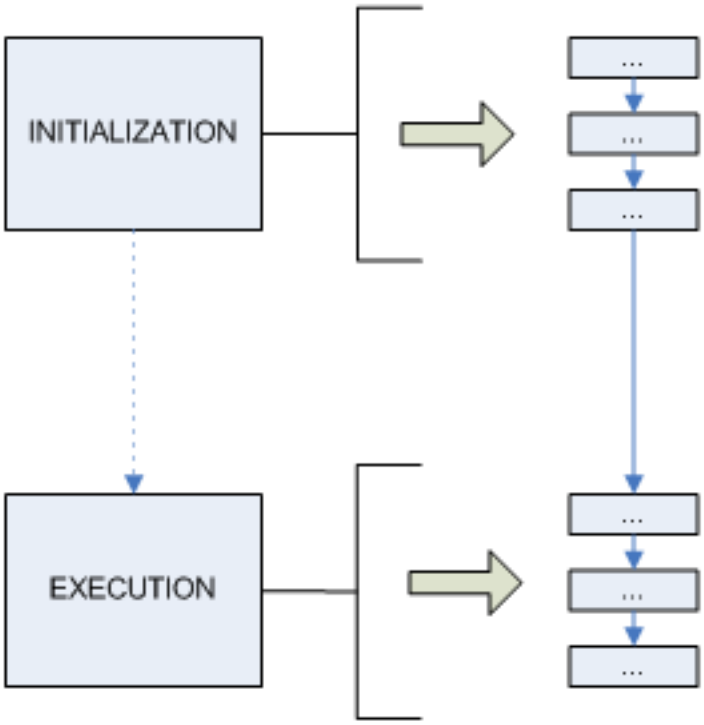

Figure 3.1: PSO - Phase Overview

While a significant volume of variants and modifications to PSO have been proposed in recent years, the majority build upon the basic PSO form. This base form consists

of two significant phases shown in Figure 3.1, a swarm initialization phase and the swarm execution phase. The pseudo-code of the two phases for the traditional form of PSO is as follows:

```
% Swarm Initialization;
foreach particle do
    Initialize particle position;
    Initialize particle velocity;
    Set pBest (Personal Best) ;
end
Solve for initial gBest (Global Best) ;
% Swarm Execution ;
while iteration count less than limit do
    foreach particle do
        Calculate particle velocity;
        Update particle position;
        if particle fitness is better than pBest then
            Update pBest;
        end
        if particle fitness is better than gBest then
            Update gBest;
        end
    end
end
```

Algorithm 1: Overview of traditional PSO

3.1 Swarm Initialization

The initialization phase of PSO consists of three significant activities shown in Figure 3.2: setting the algorithmic parameters, initializing particle positions and velocities, and determining the global best particle position gBest and assigning the personal best particle position pBest value for each particle.

3.1.1 Setting the Algorithmic Parameters

In many ways one of the most powerful characteristics of PSO is the elegance in which the nature of the optimization execution can be controlled through the judicious choice of only a few parameters. There are two important classes of algorithmic parameters in PSO - those that define the swarm and those that define its behaviour. The swarm itself is characteristically defined by the number of particles it contains. While there is no consensus of an optimal number of particles, typically between 20-100 particles are used [3]. Generally, the larger the swarm, the more significant percentage of the feasible space can be covered during the swarm's initialization, creating an algorithm that is relatively more exploratory in nature at the cost of requiring more objective function calculations to complete each iteration. An interesting solution to this may be dynamically sized swarms [22] which reduce the number of particles as the swarm begins to converge, decreasing the total number of objective function calculations required for an optimization.

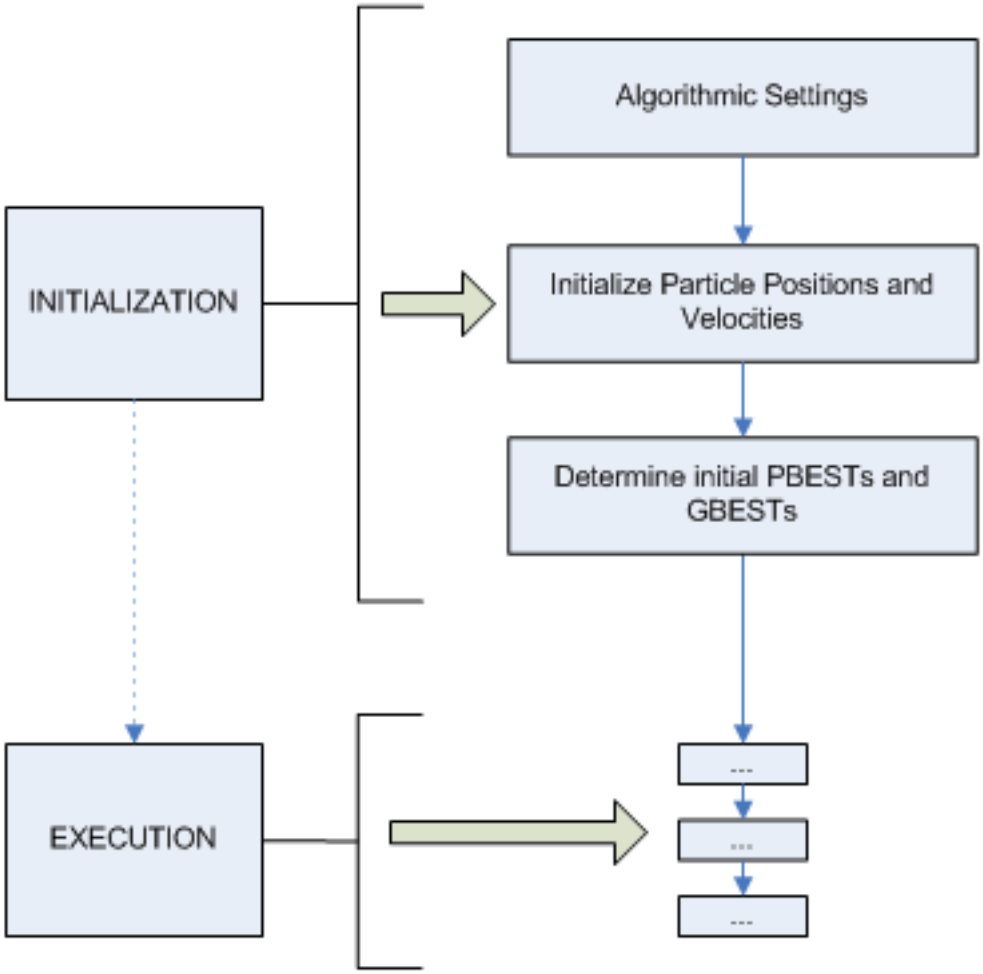

Figure 3.2: PSO - Initialization Phase

3.1.2 Position and Velocity Initialization

In the particle initialization sub-phase the search space is populated with the desired number of particles through the assignment of an initial particle position and initial particle velocity for each particle. The necessary infrastructure to record the search

histories of the individual particles and the entire swarm must be configured and any other bookkeeping activities should occur.

Most often the particles are initialized from a uniform random distribution between upper and lower bounds:

$$P_{i_d,init} = randn(n_p)(P_{max} - P_{min}) \tag{3.1}$$

where $P_{id,init}$ is the ith particle's initial position, $randn(n_p)$ is a random number drawn from a uniform distribution and P_{max} and P_{min} are some arbitrary minimum and maximum bounds on the starting position which often are dictated by systemic constraints. Occasionally, algebraic or algorithmic methods are employed [23] instead of the uniform randomly distributed scheme shown in equation (3.1). The objective is to diversely populate the search space in order to increase the chance that particles within the swarm have an opportunity to discover global phenomena.

The velocity for each particle is similarly initialized using:

$$V_{id,init} = randn(n_v)(V_{max} - V_{min}) \tag{3.2}$$

where $V_{id,init}$ is the ith particle's initial velocity, $randn(n_v)$ is a random number drawn from a uniform distribution, and V_{max} and V_{min} are some arbitrary minimum and maximum bounds on the starting velocity, which are typically also applied to the velocity update equations in order to promote search space exploration.

3.1.3 Global and Personal Best Initialization

The final activity in the initialization phase is the assignment of the personal best value for each particle and the determination of the initial global best position - two significant factors on the manner in which the search space is explored.

After starting positions for each particle have been assigned in the initial section they are also set as the personal best (P_{pb_i} - for the ith particle). The complete set of objective function values returned for the personal bests of all particles are then examined and the position corresponding to the fittest value is used as the initial global best position (P_{gb}).

3.2 Swarm Execution

Once the particles have been distributed diversely in the search space during the initialization phase and the other necessary activities have been completed, the task of searching for the global minimum begins. The execution stage of PSO is a three-phase iterative activity where the particles, representing candidate solutions to the optimization problem, search the feasible space for the global minimum as shown in Figure 3.3. While the potential to measure swarm convergence through the convergence of particle velocities to zero can be used, most often PSO optimizations run for a predetermined number of iterations [24].

With each iteration of the optimizer, the objective function for each particle is determined based on the position and any constraints, the velocity for each particle is calculated, and the position of each particle is updated.

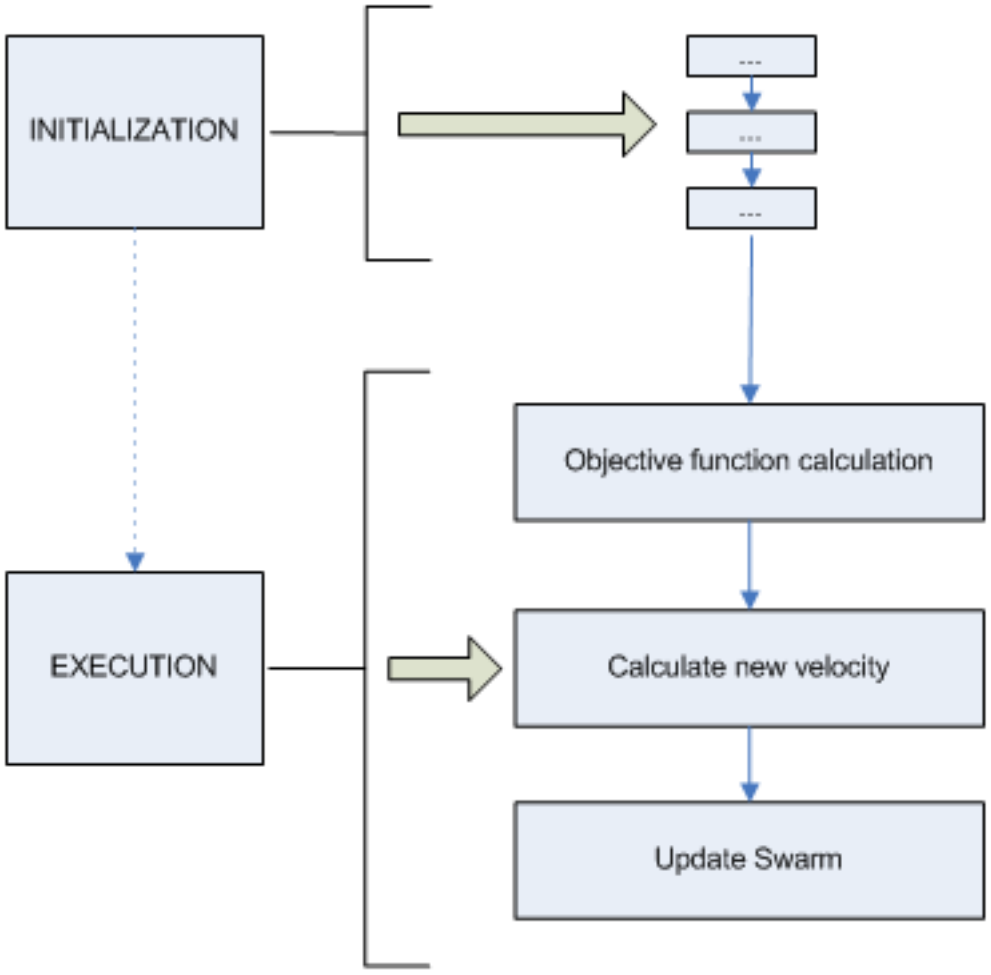

Figure 3.3: PSO - Execution Phase

3.2.1 Objective Function Calculation

The fitness or objective function calculation is completed during each iteration for every particle in the swarm. The objective function serves as a metric indicating how well the candidate solution solves the design problem in question. In traditional PSO, the design metric must be posed as a single objective function - if two (or

more) competing design considerations are to be considered they must be combined into a single objective function:

$$f_{obj} = \omega_1 f_{obj1} + \omega_2 f_{obj2} + ... + \omega_n f_{objn} \tag{3.3}$$

where f_{obj} is the particle's fitness, f_{obji} is the ith objective function value, ω_i is the weighted influence of the ith objective function component, and n is the total number of objectives.

