2008

Pheromone particle swarm optimization of stochastic systems

Paul Allan Wilhelm
Iowa State University

Pheromone particle swarm optimization of stochastic systems

by

Paul Allan Wilhelm

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Industrial Engineering

Program of Study Committee:
Douglas Gemmill, Major Professor
Sigurdur Olafsson
Eliot Winer

Iowa State University

Ames, Iowa

2008

ACKNOWLEDGEMETS

This is dedicated to my family and friends, who supported me throughout my education.

I'd like to thank Dr. Douglas Gemmill for his invaluable comments and suggestions during the writing of this paper and Sarah and Jeremy for their help editing this paper.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Pheromone particle swarm optimization (PSO) of stochastic systems tests the impact of adjustments to algorithm parameters on algorithm performance when searching for optimal solutions to stochastic simulations. To test the benefit of adjusting PSO, the tuned algorithm is compared to the results from the commercial optimization software, OptQuest. In addition, two modifications to pheromone PSO are proposed. These include utilizing orthogonal arrays as an initial position for the algorithm and biasing the release of pheromones in the first iteration based on the relative strength of the objective function. These modifications are shown to improve the average objective functions found as well as the time to convergence in the optimization of some problem types. This paper also highlights the applicability of using pheromone PSO to optimize stochastic simulations compared to commercial optimization software.

CHAPTER 1. OVERVIEW

Simulations of complex systems are continually being developed by industry as the rise in computing power and software makes it cost effective to do so. A current challenge is finding new and better ways to optimize systems via simulation. There are heuristic optimization methods that utilize tabu search, scatter search, mixed integer programming and neural networks to find optimal levels for decision variables (resource levels, number of transporters, etc.) in simulations. These work well and can be computationally efficient methods to optimize a system. The rise in the availability of computing power allows the utilization of other methods that may provide consistent solution quality in more situations. The methods being referred to are evolutionary computational techniques, which include genetic algorithms (GA) and particle swarm optimization (PSO). These types of algorithms utilize natural schemas and mimic their design and decision criteria to find potentially good solution spaces. GA's work with Darwin's theory that the strongest survive; the algorithm tries to find the good "traits" of a solution in the belief that the optimal solution will have those same "traits." PSO is based on the swarming of bees and flocking of birds, where each entity or particle is drawn to its best solution and the swarm's best solution. With this information, particles move through the design space in an effort to find the global best solution. Utilizing the before mentioned general schemas, as well as including a certain amount of randomness in the algorithm helps evolutionary algorithms avoid getting trapped in local minima and find the global minimum. The following work focuses on using variations of PSO to find optimal levels for decision variables to stochastic simulations and the impact that adjusting parameters has on this type of optimization problem.

1.1 Introduction

PSO is a relatively new optimization method that was developed by Kennedy and Eberhart in 1995. In this work, PSO will be used to heuristically optimize decision variables in simulations created in Arena, a discrete event simulation software package created by Rockwell Software. Unlike many other works in this area, the primary focus will be to test how parameter adjustments and minor modifications to the algorithm affect convergence and computational time.

1.1.1 Pheromone PSO for Stochastic Models

The first hypothesis that will be tested in this work is that pheromone PSO is a constructive addition to standard PSO and will decrease solution time when solving stochastic optimization problems.

1.1.1.1 Work on Pheromone PSO

Work by Kalivarapu et al. (2007) shows that pheromone PSO can have a positive impact on performance when used to find the minima of highly complex nonlinear mathematical functions. This will be rigorously tested on stochastic problems by comparing the solution and solution time of pheromone PSO to that of traditional PSO.

1.1.2 Parameter Optimization of Pheromone PSO

The second hypothesis tested is that a correctly tuned pheromone PSO for stochastic problems will have a decreased time for convergence and an increased quality of solution. To test this problem, a tuned pheromone PSO solution will be compared to the results of OptQuest, a commercial simulation optimization package included with Arena.

1.1.3 Modifications of Pheromone PSO

The final hypothesis tested is that utilizing two minor algorithm changes can enhance the consistency and quality of the algorithm. The modifications that will be tested use orthogonal arrays to select starting points for the algorithm, and only particles in relatively good solution space drop pheromones. These adjustments don't change the basic structure of the algorithm, but they try to ensure consistent coverage of the solution space and bias the starting direction toward the best areas found in the first iteration.

CHAPTER 2. REVIEW OF LITERATURE

2.1 General PSO

PSO is an evolutionary computational technique developed by Kennedy et al. (1995). It uses the concept of birds flocking and swarms of fish to propel the algorithm across the solution space on its way to the global optimum. After the inception of PSO, Eberhart et al. (1996) eliminated a number of extraneous parameters that didn't aid in optimization. This slimmer version is what is now known as the basic PSO algorithm. The algorithm is computationally quite simple: A swarm of particles is selected, with each particle representing a random discrete point in the solution space. Every particle is then evaluated on the strength of its current location. The strength is compared to the particle's personal best location and to the swarm's best location, updating each if the current position is found to be better. The algorithm then determines where a particle is going to move next using a velocity update equation based on its current location compared to: what direction it was moving in the last iteration, the best location it has been, and the best location the swarm has been. The velocity update equation is shown mathematically as follows, where each dimension represents a control parameter being heuristically optimized:

$v_{i,d,t}$: velocity of particle i, in dimension d, at time t

c_j: constant, controlling the local and global search ability

$rand()$: random number from [0,1]

g_d: global best location in dimension d

$p_{i,d}$: best location found for particle i, in dimension d

$x_{i,d,t}$: current location of particle i, in dimension d, at time t

$$v_{i,d,t} = v_{i,d,t-1} + c_1 * rand() * \left(p_{i,d} - x_{i,d,t-1}\right) + c_2 * rand() * (g_d - x_{i,d,t-1})$$

(2.1)

Using the result from equation 2.1, the particle's location can be updated with equation 2.2.

$$x_{i,d,t} = x_{i,d,t-1} + v_{i,d,t}$$

(2.2)

The velocity and position update are continued for each member of the swarm. The particles' new locations are evaluated and the process repeats until the swarm satisfies the preset convergence criterion.

Since PSO's inception in, there has been a large amount of research conducted on it. Poli et al. (2008) categorize the work that has been completed to date; this paper will only focus of the work relevant to this extension.

There has been a considerable interest in finding ways to reduce the time to convergence and the consistency of the PSO algorithm. To accomplish this, researchers have employed a variety of methods, from minor parameter modification to adding additional components to the velocity update equation. Some minor parameter modifications that have been made include: decreasing the weight of the inertia factor, decreasing the overall velocity of the swarm, optimizing the population depending on the dimension of the problem, and adjusting the speed of the swarm based on the iteration number. The components that researchers have added to the velocity update equation include the digital pheromone method and the neighboring best method.

2.2 Inertia Weight

The next major addition that is consistently included with PSO is the weighted inertia factor (Shi et al. 1998a, 1998b). In this addition, they attempt to modify the algorithm to give it a strong global search ability at the beginning of the optimization and a strong local search ability at the end of the optimization. This is accomplished by decreasing the weight that inertia has on the next position linearly throughout the optimization. The revised velocity update equation is as follows:

$$v_{i,d,t} = w * v_{i,d,t-1} + c_1 * rand() * \left(p_{i,d} - x_{i,d,t}\right) + c_2 * rand() * (g_d - x_{i,d,t}) \tag{2.3}$$

In equation 2.3, w is decreased gradually throughout the optimization, typically 5% per iteration, to adjust the search from a global to a local search. Alternatively, in work by Zheng et al. (2003) it is argued that for some problem types an increasing inertia weight will increase both the convergence speed and the solution precision.

2.3 Constriction Factor

PSO was originally based on modeling an algorithm to a social system, and was based on trial and error. This original PSO lacked a thorough mathematical foundation. Work done by Clerc (1999), shows that a constriction factor may be a necessary addition to ensure convergence. While a detailed explanation of the math behind the algorithm is beyond the scope of this paper, the simplified idea behind this method is to multiply the entire velocity update equation with a function K, where K is a function of c1 and c2 as shown in equation 2.4 and equation 2.5 below.

$$v_{i,d,t} = k * [v_{i,d,t-1} + c_1 * rand() * (p_{i,d} - x_{i,d,t-1}) + c_2 * rand() * (g_d - x_{i,d,t-1})]$$

(2.4)

$$k = \frac{2}{\left|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}\right|}, where\ \varphi = c_1 + c_2, \varphi > 4$$

(2.5)

In work by Eberhart and Shi (2000), φ is set to 4.1, thus making the constant multiplier K equal to 0.729. They also use a maximum velocity for each dimension i, set to the dynamic range of that dimension. This is argued to be a robust parameter selection for most problem types.

2.4 Parameter Optimization

According to El-Gallad et al. (2002), "Unlike many other computational intelligence techniques, the particle swarm optimizer has few parameters to tune. However, properly chosen values for these parameters can positively affect the accuracy of the obtained results as well as the time consumed during the search process." El-Gallad examines the impact of three parameters: swarm size, number of iterations, and velocity of particles. He argues that, for the particular function tested, as the swarm size increases, the quality of the solution increases. He also shows that as the swarm size increases one is met with diminishing gains in quality improvement. El-Gallad states that a good swarm size for the seven-dimensional test problem is 30 particles. He also argues that the relationship with swarm size and quality of solution is true for the number of iterations, and proposes a number of iterations equal to 500. He proposes that an adaptive velocity based on work by El-Gallad et al. (2001) is best. Recent work by Zhang et al. (2004) discusses optimal parameter choice for constriction

factor PSO. The parameters examined are the sum of $c_1 + c_2 = \varphi$, swarm size, and the maximum allowed velocity (γ) as a proportion of the variable range. Each variable was tested on nine common test functions. Zhang et al. (2004) concluded that φ should be set to 4.05 for highly multimodal functions and to 4.1 for unimodal functions. A γ in the range of [0.01, 1] is appropriate, with a value of 0.5 for multimodal functions and a value of 0.05 for unimodal functions. This implies that particles should take larger steps in more complicated search spaces. The optimal swarm size was set to 50 for higher dimensional problems and to 30 for lower dimensional problems.

2.5 Digital Pheromones

In addition to adjusting parameters, many researchers have worked on variations to PSO to enhance convergence. One variation is adding digital pheromones. Pheromones are scents left behind by insects to mark food and nesting locations for other members of their swarm. The scent becomes stronger as more members of the swarm make their way to that particular area and find it suitable. Utilizing this concept, the digital pheromone was born, where digital marks known as pheromones are dropped by particles when they find promising areas of the design space. Using digital pheromones to aid in solving optimization is a recent development that started in the early 1990's in ant colony optimization, where the concept was used to mark promising paths in traveling salesmen type problems (Gambardella et al. 1996; Li et al. 2003).

2.5.1 Pheromone PSO

The concept for digital pheromones was detailed in research done by Kalivarapu et al. (2007). The basic idea behind pheromone PSO is that, in addition to the global best, personal best, and momentum influencing a particle's velocity, a target pheromone has an additional influence. This is shown in equation 2.6, where tp_i is the target pheromone selected by particle i:

$$v_{i,d,t} = v_{i,d,t-1} + c_1 * rand() * (p_{i,d} - x_{i,d,t-1}) + c_2 * rand() * (g_d - x_{i,d,t-1}) + c_3 * rand() * (tp_{i,d} - x_{i,d,t-1}) \quad (2.6)$$

Along with modifying the velocity update equation, there are several other changes to the algorithm when adding pheromones to PSO. These changes include choosing when a pheromone is dropped by a particle, how to select a target pheromone, merging of pheromones in the pheromone field, and decay of pheromones. A summary of how these changes are dealt with will follow and a detailed explanation of the changes can be found in Kalivarapu et al. (2007).

2.5.2 Pheromone PSO Implementation

When implementing pheromone PSO, the dropping of pheromones must be addressed. In the work of Kalivarapu et al. (2007), two methods of pheromone release are explained: 50% of the swarm population will randomly drop a pheromone, and any swarm member that finds a better personal best location will drop a pheromone. The next issue to address is how to select a target pheromone. This is done by measuring the normalized

distance on all dimensions of the problem, subtracting that from one, and multiplying the difference by the pheromone strength (P). This is shown mathematically as follows, where:

n: number of design variables

Xp_j: location of pheromone j

X_i: location of particle i

$range_d$: allowed variable range in dimension d

$$d = \sqrt{\sum_{1}^{n} \left(\frac{Xp_j - X_i}{range_d}\right)^2}$$

$$P' = (1 - d)P$$

$$\mathrm{tp}_i = max\ (P')$$

(2.7)

To ensure that the same point doesn't have multiple pheromones dropped on it, a method of merging pheromones in the same relative location was created. This is done by determining the radius of influence of a given pheromone, and if it overlaps with another pheromone, the two are merged with a strength equal to the average of their individual strengths. In addition to managing pheromones that are placed on top of one another, pheromones released early in the optimization process may represent poor locations of the design space and could slow convergence. This is dealt with using a pheromone decay factor that reduces the strength of every pheromone with each iteration. When a pheromone is released, it starts with a strength of one, and the strength is reduced by 5% each iteration. This ensures that pheromones released early in the optimization process will have a significantly decreased influence later in the optimization process (Kalivarapu et al. 2007).

2.6 Neighboring Best Particle

The neighboring best particle method has been developed by a number of people, with some minor variations among the versions. In the neighboring best particle method, particles are influenced by a nearby particle that has a good fitness value as well as the global best particle and the personal best location. The method of determining which particle to select varies. One good method was developed by Veeramachaneni et al. (2003), using a ratio of the difference in the fitness value to the distance to the proposed particle. This is shown in the equation 2.8 below:

$FDR(i,j,d) =$ fitness distance ratio

$X_{jd} =$ location of neighboring particle j in dimension d

$X_i =$ location of particle i in dimension d

$\text{Fitness}(i) =$ fitness value of particle i

$$FDR(i,j,d) = \frac{\text{Fitness}(j) - \text{Fitness}(i)}{|X_{jd} - X_{id}|} \tag{2.8}$$

Using equation 2.8, the velocity update equation selects the neighbor particle with the best fitness improvement to distance ratio and uses this new location as a third direction in the update equation. The general form is shown below in equation 2.9:

$$v_{i,d,t} = v_{i,d,t-1} + c_1 * rand() * \left(p_{i,d} - x_{i,d,t-1}\right) + c_2 * rand() * (g_d - x_{i,d,t-1}) + c_3 * rand() * (P_{j_{i,d}} - x_{i,d,t-1}) \tag{2.9}$$

This works well with a number of applications, as shown in work by Veeramachaneni et al. (2003).

2.7 Constrained Optimization

The PSO algorithm was originally designed for an unconstrained search space. Since most optimization problems have constraints, methods of adapting the PSO algorithm to manage constraints have been created. One of the most popular methods is based on an adaption from Lagrangian relaxation, as described by Lu et al (2007). In this method, the PSO's velocity update equation is modified and there is a change in the method of evaluating the objective function. The objective function is evaluated as required in the problem with an additional term, ϕ, as shown below in equations 2.10-2.12:

$$\min \{Objective\ Funtion + \phi\} \tag{2.10}$$

$$where\ \phi = \sum_{i=1}^{g} max\{0, g_i(x)\} + \sum_{j=g+1}^{g+h} max\{0, |h_j(x)| - \partial\} \tag{2.11}$$

$$v_{i,d,t} = w|p_{i,d,t-1} - p_{i,d}|sign(v_{i,d,t-1}) + rand(1) * (p_{i,d} - x_{i,d,t}) + (1 - rand(1)) * (g_d - x_{i,d,t}) \tag{2.12}$$

The addition of ϕ to the objective function penalizes the point evaluated for violating any constraints, thus encouraging the algorithm to pick points that are feasible by causing the particles to fly toward the feasible region. This is shown to work in cases with a high ratio of feasible space to available search space.

2.8 Stochastic Simulation Optimization

Optimizing stochastic systems is a concept that has been around since there have been systems that needed improvement. The first method was to change the settings of the system and see how the output was affected. Since the inception of the computer, models have been created to allow production controllers to test proposed additions without impacting production, allowing only for the implementation of changes that are expected to have a positive impact on the system. Over time, computer processing power has increased dramatically. Correspondingly, so has the complexity of systems that are able to be modeled. The traditional method of determining a good possible scenario to model is to take and test a particular level of settings suggested by a supervisor and see if it improves the current output of the system. This particular method requires an extremely high level of system knowledge, and even so has a very low chance of selecting the global best operating parameters. Work by Al-Aomar (2000) showed that using a discrete event simulator combined with expert knowledge, it was possible to make substantial improvements to a typical linear program optimization in a product mix simulation. The next common research method is to use typical optimization routines to select parameters that are used to control the system. Since this area has a high potential for substantial savings, a large amount of research has been dedicated to this area. It has also led to some commercial products that can be used to optimize simulation models. The relevant research in this area will be discussed next.

2.8.1 Stochastic Simulation Optimization with Heuristic Methods

Meketon (1987) surveyed existing methods of selecting optimal simulation parameters. At that time, the usual methods fell into three categories: traditional non-linear

programming techniques, response surface methodologies, and stochastic approximations. Since then, there has been a large push toward utilizing heuristic search techniques to solve highly nonlinear discontinuous problems. Work by Konak et al. (2005) discusses optimizing simulation problems using tabu search, including discussion of the profound effect that parameter selection has on the performance of the search. Work by Yang et al. (2004) discusses using tabu search to optimize the parameters of a flow shop scheduling problem. Empirical results showed tabu search as a promising method to solve the flow shop scheduling problem. Simulated annealing is another heuristic search method that was first developed in 1953 by Metropolis et al. It is based on emulating the physical process of aggregating particles in a system as it is cooled. The concept has since been developed into an algorithm that can be used to solve a variety of optimization problems. This is shown in work by Manz et al. (1989), where simulated annealing was used to optimize parameters for an automated manufacturing system simulation.