3.2.2 Velocity Calculation

There are three core influences on the velocity term for each particle during each iteration (see Figure 3.4): an inertial contribution proportional to its previous velocity, an exploratory contribution proportional on how far the particle is from its personal best position, and an exploitationary contribution proportional to how far the particle is from the best position found by all particles in the entire swarm.
Upon each iteration (k) of the optimization algorithm, a new velocity is calculated for each particle using:

$$V_i = \omega\, V_i + c_1 rand(n_1)(p_{pb_i} - x_i) + c_2 rand(n_2)(p_{gb} - x_i) \tag{3.4}$$

where V_i is the ith particle's velocity, p_{pb_i} is the fittest position found by the ith particle so far, p_{gb} is the fittest position found by any particle in the swarm so far, x_i is the ith particles current position, ω is the inertial constant, c_1 is the cognitive constant - the relative influence of a particle's personal best position on its velocity,

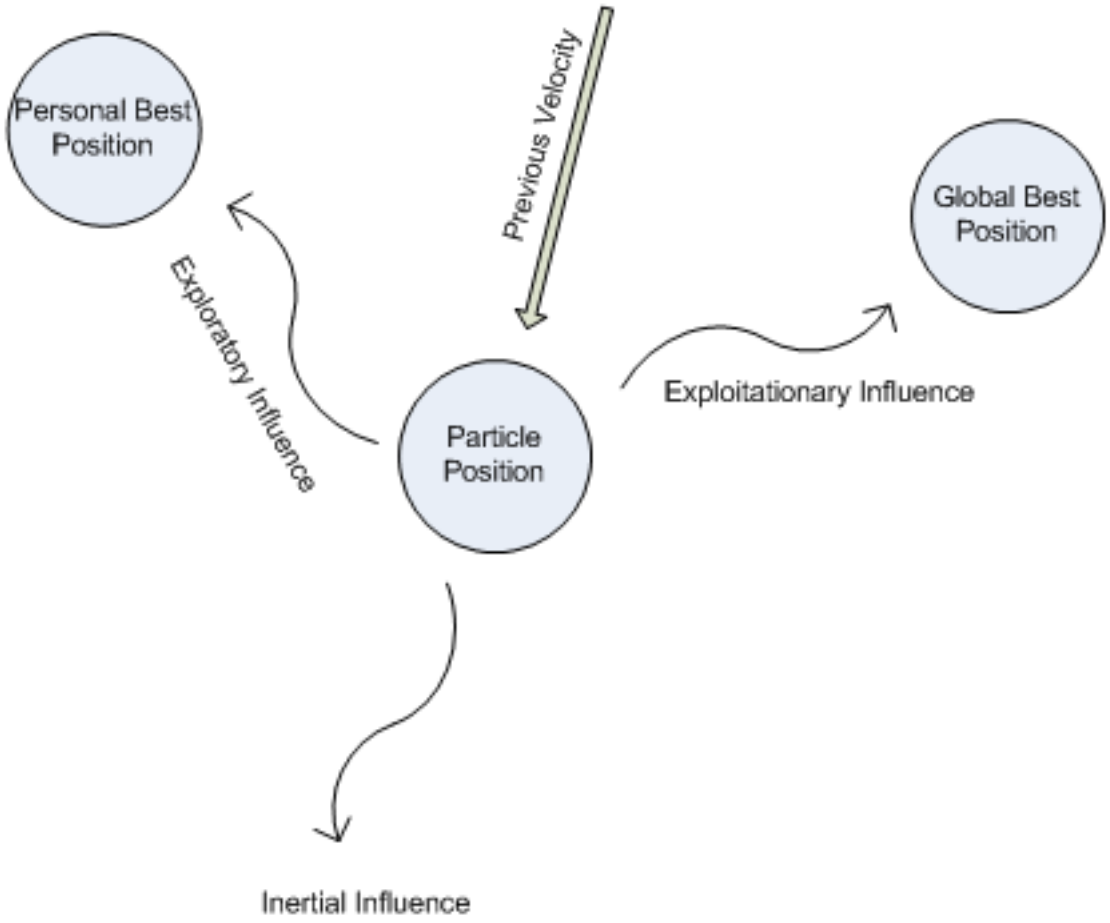

Figure 3.4: PSO - Influences on Particle Velocity

c_2 is the social constant - the relative influence of the global best position on its velocity, and $rand(n_1)$ and $rand(n_2)$ are two random numbers drawn independently from a uniform distribution.

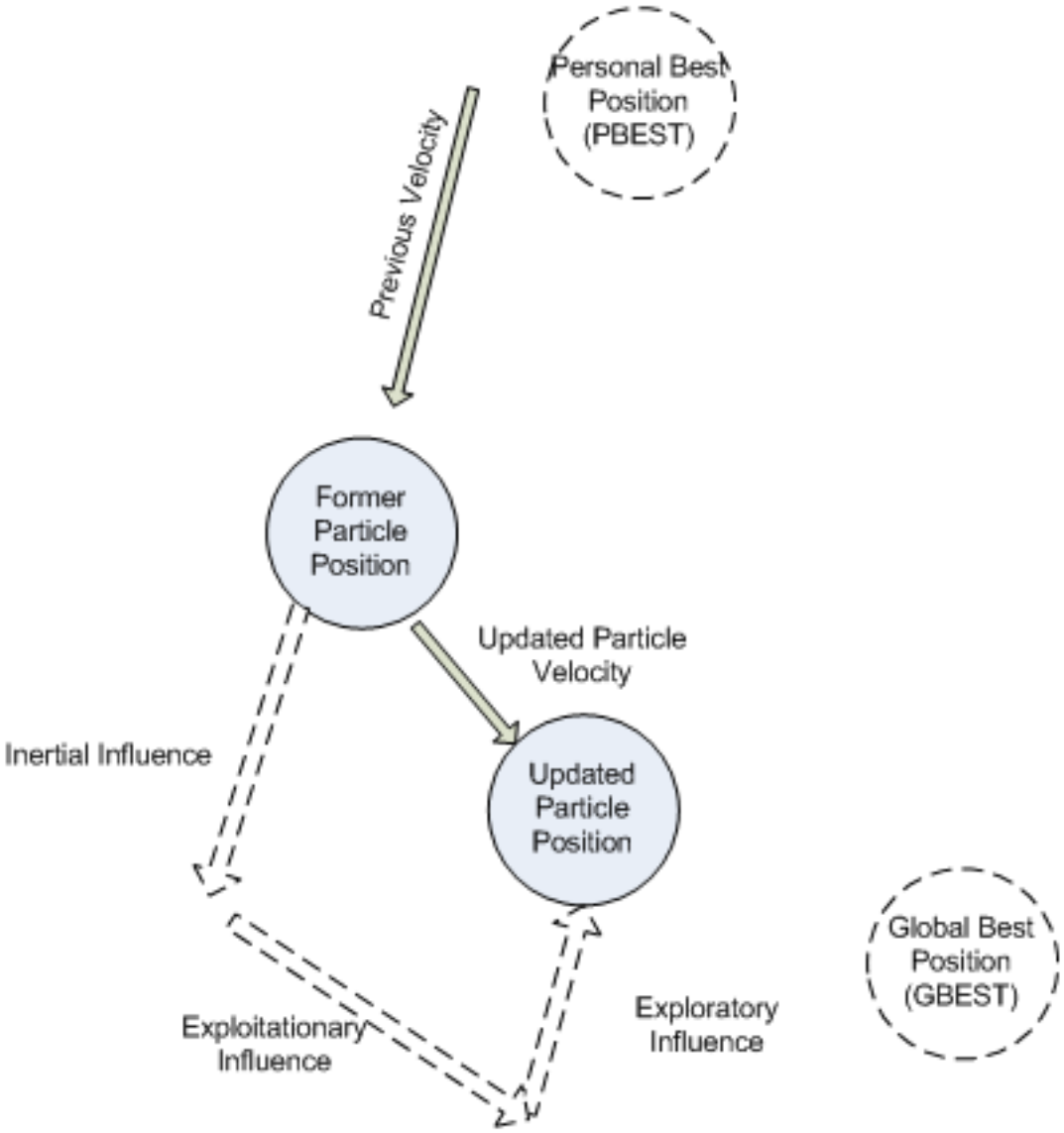

Figure 3.5: PSO - Particle Position Update

3.2.3 Updating the Swarm

Figure 3.5 shows graphically the vector summation of the position update via the velocity equation above. Once the velocities for every particle have been determined

earlier in the iteration, they are then used to determine the proposed updated position for each particle in the objective function space using:

$$x_{i_k} = x_{i_{k-1}} + V_{i_k} \tag{3.5}$$

where k is the current iteration number, x_{i_k} is the position of the ith particle in the kth iteration, $x_{i_{k-1}}$ the position of the ith particle in the $kth - 1$ iteration, and V_{i_k} is the velocity given by equation (3.4) for the ith particle in the kth iteration.

3.3 Constraint Handling

The purpose of any constraint handling scheme in PSO is to promote the exploration of feasible solutions and to minimize the number of candidate infeasible solutions proposed, thus reducing the number of objective function calculations spent on solutions that violate any prescribed constraints.

A number of constraint handling mechanisms are described in the literature. Three options considered in this work are discussed here: penalty functions [24], parameter transformations [25–27], and preservation of feasibility [10] techniques.

3.3.1 Constraint Handling via Penalty Functions

The simplest option is to apply a penalty to the objective function when the candidate solution is infeasible. By adding a penalty to infeasible solutions as opposed to replacing the infeasible objective function altogether, a capability to distinguish which solution from a set of exclusively infeasible candidates is maintained.

During the objective function calculation stage the feasibility of the proposed solution is checked and candidates whose position corresponds to solutions which violate any constraints are assigned an arbitrarily large penalty to their fitness value:

$$P_{fit} = f_{obj} + f_{pen} \tag{3.6}$$

where P_{fit} is the fitness value used by PSO, f_{obj} is the value of the objective function regardless of the feasibility of the solution, and f_{pen} is an arbitrarily large penalty applied when the solution is infeasible and 0 otherwise.

The above has the effect of guiding the particles in the swarm to feasible candidate solutions through the lower P_{pb} value associated with each particle and the lower P_{gb} of the swarm once they first find a position corresponding to a feasible solution.

The disadvantage to a pure penalty function scheme is that a particle leaving the feasible solution space can often spend several iterations overcoming the inertial influence to its velocity component before it re-enters the feasible space. Understandably, if the objective function calculations are not trivial to calculate, this overshoot can have a significant negative impact on runtime and performance.

3.3.2 Constraint Handling via Parameter Transformation

Another option is to use parameter transforms [25–27] so that each proposed solution will always be feasible. While this method requires the additional overhead of unique transform derivations for each set of constraints, every objective function evaluation will correspond to a feasible point which reduces the computational time resulting from the consideration of infeasible points.

For systems of constraints which require relatively trivial parameter transformations, this scheme is arguably the most effective method to deal with constraints - from PSO's perspective it is effectively the same effort as unconstrained optimization.
The drawback however occurs in complex constrained systems or in the pursuit of a general purpose optimization tool. In these cases, the overhead cost of determining appropriate parameter transforms is often prohibitively large or it is impossible to find a parameter transform. Any efforts to simplify the solution for proper transforms run the risk of artificially constricting the search space.

3.3.3 Constraint Handling via Preservation of Feasibility

The third method of constraint handling discussed here is the family that make efforts to preserve the feasibility of particles by discarding any proposed velocity that would cause a particle's current position corresponding to a feasible solution to become one which corresponded to an infeasible one and recalculating velocities until a feasible updated position is determined.
This method has the advantage of only performing objective function evaluations on feasible points, which makes this technique quite efficient - especially when the objective function calculation time is significantly longer than the particle update or feasibility check times. This method is employed in this work.

Chapter 4

Multi-Objective Parallel Asynchronous Particle Swarm Optimization Routine

A great deal of research has led to the proposal of a wide breadth of PSO variants over the last decade and a half. For many modelers of sophisticated engineering design problems one of the most powerful examples are the class of multi-objective handling schemes. One routine, multi-objective particle swarm optimization (MOPSO) [10] has been used extensively since its first proposal on a wide variety of engineering tasks with great success [28, 29].

The power of the MOPSO algorithm does come at an increased computational cost when compared with singular-objective function PSO optimizations. Specifically, swarm sizes tend to be significantly larger which can result in substantially longer

run-times, especially when the objective function calculations are not computationally trivial.

More recently, studies in the parallelization of PSO [14, 15] have shown that remarkable computational performance can be realized by completing the optimization effort in a distributed environment. This presents an exciting opportunity for a potential to reduce the relatively lengthy MOPSO execution time by completing the administrative and executive functions in parallel as opposed to serially, which until now has almost always been the case.

The motivation for this work is to combine the work in these two domains, which are quite complimentary. The idea is to use the significant performance improvement available through parallelization of PSO tasks to deliver algorithmic performance identical to serial MOPSO, but achieve similar results in less time through distribution on a grid of processors. This is a topic of very limited research to date.

This chapter consists of three sections, in the first two sections significant contributing forms of PSO to this work, MOPSO and parallel asynchronous particle swarm optimization (PAPSO), are reviewed. In the third a significant contribution of this work, a new multi-objective parallel asynchronous particle swarm optimization (MOPAPSO) algorithm is presented, and its performance evaluated in comparison to the serial MOPSO on several benchmark examples is shown.

4.1 Multi-Objective Particle Swarm Optimization

MOPSO proposes modifying the traditional PSO form described in the previous chapter to use a Pareto [30] ranking scheme in order to handle multi-objective problems. As opposed to traditional PSO which delivers a single solution as the result of an optimization effort, MOPSO aims to deliver a set of solutions comprised of the Pareto dominant particle personal best positions from the swarm.

4.1.1 Pareto Dominance

Consider the general expression for a multi-objective optimization problem:

$$minimize\ \mathbf{F}(\mathbf{x}) = (\mathbf{f}_1(\mathbf{x}), \mathbf{f}_2(\mathbf{x}), ..., \mathbf{f}_n(\mathbf{x})) \tag{4.1}$$

$$\mathbf{x} \in C \tag{4.2}$$

where $\mathbf{x}$ is the candidate selection, C is the set of feasible solutions, and $\mathbf{F}$ represents the set of competing $\mathbf{f}$ contributions for the n optimization objectives.

A candidate solution $\mathbf{x}_1 \in C$ is said to dominate $\mathbf{x}_2 \in C$ (denoted by $\mathbf{x}_1 \prec \mathbf{x}_2$) if the candidate solution $\mathbf{x}_1$ is not worse then $\mathbf{x}_2$ in all objectives and superior to $\mathbf{x}_2$ in at least one objective. *Pareto-optimal* solutions are those which are not dominated by any other solutions, the set of which form the Pareto-optimal set. In the objective function space, this set forms a Pareto-front. Evaluating this Pareto-front at the conclusion of the optimization allows the designer to potentially choose a better candidate set by exposing candidate solutions which may have been overlooked by pre-weighting objective functions.

4.1.2 Implementation

Using PSO as the mechanism to propose candidate solutions to perform multi-objective optimization using MOPSO requires implementing the traditional form of PSO described in Chapter 3 with some modifications. The most significant modifications are to the arguments used to determine the social swarm influences on particle behaviour.

In traditional PSO, the social component contribution to the velocity is proportional to the distance between the particle's current position and the *global best* position, P_{gb} found by all particles in the swarm (the third term in equation (3.4)). All the particles in the swarm tend to converge on a single point in the objective function space corresponding to the best solution to the singularly defined optimization problem.

Here the target is not to identify a single solution, but to find the set of non-dominated solutions for the multi-objective problem. This requires modifications to the particle update routine, using one point based on the fitness of a single function value is no longer appropriate. Instead, the social influence is taken from the set of non-dominated solutions. MOPSO does this by creating a repository of non-dominated solutions, and selecting one for each particle update, as per the following algorithm:

```
% Swarm Initialization;
foreach particle do
    Initialize particle position;
    Initialize particle velocity;
    Set pBest;
end
Initialize repository of non-dominated solutions ;
% Swarm Execution ;
while iteration count less than limit do
    foreach particle do
        Calculate particle velocity;
        Update particle position;
        if particle dominates pBest then
            Update pBest;
        end
    end
    Update repository of non-dominated solutions;
end
```

Algorithm 2: Overview of MOPSO

4.1.2.1 Repository of Non-Dominated Solutions

Key to the execution of the MOPSO routine is the utilization of a geographically-based repository of non-dominated solutions used to serve as the social leader in the particle velocity equation. After the particles have been initialized in the first phase of MOPSO, the set of candidate solutions corresponding to the particles' positions

are investigated and those that represent non-dominated solutions are stored in the repository. At the end of each iteration, the contents of the repository are updated by inserting all non-dominated particle personal best positions and eliminating any newly dominated members.

After population, the repository is sorted geographically, with particles placed in regions defined by their objective function values [30]. By grouping the dominated solutions geographically in the objective functions space, it is possible to promote diversification along the Pareto-front by increasing the probability that the social leader supplied by the repository will be from a less populated region of the front.

4.1.2.2 MOPSO Swarm Execution

Swarm execution in MOPSO is fundamentally very similar to that of the traditional single objective PSO with one exception, the velocity assignment uses a member of the repository as the social leader as opposed to the global best used in PSO. Specifically, the velocity update equation for the ith particle in the kth iteration of MOPSO is:

$$V_i = \omega\, V_i + c_1 rand(n_1)(p_{pb_i} - x_i) + c_2 rand(n_2)(p_{rep_h} - x_i) \tag{4.3}$$

where V_i is the ith particle's velocity, p_{pb_i} is the fittest position found by the ith particle so far, x_i is the ith particle's current position, ω is the inertial constant, c_1 is the cognitive constant - the relative influence of a particle's personal best position on its velocity, c_2 is the social constant - the relative influence of the global best position on its velocity, $rand(n_1)$ and $rand(n2)$ are two random numbers drawn independently

from a uniform distribution, and p_{rep_h} is the position of the hth particle in the repository of non-dominated solutions rep.

The index h is selected from the repository using roulette-wheel selection from those hyper-cubes which contain at least one non-dominated solution, proportional to a fitness value assigned to each:

$$Fit_j = 1/Pop_j \tag{4.4}$$

where Fit_j is the fitness value assigned to each populated hypercube, and Pop_j is the number of non-dominated solutions which populate the jth hypercube. If there is more than one solution in the selected hypercube, the social leader is selected randomly from that population. This has the effect of promoting diverse coverage of the Pareto-front by proposing social leaders from less populated segments of the front more often than not.

4.1.2.3 Advantages of MOPSO

The significant benefit to MOPSO is the elimination of the requirement to create single arbitrary objective functions, by using weighted sums or constraints, when optimizing problems with multiple competitive objectives. These pseudo multi-objective optimization techniques have problems, not the least of which is that significant prior knowledge of the objective functions is required to remove the opportunity to prejudice the results of the optimization effort to the point where the optimized solution is directly dependent on the weights assigned to each objective.

In MOPSO the resulting Pareto-front is completely independent of any weightings of

the objective functions, it is simply a curve in the objective function space representing the best possible solutions for all values of the set of objectives. This effectively moves the design decision to a point completely after the optimization is complete when all information about the relationships between the competing objectives is available as opposed to before as is the case in pseudo-multi-objective with PSO via weighted contributions to a single objective function.

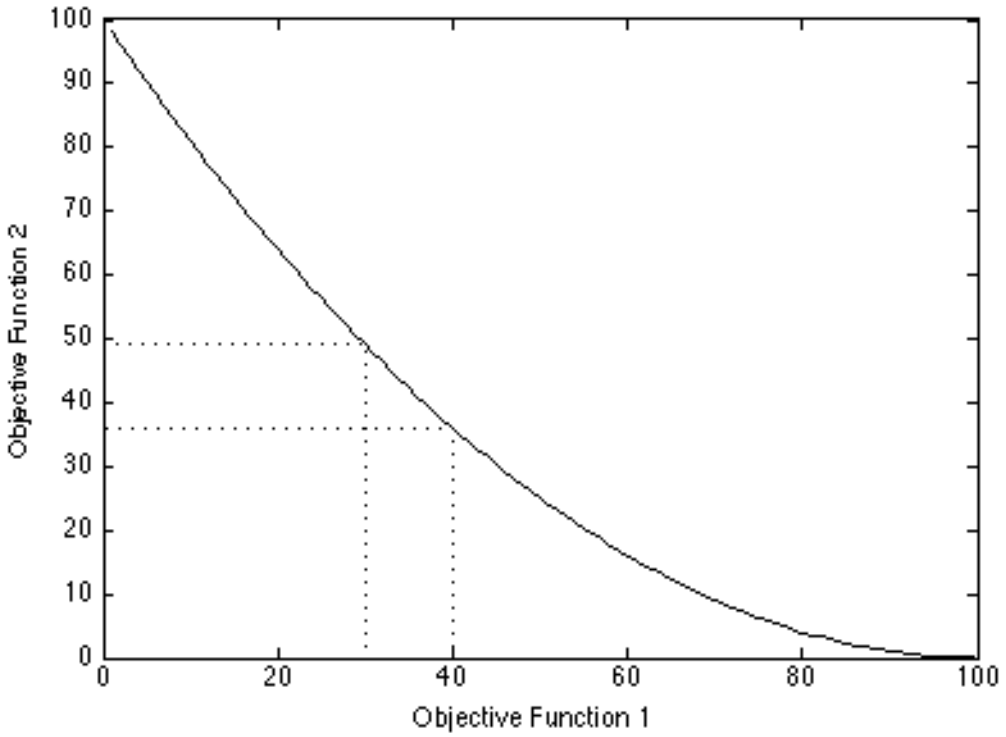

Figure 4.1: Pareto Front Example

Consider the arbitrary Pareto-front shown in Figure 4.1. If the pseudo-multi-objective technique of mandating that the first objective function OF_1 should be no lower then 40, one would be unaware that a 25% reduction in its value would realize a

25% improvment in OF_2, insight that would be available through investigation of the Pareto-front generated by MOPSO.

4.2 Distributed Particle Swarm Optimization

The power of PSO, in particular its superior performance on many non-linear problems compared to deterministic algorithms, comes at the cost of the requirement of an elevated number of objective function calculations compared to traditional methods. If the objective function calculations are not trivial, optimization using evolutionary optimization techniques instead of deterministic techniques may take longer to complete - proportional to the ratio of iterations required when run serially. Fortunately, evolutionary algorithms often afford opportunities to leverage parallel computing techniques to reduce run-times and PSO is no exception. Unlike most deterministic algorithms where the candidate solution necessitating each objective function calculation is rigidly dependent on the immediately prior proposal for convergence, PSO has been shown to be nearly as efficient when particle updates are farmed to nodes on a cluster and run in parallel as it does when they are run serially without parallelizing [14]. In fact, with little affect on the end results reported by the optimization run, parallel asynchronous PSO (PAPSO) leads to runtime improvements [15]:

$$PAPSO_{eff} \sim N_{processors} * \frac{T_{ObjFunCalc,ave}}{T_{Network,ave}} \tag{4.5}$$

where $PAPSO_{eff}$ is the proportional effective runtime improvement relative to serial

PSO, $N_{processors}$ is the number of processors used, $T_{ObjFunCalc,ave}$ is the average objective function calculation time, and $T_{Network,ave}$ is the average network communication time.

4.2.1 Parallel Asynchronous Particle Swarm Optimization

PAPSO achieves runtime improvement by farming out the tasks of objective function calculation and particle updates to nodes in a cluster using a master-slave architecture. In the basic form of PAPSO, a master process initializes the particles, farms out the tasks of objective function calculations and particle updates to the slave processes, and book-keeps the histories of the particles and swarm as required. This requires three significant modifications to the basic PSO algorithm reviewed earlier: division of tasks to be completed by the master processor and slave processors, creation of a queue to farm particle update tasks to the slave processors, and infrastructure to handle the communication between the master and slave processors.

4.2.1.1 Division of Tasks for the Master and Slave Processors

In order to run in a distributed fashion, PAPSO requires the division of tasks into two groups - those completed by the master process and those completed by the slave processors.

Typically this segmentation sees the master processor taking responsibility for the initialization of the swarm, assigning particle update tasks to the slave processors, recording the personal history of each particle, and determining the global best particle position. The slave processors are then tasked with objective function determi-

nation and particle velocity and position updates.

Calculating the objective function and performing particle updates simultaneously on multiple slave processors instead of consecutively on one processor is what allows PAPSO optimizations to be completed significantly faster than traditional PSO with little or no ill effects on the results.

4.2.1.2 Particle Queue

If the population of the swarm exceeds the number of processors then some method to decide which particle is updated by the next available processor is required. In PAPSO, this is typically accomplished using a first-in first-out (FIFO) queue. After initialization, PAPSO assigns particle updates to all available slaves and the rest are placed in a queue. As particle updates are completed, they are added to the back of the queue and the particle at the front of the queue is farmed to the newly available slave processor.

This creates a couple of differences with traditional PSO that are worth noting. First, no explicit effort is expended in ensuring that the set of particles all complete the same number of updates during a PAPSO optimization. Depending on consistency of the objective function calculation times required, it is possible to see that some particles may be updated more than others. Similarly, this mode of asynchronous updating of particles using a FIFO queue removes the opportunity to coordinate the determination of P_{gb} once per iteration - instead this social bookkeeping is usually done before each particle is passed to the slave processor, resulting in a PSO run that instantaneously updates the social leader characteristics used in the velocity

equation for each particle update.

4.2.1.3 Network Communication Infrastructure

The third significant requirement is a capability to distribute information required to perform the objective function calculation and particle update from the master processor, and, likewise, the updated particle information from the slave processors. As there are a number of options available to handle this, it has been included in this list for completeness. The component used in this work will be described in detail in the next section.

4.3 Multi-Objective Parallel Asynchronous Particle Swarm Optimization

The purpose of this work was to develop a general purpose algorithm which combines the most advantageous characteristics of MOPSO and PAPSO, using task distribution to deliver a formal multi-objective handling capability quicker than traditional MOPSO implementations. Efforts were made to segregate model specific code from the MOPSO library in an effort to allow easy application to other multi-objective problems.

In many respects, the underlying core of MOPSO and PAPSO allow for a relatively seamless integration as the modifications applied to the traditional PSO form are quite complimentary. Where the modifications suggested in MOPSO are chiefly restricted to the mechanics of swarm behavior, specifically the source of the social

component to the velocity, and a redefinition of the result to a front of non-dominated solutions; the changes corresponding with a jump to PAPSO are very much applied strictly to the mechanics of the computation itself, and in most respects independent of the swarm behaviour.