2.8.2 Stochastic Simulation Optimization with Evolutionary Algorithms

As an alternative to these types of heuristic search methods, evolutionary search methods such as Genetic Algorithms (GA) and Particle Swarm Optimization (PSO) are showing a wide range of applicability and robustness. These methods work well to solve a wide variety of problems and have received a large amount of research. In work by Joines et al. (2002), GA is used to optimize simulations of a supply chain to set optimal order quantity and time between orders. Koyama et al. (2004) worked on optimizing routing algorithms with GA using a simulated system. Wang et al. (2003) used a fuzzy adaptation of GA to optimize the industrial fermentation tower process and improved the expert system being

utilized. Dahal et al. (2005) used a standard GA optimizer to solve a simulation of an actual port facility to minimize total costs by reducing delays. Persson (2006) was able to use GA to solve a multi-objective mail sorting simulation created in Arena.

Yun et al. (2006) compared the results of GA to that of simulated annealing and OptQuest. The performance of these three algorithms was compared using a traffic management optimization problem. It was found that GA converged to a better solution in a shorter amount of time on a per iteration basis. This raises a concern because GA is a swarm based algorithm and each iteration takes 20-50 times the amount of computational time as OptQuest or simulated annealing, depending on the population used in the GA algorithm.

PSO has recently been used to solve the stochastic optimization of simulation problems. Zhang (2008) used PSO to solve a multi-objective simulation of earthmoving operations, comparing results to those of an exhaustive search. Alkhamis (2005) used PSO to optimize a repairable item inventory system, which examines working with discrete and integer variables by solving the continuous case and rounding to the nearest integer value. The work by Wu et al. (2005) compares the results of optimization of a ready mixed concrete simulation using PSO and GA. In this optimization case, PSO with constriction factor and weighted inertia reaches a better solution significantly faster than a GA-based optimization. While this is a very useful finding, the lack of variations and adjustments of the PSO and GA algorithms, as well as only testing it on one case shows there is still a need to optimally adjust PSO for stochastic discrete event simulation.

2.8.3 Stochastic Simulation Optimization with Commercial Products

In addition to these heuristic search methods, some commercial heuristic simulation optimization programs have been created, including OptQuest and ProModel Optimization Software Suite. These software programs utilize popular optimization techniques and are integrated with simulation programs. They can be easily interfaced with a simulation model to find optimal decision variable levels. April et al. (2001) describe the algorithm that is used in the OptQuest simulation optimization package as an integrated set of methods, including tabu search, scatter search, mixed integer programming, and neural network. While the exact method in which these algorithms are utilized is not public, work by Kleijnen (2006) shows the diverse applicability of the software. Kleijnen's work also discusses the alternative methods of setting convergence criteria. The work shows that, while the default settings are appropriate for most problem types, a large amount of efficiency is gained by good selection of the starting solution, choice of suggested solutions, and size of the search area.

2.9 Literature Conclusion

As noted previously, much work has been focused on optimizing PSO for solving linear and nonlinear deterministic systems. Significantly less research has been done on solving stochastic problems, especially when dealing with simulation where steady state is not guaranteed. In this type of volatile problem, little research has been completed.

CHAPTER 3. METHODOLOGY

3.1 Overview of Methodology

Pheromone PSO was tested with stochastic simulations by interfacing multiple versions of PSO, including pheromone PSO, with Arena models. Once it was shown that pheromone PSO was a constructive addition to PSO, the input parameters for pheromone PSO were tuned to solve stochastic problems. The solution time and solution quality achieved with the optimization was compared to the solution time and solution quality found when solving the same problem with OptQuest. Once this test was completed, two minor modifications were made in order to further enhance the algorithm: orthogonal arrays were used to place the initial solutions, and pheromones were initially placed only at points with relatively high objective function values.

3.2 Development of Models to Optimize

The first step was to create the simulations to be optimized and establish cost metrics on which to rate the simulation. Following, this process is described for four test simulation models.

3.2.1 Asynchronous Automated Assembly System (AAAS)

The first model optimized was a closed loop asynchronous automated assembly system (AAAS) based on the model described in Stochastic Optimization of Cost of Automatic Assembly System (Tandiono et al., 1994). An AAAS is a high speed production system containing a number of stations that are set in a predetermined order. The product starts at the first station and is processed at each subsequent station until it has been

processed on every machine. At that point the product is completed and is ready for the next stage in its production. Being asynchronous, processing times at each station are not equal. When a particular operation is completed, the product moves to the next station. A buffer zone exists between the stations to allow the previous machine to keep running even if the following station hasn't finished production. Since this is a closed loop system, the product is loaded onto a pallet at the first station and the finished product is unloaded at the final station. The empty pallet then returns to the first station. The two major items that can be adjusted and must to be optimized are the number of pallets in the system and the length of the buffers between stations. If the two are not set correctly, starvation and/or blocking can occur throughout the system. A diagram of this model can be seen in Appendix V – AAAS Model. Based on the work by Tondiono et al. (1994) the optimal setting for minimizing the cost function can be determined by:

CL: conveyor length

P: number of pallets

TP: total units produced during the simulation

$Cost$: total cost of a particular setup

$$Cost = 15000 * (CL + 10) * 0.1627 + 500 * P * 0.1627 + 1500 * (CL + 10) * \left(0.2259 + 0.0314 * (CL + 10)\right) + 15000 + 100 * P * 0.1 + 52 * 4 * 10 * (2200 - TP) \quad (3.13)$$

3.2.2 Catalog Center Model

The second model is based on a catalog center using the example model included in Simulation with Arena by Kelton et al. (2004). This model has calls arriving to a catalog

center where a number of representatives take the orders. The order must be filled by warehouse workers and the requested items must be loaded on a delivery truck. Once the entire order is shipped, a copy of the ticket is sent to the billing department and sales department where the invoice and future mailing lists are generated, respectively. A diagram of this model can be seen in Appendix VII – Catalog Center Model. The model's optimal parameter settings are adjusted using the cost equation that follows:

BKS: number of customer that baulk

$CatRep$: number of catalog representatives

WHW: number of warehouse workers

DR: number of truck drivers

SC: number of scanners and operators

CK: number of billing clerks

BCH: shipping batch size

$Cost$: total cost of a particular setup

$$Cost = 7500 * (BKS) + 60000 * CatRep + 45000 * (WHW + DR + SC + CK) + 1000 * \frac{4400 - BKS}{2160} + \left(\frac{4400 - BKS}{BCH}\right) * 20 * 3000 \quad (3.14)$$

3.2.3 Lamp Assembly System

The third model that was analyzed is a lamp assembly line as described in work by Mo (2007). The twenty-one step, five-stage lamp supply chain is modeled using times and flows from Mo (2007), with an additional storage constraint limiting the total amount of

work in progress (WIP). The goal of this simulation is different from that described in the work by Mo; the objective is to maximize the profit of the system instead of minimizing the cost of WIP. A diagram of this model can be seen in Appendix IV – Lamp Model. The operating cost is calculated as follows:

$Lamps$: number of lamp assemblies created

$Returns$: number of lamps returned to supplier

$Employees$: sum of all assembly employees

IAT: the requested time between supply shipments

$$Profit = Min(Lamps, 5184) * 500 - 3650 * (Returns) - 60000 * Emplyees + 5184 - Lamps - 31{,}553{,}600/IAT \quad (3.15)$$

3.2.4 Distribution Center Model

The fourth model that was included is a ten-door, pallet-based, long-haul distribution center, with five incoming and five outgoing doors. The model attempts to determine the optimal door location and number of fork trucks in the system for this distribution center. A diagram of this model can be seen in Appendix VI – Distribution Center Model. The objective function is calculated as follows:

$ATIS$: average time a shipment is in the system

SOT: number of shipments that exceeded the maximum time allowed in the system

$Fork$: number of fork trucks

$Forkspeed$: speed factor that allows increase movement from all fork trucks

$Cost$: total cost of a particular setup

$$Cost = 12500 * ATIS + 25 * 58.4 * SOT + 20000 * (Fork) + 2000 * Forkspeed * Fork \quad (3.16)$$

3.2.5 Steady State

Once the models were defined and created in Arena, steady state was evaluated to establish the appropriate warm-up time for the simulation. Once established, that number was increased by an arbitrary amount because as the different input parameters are changed, the time it takes to reach steady state could potentially increase. This process to determine time until steady state is described below:

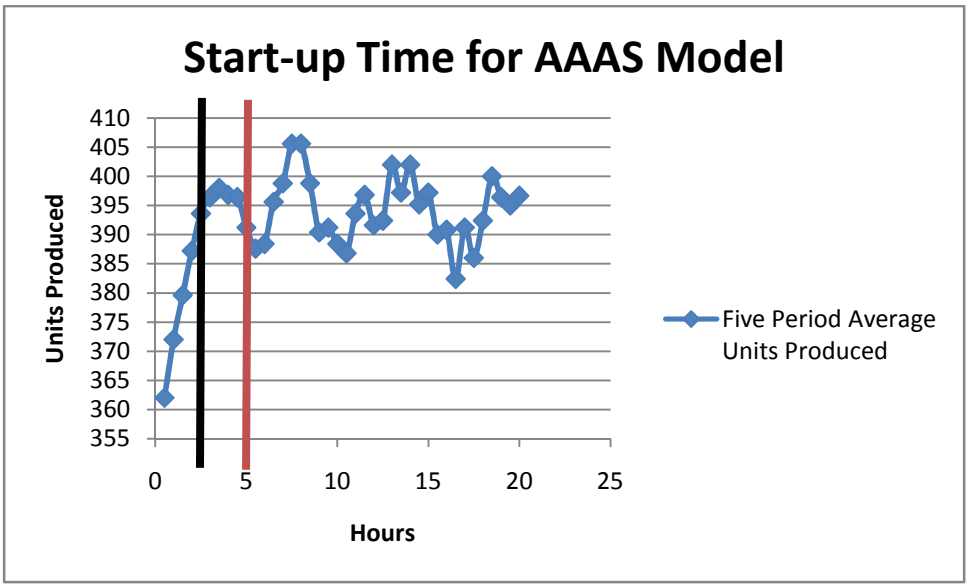


Figure 1 - Startup time for AAAS model.

Steady state is described as the standard running condition of a particular process not influenced by start-up. In Figure 1 above, after three and a half hours of run time (marked by the black line) the system levels off and is in steady state. Then, to ensure that all variations

of this simulation were in steady state, a warm-up period of five hours was used (shown as the red line).

3.3 PSO Model Development

Once the models were created in Arena, the PSO algorithm was set up to control the algorithm, and update and control the Arena models. The adjustment and control of the Arena models was required in order to evaluate the objective function at different points in the design space. PSO was coded into Arena using Visual Basic because of the ease of adjusting the simulation model as needed by the algorithm. The PSO and pheromone PSO code for the AAAS model can be seen in their entirety in Appendix III and VI, respectively.

The general form for the algorithm/simulation interaction implemented can be seen in Figure 2 below:

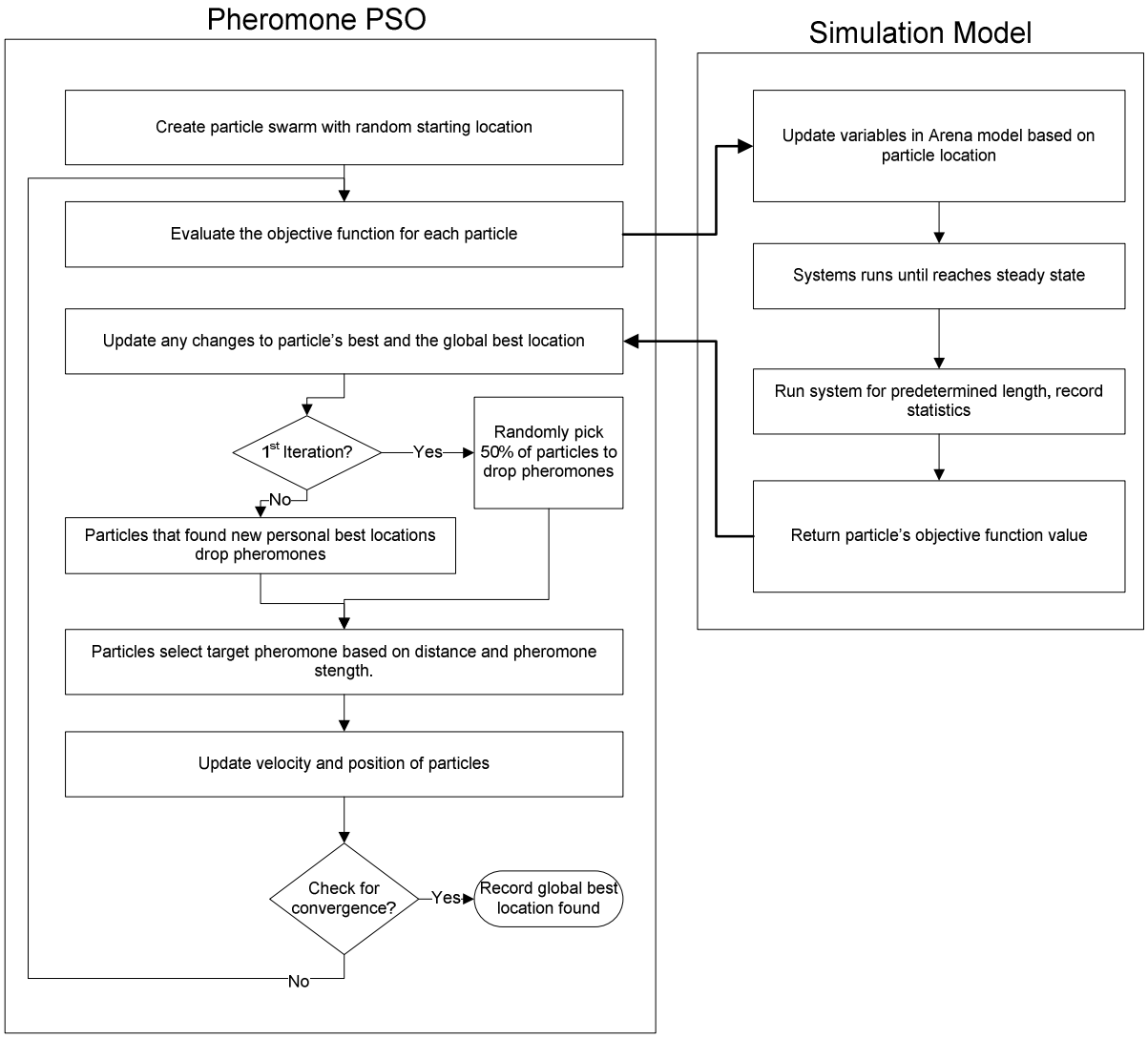


Figure 2 - Pheromone PSO flowchart

Figure 2 shows that once a PSO algorithm was created as an object in the Arena model, it filled the role of the optimization package. It initially picks population parameters, populates them in the Arena model, and starts the Arena model running. When the model is finished running it writes the measured metrics to a text file which the PSO program uses to determine the next test location. The convergence criterion for the algorithm is met when the algorithm completes the required number of iterations without a change to the global best

solution. When this happens, the algorithm stops. Otherwise, the algorithm sends the updated population parameters to the Arena model and the process is repeated until the convergence criterion is met.

3.4 PSO vs. Pheromone PSO

Linking Arena and Visual Basic, a comparison between PSO and pheromone PSO was made using the four models described in section 3.2. Standard levels derived from literature (El-Gallad et al. 2002, Zhang, et al. 2004 and Kalivarapu et al 2007) were used for the parameters in PSO; the values for these levels are shown in Table 1 on page 26. PSO and pheromone PSO results were compared using average solution quality and time to convergence, based on the number of calculations using the objective function. Objective function calculations were used as a measure of time to convergence because in optimizing simulations, the time it takes to run the simulations is orders of magnitude higher than the time it takes to run the rest of the optimization code. As shown in work by Kalivarapu (2008), even very complicated functions of up to 50 dimensions took less than 100 seconds to optimize using pheromone PSO, compared to 1200 seconds for a comparatively simple simulation model optimization.

3.5 Parameter Optimization for Pheromone PSO

In order to determine what parameters work best for pheromone PSO on stochastic problems, four simulations were used. These were used to optimize the input parameters of a pheromone PSO with weighted inertia. The results were used to tune the algorithm and compare the results of all the simulations to the results of OptQuest. OptQuest is a well-

developed program that utilizes tabu search, scatter search, mixed integer programming and neural networks to optimize complex simulations and, according to Kleijnen (2006), sets a good bar for comparison to see if PSO is adept at solving stochastic optimization problems. The models were compared on three criteria: solution quality, consistency, and computational time.