The most significant conflict results from the fact that there are no formal iterations in PAPSO and that the timing of algorithmic tasks in MOPSO are usually completed at the end of each iteration. In the case of single objective optimization problems, this fact does not cause much of an issue. Instead of updating the P_{gb} value used in the velocity updates for each particle at the end of each iteration, it is simply updated whenever particle information is received from the swarm (a trivial effort computationally). In MOPSO however, the social leader is drawn from a repository of non-dominated solutions. This effort is usually done once per iteration in traditional MOPSO. Since the computation of the non-dominated solution is not computationally trivial, computing it every time a particle update is received is not a viable option. Instead, an algorithmic parameter representing the frequency that the repository of non-dominated solutions should be evaluated was instituted, and the repository was updated accordingly by one of the slave processors.

Other then the source of the social leader, the velocity and particle update procedure are identical to both traditional PSO and MOPSO routines described earlier with one addition. In order to provide a formal constraint handling capability on the optimization routine, a preservation of feasibility scheme was implemented where particles which leave the feasible space have their velocities recalculated in the slave process until a feasible position is proposed:

```
if particle becomes infeasible then
 |  ParticlePosition = PreviousPosition - ParticleVelocity;
end
if particle remains infeasible then
 |  while particle remains infeasible do
 |   |  ParticleVelocity=SocialContribution + PersonalContribution;
 |  end
end
```

Algorithm 3: Preservation of Feasibility

4.3.1 MOPAPSO Benchmark Tests

The first significant performance metric considered was to evaluate results obtained using MOPAPSO on two general benchmarks problems. The solutions for both have been well established either analytically or via other GA techniques [10, 31]. The objective of these two tests is to confirm the effectiveness of MOPAPSO on two classes of multi-objective optimization problems, one with a continuous convex Pareto-front and another with a discontinuous Pareto-front.

The algebraic objective functions used in this section are not expected to flatter the runtime performance of MOPAPSO (recall equation (4.5)), only to confirm the capability of MOPAPSO to locate the Pareto-front. In Chapter 5 of this work, where MOPAPSO is used in the significantly more computationally expensive task of OBMS, the runtime improvements made possible by MOPAPSO compared with MOPSO will be discussed.

These two studies and those that follow in the next chapter have all been completed on a pair of dual-core duo PC's running XP, the first hosts the master process and two slave process running at 2.4GHz with 4GB of RAM; the second hosts three slave processes running at 2.6GHz with 4 GB of RAM. All tasks are completed in the acslX InterpConsole m-language interpreter v 2.4.1 from AEgis Technologies (http://www.acslx.com) using the MPICH2 message passing interface freely available from Argonne National Labs (http://www.mcs.anl.gov/research/projects/mpich2).

4.3.1.1 Test Function One

The first benchmark example was a simple two-objective problem with two parameters x_1 and x_2 [31]:

$$Min\, f_1(x_1, x_2) = x_1 \tag{4.6}$$

$$Min\, f_2(x_1, x_2) = \frac{g(x_2)}{x_1} \tag{4.7}$$

where $g(x_2)$ is a function of x_2 only and:

$$x_1, x_2 > 0 \tag{4.8}$$

For a fixed value of $g(x_2) = c$, the Pareto-front becomes a hyperbola ($f_1 f_2 = c$). The results of this optimization are shown in Figure 4.2, the solid line represents the analytical solution for the hyperbolic segment (when $c = 1$) in the feasible space and the points represent the non-dominated solutions determined using MOPAPSO with

40 particles, 100 iterations, on two slaves.

4.3.1.2 Test Function Two

The second benchmark test includes a set of objective functions which generate a discontinuous Pareto-front [32]:

$$Min\, f_1(x) = \begin{cases} -x & \text{if } x \leq 1 \\ -2 + x & \text{if } 1 \leq x \leq 3 \\ 4 - x & \text{if } 3 \leq x \leq 4 \\ 4 + x & \text{if } x > 4 \end{cases} \tag{4.9}$$

$$Min\, f_2(x) = (x - 5)^2 \tag{4.10}$$

where:

$$-5 \leq x \leq 10 \tag{4.11}$$

The results obtained from MOPSO [10] for the second test function are shown in Figure 4.3. The results from the same system optimized using MOPAPSO as above are shown in Figure 4.4.

4.3.2 Discussion

The results for both cases show the potential for MOPAPSO to be considered a viable tool for the identification of Pareto-front. Figure 4.2 shows that the MOPAPSO algorithm can clearly identify the Pareto-front. Comparing Figures 4.3 and 4.4 shows

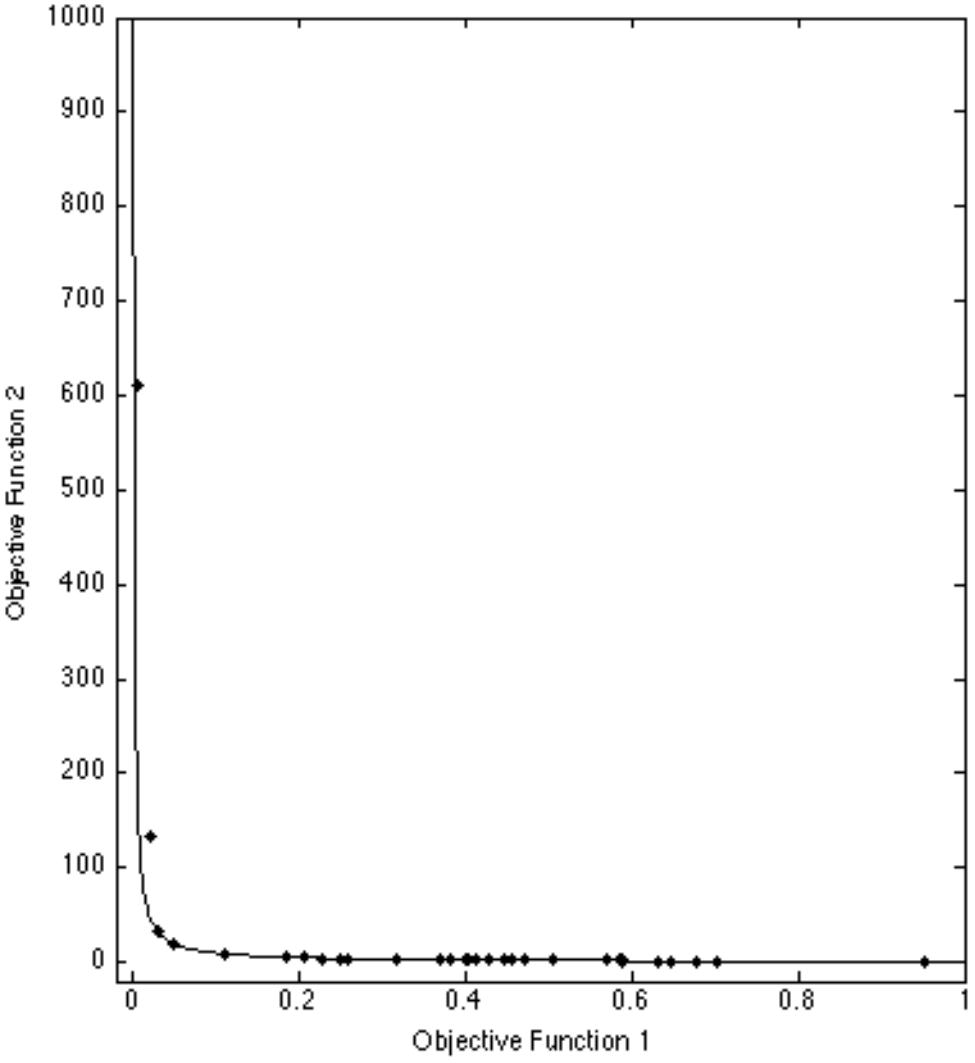

Figure 4.2: Pareto-Front for Test Function One using MOPAPSO

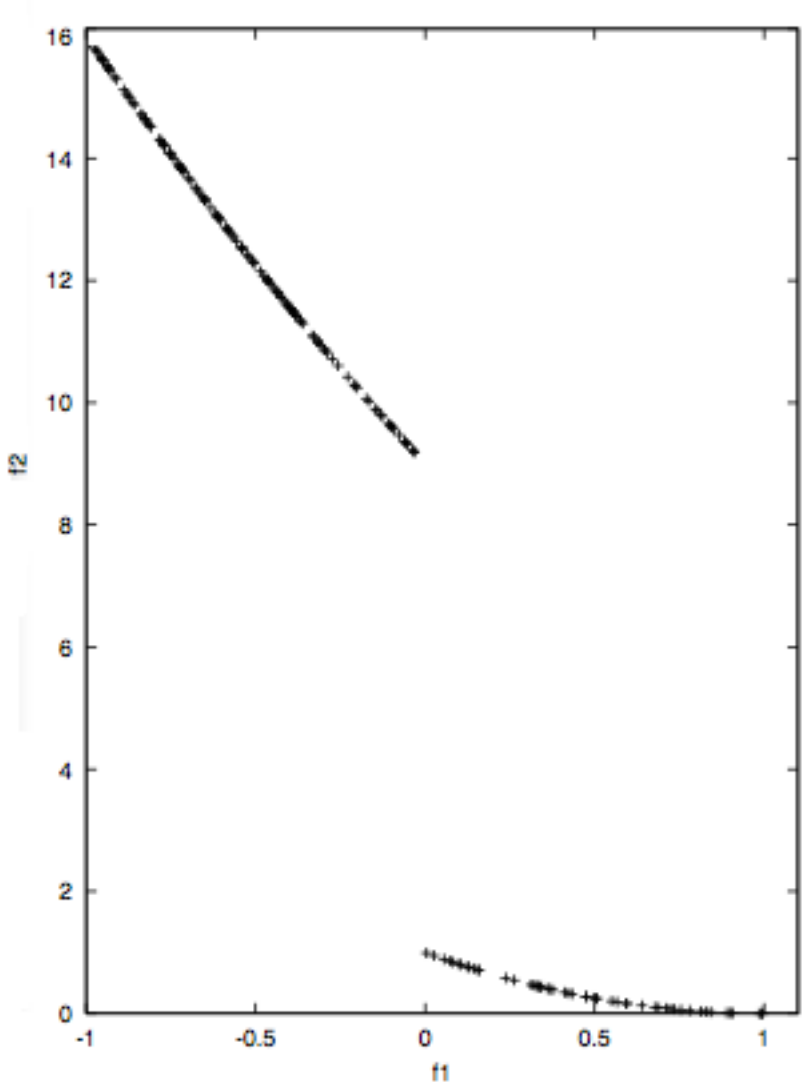

Figure 4.3: Pareto-Front for Test Function Two using MOPSO

that MOPAPSO can generate the same disconnected Pareto-front as that achieved by MOPSO.

Distribution of swarm activities from a master process would seem to have negligible effect on the optimization capabilities of MOPAPSO, and while the increased com-

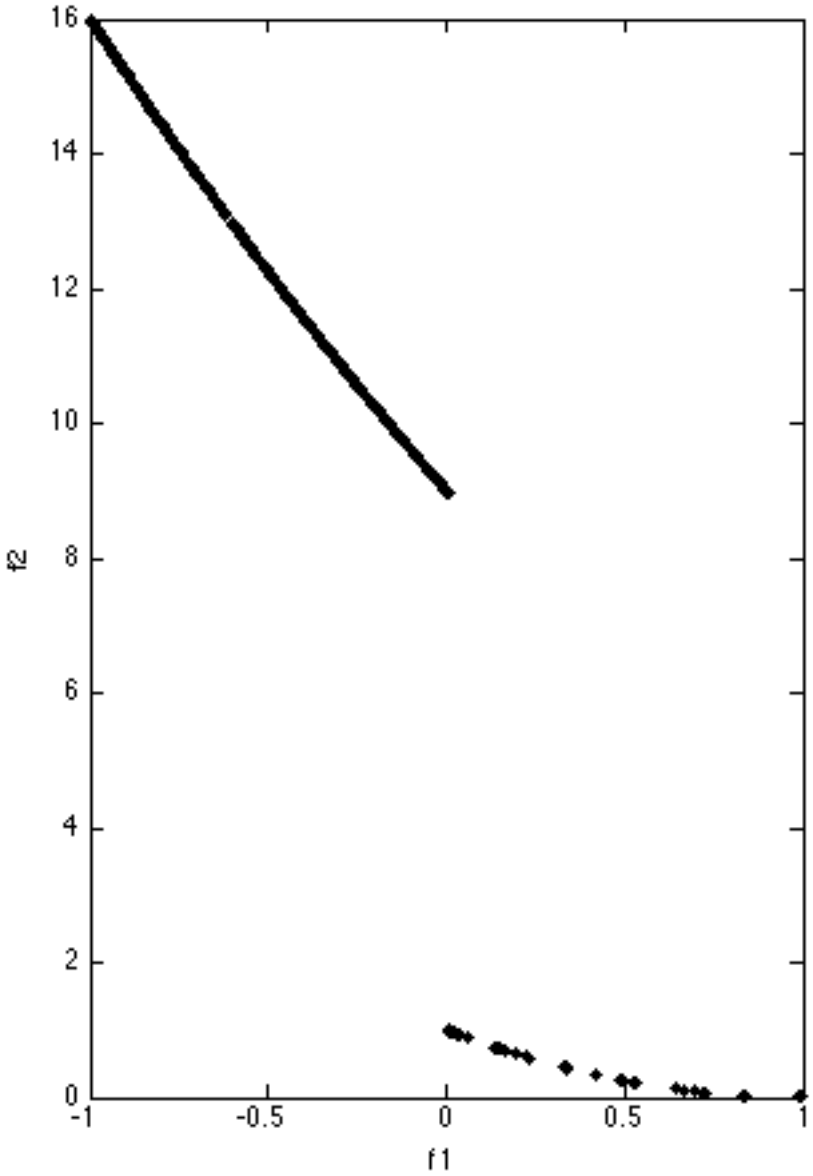

Figure 4.4: Pareto-Front for Test Function Two using MOPAPSO

putational overhead required to handle the distribution is significant in these two cases, it can be expected that significant performance improvements can be realized

when optimizing problems with more substantive objective functions.