The PSO algorithm was tuned to solve stochastic optimization. All of the input parameters that can be adjusted must be studied. In this case, the simplest parameters are the size of the swarm, the number of iterations until convergence, and the maximum velocity. Additional parameters include the number of pheromones dropped in the first iteration, the decay rate of the pheromone field, and values of the coefficients associated with global best, personal best, and pheromone selected (alternatively known as c1, c2, and c3, respectively). From previous literature, there are predetermined parameter values that are normally used. These, as well as a range of acceptable values, can be seen in Table 1:

Table 1 - Parameter ranges and standard values

Parameter	Minimum Value	Standard Value	Maximum Value	Units
c1	1	2	4	NA
c2	1	2	4	NA
c3	1	4	6	NA
Weighted inertia factor	1%	5%	25%	Percentage decrease
Pheromone decay factor	1%	5%	25%	Percentage decrease
Pheromones dropped during first iteration	20%	50%	100%	Percentage released
Maximum velocity	20%	80%	100%	Percentage of variable range
Swarm size	10	30	50	Number of members
Iterations until convergence	10	20	50	Number of iterations without an improving solution

These parameters were run five times at each incremental adjustment of the variable, and the results from the optimization algorithm were recorded into data sets to show how the algorithm behaved as the parameters were adjusted.

3.6 Modifications of Pheromone PSO

While pheromone PSO is a well developed algorithm, there are a number of modifications to the initialization of the algorithm that have the potential to increase its consistency and speed. These modifications include using an orthogonal array to set the particles' starting locations and allowing only the good particles to drop pheromones.

3.6.1 Pheromone Release Modification

Pheromone release during the first iteration was initially set to allow each particle a 50% chance of releasing a pheromone. This was done to allow a thorough exploration of the solution space. The modification tested was to only allow particles that have an objective function value within some percentage of the global best value to drop pheromones. This ensures that only areas of the solution space that are relatively good compared to the current global best have a pheromone dropped in them. While this biases the search direction toward the area of the search space where good solutions were found, it has the potential to decrease the time to convergence.

3.6.2 Orthogonal Arrays

Orthogonal arrays were developed in statistics as a systematic way of setting test levels for variables in a problem. In an orthogonal array, each vector is designed to be perpendicular to every other vector, and conveys unique information about the test to avoid redundancy in testing. This concept was used to set the starting location for PSO. For the selected problems, the largest number of variables is thirteen. By setting each decision variable to three levels (the minimum, the mean, and the maximum), an L27 array can be used to contain the problem. This array size works well for PSO applications because 27, the number of tests needed to define the thirteen variables at three levels, is close to the standard value for the number of particles in a swarm. A list of the vectors for the thirteen variables and three levels can be found in Appendix 1.

CHAPTER 4. RESULTS

4.1 Overview of Results

Once pheromone PSO was shown to be a constructive addition to PSO, the different parameters were adjusted to see what levels were appropriate. Once the levels were determined, two modifications to PSO were implemented: using orthogonal arrays to determine particle starting locations, and biasing pheromone dropping based on the objective function value.

4.2 PSO vs. Pheromone PSO

To see if pheromone PSO was a good addition to PSO, the results of the standard PSO algorithm were compared to those of the pheromone PSO algorithm. The results of these comparisons are shown below for all four models:

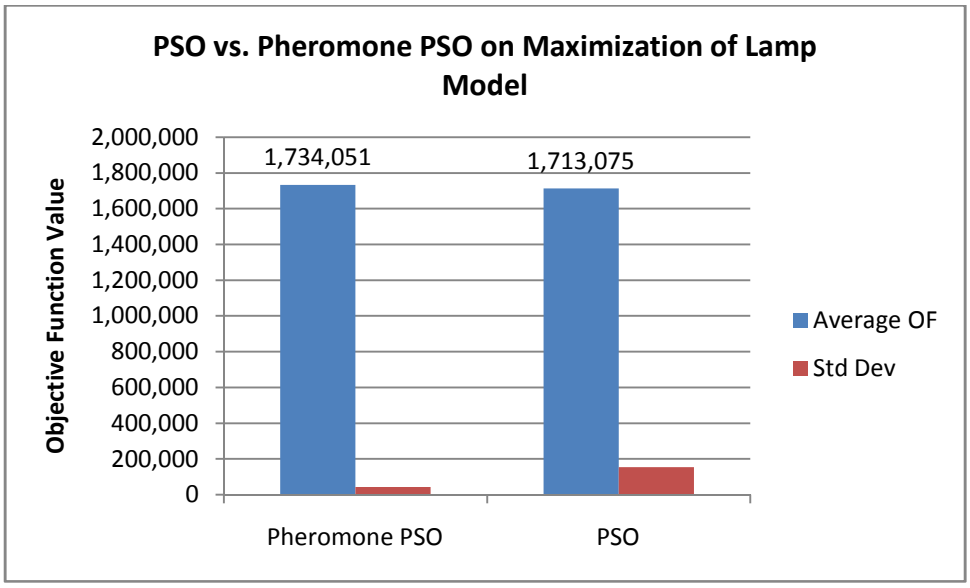


Figure 3 - Comparison between PSO and pheromone PSO on the lamp model.

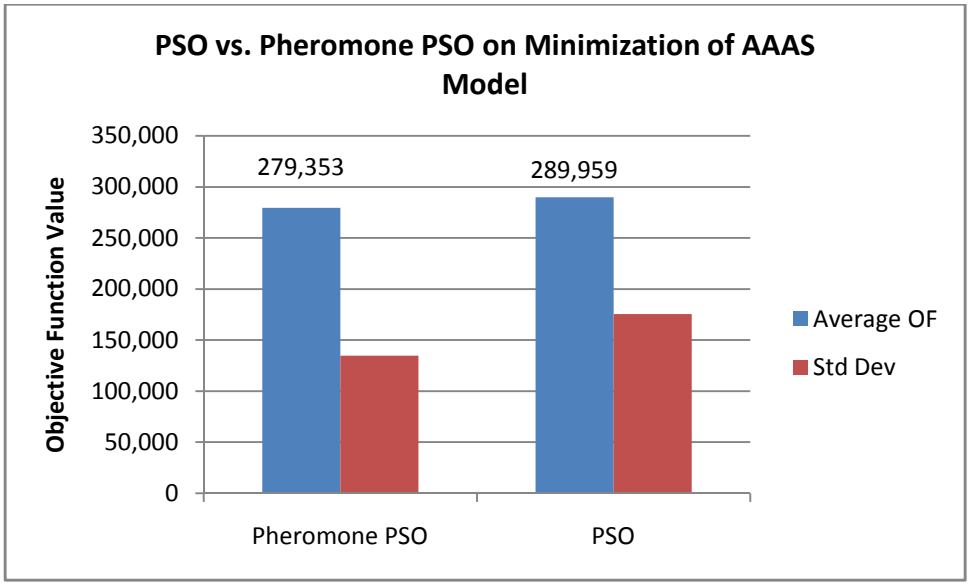


Figure 4 - Comparison between PSO and pheromone PSO on AAAS model.

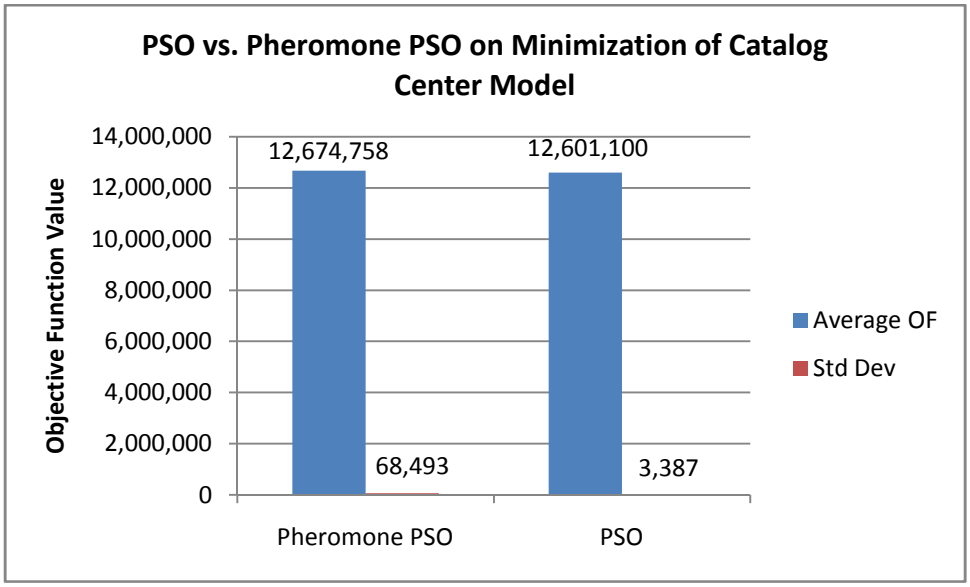


Figure 5 - Comparison between PSO and pheromone PSO on catalog center model.

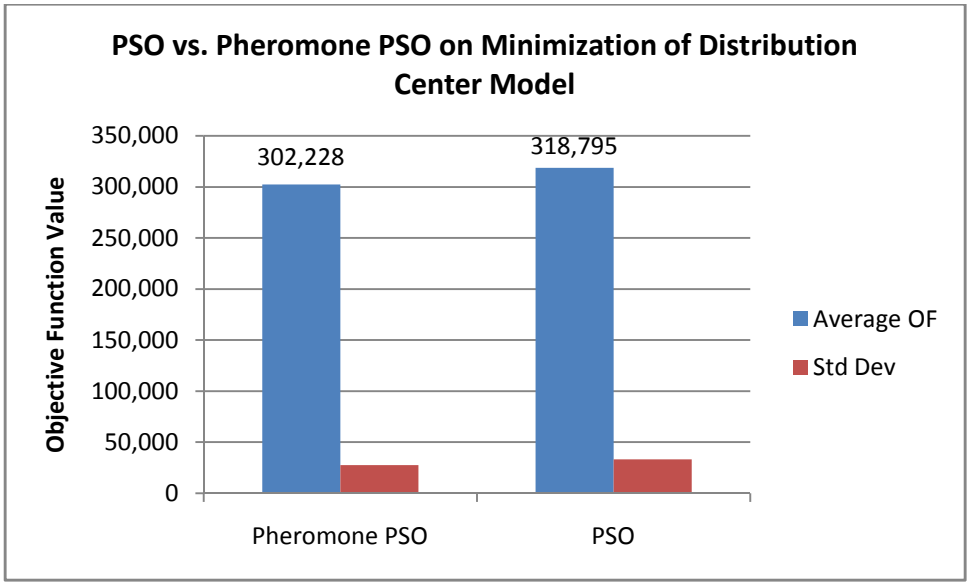


Figure 6 - Comparison between PSO and pheromone PSO on distribution center model.

As illustrated in Figure 3 - Figure 6, the model using pheromone PSO resulted in a better average objective function and lower standard deviation in three of the four models tested. This shows that for most of the models tested, pheromone PSO solutions were better than those of standard PSO.

4.3 Selection of Parameter Levels for Pheromone PSO

The next step was to determine parameter levels for the pheromone PSO algorithm. Pheromone PSO was tested to see, if properly configured, how its solutions compared to those of OptQuest on a number of objective function calculations.

4.3.1 Developing Parameter Settings

To develop parameter settings, the sensitivity of all pertinent controls described in section 3.5 were tested individually, varying the test parameter and holding the rest of the parameters constant. This was done for all four models. One example for each parameter is

shown below, with brief discussion of the resultant trends, the additional graphs can be seen in Appendix IX – Parameter Adjustment.

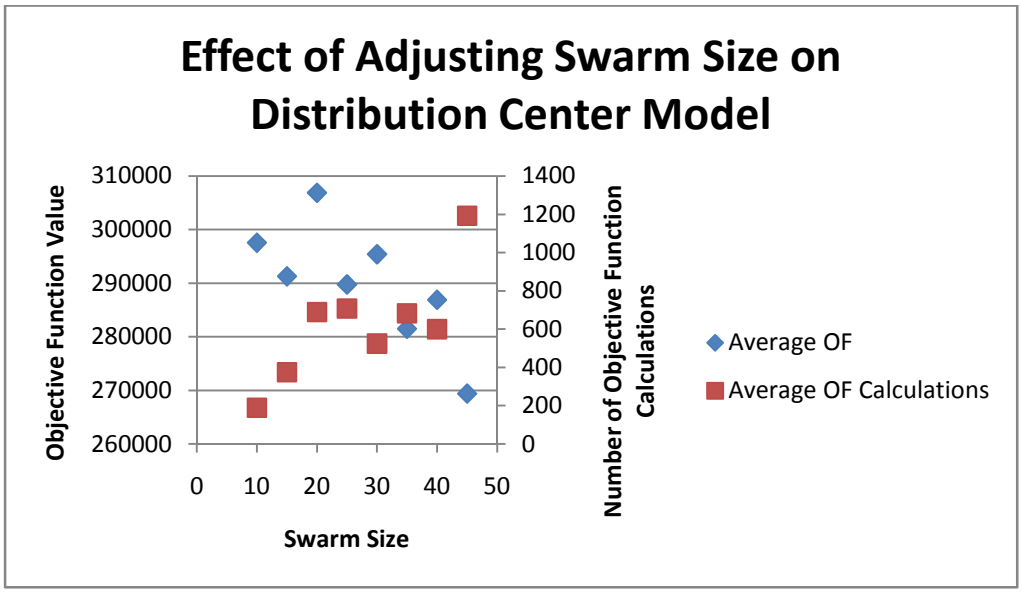


Figure 7 - Effect of adjusting swarm size on distribution center model.

Figure 7 shows that, for the distribution center model, as the swarm size increased the average objective function decreased and the number of objective function calculations increased. The AAAS and catalog center models had the same relationships between swarm size, objective function, and objective function calculations. Conversely, the lamp model had no consistent decrease in objective function as the swarm size was increased. The trend in Figure 7 makes intuitive sense, being that when more particles are in the system, more particles are trying to find the best solution. Consequently, the objective function is calculated more, and the model takes more computational time to converge on a solution. Figure 7 also illustrates that a good setting for this problem is a swarm size of approximately 25-35 particles, since this range has relatively low objective function values compared to the number of iterations needed to converge.

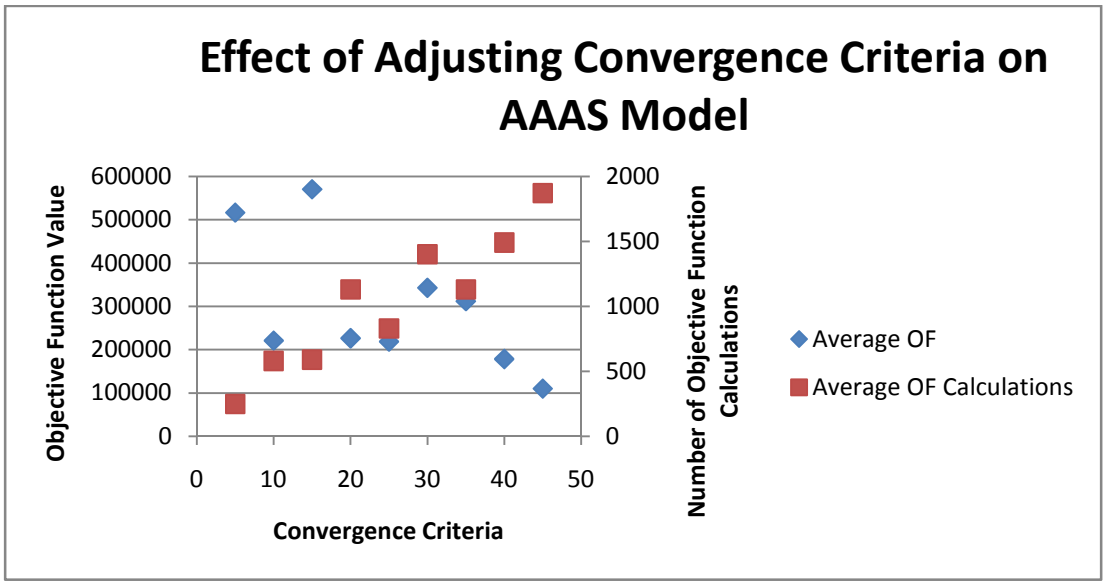


Figure 8 - Effect of adjusting convergence criterion on AAAS model.

Figure 8 shows the effect that the convergence criterion has on the objective function value and the number of objective function calculations. In Figure 8, as the convergence criterion was increased, the average objective function value improved and the number of calculations increased. The other three models followed the same trend. This makes intuitive sense, because the more time the algorithm has to find a better solution, the better the chance that the algorithm will converge. Figure 8 also shows that a good setting for the convergence criterion in this problem is 20 – 25 iterations, since this range has a good ratio of the average objective function value to number of objective function calculations.

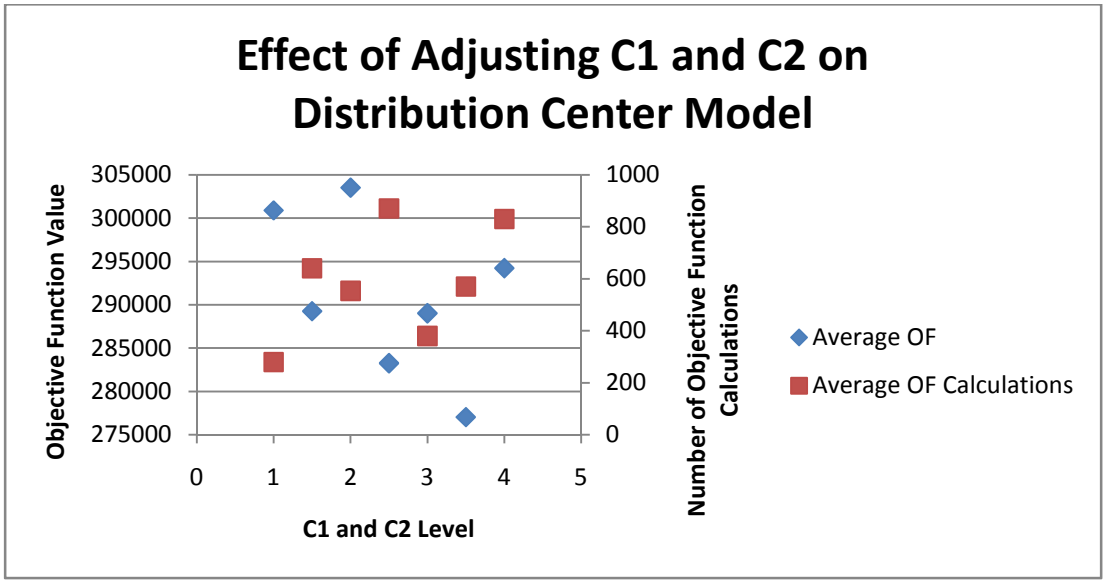


Figure 9 - Effect of adjusting C1 and C2 on distribution center model.

Figure 9 shows the effect that adjusting c1 and c2 had on the objective function value and the number of objective function calculations. As shown in Figure 9, there was not a clear trend on the effect of adjusting c1 and c2, but values of 1.5 and 3-3.5 appear to be good choices. The other three models had a similar lack of general trend. In the AAAS model, a value of 2 appeared to be a good level, and the other two models had no apparent best location.

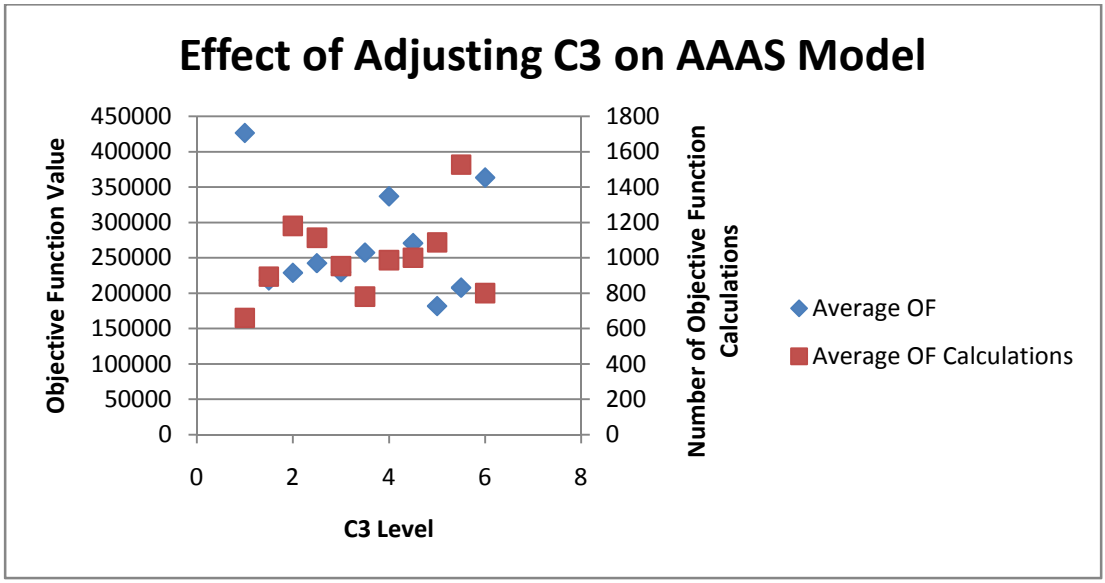


Figure 10 - Effect of adjusting C3 on AAAS model.

Figure 10 shows the effect that adjusting c3 had on the objective function value and the number of objective function calculations. As shown in Figure 10, the number of objective function calculations behaved like a fourth-order polynomial equation, and the average objective function value behaved like a quadratic equation. This makes sense because, at the extreme locations around one and six, the algorithm moved too chaotically and converged prematurely. In the middle (around 3.5-4) the algorithm converged quickly and to a good location. While the second and fourth order relationships were not nearly as apparent on the other three models, all of the models performed well with a c3 value of approximately 3-4.

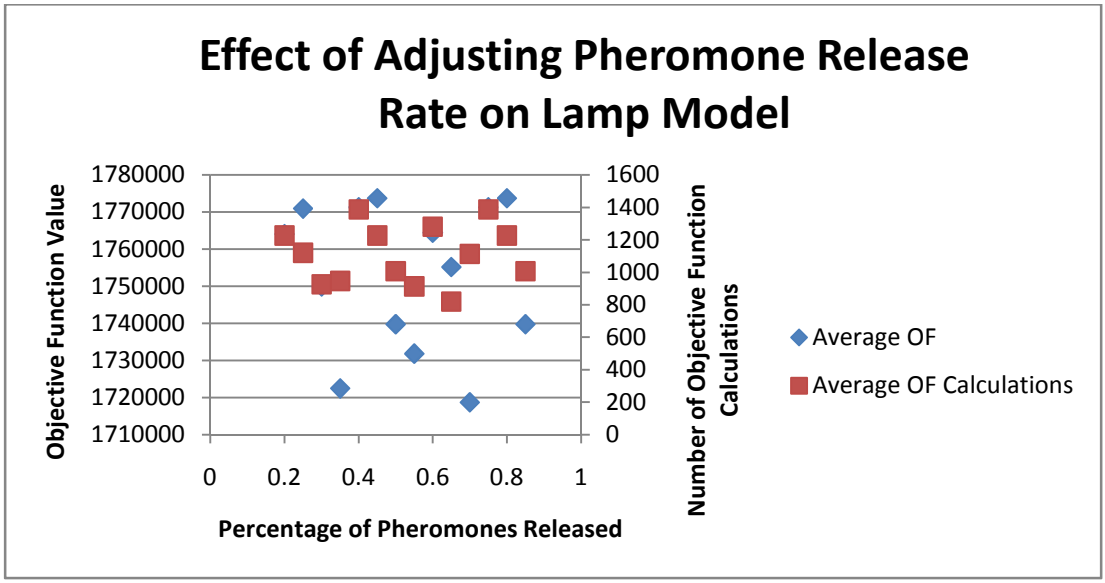


Figure 11 - Effect of adjusting pheromone release rate on lamp model.

Figure 11 shows the effect that adjusting pheromone release rate had on the objective function value and the number of objective function calculations. As shown in Figure 11, there was not a clear trend on the effect of adjusting the pheromone release rate, but a rate of 45-50 percent appears to be a good choice, being that they were the only consecutive parameter levels to converge to an objective function value less than 1,740,000 in fewer than 1000 iterations. The other three models also performed well in the 40 to 50 percent range.

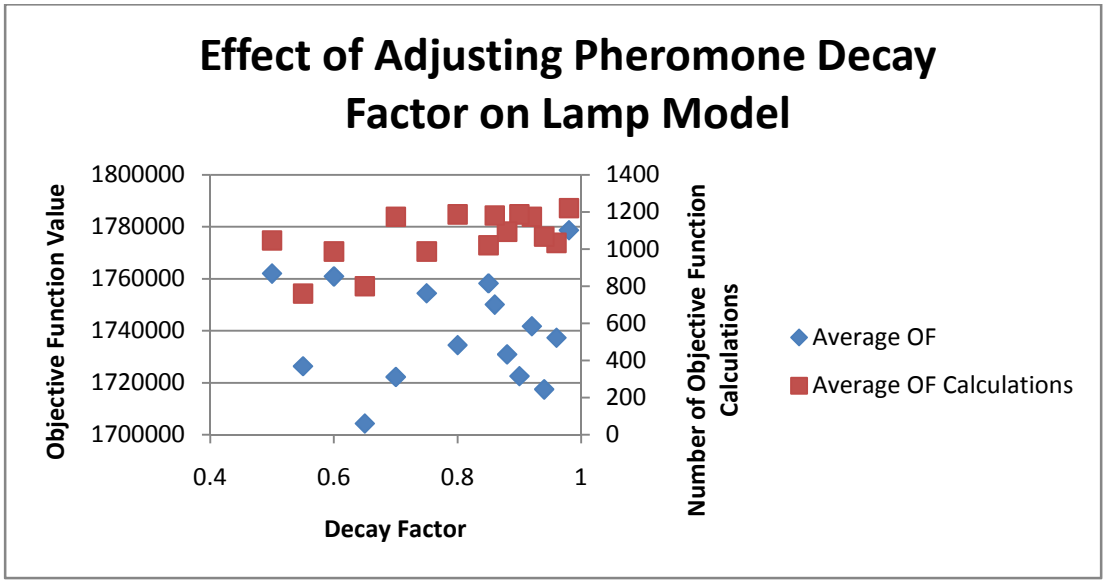


Figure 12 - Effect of adjusting pheromone decay factor on lamp model.

Figure 12 shows the effect that adjusting the pheromone decay factor had on the objective function value and the number of objective function calculations. As shown in Figure 12, there was not a clear trend on the effect of adjusting the pheromone decay factor, but as the parameter reached 0.8 and above, the results seem significantly more consistent, suggesting that a value of 0.9-0.95 may be appropriate. The other three models had no clear trends, with the best values ranging from 0.8-0.95.

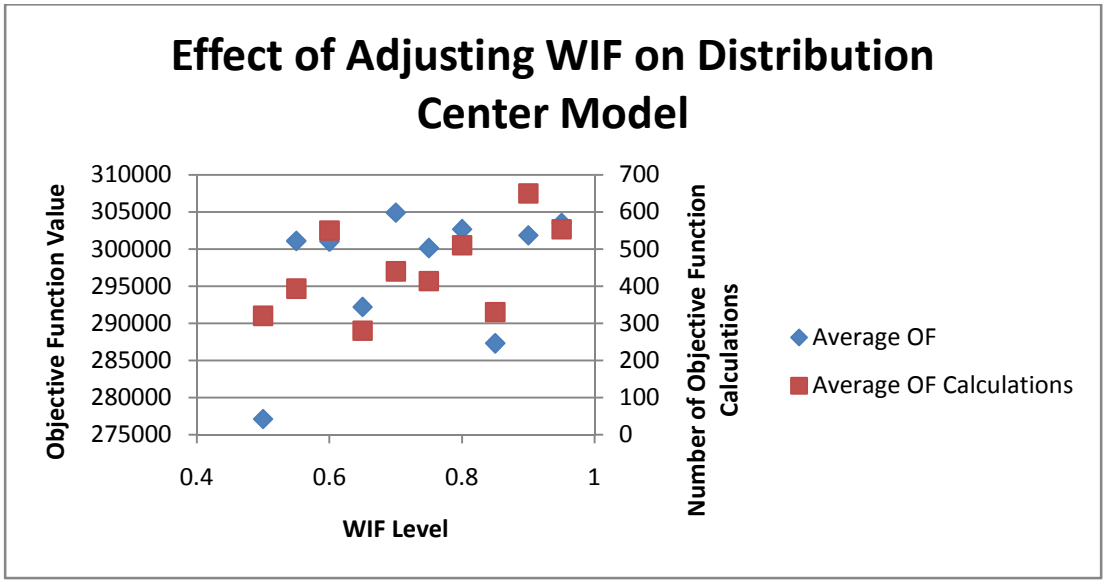


Figure 13 - Effect of adjusting weighted inertia factor on distribution center model.

Figure 13 shows the effect that adjusting the weighted inertia factor had on the objective function value and the number of objective function calculations. As shown in Figure 13, there was not a clear trend on the effect of adjusting the weighted inertia factor. In the AAAS model and the catalog center models, a value of 0.9 performed well.

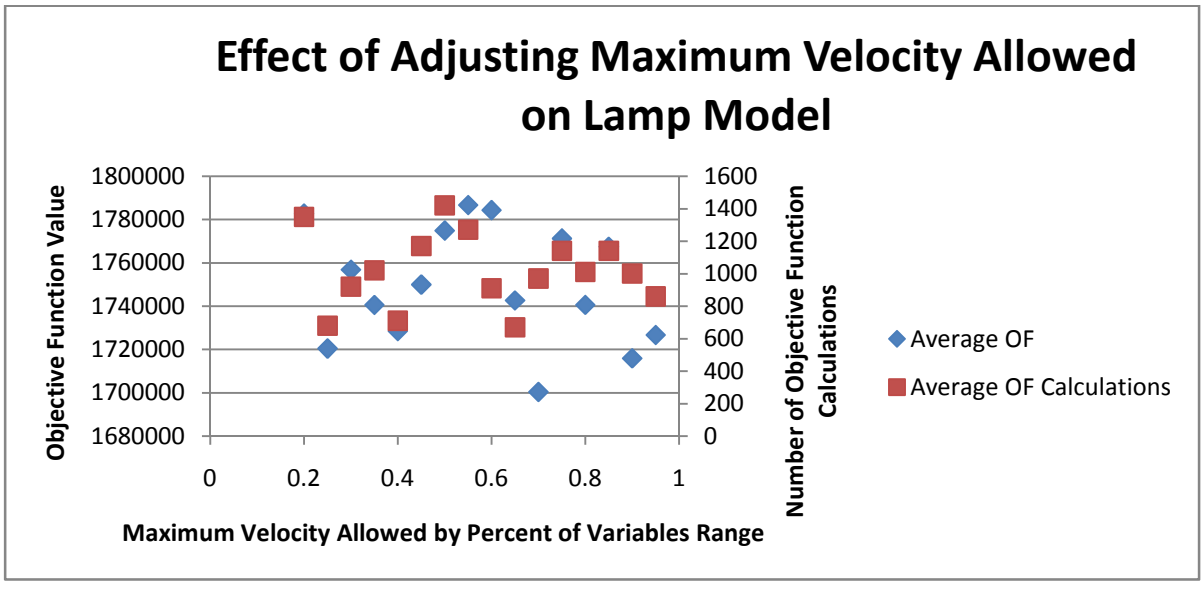


Figure 14 - Effect of adjusting maximum velocity allowed on lamp model.

Figure 14 shows the effect that adjusting the maximum velocity allowed had on the objective function value and the number of objective function calculations. As shown in Figure 14, there was not a clear trend on the effect of adjusting the maximum velocity allowed, as there are local minima around 0.25, 0.7 and 0.95. A maximum velocity allowance of 95% appears to be the best of that subset, with neighboring points also being good solutions.

4.3.2 Summarizing Parameter Effects

Based on the data collected from all four models for the nine control parameters for PSO, the settings that work well for the tested models are described in Table 2 :

Table 2 - Tuned parameter levels

Parameter	Standard Value	Tuned Level	Units
c1	2	1.75	NA
c2	2	1.75	NA
c3	4	3.75	NA
Weighted inertia factor	5%	10%	Percentage decrease
Pheromone decay factor	5%	7.5%	Percentage decrease
Pheromones dropped during first iteration	50%	47.5%	Percentage of global best
Maximum velocity	80%	95%	Percentage of variable range
Swarm size	30	25	Number of members
Iterations until convergence	20	23	Number of iterations without an improving solution

4.3.3 Testing PSO vs. OptQuest

The settings from the previous section were used to compare the speed and quality of solutions from pheromone PSO to those of the commercial optimization software, OptQuest. Each sample point was run ten times and the average was used to calculate the objective function value. The results from these tests are shown in the graphs that follow:

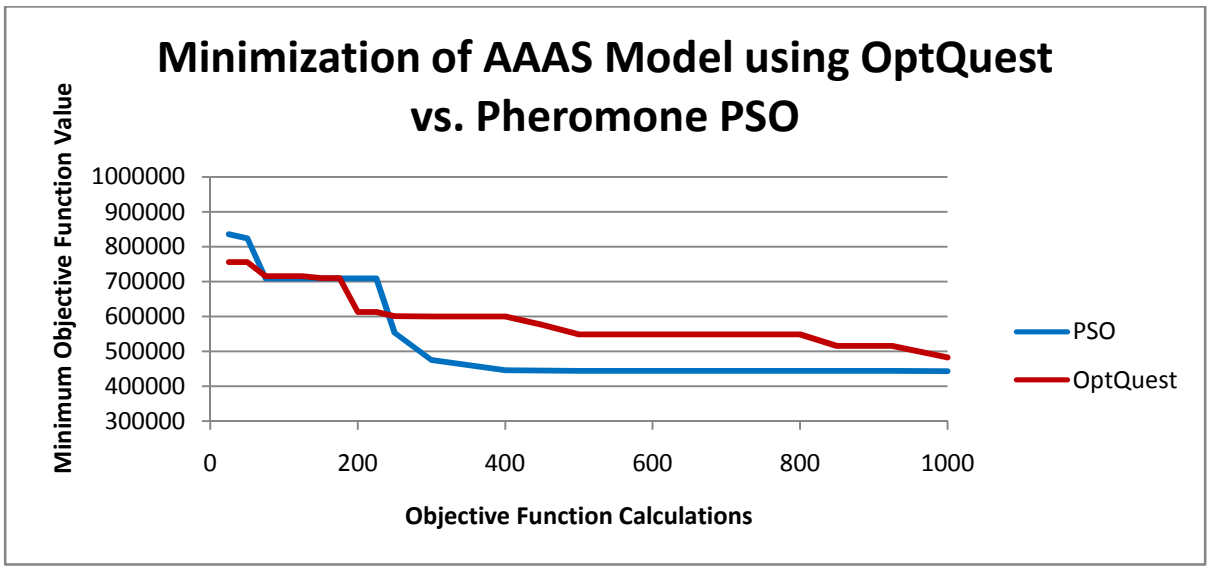


Figure 15- Minimization of AAAS using OptQuest vs. pheromone PSO.