Chapter 5

Optimization-Based Mechanism Synthesis Using Multi-Objective Parallel Asynchronous Particle Swarm Optimization

The initial motivation for this exploration of the potential effectiveness of a distributed multi-objective PSO was an application to the computationally intensive task of OBMS. Preliminary investigations [8, 9] of the effectiveness of traditional PSO in OBMS revealed two distinct areas for improvement that have until this point been relatively under-serviced in the literature and are importantly extensible to many other engineering design problems - first, the application of MOPSO to OBMS and, secondly, implementation improvements to reduce the time taken to

complete OBMS using PSO.

5.1 MOPAPSO OBMS Examples

The proceeding chapter introduced MOPAPSO which showed promise in its ability to deliver MOPSO like optimization performance faster by distributing tasks onto multiple processors. On the two benchmark problems used to confirm the outcomes of MOPAPSO versus MOPSO near-identical results were delivered. However, in these benchmarks, the objective functions were all analytical equations, the solution of which, when compared to the computational effort of more complex objective function determinations requiring the execution of system models, was relatively trivial.

In this chapter, the performance of MOPAPSO is examined on the more sophisticated problem of OBMS. First, the capability to effectively synthesize four-bar mechanisms will be evaluated by comparing MOPAPSO's performance with recently published work. Secondly, the performance of MOPAPSO on the synthesis of both four and five-bar mechanisms will be evaluated with special attention paid to the effects of computational distribution on results and runtime.

5.1.1 Four-Bar Grashof Mechanism Synthesis

The first multi-objective OBMS problem considered in this chapter is the four-bar Grashof system design problem defined by Nariman-Zadeh et al. [13] with the competing objectives of following a path as closely as possible and minimizing the devi-

ation from the optimal transmission angle. The first objective function is:

$$Min\, f_{obj_1}(r_1, r_2, ..., n) = \sum_{j=1}^{m}\sum_{k=1}^{n}(x_{jk} - p_{jk})^2 \tag{5.1}$$

where f_{obj_1} represents the deviation from the specified precision points, $(r_1, r_2, ..., n)$ is the set of n design parameters under investigation, m is the number of points specifying the path, n is the number of parameters being searched, x_{ij} is the jth parameter of the point closest on the path to the ith point in the path and p_{ij} is the jth parameter of the ith point in the specified straight-line path:

$$p = (20, 20), (20, 25), (20, 30), (20, 35), (20, 40), (20, 45) \tag{5.2}$$

The competing objective function is:

$$Min\, f_{obj_2}(r_1, r_2..., n) = [(\gamma_{max} - 90\deg)^2 + (\gamma_{min} - 90\deg)^2] \tag{5.3}$$

where f_{obj_2} quantifies the deviation of transmission angle, $(r_1, r_2, ..., n)$ is the set of n design parameters under investigation, and γ_{max} and γ_{min} are the maximum and minimum transmission angles per cycle determined using the following equations [33]:

$$\gamma_{max} = \left[\frac{r_4^2 - (r_1 + r_2)^2 + r_3^2}{2r_4 r_3}\right] \tag{5.4}$$

$$\gamma_{min} = \left[\frac{r_4^2 - (r_1 - r_2)^2 + r_3^2}{2r_4 r_3}\right] \tag{5.5}$$

In addition to enforcement of the Grashof criteria for four-bar mechanisms discussed

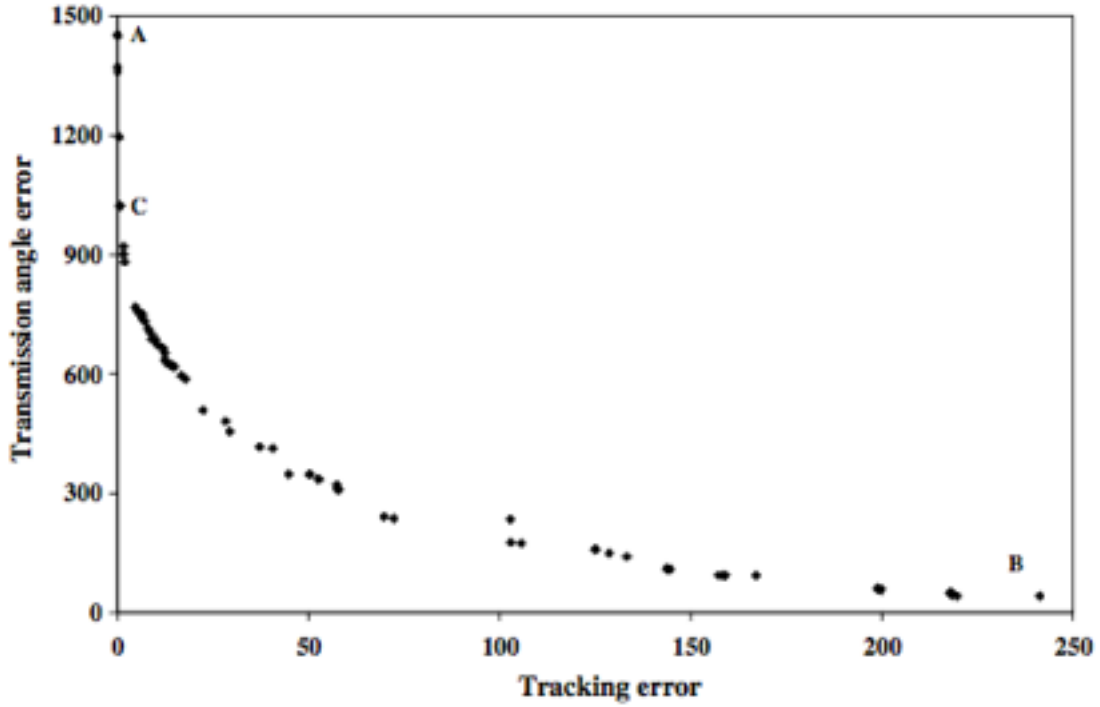

Figure 5.1: Pareto-Front of Tracking Error and Transmission Angle from [13]

in Section 2.1.1, constraints were also applied to the mechanism parameters as well. Link lengths r_1 to r_4 were subject to the following limits $0.0 < r_i \leq 60.0$, while lengths r_5 and r_6 were constrained to -60 $\leq r_i \leq$ 60. Furthermore, $0 \leq \alpha \leq 2\pi$ and (-60,-60) $\leq P(x_2,y_2) \leq$ (60,60) were also prescribed. The results from a prior study [13] using the Niched Genetic Simulated Anealling II optimization algorithm (NGSA-II) and then a deterministic optimization algorithm are shown in Figure 5.1 for reference.

In this study MOPAPSO was used as the optimization technique in OBMS. The algorithmic settings used were a 200 particle swarm, completing 140,000 objective function calculations (equivalent to 700 swarm iterations in traditional PSO) dis-

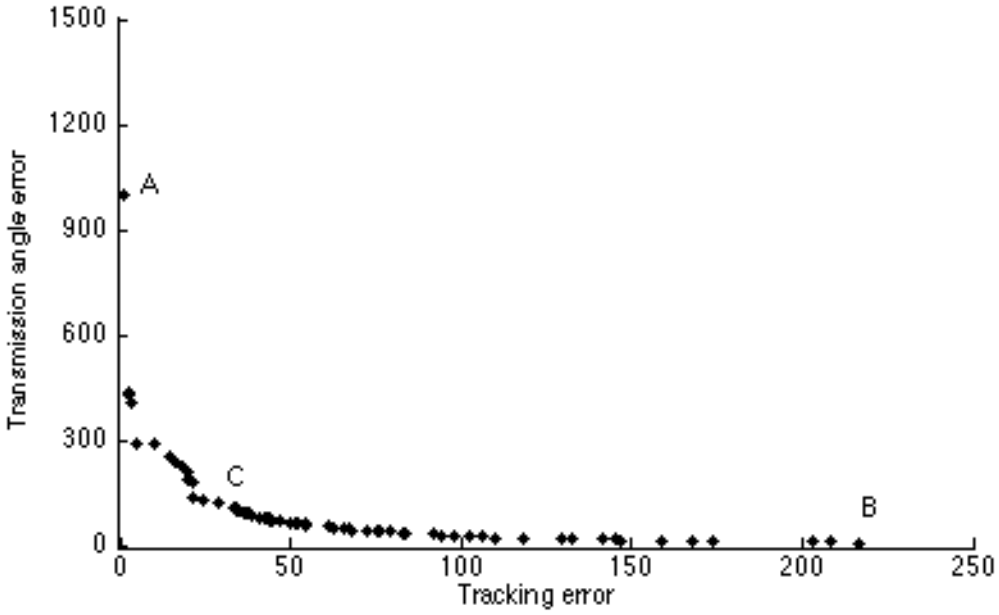

Figure 5.2: Pareto-Front of Tracking Error and Transmission Angle Using MOPAPSO

Table 5.1: Four-Bar Mechanism Optimized Parameters - Point "A"

Parameter	Optimized Value
r_1	12.2650
r_2	9.8261
r_3	13.6490
r_4	15.5613
r_5	18.0580
r_6	-0.4254
α	2.0928
P_x	10.1192
P_y	21.9078
f_{obj_1}	1.1100
f_{obj_2}	1001.7

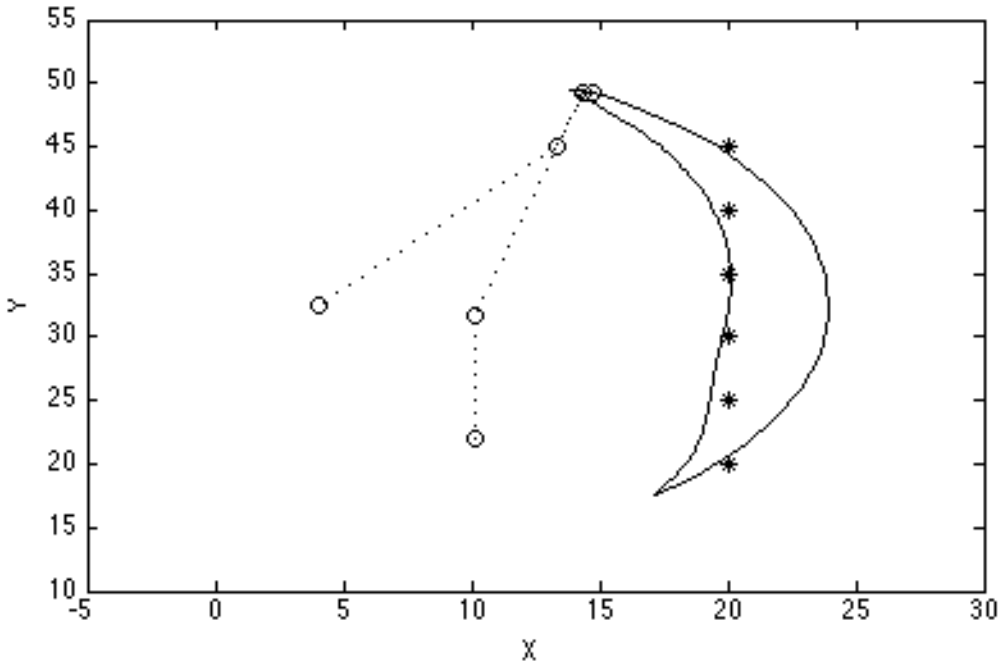

Figure 5.3: Four-Bar Mechanism Path Corresponding to Point"A" in Figure 5.2

tributed on five slave processors. The resulting Pareto-front is shown in Figure 5.2. Comparing Figures 5.1 and 5.2 shows that by the performance of MOPAPSO on this problem was superior to the previous work in exploring nearly all of the front, especially in the area marked "C" since the second solution set Pareto dominates the first. In this central section, where the two competing objectives each exert significant influence on the optimization, MOPAPSO delivered significantly superior results.

The mechanism paths for two extreme points marked "A", where the objective is to get as close as possible to all points regardless of path shape, and "B" where the transmission angle error is optimal are shown in Figures 5.3 and 5.4, respectively.