As the above graph clearly shows, the PSO algorithm outperformed the OptQuest model on the AAAS model; PSO converged to a solution that was 39,000 units lower (443,469 units vs. 482,536 units).

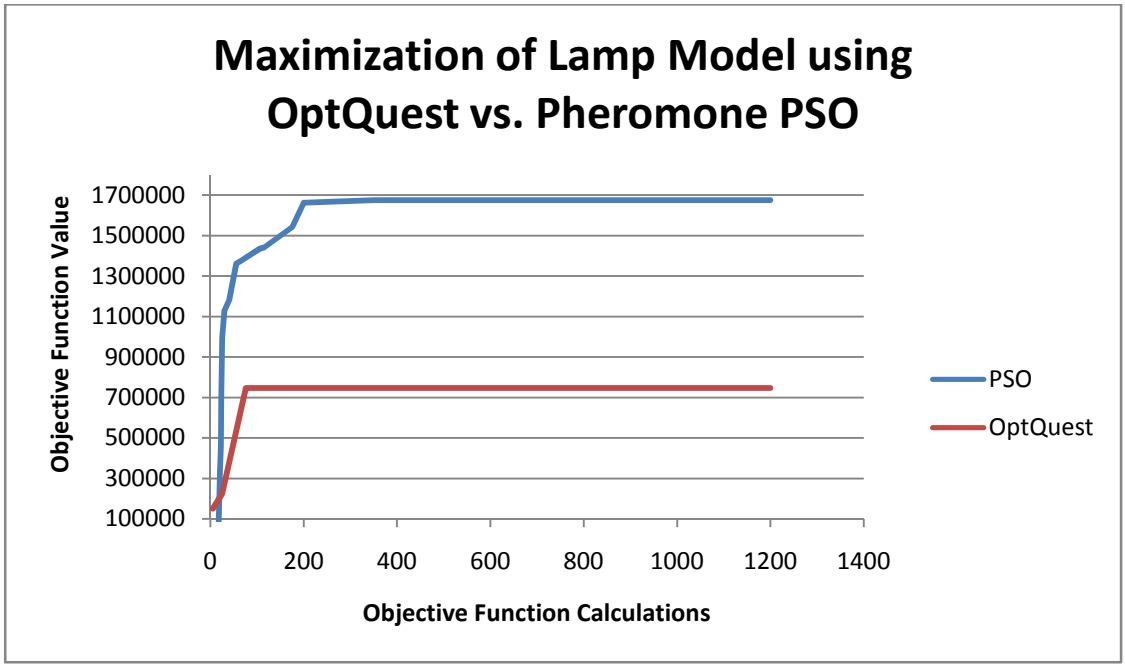


Figure 16 - Maximization of lamp model using OptQuest vs. pheromone PSO.

As illustrated by the graph above, the PSO algorithm outperformed the OptQuest model on the lamp model; PSO converged to a solution that had a total profit of 929,005 units higher than that of OptQuest (1,674,900 units vs. 745,895 units), or a 124% increase.

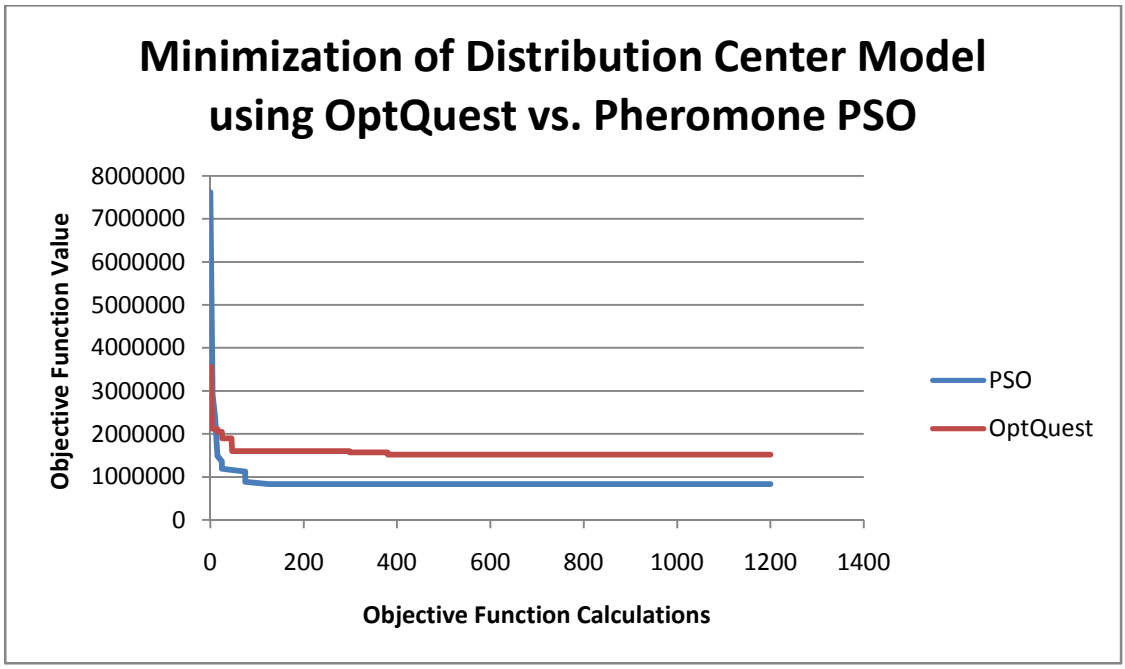


Figure 17 - Minimization of distribution center model using OptQuest vs. pheromone PSO.

In the comparison of OptQuest vs. pheromone PSO on the distribution center model, a few modifications to the model were necessary to allow OptQuest to adjust the door locations. The adapted model was used for both tests, and the optimization runs were then compared. The graph above shows that PSO outperformed OptQuest on optimizing the distribution center model; PSO converged to a solution that had a total profit of 689,281 units lower than that of OptQuest (1,519,528 units vs. 830,247 units), or a 45% decrease. Attempting to solve this model also exposed a limitation to OptQuest: OptQuest only recognizes resources and variables as controls that can be adjusted in its optimization. This increased flexibility is an additional benefit to using direct coding approach to control and heuristically optimize the decision variables in the simulation.

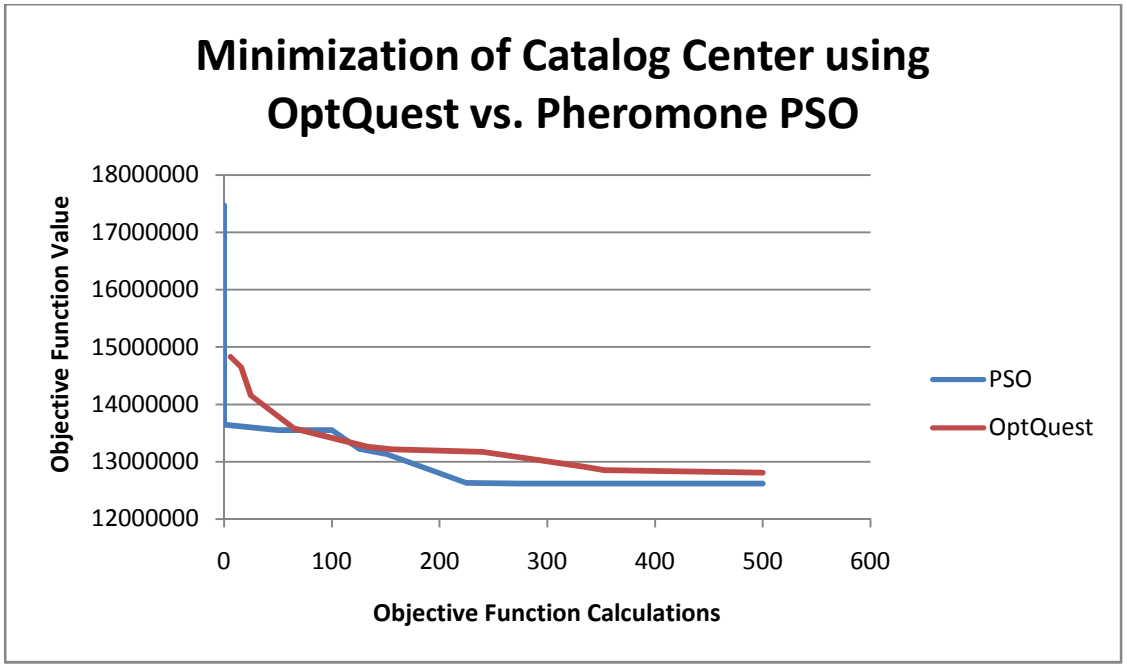


Figure 18 - Minimization of catalog center using OptQuest vs. pheromone PSO.

In the graph above, PSO algorithm performed slightly better than OptQuest on the catalog center model; PSO converged to a solution that had a total cost 185,240 units lower than OptQuest (12,620,760 units vs. 12,806,000 units), or a 1.4% decrease.

These tests show that, for the models tested, pheromone PSO was able to find a better solution than an OptQuest model in the same number of objective function calculations. Although, only in the lamp model can the solution be shown as statistically different.

4.4 Modification using Orthogonal Arrays and Biased Starting Location

The final experiment tested was to see if two modifications to the algorithm could improve the solution quality and/or decrease the time to convergence. These changes were made in all four models, and the results from the tests are shown below:

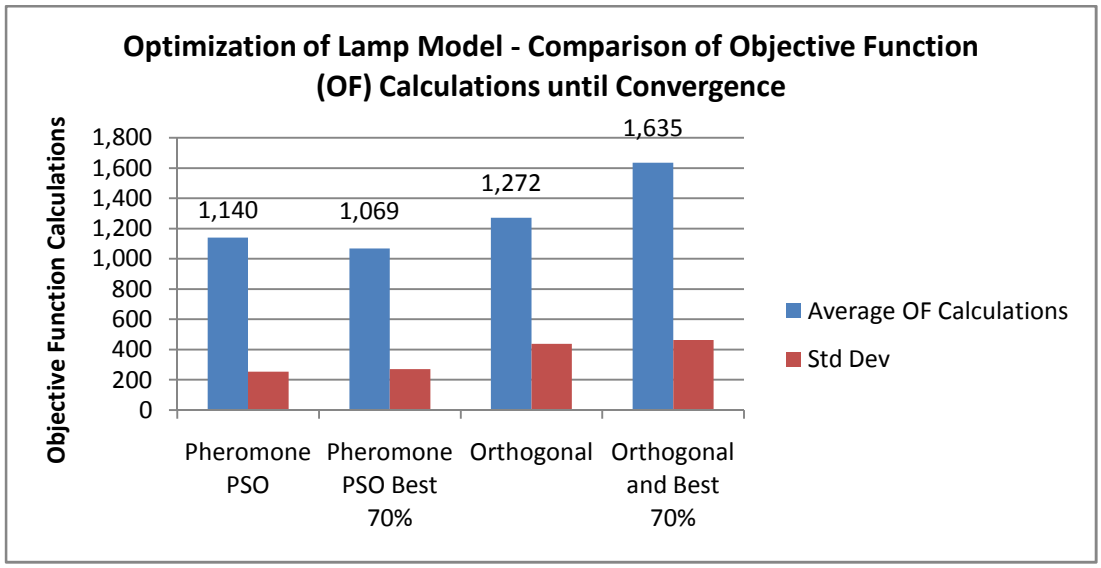


Figure 19 - Effects of modifications to pheromone PSO on number of objective function calculations for lamp model.

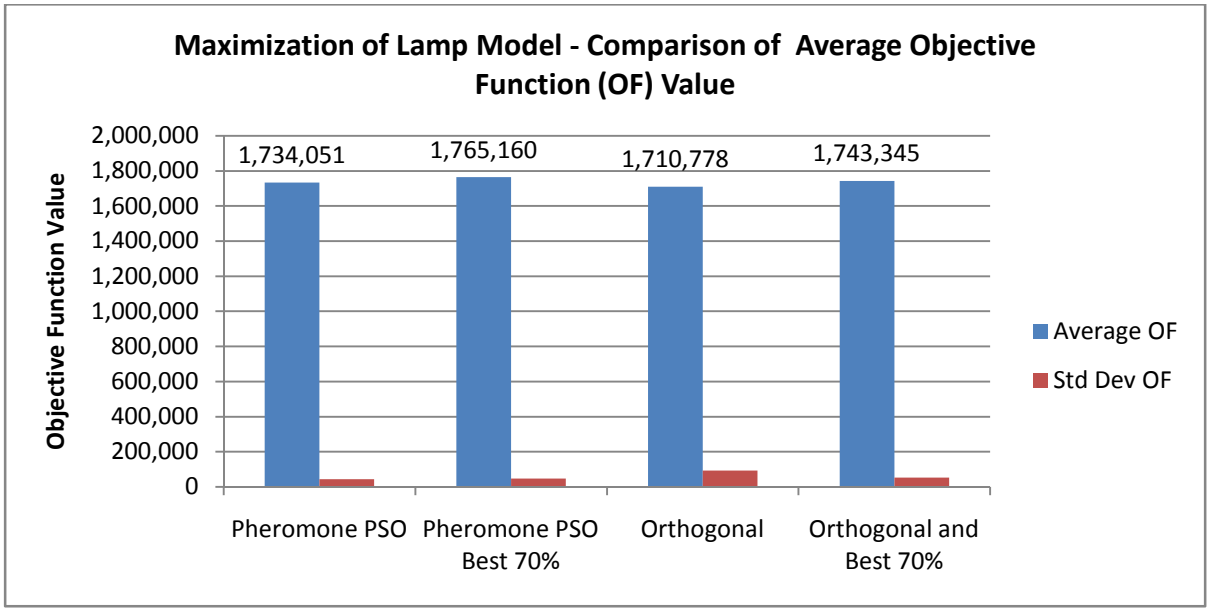


Figure 20 - Effects of modifications to pheromone PSO on maximization of objective function for lamp model.

Above, Figure 19 and Figure 20 show that using pheromone PSO with the best 70% of the particles releasing pheromones converged in the shortest amount of time and to the best solution. While pheromone PSO with the best 70% of particles releasing pheromones had a higher average solution using a 95% C.I., there is no statistical difference between the four tests.

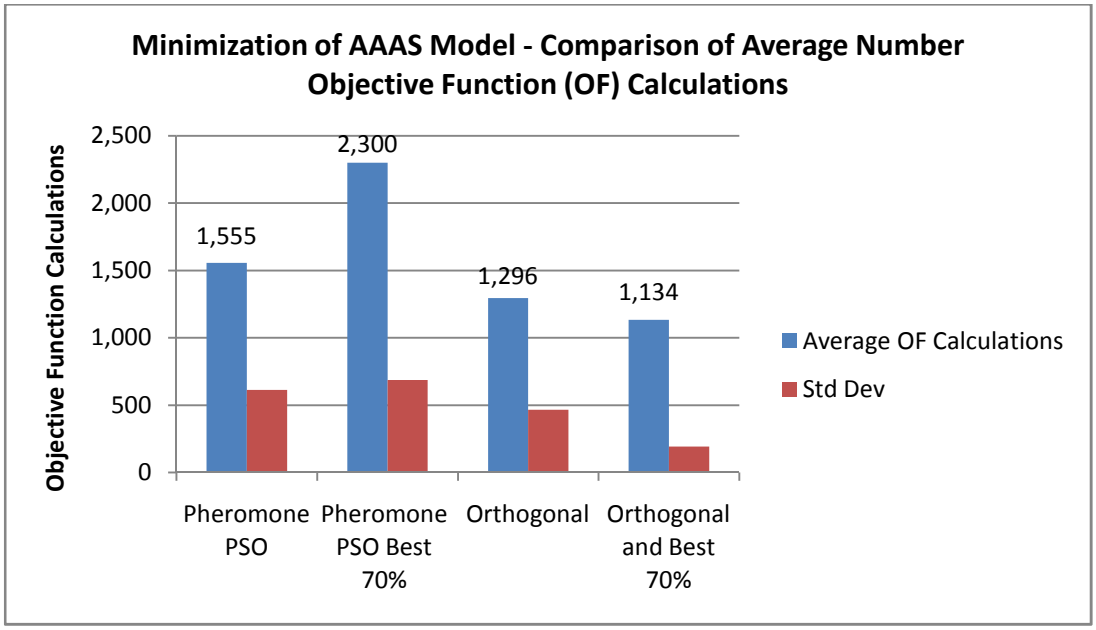


Figure 21 - Effects of modifications to pheromone PSO on number of objective function calculations for AAAS model.

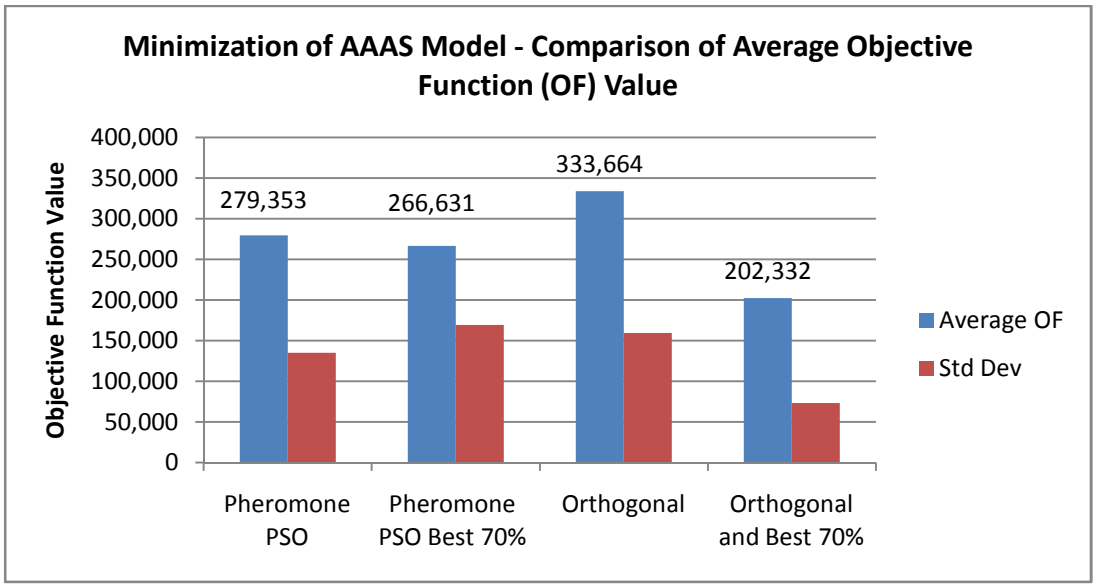


Figure 22 - Effects of modifications to pheromone PSO on minimization of objective function for AAAS model.

Above, Figure 21 and Figure 22 show that when using an orthogonal array to place the initial particles and having the best 70% of the particles release a pheromone, the

algorithm converged in the shortest amount of time and to the best solution. It is noteworthy that when using a 95% C.I. there is not a statistical difference between the orthogonal test and the pheromone tests. An example of the statistical comparison can be seen in Appendix VIII.

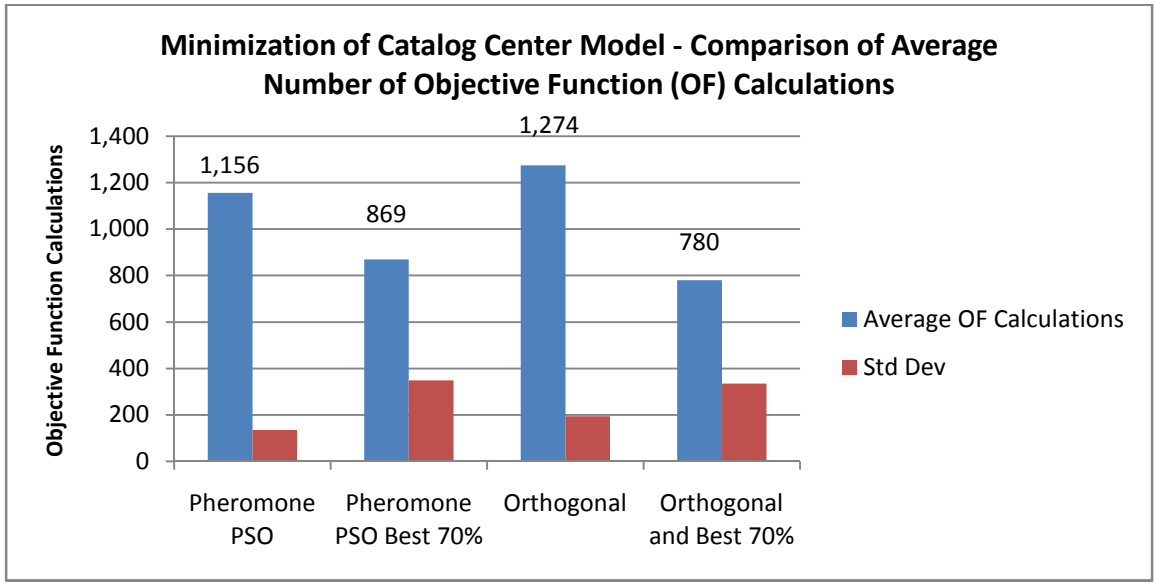


Figure 23 - Effects of modifications to pheromone PSO on number of objective function calculations for catalog center model.

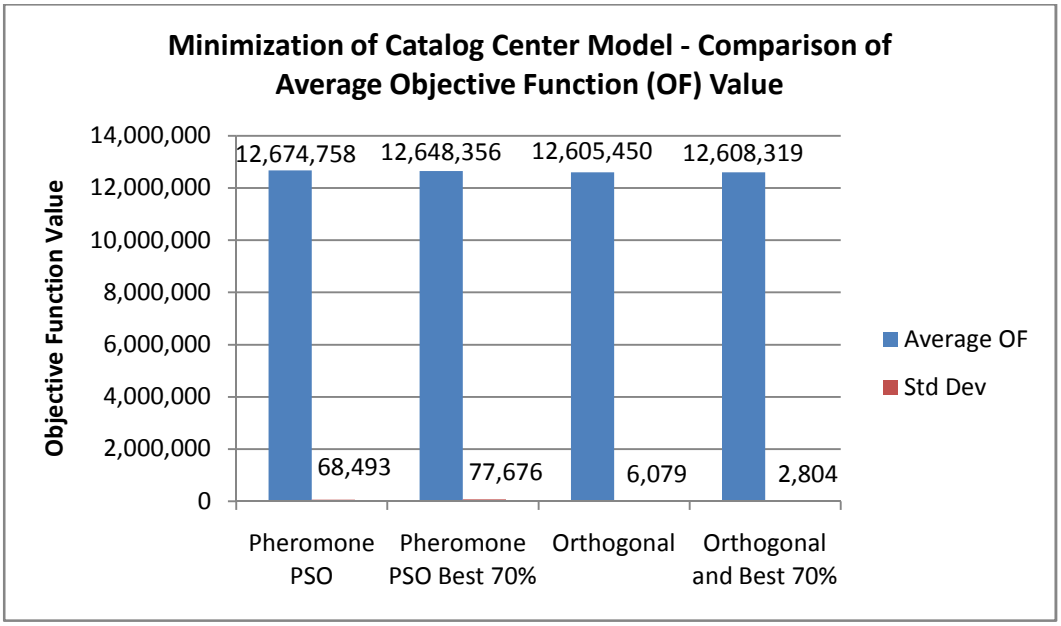


Figure 24 - Effects of modifications to pheromone PSO on minimization of objective function for catalog center model.

Above, Figure 23 and Figure 24 show that using an orthogonal array and having the best 70% of the particles releasing pheromones converged in the shortest amount of time. However, the orthogonal array method had the best average solution. There is not a statistical difference between the orthogonal test and the pheromone tests.

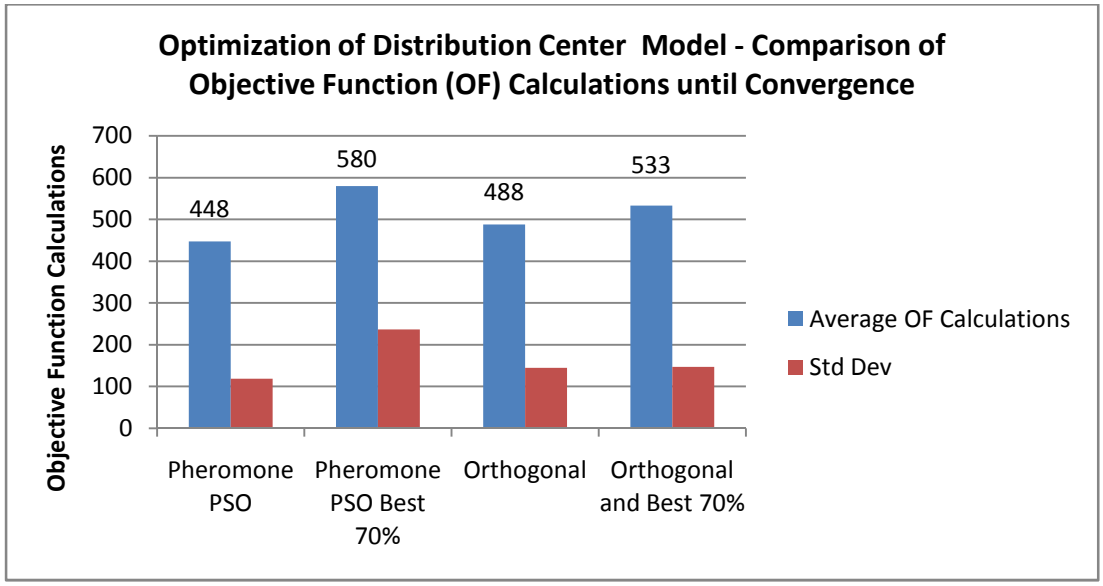


Figure 25 - Effects of modifications to pheromone PSO on number of objective function calculations for distribution center model.

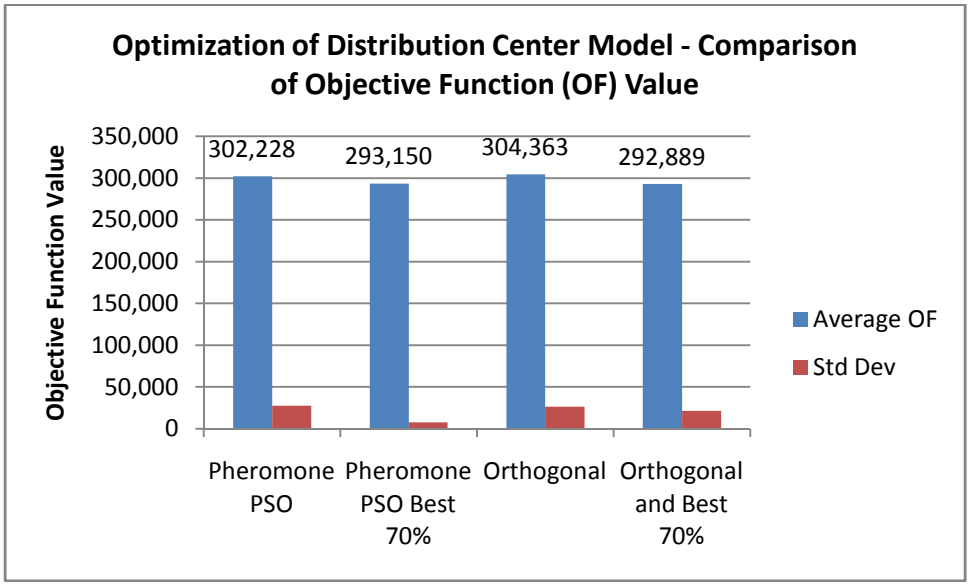


Figure 26 - Effects of modifications to pheromone PSO on minimization of objective function for distribution center model.

Above, Figure 25 and Figure 26 show that using pheromone PSO with random particles releasing pheromones converged in the shortest amount of time, and the pheromone PSO with orthogonal starting locations and the best 70% of the particles releasing

pheromones had the best average solution. There is not a statistical difference between the four tests.

4.4.1 Summary of Comparison

As illustrated by Figure 19 – Figure 26, utilizing orthogonal arrays and biasing the release of pheromones to particles that have a comparatively good objective function value has a positive impact on the quality of the solution for some models. In none of the cases was the objective function in one method found to be statistically different and better than pheromone PSO with random particles releasing pheromones. There was no clear best method displaying a consistent best time to convergence. Three of the methods had the smallest number of objective function calculations on at least one of the models tested.

CHAPTER 5. CONCLUSIONS AND FUTURE WORK

This paper tested three items: the performance of pheromone PSO vs. PSO, the performance of pheromone PSO vs. OptQuest, and the performance benefit of algorithm adjustments. This paper showed that adding digital pheromones to PSO is a constructive addition to PSO. Pheromone PSO was compared to OptQuest, and pheromone PSO was able to outperform on all models compared. This highlights the robustness of the algorithm. An additional benefit that pheromone PSO had over OptQuest was that pheromone PSO was able to use more controls as decision variables. OptQuest only allows users to adjust resource levels and variables as decision variables, whereas with proper coding in pheromone PSO, an additional production line can be added as a decision variable. Finally, two modifications that adjusted the start of the algorithm were tested, and it was shown that these modifications were not able to consistently benefit the algorithm.

Possible extensions of this paper include testing more models to see if the same trends hold true and testing how pheromone PSO works on models with more than 50 variables. Additionally, testing the functionality and usability of using pheromone PSO to control major adjustments to modeled production systems could prove beneficial, such as in cases where complete revisions to production lines are decision variables. These tests would give more insight on the performance of pheromone PSO with different problem types.

REFERENCES

Al-Aomar, R. (2000) Product-Mix Analysis with Discrete Event Simulation. 2000 Winter Simulation Conference, pp 1385—1392.

Alkhamis, T. and Ahmed, M. (2005) Simulation-based optimization for repairable systems using particle swarm algorithm. 2005 Winter Simulation Conference, pp 857—861.

April, J., Glover, F., Kelly, J. and Laguna, M. (2001) Simulation/optimization "real-world" applications. 2001 Winter Simulation Conference, pp 134—138.

Clerc, M. (1999) The swarm and the queen: toward a deterministic and adaptive particle swarm optimization. Evolutionary Computation, Vol. 3, pp 1957.

Dahal, K.P., Galloway, S.J., Burt, G.M., McDonald, J.R. and Hopkins, I. (2005) A case study of process facility optimization using discrete event simulation and genetic algorithm. GECCO 2005, pp 2197—2198.

Eberhart, R. and Shi, Y. (2000) Comparing inertia weights and constriction factors in particle swarm optimization. Evolutionary Computation, Vol. 1, pp 84—88.

Eberhart, R., Simpson, P. and Dobbins, R. (1996) Computational intelligence PC tools. Boston: Academic Press Professional.

El-Gallad, A., El-Hawary, M. and Sallam A. (2001) Swarming of intelligent particles for solving nonlinear constrained optimization problems. International Journal of Engineering Intelligent Systems, Vol. 9, No. 3, pp 155—163.

El-Gallad, A., El-Hawary, M., Sallam, A. and Kalas, A. (2002) Enhancing the particle swarm optimizer via proper parameter selection. 2002 IEEE Canadian Conference on Electrical & Computer Engineering, pp 792—797.

Gamardella, L. and Dorigo, M. (1996) Solving symmetric and asymmetric TSP's by ant colonies. Evolutionary Computation, pp 622—627.

Kalivarapu, V., Foo, J. and Winer, E. (2008) Improving solution characteristics of particle swarm optimization using digital pheromones, 48th AIAA/ASME/ASCE/AHS/ASC Structural Dynamics, and Materials Conference, pp 2180—2191.

Kelton, W., Sadowski, R. and Sturrock, D. (2004) Simulation with Arena. The McGraw-Hill Companies, Inc.

Kennedy, J. and Eberhart, R. (1995) Particle swarm optimization. IEEE International Conference on Neural Networks, vol. 4, pp 1942—1948.

Kleijnen, J. and Wan, J. (2006) Optimization of simulated systems: OptQuest and alternatives. Simulation Modeling Practice and Theory, Vol. 15, pp 354—362.

Konak, A. and Kulturel-Konak, S. (2005) Simulation optimization using tabu search: an empirical study. 2005 Winter Simulation Conference, pp 2686-2692.

Koyama, A., Barolli, L., Matsumoto, K. and Apduhan, B. (2004) A GA-based multi-purpose optimization algorithm for QoS routing. 18th International Conference of Advanced Information Networking and Applications, vol. 1, pp 23—28.

Joines, J., Gupta, D., Gokee, M., King, R. and Kay, M. (2002) Supply chain multi-objective simulation optimization. 2002 Winter Simulation Conference, pp 1306—1314.

Jones, M. and White, K. (2004) Stochastic approximation with simulated annealing as an approach to global discrete-event simulation optimization. 2004 Winter Simulation Conference, pp 500—507.

Lei, X. Shi, Z. (2007) The variations, combination strategies analysis of particle swarm optimization. Third International Conference on Natural Computation.

Li, Y. and Gong, S. (2003) Dynamic ant colony optimization for TSP. The International Journal of Advanced Manufacturing Technology, Vol. 22, pp 528—533.

Lu, H. and Chen, W. (2008) Self-adaptive velocity particle swarm optimization for solving constrained optimization problems. Journal of Global Optimization, vol. 41, pp 427—445.

Manz, E., Haddock, J. and Mittenthal, J. (1989) Optimization of an automated manufacturing system simulation model using simulated annealing. 1989 Winter Simulation Conference Proceedings, pp 390—395.

Meketon, M. (1987) Optimization in simulation: A survey of recent results. 1987 Winter Simulation Proceedings, pp 58-67.

Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A. and Teller, E. (1953) Simulated Annealing. Journal of Chemical Physics, vol. 21, pp 1087—1092.

Mo, J., Huang, M. and Wang, X. (2007) Optimal design of the real-time production control system for a general single-product assembly line based on fuzzy logic control, genetic algorithm and simulation. Fourth International Conference on Fuzzy systems and Knowledge Discovery.

Persson, A., Grimm, H., Ng, A., Lezama, T., Ekberg, J., Falk, S. and Stablum, P. (2006) Simulation-based multi-objective optimization of real-world scheduling problem. 2006 Winter Simulation Conference, pp 1757—1764.

Poli, R. (2008) Analysis of the publications on the applications of particle swarm optimization. Journal of Artificial Evolution and Applications. vol. 8.

Shi, Y. and Eberhart, R. (1998a) A modified particle swarm optimizer. Evolutionary Computation Proceedings, pp 69—73.

Shi, Y. and Eberhart, R. (1998b) Parameter selection in particle swarm optimization. Lecture Notes in Computer Science. Vol. 1447, pp 591.

The University of York. Math Department. L27 Orthogonal Array. http://www.york.ac.uk/depts/maths/tables/l27.htm. Accessed September, 29th 2008.

Tandiono, E. and Gemmill, D. (1994) Stochastic optimization of the cost of automatic assembly systems. European Journal of Operational Research, vol. 77, pp 303—313.