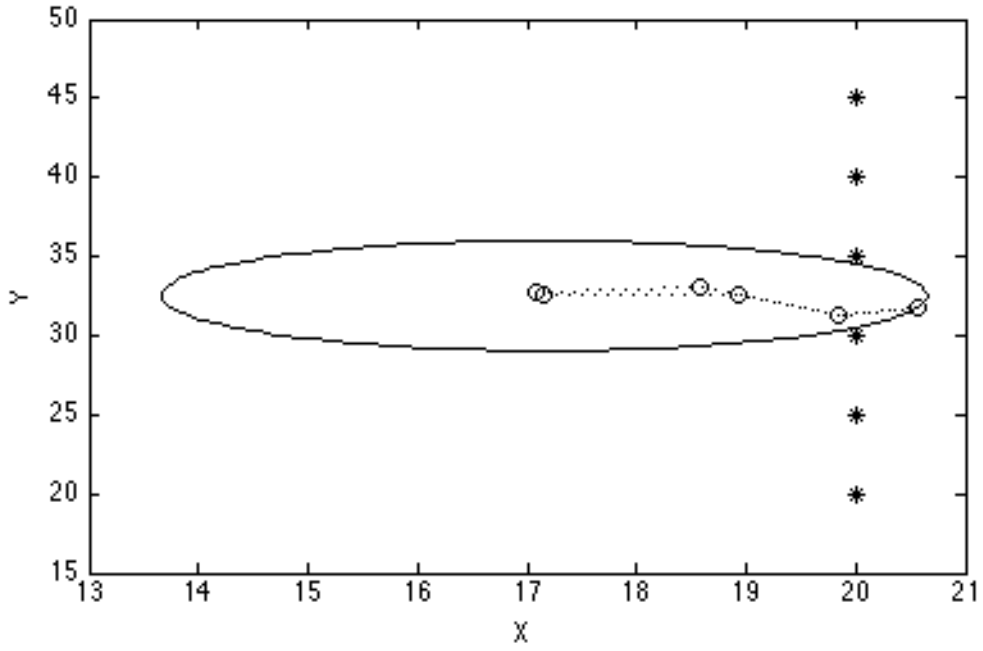

Figure 5.4: Four-Bar Mechanism Path Corresponding to Point "B" in Figure 5.2

Table 5.2: Four-Bar Mechanism Optimized Parameters - Point "B"

Parameter	Optimized Value
r_1	0.1461
r_2	0.6654
r_3	0.5879
r_4	0.5546
r_5	-0.44650
r_6	1.3137
α	0.6992
P_x	20.2603
P_y	31.3350
f_{obj_1}	216.44
f_{obj_2}	9.2992

Note, in the mechanism figures, the dotted line denotes the mechanism link positions. The only notable region where the previous study out-preforms this study is in area "A". This discrepancy is most likely due to the fact that in [13], once the Pareto-front was obtained, these points were then used as the starting point for a deterministic algorithm.

The objective function determination here was significantly more complex than in the previous chapter, as a result the effects of parallelization of tasks on runtime were significant. Three studies each consisting of 250,000 objective function calculations were completed using both one slave processor and five slave processors. The average runtime for the one slave configuration was 182 minutes while the five slave configuration completed the task in an average of 61 minutes. All six studies delivered similar results compared to the previous study, providing improved solutions in regions "B" and "C" and competitive results in "A". Tables 5.1 and 5.2 show the mechanism parameters for points "A" and "B", respectively.

5.1.2 Geared Five-Bar Grashof Mechanism Synthesis

The second applied example of MOPAPSO investigated the OBMS of geared five-bar Grashof mechanisms. The path specified for the mechanism and the tracking error objectives were the same as the first example:

$$Min\, f_{obj_1}(r_1, r_2, ..., n) = \sum_{j=1}^{m} \sum_{k=1}^{n} (x_{jk} - p_{jk})^2 \tag{5.6}$$

where f_{obj_1} representing the deviation from the specified precision points is the objec-

tive function being minimized, $(r_1, r_2, ..., n)$ is the set of n design parameters under investigation, m is the number of points specifying the path, n is the number of parameters being searched, x_{ij} is the jth parameter of the point closest on the path to the ith point in the path and p_{ij} is the jth parameter of the ith point in the specified straight-line path:

$$p = (20, 20), (20, 25), (20, 30), (20, 35), (20, 40), (20, 45) \tag{5.7}$$

Here, instead of minimizing the transmission angle as in the previous example, the second objective function consisted of minimizing the mechanism path length:

$$Min\, f_{obj_2}(r_1, r_2, ..., n) = |PL_{cand} - PL_{min}| \tag{5.8}$$

where PL_{cand} is the length of path of the candidate mechanisms and PL_{min} is the shortest potential path-length for an optimal solution, twice the length of the straight-line segment in p.

In addition to enforcement of the Grashof criteria for five-bar mechanisms discussed in Section 2.2.2, constraints were also applied to the mechanism parameters as well. Link lengths r_1 to r_5 were subject to the following limits $0.0 < r_i \leq 60.0$, while lengths r_5 and r_6 were constrained to -60 $\leq r_i \leq$ 60. Furthermore, $0 \leq \alpha \leq 2\pi$ and (-60,-60) $\leq P(x_2,y_2) \leq$ (60,60) were also prescribed and the Gear Ratio between angles θ_2 and θ_5 was set to 2.

In this study MOPAPSO was used as the optimization technique in OBMS. The algorithmic settings used were a 125 particle swarm, completing 225,000 objective

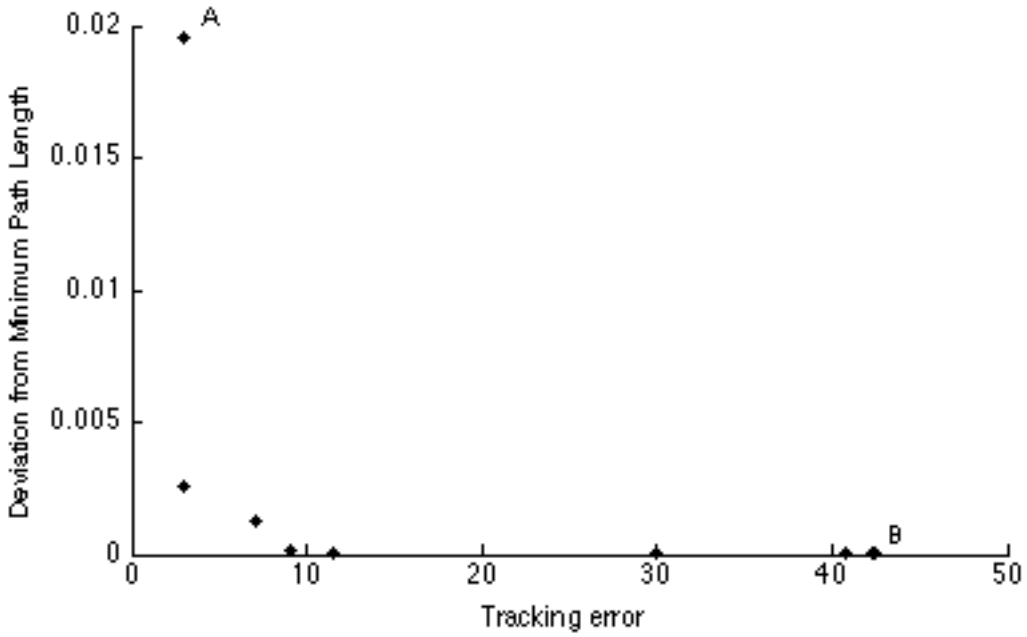

Figure 5.5: Pareto-Front of Tracking Error and Deviation From Minimum Path-Length Using MOPAPSO

function calculations (equivalent to 1,500 swarm iterations in traditional PSO) distributed on five slave processors. The resulting Pareto-front is shown in Figure 5.5. The mechanism paths for the two extreme points marked "A" and "B" in the MOPAPSO study are shown in Figures 5.6 and 5.7, and the corresponding mechanism parameters in Tables 5.3 and 5.4. The path shown in Figure 5.6 illustrates the result of the mechanism configuration which best minimizes path deviation, while Figure 5.7 presents the solution where path-length is optimized.

There has been little or no published investigation of multi-objective OBMS studies on geared five-bar Grashof mechanisms, but the results in "A" compared well to previous work [9] achieved with PSO in OBMS (in a single objective function

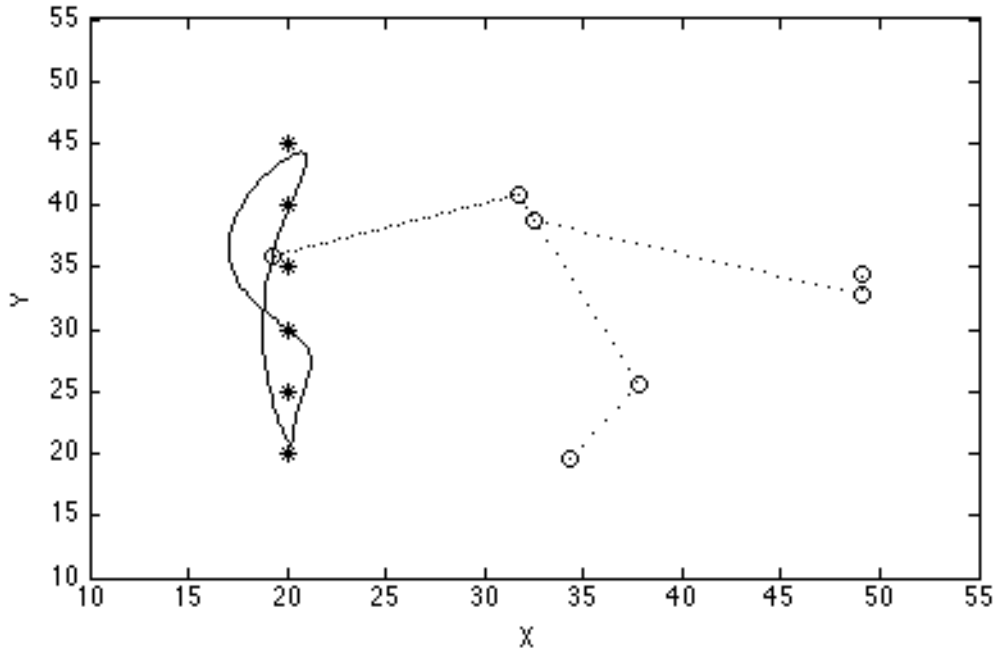

Figure 5.6: Geared Five-Bar Mechanism Path Corresponding to Point "A" in Figure 5.5

scenario).

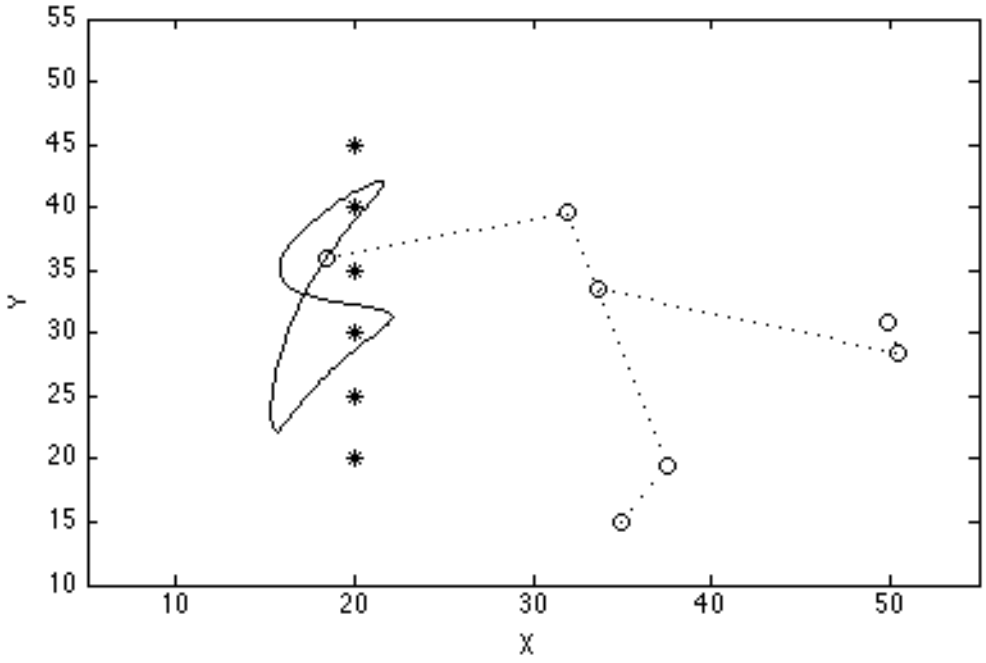

Figure 5.7: Geared Five-Bar Mechanism Path Corresponding to Point "B" in Figure 5.5

Table 5.3: Geared Five-Bar Mechanism Optimized Parameters - Point "A"

Parameter	Optimized Value
r_1	20.9783
r_2	7.0882
r_3	14.2155
r_4	17.6788
r_5	1.6483
r_6	16.3796
r_7	13.4591
α	0.7897
$\theta_{2,ic}$	2.6849
P_x	34.2789
P_y	19.5084
G_r	2
f_{obj_1}	2.9008
f_{obj_2}	0.0196

Table 5.4: Geared Five-Bar Mechanism Optimized Parameters - Point "B"

Parameter	Optimized Value
r_1	21.8338
r_2	5.0426
r_3	14.6429
r_4	17.5720
r_5	2.5399
r_6	21.0185
r_7	14.0715
α	0.8186
$\theta_{2,ic}$	2.8533
P_x	34.9868
P_y	15.0185
G_r	2
f_{obj_1}	42.4781
f_{obj_2}	$5.6954e-07$

Chapter 6

Conclusions and Recommendations for Future Work

6.1 Conclusions

The most significant new contribution of this work is development of an algorithm, MOPAPSO, which exploits the runtime improvement delivered by parallelization of PSO to deliver a formal multi-objective optimization routine which was shown to match a previous segment leader MOPSO in performance with reduced runtime.

While both the inspirational variants which inspired this work, PAPSO and MOPSO, have been well utilized in the literature, investigations of their integration seem absent, and certainly have not been applied to OBMS before.

The results presented in this work show the effectiveness of MOPAPSO at locating Pareto-fronts for objectives of increasing complexity and with varying number of

adjustable parameter set sizes.

In Chapter 4, the capability of MOPAPSO to find Pareto-fronts for test functions from the literature were compared with previously published results for the same functions using the popular MOPSO multi-objective function. The effectiveness of MOPAPSO to identify the Pareto-front in OBMS for four and five-bar Grashof mechanism synthesis was also investigated and compared with available examples from the literature in Chapter 5.

The results show that MOPAPSO was able to match the Pareto-fronts delivered by MOPSO on test-functions from the literature [10] in point value and in the quantity of points on the front.

This indicates that modifications required to allow the objective function calculations to be distributed on a network and calculated in parallel do not seem to adversely affect the results of the optimization algorithm, a fundamental requirement for the new optimization algorithm.

The results for obtaining the Pareto-front for four-bar Grashof mechanism synthesis using MOPAPSO greatly exceeded the point values from the literature [13], though the front itself was much more sparsely populated by MOPAPSO. The discrepancy in front population can be accounted for by the fact that in [13] each point initially found by the multi-objective optimizer was then subject to a post-run single objective optimization with respect to each of the two objective functions. This procedural difference makes the improved values reported by MOPAPSO in the four-bar Grashof OBMS especially impressive. The results for five-bar Grashof mechanism synthesis show that OBMS using MOPAPSO can successfully locate Pareto-fronts

for this configuration as well. Significant runtime-performance was also realized due to parallelization, both four and five-bar OBMS efforts completed around three times quicker with the use of five slave processors.

In problems with a small number of design parameters like the test functions in Chapter 4, the Pareto-front discovered by MOPAPSO using the same number of objective function calculations was nearly identical to that delivered by MOPSO, and contained the absolute Pareto optimum positions for each objective function pair. In more highly parameterized systems, finding the absolute Pareto-optimum solution set in a fixed number of iterations becomes less likely.

As a result, the optimal role of MOPAPSO in OBMS is likely in a binary hybrid, where MOPAPSO is used to select the starting position for traditional deterministic algorithms based on its location on the Pareto-front. This effectively balances the benefits of removing the effects of pre-weighting competing objectives from the optimization altogether while delivering the precision accorded by using deterministic optimization algorithms.

6.2 MOPAPSO m-language Toolkit

An implementation of a general purpose m-language MOPAPSO toolkit was presented. The toolkit was developed on acslX v2.4.1 m-language InterpConsole (http://www.acslX.com) and compiled MPICH mfx functions using the MPICH2 library from Argonne National Labs (http://www.mcs.anl.gov/research/projects/mpich2). The library was constructed

with ease of extensibility in mind with respect to algorithmic portability and system specification.

The MOPAPSO toolkit consists of two sets of m-files, one for the master process and one for the slave processors. The code presented here was used for OBMS using MOPAPSO, but with straightforward modifications requiring minimal effort this toolkit can easily be applied to other applications.

The toolkit can be easily extend to run on the Matlab m-language interpreter (http://mathworks.com) with little additional effort. The code for the toolkit can be found in Appendix A.

6.3 Recommendations for Future Work

The repository bookkeeping tasks, including tracking the dominant solutions, sorting the repository solutions into geographical segments and choosing which dominant solution to use as the social leader, is currently in the main master process creating a bottleneck. Some research into the consequences of farming these tasks out to a slave process could result in substantial reduction in the overhead associated with support of a distributed multi-objective capability.

The current m-code implementation of MOPAPSO requires execution through an m-language interpreter which allows easy application to models composed in a higher level programming language. However, the porting of this code into a compiled library that can be called from the same interpreter could substantially improve runtime performance.

Finally, the application of MOPAPSO to other problems, in any design and control domains warrants significant consideration. In complex problems from these domains, with non-linear objective function surfaces and complex objective function calculations, MOPAPSO may prove to be a valuable member in any designers arsenal of optimization tools.

References

[1] Sandor, G., and Erdman, A., 1984. *Advanced Mechanism Design: Analysis and Synthesis - Volume 2.* Toronto Prentice Hall, Toronto, ON.

[2] Fogel, D., 1994. "An introduction to simulated evolutionary optimization". *Neural Networks, IEEE Transactions on,* **5**(1), Jan, pp. 3–14.

[3] Eberhart, R. C., and Yuhui, S., 2001. "Particle swarm optimization: developments, applications and resources". In Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2001).

[4] Hu, X., Eberhart, R., and Shi, Y., 2003. Engineering optimization with particle swarm.

[5] Fukuyama, Y., Takayama, S., Nakanishi, Y., and Yoshida, H., 1999. "A particle swarm optimization for reactive power and voltage control in electric power systems". In Proc. of the Genetic and Evolutionary Computation Conf. GECCO-99, W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, eds., Morgan Kaufmann, pp. 1523–1528.

[6] Cabrera, J., Simon, A., and Prado, M., 2002. "Optimal synthesis of mechanisms

with genetic algorithms". *Mechanism and Machine Theory,* **37**(10), October, pp. 1165–1177.

[7] Smaili, A. A., Diab, N. A., and Atallah, N. A., 2005. "Optimum synthesis of mechanisms using tabu-gradient search algorithm". *Journal of Mechanical Design,* **127**(5), September, pp. 917–923.

[8] McDougall, R., and Nokleby, S., 2008. "Synthesis of Grashof four-bar mechanisms using particle swarm optimization". In Proceedings of the 2008 International Design Engineering Technical Conferences and & Computers and Information in Engineering Conference.

[9] McDougall, R., and Nokleby, S., 2008. "Synthesis of Grashof five-bar mechanisms using particle swarm optimization". In Proceedings of the Canadian Society for Mechanical Engineers Forum 2008.

[10] Coello, C., and Lechunga, M., 2002. MOPSO: A proposal for multiple objective particle swarm optimization. Computational Intelligence, Hawaii, May 12-17, 2002. IEEE Press.

[11] U. Baumgartner, C. M., and Renhart, W., 2004. "Pareto optimality and particle swarm optimization". *IEEE Transactions on Magnetics,* **40**, March, pp. 1172–1175.

[12] Francesco di Pierro, S.-T. K., and Savic, D., 2007. "An investigation on preference order ranking scheme for multiobjective optimization". In IEEE: Transactions on Evolutionary Computation, Vol. 11.

[13] Nariman-Zadeh, N., Felezi, M., Jamali, A., and Ganji, M., 2008. "Pareto optimal synthesis of four-bar mechanisms". *Mechanism and Machine Theory.*

[14] Schutte, J., Reinbolt, J., Fregly, B., Haftka, R., and George, A., 2004. "Parallel global optimization with particle swarm algorithm". *International Journal for Numerical Methods in Engineering,* **61**, pp. 465–474.

[15] Koh, B., George, A., Haftka, R., and Fregly, B., 2006. "Parallel asycnhronous particle swarm optimization". *International Journal for Numerical Methods in Engineering,* **67**, pp. 578–595.

[16] Sandor, G., and Erdman, A., 1991. *Advanced Mechanism Design: Analysis and Synthesis - Volume 1, Second Edition.* Toronto Prentice Hall, Toronto, ON.

[17] Nokleby, S. B., 1999. "Optimization-Based Mechanism Synthesis". MASc Thesis, University of Victoria, Victoria, BC.

[18] Ting, K., 1994. "Mobility criteria of geared five-bar linkages". *Mechanism and Machine Theory,* **29**(2), pp. 251–254.

[19] Fletcher, R., 1987. *Practical Methods of Optimization.* John Wiley & Sons, Toronto, ON.

[20] Nokleby, S., and Podhorodeski, R., 2000. "Optimization-based synthesis of a deep-digging tillage mechanism". *Transactions of the CSME,* **24**(1A), pp. 61–78.

[21] Eberhart, R., and Kennedy, J., 1995. "New optimizer using particle swarm theory". In Proceedings of the International Symposium on Micromechatronics Human Science, pp. 39–43.

[22] Soudan, B., and Saad, M., 2008. "An evolutionary dynamic population size pso implementation". *Information and Communication Technologies: From Theory to Applications, 2008. ICTTA 2008. 3rd International Conference on*, April, pp. 1–5.

[23] Richards, M., and Ventura, D., 2004. "Choosing a starting configuration for particle swarm optimization". *Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on,* **3**, July, pp. 2309–2312.

[24] Engelbrecht, A. P., 2006. *Fundamentals of Computational Swarm Intelligence*. John Wiley & Sons.

[25] Podhorodeski, R., and Fang, X., 1996. "Optimization-based Grashof-mechanism synthesis via sub-type specific parameter transforms". In Proceedings of the 1996 ASME Design Engineering technical Conferences and Computers in Engineering Conference, pp. 1–10.

[26] Fang, X., and Podhorodeski, R., 1996. "Synthesis of a deep-digging tillage mechanism". In Proceedings 15th Canadian Congress of Applied Mechanics, pp. 248–249.

[27] Nokleby, S. B., and Podhorodeski, R. P., 2001. "Optimization-based synthesis of Grashof geared five-bar mechanisms". *Journal of Mechanical Design,* **123**(4), December, pp. 529–534.

[28] Mostaghim, S., Hoffmann, M., Konig, P., Frauenheim, T., and Teich, J., 2004. "Molecular force field parametrization using multi-objective evolutionary algo-

rithms". *Evolutionary Computation, 2004. CEC2004. Congress on,* **1**, June, pp. 212–219 Vol.1.

[29] Reddy, M. J., and Kumar, D. N., 2007. "Multi-objective particle swarm optimization for generating optimal trade-offs in reservoir operation". *Huydrological Processes,* **21**, pp. 2897 – 2909.

[30] Knowles, J. D., and Corne, D. W., 2000. "Approximating the nondominated front using the pareto archived evolution strategy". *Evolutionary Computation,* **8**, pp. 149–172.

[31] Deb, K., 1999. Multi-objective genetic algorithms: Problem difficulties and construction.

[32] Schaffer, J. D., 1984. "Multiple Objective Optimization with Vector Evaluated Genetic Algorithms". PhD Thesis, Vanderbilt University.

[33] Waldron, K. J., and Kinzel, G. L., 1999. *Kinematics, dynamics, and design of machinery.*

Appendix A

MOPAPSO Toolkit

A.1 Master Processor Code

This section includes the functions which are run on the master processor.