Veeramachaneni, K., Peram, T., Mohon, C. and Osadciw, L. (2003) Optimization using particle swarms with near neighbor interactions. GECCO 2003, pp 110—121.

Wang, B., Wang, S., Du, H. and Qu, P. (2003) Parameter optimization in complex industrial process control based on improved fuzzy-GA. International Conference on Machine Learning and Cybernetics, vol. 4, pp 2512—1215.

Wu, D., Lu, M. and Zhang, J. (2005) Efficient optimization procedures for stochastic simulation systems. Fourth International Conference on Machine Learning and Cybernetics, pp 2895—2900.

Yang, T., Kuo, Y. and Chang, I. (2004) Tabu-search simulation optimization approach for flow-shop scheduling with multiple processors — a case study. International Journal of Production Research, 42:19, 4015—4030.

Yun, I. and Park, B. (2006) Application of stochastic optimization method for an urban corridor. 2006 Winter Simulation Conference, pp 1493—1499.

Zhang, H. (2008) Multi-objective simulation-optimization for earthmoving operations. Automation in Construction, vol. 18, pp 79-86.

Zhang, L., Yu, H. and Hu, S. (2005) Optimal choice of parameters for particle swarm optimization. Journal of Zhenjiang University: Science, Vol. 6, pp. 528—534.

Zheng, Y., Ma, L., Zhang, L. and Quin, J. (2003) On the convergence analysis and parameter selection in particle swarm optimization. Second International Conference on Machine Learning and Cybernetics, pp 1802—1807.

APPENDIX I – L27 ARRAY

Table 3 - L27 array from The University of York

Run	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	X11	X12	X13
1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	2	2	2	2	2	2	2	2	2
3	1	1	1	1	3	3	3	3	3	3	3	3	3
4	1	2	2	2	1	1	1	2	2	2	3	3	3
5	1	2	2	2	2	2	2	3	3	3	1	1	1
6	1	2	2	2	3	3	3	1	1	1	2	2	2
7	1	3	3	3	1	1	1	3	3	3	2	2	2
8	1	3	3	3	2	2	2	1	1	1	3	3	3
9	1	3	3	3	3	3	3	2	2	2	1	1	1
10	2	1	2	3	1	2	3	1	2	3	1	2	3
11	2	1	2	3	2	3	1	2	3	1	2	3	1
12	2	1	2	3	3	1	2	3	1	2	3	1	2
13	2	2	3	1	1	2	3	2	3	1	3	1	2
14	2	2	3	1	2	3	1	3	1	2	1	2	3
15	2	2	3	1	3	1	2	1	2	3	2	3	1
16	2	3	1	2	1	2	3	3	1	2	2	3	1
17	2	3	1	2	2	3	1	1	2	3	3	1	2
18	2	3	1	2	3	1	2	2	3	1	1	2	3
19	3	1	3	2	1	3	2	1	3	2	1	3	2
20	3	1	3	2	2	1	3	2	1	3	2	1	3
21	3	1	3	2	3	2	1	3	2	1	3	2	1
22	3	2	1	3	1	3	2	2	1	3	3	2	1
23	3	2	1	3	2	1	3	3	2	1	1	3	2
24	3	2	1	3	3	2	1	1	3	2	2	1	3
25	3	3	2	1	1	3	2	3	2	1	2	1	3
26	3	3	2	1	2	1	3	1	3	2	3	2	1
27	3	3	2	1	3	2	1	2	1	3	1	3	2

APPENDIX II – SAMPLE CODE PHEROMONE PSO

```
Sub Pheromone PSO()

Dim Xmin(10) As Integer 'Array for Minimum control values
Dim Xmax(10) As Integer 'Array for Minimum control values
Dim X(100, 10) As Integer 'Array for each particles current control values
Dim F(100) As Double 'Objective Function Value for each particle
Dim V(100, 10, 1) As Single 'Current and last iterations velocity for each particle,
Dim Pbest(100, 11) As Single 'personal best location for each particle(P(i,0-10)) and
personal best objective function (P(i,11))
Dim Pheromone(1000, 11) As Single 'location (Pheromone(i,0-10)) and strength
(Pheromone(i,11))of pheromone field
Dim Gbestpoint(10) As Single 'Gbest location
Dim Gbest As Single 'Global best objective function value
Dim c As Integer ' convergence iteration counter
Dim w As Single ' weighted Inertia starting value
Dim PST As Single 'Target Pheromone's Strength
Dim PS As Single 'Pheromone Strength
Dim TP As Integer 'Target Pheromone
Dim count As Integer 'total iteration counter, counting 0
Dim phr As Single 'Percent of objective in which Pheromones are released
Dim RLF As Single 'Range Limit Factor
Dim pdf As Single ' Pheromone Decay Factor
Dim WIF As Single 'Weighted inertia factor
Dim ConCrit As Integer ' Number of interations until convergence
Dim c1 As Single 'weighted factor for global best influence
Dim c2 As Single 'weighted factor for personal best influence
Dim c3 As Single 'weighted factor for pheromone field influence
Dim N As Integer 'Number of Particles, maximum of 100 allowed based on arrays defined
earlier
Dim modelout As String 'data extracted from text file from arena
Dim Starttime As String ' Start time of the program
Dim Finishtime As String 'Completion time of the program
Dim Cp As Single 'variable to compute objective function
Dim Cf As Single 'variable to compute objective function
Dim CC As Single 'variable to compute objective function
Dim Ch As Single 'variable to compute objective function
Dim Cg As Single 'variable to compute objective function
Dim i As Integer 'counter
Dim j As Integer 'counter
Dim k As Integer 'counter
Dim l As Integer 'counter
Dim p As Integer 'counter for total number of pheromones in the system
```

```
'Dimension the variable m as a model object

Dim m As Model
Dim m1 As Model
Dim m2 As Model
Dim m3 As Model
Dim m4 As Model
Dim m5 As Model
Dim m6 As Model
Dim m7 As Model
Dim m8 As Model
Dim m9 As Model
Dim m10 As Model

'Set m equal to this models object

Set m = ActiveModel

'm1-m10 are used to define various submodels in the program
Set m1 = m.Submodels(m.Submodels.Find(smFindTag, "object.S1")).Model
Set m2 = m.Submodels(m.Submodels.Find(smFindTag, "object.S2")).Model
Set m3 = m.Submodels(m.Submodels.Find(smFindTag, "object.S3")).Model
Set m4 = m.Submodels(m.Submodels.Find(smFindTag, "object.S4")).Model
Set m5 = m.Submodels(m.Submodels.Find(smFindTag, "object.S5")).Model
Set m6 = m.Submodels(m.Submodels.Find(smFindTag, "object.S6")).Model
Set m7 = m.Submodels(m.Submodels.Find(smFindTag, "object.S7")).Model
Set m8 = m.Submodels(m.Submodels.Find(smFindTag, "object.S8")).Model
Set m9 = m.Submodels(m.Submodels.Find(smFindTag, "object.S9")).Model
Set m10 = m.Submodels(m.Submodels.Find(smFindTag, "object.S10")).Model
Model.QuietMode = True

'Set start time of model

   Starttime = Format(Time, "Long Time")

'Set starting values for various parameters

   RLF = 1
   phr = 0.5
   pdf = 0.95
   c1 = 2
   c2 = 2
   c3 = 4
   WIF = 0.95
   ConCrit = 5
```

```
count = 0
c = 1
w = 1
p = 0

'Enter population size less than or equal to 100

N = 20

'Set starting Gbest value as a large value for minimization problem

Gbest = 100000000

'Set min and max values for all model controls

Xmax(0) = 80
Xmin(0) = 1
Xmax(1) = 8
Xmin(1) = 1
Xmax(2) = 8
Xmin(2) = 1
Xmax(3) = 8
Xmin(3) = 1
Xmax(4) = 8
Xmin(4) = 1
Xmax(5) = 8
Xmin(5) = 1
Xmax(6) = 8
Xmin(6) = 1
Xmax(7) = 8
Xmin(7) = 1
Xmax(8) = 8
Xmin(8) = 1
Xmax(9) = 8
Xmin(9) = 1
Xmax(10) = 8
Xmin(10) = 1

'Set the starting value for all particles as a random location between controls
minimum and maximum

For i = 0 To N
   For j = 0 To 10
      X(i, j) = Xmin(j) + Rnd() * (Xmax(j) - Xmin(j))
   Next
Next
```

```
'Calculate the objective function for all particles

   For i = 0 To N
'Based on particles location, adjusts controls in Arena

 m1.Modules(m1.Modules.Find(smFindTag, "object.1a")).Data("Value") = "NQ(Station 2
Process.Queue) < " & X(i, 1)
m1.Modules(m1.Modules.Find(smFindTag, "object.1b")).Data("Expression") = X(i, 1)
m2.Modules(m2.Modules.Find(smFindTag, "object.2a")).Data("Value") = "NQ(Station 3
Process.Queue) < " & X(i, 2)
m2.Modules(m2.Modules.Find(smFindTag, "object.2b")).Data("Expression") = X(i, 2)
m3.Modules(m3.Modules.Find(smFindTag, "object.3a")).Data("Value") = "NQ(Station 4
Process.Queue) < " & X(i, 3)
m3.Modules(m3.Modules.Find(smFindTag, "object.3b")).Data("Expression") = X(i, 3)
m4.Modules(m4.Modules.Find(smFindTag, "object.4a")).Data("Value") = "NQ(Station 5
Process.Queue) < " & X(i, 4)
m4.Modules(m4.Modules.Find(smFindTag, "object.4b")).Data("Expression") = X(i, 4)
m5.Modules(m5.Modules.Find(smFindTag, "object.5a")).Data("Value") = "NQ(Station 6
Process.Queue) < " & X(i, 5)
m5.Modules(m5.Modules.Find(smFindTag, "object.5b")).Data("Expression") = X(i, 5)
m6.Modules(m6.Modules.Find(smFindTag, "object.6a")).Data("Value") = "NQ(Station 7
Process.Queue) < " & X(i, 6)
m6.Modules(m6.Modules.Find(smFindTag, "object.6b")).Data("Expression") = X(i, 6)
m7.Modules(m7.Modules.Find(smFindTag, "object.7a")).Data("Value") = "NQ(Station 8
Process.Queue) < " & X(i, 7)
m7.Modules(m7.Modules.Find(smFindTag, "object.7b")).Data("Expression") = X(i, 7)
m8.Modules(m8.Modules.Find(smFindTag, "object.8a")).Data("Value") = "NQ(Station 9
Process.Queue) < " & X(i, 8)
m8.Modules(m8.Modules.Find(smFindTag, "object.8b")).Data("Expression") = X(i, 8)
m9.Modules(m9.Modules.Find(smFindTag, "object.9a")).Data("Value") = "NQ(Station 10
Process.Queue) < " & X(i, 9)
m9.Modules(m9.Modules.Find(smFindTag, "object.9b")).Data("Expression") = X(i, 9)
m10.Modules(m10.Modules.Find(smFindTag, "object.10a")).Data("Value") = "NQ(Station 1
Process.Queue) < " & X(i, 10)
m10.Modules(m10.Modules.Find(smFindTag, "object.10b")).Data("Expression") = X(i, 10)
m1.Modules(m1.Modules.Find(smFindTag, "object.56")).Data("Batch Size") = X(i, 0)
m2.Modules(m2.Modules.Find(smFindTag, "Clear Jam 2")).Data("Expression") =
"EXPO(36," & CInt(Rnd() * 10) & ")"
m4.Modules(m4.Modules.Find(smFindTag, "Clear Jam 4")).Data("Expression") =
"EXPO(36," & CInt(Rnd() * 10) & ")"
m8.Modules(m8.Modules.Find(smFindTag, "Clear Jam 8")).Data("Expression") =
"EXPO(36," & CInt(Rnd() * 10) & ")"

'runs the model

        m.Go
```

```
'Waits until the model is finished running

        If m.SIMAN.RunMaximumReplications = m.SIMAN.RunCurrentReplication
        Then m.End
        End If

'Collected output data from text files and sets as modelout

    Open "C:\ AASOutput.txt" For Input As #1
      Line Input #1, modelout
    Close #1
'Based on results from model calculates objective function

    Cp = 500 * CSng(X(i, 0)) * 0.1627 '.1627 is the A/P factor that spreads the cost of the
pallets over 10 years
    ConveyorLength = CSng(CInt(X(i, 1)) + CInt(X(i, 2)) + CInt(X(i, 3)) + CInt(X(i, 4)) +
CInt(X(i, 5)) + CInt(X(i, 6)) + CInt(X(i, 7)) + CInt(X(i, 8)) + CInt(X(i, 9)) + CInt(X(i, 10)))
    Cf = CSng(1500 * (ConveyorLength + 10) * (0.2259 + 0.0314 * (ConveyorLength +
10)))
    CC = CSng((ConveyorLength + 10) * 15000 * 0.1627) '.1627 is the A/P factor that
spreads the cost of the pallets over 10 years
    Ch = 15000 + 100 * CInt(X(i, 0)) * 0.1
    Cg = 52 * 4 * 10 * (2200 - CSng(modelout)) 'demand is a 6 second cycle time, and $10
per unit

    F(i) = Cp + Cf + CC + Ch + Cg

'Since this is the first iteration sets the current location as the particles best
location

    Pbest(i, 11) = F(i)
    Pbest(i, 0) = X(i, 0)
    Pbest(i, 1) = X(i, 1)
    Pbest(i, 2) = X(i, 2)
    Pbest(i, 3) = X(i, 3)
    Pbest(i, 4) = X(i, 4)
    Pbest(i, 5) = X(i, 5)
    Pbest(i, 6) = X(i, 6)
    Pbest(i, 7) = X(i, 7)
    Pbest(i, 8) = X(i, 8)
    Pbest(i, 9) = X(i, 9)
    Pbest(i, 10) = X(i, 10)
'Checks to see if this particles current location is better than the current global
best, if it is better sets current particles location as gbest

    If Pbest(i, 11) < Gbest Then
```

```
            Gbest = Pbest(i, 11)
            Gbestpoint(0) = X(i, 0)
            Gbestpoint(1) = X(i, 1)
            Gbestpoint(2) = X(i, 2)
            Gbestpoint(3) = X(i, 3)
            Gbestpoint(4) = X(i, 4)
            Gbestpoint(5) = X(i, 5)
            Gbestpoint(6) = X(i, 6)
            Gbestpoint(7) = X(i, 7)
            Gbestpoint(8) = X(i, 8)
            Gbestpoint(9) = X(i, 9)
            Gbestpoint(10) = X(i, 10)
'Records in a test file when a new gbest is found

            Open "C:\ModelRun.txt" For Append As #1
               Print #1, "New G Best"; Tab; count; Tab; Gbest
            Close #1
         End If
      Next

'Random drops pheromones for (PHR)% of the population

      For i = 0 To N
         If Rnd() < phr Then
            'drop pheromone
            Pheromone(p, 0) = X(i, 0)
            Pheromone(p, 1) = X(i, 1)
            Pheromone(p, 2) = X(i, 2)
            Pheromone(p, 3) = X(i, 3)
            Pheromone(p, 4) = X(i, 4)
            Pheromone(p, 5) = X(i, 5)
            Pheromone(p, 6) = X(i, 6)
            Pheromone(p, 7) = X(i, 7)
            Pheromone(p, 8) = X(i, 8)
            Pheromone(p, 9) = X(i, 9)
            Pheromone(p, 10) = X(i, 10)
            'pheromone strength
            Pheromone(p, 11) = 1
'Marks counter p that another pheromone has been dropped

            p = p + 1
         End If
      Next
'Sets the starting velocity for all particles as 0

      For i = 0 To N
```

```
      For j = 0 To 10
         V(i, j, 0) = 0
      Next
   Next
'This completes the start up of the algorithm


'Algorithm then continues until ConCrit number of iterations pass without a
change in the objective function value

   Do Until c > ConCrit
'Updates iterations counters

      count = count + 1
      c = c + 1
'Sets target pheromones for all particles

      For i = 0 To N
         PS = 0
         PST = -1000
         For j = 0 To p - 1
'Calculates the -distance*strength between each particle and each pheromone

            PS = (1 - ((X(i, 0) - Pheromone(j, 0)) ^ 2 + (X(i, 1) - Pheromone(j, 1)) ^ 2 + (X(i,
2) - Pheromone(j, 2)) ^ 2 + (X(i, 3) - Pheromone(j, 3)) ^ 2 + (X(i, 4) - Pheromone(j, 4)) ^ 2 +
(X(i, 5) - Pheromone(j, 5)) ^ 2 + (X(i, 6) - Pheromone(j, 6)) ^ 2 + (X(i, 7) - Pheromone(j, 7))
^ 2 + (X(i, 8) - Pheromone(j, 8)) ^ 2 + (X(i, 9) - Pheromone(j, 9)) ^ 2 + (X(i, 10) -
Pheromone(j, 10)) ^ 2) ^ 0.5) * Pheromone(j, 11)
'If a pheromone is found to have a better strength it is selected as the target
pheromone

            If PS > PST Then
               PST = PS
               TP = j
            End If
         Next
'Updates the velocity vector

         For j = 0 To 10
            V(i, j, 1) = w * V(i, j, 0) + c1 * Rnd() * (Pbest(i, 1) - X(i, j)) + c2 * Rnd() *
(Gbestpoint(j) - X(i, j)) + c3 * Rnd() * (Pheromone(TP, j) - X(i, j))
'If velocity is greater than the variable range * RLF, the velocity is set to the
variable range * RLF

            If V(i, j, 1) > (Xmax(j) - Xmin(j) * RLF) Then
               V(i, j, 1) = (Xmax(j) - Xmin(j) * RLF)
            End If
```

```
'If velocity is less than the variable range * RLF, the velocity is set to the
variable range * RLF

            If V(i, j, 1) < -(Xmax(j) - Xmin(j) * RLF) Then
               V(i, j, 1) = -(Xmax(j) - Xmin(j) * RLF)
            End If
            V(i, j, 0) = V(i, j, 1)
         Next
      Next
'Adjusts particles position based on velocity

      For i = 0 To N
         For j = 0 To 10
            X(i, j) = X(i, j) + V(i, j, 1)
            If X(i, j) < Xmin(j) Then
               X(i, j) = Xmin(j)
            End If
         Next
      Next

For i = 0 To N
'Based on particles location, adjusts controls in Arena

m1.Modules(m1.Modules.Find(smFindTag, "object.1a")).Data("Value") = "NQ(Station 2
Process.Queue) < " & X(i, 1)
m1.Modules(m1.Modules.Find(smFindTag, "object.1b")).Data("Expression") = X(i, 1)
m2.Modules(m2.Modules.Find(smFindTag, "object.2a")).Data("Value") = "NQ(Station 3
Process.Queue) < " & X(i, 2)
m2.Modules(m2.Modules.Find(smFindTag, "object.2b")).Data("Expression") = X(i, 2)
m3.Modules(m3.Modules.Find(smFindTag, "object.3a")).Data("Value") = "NQ(Station 4
Process.Queue) < " & X(i, 3)
m3.Modules(m3.Modules.Find(smFindTag, "object.3b")).Data("Expression") = X(i, 3)
m4.Modules(m4.Modules.Find(smFindTag, "object.4a")).Data("Value") = "NQ(Station 5
Process.Queue) < " & X(i, 4)
m4.Modules(m4.Modules.Find(smFindTag, "object.4b")).Data("Expression") = X(i, 4)
m5.Modules(m5.Modules.Find(smFindTag, "object.5a")).Data("Value") = "NQ(Station 6
Process.Queue) < " & X(i, 5)
 m5.Modules(m5.Modules.Find(smFindTag, "object.5b")).Data("Expression") = X(i, 5)
m6.Modules(m6.Modules.Find(smFindTag, "object.6a")).Data("Value") = "NQ(Station 7
Process.Queue) < " & X(i, 6)
m6.Modules(m6.Modules.Find(smFindTag, "object.6b")).Data("Expression") = X(i, 6)
m7.Modules(m7.Modules.Find(smFindTag, "object.7a")).Data("Value") = "NQ(Station 8
Process.Queue) < " & X(i, 7)
m7.Modules(m7.Modules.Find(smFindTag, "object.7b")).Data("Expression") = X(i, 7)
m8.Modules(m8.Modules.Find(smFindTag, "object.8a")).Data("Value") = "NQ(Station 9
Process.Queue) < " & X(i, 8)
```

```
m8.Modules(m8.Modules.Find(smFindTag, "object.8b")).Data("Expression") = X(i, 8)
m9.Modules(m9.Modules.Find(smFindTag, "object.9a")).Data("Value") = "NQ(Station 10
Process.Queue) < " & X(i, 9)
m9.Modules(m9.Modules.Find(smFindTag, "object.9b")).Data("Expression") = X(i, 9)
m10.Modules(m10.Modules.Find(smFindTag, "object.10a")).Data("Value") = "NQ(Station 1
Process.Queue) < " & X(i, 10)
m10.Modules(m10.Modules.Find(smFindTag, "object.10b")).Data("Expression") = X(i, 10)
m1.Modules(m1.Modules.Find(smFindTag, "object.56")).Data("Batch Size") = X(i, 0)
m2.Modules(m2.Modules.Find(smFindTag, "Clear Jam 2")).Data("Expression") =
"EXPO(36," & CInt(Rnd() * 10) & ")"
m4.Modules(m4.Modules.Find(smFindTag, "Clear Jam 4")).Data("Expression") =
"EXPO(36," & CInt(Rnd() * 10) & ")"
m8.Modules(m8.Modules.Find(smFindTag, "Clear Jam 8")).Data("Expression") =
"EXPO(36," & CInt(Rnd() * 10) & ")"
'Runs the model

        m.Go
'Waits until the model is finished running

 If m.SIMAN.RunMaximumReplications = m.SIMAN.RunCurrentReplication Then
        m.End
End If
'Collected output data from text files and sets as modelout

      Open "C:\Thesis\AASOutput.txt" For Input As #1
        Line Input #1, modelout
      Close #1
'Based on results from model calculates objective function

      Cp = 500 * CSng(X(i, 0)) * 0.1627 '.1627 is the A/P factor that spreads the cost of the
pallets over 10 years
      ConveyorLength = CSng(CInt(X(i, 1)) + CInt(X(i, 2)) + CInt(X(i, 3)) + CInt(X(i, 4)) +
CInt(X(i, 5)) + CInt(X(i, 6)) + CInt(X(i, 7)) + CInt(X(i, 8)) + CInt(X(i, 9)) + CInt(X(i, 10)))
      Cf = CSng(1500 * (ConveyorLength + 10) * (0.2259 + 0.0314 * (ConveyorLength +
10)))
      CC = CSng((ConveyorLength + 10) * 15000 * 0.1627) '.1627 is the A/P factor that
spreads the cost of the pallets over 10 years
      Ch = 15000 + 100 * CInt(X(i, 0)) * 0.1
      Cg = 52 * 4 * 10 * (2200 - CSng(modelout)) 'demand is an 6 second cycle time, and $10
per unit

      F(i) = Cp + Cf + CC + Ch + Cg

        'If the current particles location is greater than that particles personal
best and particle is feasible, then pbest is updated

        If F(i) < Pbest(i, 11) Then
```

```
For j = 0 To 10

    'Checks if current particle is feasible

  If X(i, j) > Xmax(j) Then
    Exit For
  End If
  If X(i, j) < Xmin(j) Then
    Exit For
  End If
  If j = 10 Then
    Pbest(i, 11) = F(i)
    Pbest(i, 0) = X(i, 0)
    Pbest(i, 1) = X(i, 1)
    Pbest(i, 2) = X(i, 2)
    Pbest(i, 3) = X(i, 3)
    Pbest(i, 4) = X(i, 4)
    Pbest(i, 5) = X(i, 5)
    Pbest(i, 6) = X(i, 6)
    Pbest(i, 7) = X(i, 7)
    Pbest(i, 8) = X(i, 8)
    Pbest(i, 9) = X(i, 9)
    Pbest(i, 10) = X(i, 10)

    'A pheromone is release each time a particle finds a new p best

    Pheromone(p, 0) = X(i, 0)
    Pheromone(p, 1) = X(i, 1)
    Pheromone(p, 2) = X(i, 2)
    Pheromone(p, 3) = X(i, 3)
    Pheromone(p, 4) = X(i, 4)
    Pheromone(p, 5) = X(i, 5)
    Pheromone(p, 6) = X(i, 6)
    Pheromone(p, 7) = X(i, 7)
    Pheromone(p, 8) = X(i, 8)
    Pheromone(p, 9) = X(i, 9)
    Pheromone(p, 10) = X(i, 10)

    'Pheromone strength

    Pheromone(p, 11) = 1
    p = p + 1

        'If the current location is also better then Global best, gbest is
updated
```

```
                If Pbest(i, 11) < Gbest Then

                    'Convergence criterion counter is updated

                  c = 1
                  Gbest = Pbest(i, 11)
                  Gbestpoint(0) = X(i, 0)
                  Gbestpoint(1) = X(i, 1)
                  Gbestpoint(2) = X(i, 2)
                  Gbestpoint(3) = X(i, 3)
                  Gbestpoint(4) = X(i, 4)
                  Gbestpoint(5) = X(i, 5)
                  Gbestpoint(6) = X(i, 6)
                  Gbestpoint(7) = X(i, 7)
                  Gbestpoint(8) = X(i, 8)
                  Gbestpoint(9) = X(i, 9)
                  Gbestpoint(10) = X(i, 10)

                    'Prints current solution to file

                  Open "C:\ModelRun.txt" For Append As #1
                     Print #1, "New G Best"; Tab; count; Tab; Gbest
                  Close #1
                End If
            End If
        Next
    End If
Next

'Before the start of the next iteration the pheromone field is decayed

For k = 0 To p
    Pheromone(k, 11) = Pheromone(k, 11) * pdf
Next

'Merges pheromones together if they are effectively overlapping

For k = 0 To p
    For l = k To p
        If k <> l Then
            If ((Pheromone(k, 0) - Pheromone(l, 0)) ^ 2 + (Pheromone(k, 1) -
            Pheromone(l, 1)) ^ 2 + (Pheromone(k, 2) - Pheromone(l, 2)) ^ 2 +
            (Pheromone(k, 3) - Pheromone(l, 3)) ^ 2 + (Pheromone(k, 4) - Pheromone(l,
            4)) ^ 2 + (Pheromone(k, 5) - Pheromone(l, 5)) ^ 2 + (Pheromone(k, 6) -
            Pheromone(l, 6)) ^ 2 + (Pheromone(k, 7) - Pheromone(l, 7)) ^ 2 +
```

```
                (Pheromone(k, 8) - Pheromone(l, 8)) ^ 2 + (Pheromone(k, 9) - Pheromone(l,
                9)) ^ 2 + (Pheromone(k, 10) - Pheromone(l, 10)) ^ 2) ^ 0.5 < 2 Then
                Pheromone(k, 11) = (Pheromone(k, 11) + Pheromone(l, 11)) / 2
                Pheromone(l, 11) = 0
            End If
        End If
    Next
Next

  'Reduced weighted inertia factor and then optimization is continued until
convergence

    w = w * WIF
  Loop
'Once the convergence criterion, the finish time is recorded

Finishtime = Format(Time, "Long Time")
'Run statistics and best solution found is recorded to a file

Open "C:\OptimalOut.txt" For Append As #1
  Print #1, "Run length of " & count
  Print #1, Gbest
  Print #1, Gbestpoint(0)
  Print #1, Gbestpoint(1)
  Print #1, Gbestpoint(2)
  Print #1, Gbestpoint(3)
  Print #1, Gbestpoint(4)
  Print #1, Gbestpoint(5)
  Print #1, Gbestpoint(6)
  Print #1, Gbestpoint(7)
  Print #1, Gbestpoint(8)
  Print #1, Gbestpoint(9)
  Print #1, Gbestpoint(10)
  Print #1, Starttime
  Print #1, Finishtime
Close #1

End Sub
```