A.1.1 mopapsomaster.m

```
%-----------------------------------------------
%  papso_master.m
%
%  PAPSO m-language (acslX) Toolkit
%
%    Developed by:  Robin McDougall
%    University of Ontario Institute of Technology
%    Last Modified: 10/20/2008
```

```
%
%-------------------------------------------------
z=clock
z=z(6)
for i=[1:z]
rand()
end



%-- Algorithmic Settings
NumberOfSlaves = 5;
NumberOfParameters = 12;
NumberOfParticles = 125;
NumberOfIterations = 6000;
startstats=[NumberOfSlaves NumberOfParameters NumberOfParticles NumberOfIterations cl
save startstats @format=ascii @file=startstats.dat

for i=[1:1:NumberOfSlaves]
mpiSend(NumberOfParameters,i,0,MPI_COMM_WORLD);
mpiSend(NumberOfParticles,i,0,MPI_COMM_WORLD);
end
%-- Statistical Variables
Nodes=[];
```

```
%-- Model Characteristics
%ParmStarting =[1 1 ; 0 0];
%MaxVel = ones(NumberOfParticles,1)*.5;
ParmStarting =[60.0 60.0 60.0 60.0 60.0  60.0 60.0 2*pi() 2*pi() 60 60 2 ;  0.1 0.1 0
MaxVel = zeros(NumberOfParameters);
%-- Initialize the Swarm
PPOS=zeros(NumberOfParticles,NumberOfParameters)
PVEL=zeros(NumberOfParticles,NumberOfParameters);
FPPOS1=5.55e33*ones(NumberOfParticles,1);
FPPOS2=5.55e33*ones(NumberOfParticles,1);
PBEST=5.55e33*PPOS;
EverFeasible=zeros(NumberOfParticles,1);

FPBEST1=5.55e33*ones(NumberOfParticles,1);
FPBEST2=5.55e33*ones(NumberOfParticles,1);
for i=[1:1:NumberOfParticles]
for j=[1:1:NumberOfParameters]
PPOS(i,j)=ParmStarting(2,j)+rand()*(ParmStarting(1,j)-ParmStarting(2,j));
PVEL(i,j)= 0.0;
end
end
for i=[1:1:NumberOfParticles]
[FPPOS1(i),FPPOS2(i)]=CalcObjFun(PPOS(i,:));
```

```
end

%-- Initialize Particle Queue
PQueue=ones(1,NumberOfParticles);
for i=[2:1:NumberOfParticles]
PQueue(i)=PQueue(i-1)+1;
QueueSize=NumberOfParticles;
end
Rep=[FPPOS1 FPPOS2 PPOS]
  % F1,F2,P1,P2....PN
Dominators=FindDom(Rep(:,1),Rep(:,2));
%-- Run Swarm
for i=[1:1:NumberOfParticles*NumberOfIterations]
if(i<=NumberOfSlaves)
Node=i;
end
if(mod(i,NumberOfParticles)==1 | i==1)
i
Rep=[Rep; FPBEST1 FPBEST2 PBEST];
Dominators=FindDom(Rep(:,1),Rep(:,2));
DomVol=size(Dominators);
DomVol=DomVol(1)
UpdatedRep=[];UpdatedDoms=[];
```

```
for iii=[1:DomVol]
UpdatedRep=[UpdatedRep;Rep(Dominators(iii),:)];
UpdatedDoms=[UpdatedDoms iii];
end
sum(EverFeasible)
Rep=UpdatedRep;
Dominators=UpdatedDoms;
[Bins,BinPops]=UpdateBins(Rep(:,1),Rep(:,2),Dominators);
end

% Dom=ceil(rand()*Dom);
if(i>NumberOfSlaves)
FromSlave=mpiRecv(zeros(1,2*NumberOfParameters+5),MPI_ANY_SOURCE,MPI_ANY_TAG, MPI_COM
Particle=FromSlave(2*NumberOfParameters+4);
PPOS(Particle,:)=FromSlave(1:NumberOfParameters);
PVEL(Particle,:)=FromSlave(NumberOfParameters+1:2*NumberOfParameters);
FPPOS1(Particle)=FromSlave(2*NumberOfParameters+1);
FPPOS2(Particle)=FromSlave(2*NumberOfParameters+2);
Node=FromSlave(2*NumberOfParameters+3);
EverFeasible(Particle)=FromSlave(2*NumberOfParameters+5);
PQueue(QueueSize+1)=Particle;
QueueSize=QueueSize+1;
if(FPPOS1(Particle)<=FPBEST1(Particle) & FPPOS2(Particle)<=FPBEST2(Particle) & FPPOS1
```

```
FPBEST1(Particle)=FPPOS1(Particle);
FPBEST2(Particle)=FPPOS2(Particle);
PBEST(Particle,:)=PPOS(Particle,:);
end
end
if(i<=NumberOfParticles*NumberOfIterations)
Particle=PQueue(1);
PQueue=PQueue(2:QueueSize);
QueueSize=QueueSize-1;
ToSlave=[PPOS(Particle,:),PVEL(Particle,:),Rep(ChooseDom(Bins,BinPops,Dominators),3:N
mpiSend(ToSlave,Node,0,MPI_COMM_WORLD);
end
Nodes=[Nodes Node];

if(mod(i,5000)==1 | i==1)
save Rep @file=rep.dat @format=ascii
end



end
Rep
save Rep @file=rep.dat @format=ascii
save Nodes @file=nodes.dat @format=ascii
```

```
startstats=[NumberOfSlaves NumberOfParameters NumberOfParticles NumberOfIterations cl
save startstats @format=ascii @file=endstats.dat
```

A.1.2 FindDom.m

```
%-------------------------------------------------
%  FindDom.m
%
%  MOPAPSO m-language (acslX) Toolkit
%
%    Developed by:  Robin McDougall
%    University of Ontario Institute of Technology
%    Last Modified: 10/20/2008
%
%-------------------------------------------------

function ret=FindDom(FPPOS1,FPPOS2)
zs=size(FPPOS1);
zs=zs(1);
isdominant=ones(zs,1);
for i=[1:1:zs]
```

```
 for j=[1:1:zs]
  if(i~=j)
if(FPPOS1(i)<=FPPOS1(j))
if(FPPOS2(i)<=FPPOS2(j))
isdominant(j)=0;
end
elseif (FPPOS2(i)<=FPPOS2(j))
isdominant(i)=isdominant(i)*1;
else
isdominant(i)=0;
end
 end
 end
end
ret=[];
for k=[1:1:zs]
if(isdominant(k))
ret=[ret; k];
end
end
if(sum(size(ret))==0)
ret=1;
end
```

```
end
```

A.1.3 ChooseDom.m

```
%-------------------------------------------------
%  ChooseDom.m
%
%  MOPAPSO m-language (acslX) Toolkit
%
%    Developed by:  Robin McDougall
%    University of Ontario Institute of Technology
%    Last Modified: 10/20/2008
%
%-------------------------------------------------
function [Dom]=ChooseDom(Bins,BinsPop,Doms)
NumberOfDoms=size(Doms);
NumberOfDoms=NumberOfDoms(1);
NumberOfBins=size(BinsPop);
NumberOfBins=NumberOfBins(1);
BinDrawData=[];Multiples=0;
for i=[1:NumberOfBins]
```

```
if(BinsPop(i)~=0)
if(BinsPop(i)==1)
BinDrawData=[BinDrawData; i 1.];
else
BinDrawData=[BinDrawData; i 1/(BinsPop(i)*10)];
Multiples=Multiples+1;
end
end
end
TotalOdds=sum(BinDrawData(:,2));
NumberOfHorses=size(BinDrawData(:,1));NumberOfHorses=NumberOfHorses(1);
Delta=TotalOdds/NumberOfHorses;
Winner=rand();
i=0;nowinner=1;sofar=0;
while(nowinner)
i=i+1;
if(BinDrawData(i,2)/TotalOdds+sofar>=Winner)
nowinner=0;
WinningBin=BinDrawData(i,1);
end
sofar=sofar+BinDrawData(i,2)/TotalOdds;
end
Winners=[];
```

```
for i=[1:NumberOfDoms]
if(WinningBin==Bins(i))
Winners=[Winners; Doms(i)];
end
end
WinningBinPop=size(Winners);WinningBinPop=WinningBinPop(1);
if(WinningBinPop==1)
Dom=Winners;
else
i=0;nowinner=1;Winner=rand();
while(nowinner)
i=i+1;
if((i/WinningBinPop)>=Winner)
nowinner=0;
Dom=Winners(i);
end
end
```

```
%notfound=1;i=1;
%while(notfound)
% if(Bins(i)==WinningBin)
```

```
% Dom=Doms(i);
% notfound=0;
% end
% i=i+1;
%end
end
```

A.2 Slave Processor Code

This section includes the functions which are run on the slave processors.

A.2.1 mopapsoslave.m

```
%-----------------------------------------------
%  MOPAPSOSlave.m
%
%  MOPAPSO m-language (acslX) Toolkit
%
%    Developed by:  Robin McDougall
%    University of Ontario Institute of Technology
%    Last Modified: 10/20/2008
%
%-----------------------------------------------
```

```
NumberOfParameters=mpiRecv([0],0,MPI_ANY_TAG,MPI_COMM_WORLD)
NumberOfParticles=mpiRecv([0],0,MPI_ANY_TAG,MPI_COMM_WORLD)
for i=[1:33]
rand();
end
for i=[1:1:9999999]
FromMaster=mpiRecv(zeros(1,4*NumberOfParameters+3),0,MPI_ANY_TAG,MPI_COMM_WORLD);
PPOS=FromMaster(1:NumberOfParameters);
PVEL=FromMaster(NumberOfParameters+1:2*NumberOfParameters);
GBEST=FromMaster(2*NumberOfParameters+1:3*NumberOfParameters);
PBEST=FromMaster(3*NumberOfParameters+1:4*NumberOfParameters);
Node=FromMaster(4*NumberOfParameters+1);
Particle=FromMaster(4*NumberOfParameters+2);
EverFeasible=FromMaster(4*NumberOfParameters+3);
[PPOS,PVEL]=UpdateParticle(PPOS,PVEL,PBEST,GBEST,NumberOfParameters,EverFeasible) ;
[FPPOS1,FPPOS2]=CalcObjFun(PPOS);
ToMaster=[PPOS PVEL FPPOS1 FPPOS2 Node Particle EverFeasible];
mpiSend(ToMaster,0,0,MPI_COMM_WORLD);
end
```

```
\subsection{UpdateParticle.m}
```

```
\begin{verbatim}
%-------------------------------------------------
%  UpdateParticle.m
%
%  MOPAPSO m-language (acslX) Toolkit
%
%    Developed by:  Robin McDougall
%    University of Ontario Institute of Technology
%    Last Modified: 10/20/2008
%
%-------------------------------------------------
function [a,b]=UpdateParticle(PPOS,PVEL,PBEST,GBEST,NumberOfParameters,everfeasible)
check=0;iter=1;PPOSin=PPOS;
while(check==0),
w=(0.5 + rand()/2);c1=sqrt(2);c2=sqrt(2);maxvel=5;minvel=-5;
for j = [1:1:NumberOfParameters]
PVEL(j)=max(minvel,min(maxvel,w*PVEL(j)+c1*rand()*(PBEST(j)-PPOS(j))+c2*rand()*(GBEST
PPOS(j)=PPOS(j)+PVEL(j);
end

if(everfeasible==1 & iter<10001)
check=IsFeasible(PPOS);
if(check==0)
```

```
PPOS=PPOSin-PVEL;

PVEL=-PVEL;

if(IsFeasible(PPOS)==1)

check=1;

else

PPOS=PPOSin;

PVEL=0;

end

end

else

check=1;

end

  iter=iter+1;

end

b=PVEL;

a=PPOS;


end
```

A.2.2 CalcObjFun.m

```
%-------------------------------------------------

%  CalcObjFun.m
```

```
%
%  MOPAPSO m-language (acslX) Toolkit
%
%    Developed by:  Robin McDougall
%    University of Ontario Institute of Technology
%    Last Modified: 10/20/2008
%
%-------------------------------------------------
function [f1,f2]=CalcObjFun(PPOS)
FP=IsFeasible(PPOS);
[f1,f2]=ObjFun(PPOS,FP);
```

A.2.3 ObjFun.m

```
%-------------------------------------------------
%  ObjFun.m
%
%  MOPAPSO m-language (acslX) Toolkit
%
%    Developed by:  Robin McDougall
%    University of Ontario Institute of Technology
%    Last Modified: 9/20/2008
%
%-------------------------------------------------
```

```
function [f1,f2,path]=ObjFun(r,FP)

NumberOfParameters=9;

%--nairman paper
px_abs=[20 20 20 20 20 20];
py_abs=[20 25 30 35 40 45];



if(FP==0)
f1=10^100;
f2=f1;
else
r1=r(1);r2=r(2);r3=r(3);r4=r(4);r5=r(5);r6=r(6);alpha=r(7);x2=r(8);y2=r(9);
path=[];
for theta2=[0:0.01*pi:2*pi]
A1=r2*cos(theta2)-r1*cos(alpha);
A2=r2*sin(theta2)-r1*sin(alpha);
A3=-(A1^2+A2^2+r3^2-r4^2)/(2*r3);
if(r6<=0)
theta3 = atan2(A2,A1)+atan2(sqrt(A1^2+A2^2-A3^2),A3);
else
theta3 = atan2(A2,A1)-atan2(sqrt(A1^2+A2^2-A3^2),A3);
```

```
end

Px=x2+r2*cos(theta2)+r5*cos(theta3)+r6*cos(theta3+3.1415/2);
Py=y2+r2*sin(theta2)+r5*sin(theta3)+r6*sin(theta3+3.1415/2);

path=[path; Px Py];
end

f1=0;
f2=0;

numberofpoints=size(px_abs);
numberofpoints=numberofpoints(2);
min_contribution=10000*ones(numberofpoints,1);

for i=[1:1:200]
for j=[1:1:numberofpoints]
contribution=0;
contribution=(px_abs(j)-path(i,1))^2;
contribution=contribution+(py_abs(j)-path(i,2))^2;
if(min_contribution(j)>contribution)
min_contribution(j)=contribution;
end
```

```
end
end
f1=sum(min_contribution);
gammamax=acos((r4^2-(r1+r2)^2+r3^2)/(2*r4*r3));
gammamin=acos((r4^2-(r1-r2)^2+r3^2)/(2*r4*r3));
f2a=(gammamax*180/pi()-90)^2+(gammamin*180/pi()-90)^2;
f2=r1^2+r2^2+r3^2+r4^2+r5^2+r6^2;
end
end
```

A.2.4 IsFeasible.m

```
%-------------------------------------------------
%  IsFeasible.m
%
%  MOPAPSO m-language (acslX) Toolkit
%
%    Developed by:  Robin McDougall
%    University of Ontario Institute of Technology
%    Last Modified: 10/20/2008
%
%-------------------------------------------------
function ret=IsFeasible(PPOS)
r=PPOS;
```

```
r_up=[60.0 60.0 60.0 60.0 60.0  60.0 2*pi() 60 60 ];
r_low=[0.1 0.1 0.1 0.1 -60 -60.0 0.0 -60 -60];


ret=1;
links=r(1:4);
links=sort(links);

if((links(1)+links(4)>links(2)+links(3)) | (abs(links(1)-r(3))<1e-3) | (abs(links(1)-
ret=0;
end

for i=[1:1:9]
if((r(i)>r_up(i)) | r(i)<r_low(i))
ret=0;
end
end


end
```