APPENDIX III – SAMPLE CODE PSO

```
Sub PSO()

Dim Xmin(10) As Integer 'Array for Minimum control values
Dim Xmax(10) As Integer 'Array for Minimum control values
Dim X(100, 10) As Integer 'Array for each particles current control values
Dim F(100) As Double 'Objective Function Value for each particle
Dim V(100, 10, 1) As Single 'Current and last iterations velocity for each particle,
Dim Pbest(100, 11) As Single 'personal best location for each particle(P(i,0-10)) and
personal best objective function (P(i,11))
Dim Gbestpoint(10) As Single 'Gbest location
Dim Gbest As Single 'Global best objective function value
Dim c As Integer ' convergence iteration counter
Dim w As Single ' weighted Inertia starting value
Dim count As Integer 'total iteration counter, counting 0
Dim RLF As Single 'Range Limit Factor
Dim WIF As Single 'Weighted inertia factor
Dim ConCrit As Integer ' Number of interations until convergence
Dim c1 As Single 'weighted factor for global best influence
Dim c2 As Single 'weighted factor for personal best influence
Dim c3 As Single 'weighted factor for pheromone field influence
Dim N As Integer 'Number of Particles, maximum of 100 allowed based on arrays defined
earlier
Dim modelout As String 'data extracted from text file from arena
Dim Starttime As String ' Start time of the program
Dim Finishtime As String 'Completion time of the program
Dim Cp As Single 'variable to compute objective function
Dim Cf As Single 'variable to compute objective function
Dim CC As Single 'variable to compute objective function
Dim Ch As Single 'variable to compute objective function
Dim Cg As Single 'variable to compute objective function
Dim i As Integer 'counter
Dim j As Integer 'counter
Dim k As Integer 'counter
Dim l As Integer 'counter

'Dimension the variable m as a model object

Dim m As Model
Dim m1 As Model
Dim m2 As Model
Dim m3 As Model
Dim m4 As Model
Dim m5 As Model
```

```
Dim m6 As Model
Dim m7 As Model
Dim m8 As Model
Dim m9 As Model
Dim m10 As Model

'Set m equal to this models object

Set m = ActiveModel

'm1-m10 are used to define various submodels in the program
Set m1 = m.Submodels(m.Submodels.Find(smFindTag, "object.S1")).Model
Set m2 = m.Submodels(m.Submodels.Find(smFindTag, "object.S2")).Model
Set m3 = m.Submodels(m.Submodels.Find(smFindTag, "object.S3")).Model
Set m4 = m.Submodels(m.Submodels.Find(smFindTag, "object.S4")).Model
Set m5 = m.Submodels(m.Submodels.Find(smFindTag, "object.S5")).Model
Set m6 = m.Submodels(m.Submodels.Find(smFindTag, "object.S6")).Model
Set m7 = m.Submodels(m.Submodels.Find(smFindTag, "object.S7")).Model
Set m8 = m.Submodels(m.Submodels.Find(smFindTag, "object.S8")).Model
Set m9 = m.Submodels(m.Submodels.Find(smFindTag, "object.S9")).Model
Set m10 = m.Submodels(m.Submodels.Find(smFindTag, "object.S10")).Model
Model.QuietMode = True

'Set start time of model

   Starttime = Format(Time, "Long Time")

'Set starting values for various parameters

   RLF = 1
   c1 = 2
   c2 = 2
   c3 = 4
   WIF = 0.95
   ConCrit = 5
   count = 0
   c = 1
   w = 1
   p = 0

'Enter population size less than or equal to 100

   N = 20

'Set starting Gbest value as a large value for minimization problem

   Gbest = 100000000
```

```
'Set min and max values for all model controls

   Xmax(0) = 80
   Xmin(0) = 1
   Xmax(1) = 8
   Xmin(1) = 1
   Xmax(2) = 8
   Xmin(2) = 1
   Xmax(3) = 8
   Xmin(3) = 1
   Xmax(4) = 8
   Xmin(4) = 1
   Xmax(5) = 8
   Xmin(5) = 1
   Xmax(6) = 8
   Xmin(6) = 1
   Xmax(7) = 8
   Xmin(7) = 1
   Xmax(8) = 8
   Xmin(8) = 1
   Xmax(9) = 8
   Xmin(9) = 1
   Xmax(10) = 8
   Xmin(10) = 1

'Set the starting value for all particles as a random location between controls
minimum and maximum

   For i = 0 To N
      For j = 0 To 10
         X(i, j) = Xmin(j) + Rnd() * (Xmax(j) - Xmin(j))
      Next
   Next
'Calculate the objective function for all particles

   For i = 0 To N
'Based on particles location, adjusts controls in Arena

 m1.Modules(m1.Modules.Find(smFindTag, "object.1a")).Data("Value") = "NQ(Station 2
Process.Queue) < " & X(i, 1)
m1.Modules(m1.Modules.Find(smFindTag, "object.1b")).Data("Expression") = X(i, 1)
m2.Modules(m2.Modules.Find(smFindTag, "object.2a")).Data("Value") = "NQ(Station 3
Process.Queue) < " & X(i, 2)
m2.Modules(m2.Modules.Find(smFindTag, "object.2b")).Data("Expression") = X(i, 2)
```

```
m3.Modules(m3.Modules.Find(smFindTag, "object.3a")).Data("Value") = "NQ(Station 4
Process.Queue) < " & X(i, 3)
m3.Modules(m3.Modules.Find(smFindTag, "object.3b")).Data("Expression") = X(i, 3)
m4.Modules(m4.Modules.Find(smFindTag, "object.4a")).Data("Value") = "NQ(Station 5
Process.Queue) < " & X(i, 4)
m4.Modules(m4.Modules.Find(smFindTag, "object.4b")).Data("Expression") = X(i, 4)
m5.Modules(m5.Modules.Find(smFindTag, "object.5a")).Data("Value") = "NQ(Station 6
Process.Queue) < " & X(i, 5)
m5.Modules(m5.Modules.Find(smFindTag, "object.5b")).Data("Expression") = X(i, 5)
m6.Modules(m6.Modules.Find(smFindTag, "object.6a")).Data("Value") = "NQ(Station 7
Process.Queue) < " & X(i, 6)
m6.Modules(m6.Modules.Find(smFindTag, "object.6b")).Data("Expression") = X(i, 6)
m7.Modules(m7.Modules.Find(smFindTag, "object.7a")).Data("Value") = "NQ(Station 8
Process.Queue) < " & X(i, 7)
m7.Modules(m7.Modules.Find(smFindTag, "object.7b")).Data("Expression") = X(i, 7)
m8.Modules(m8.Modules.Find(smFindTag, "object.8a")).Data("Value") = "NQ(Station 9
Process.Queue) < " & X(i, 8)
m8.Modules(m8.Modules.Find(smFindTag, "object.8b")).Data("Expression") = X(i, 8)
m9.Modules(m9.Modules.Find(smFindTag, "object.9a")).Data("Value") = "NQ(Station 10
Process.Queue) < " & X(i, 9)
m9.Modules(m9.Modules.Find(smFindTag, "object.9b")).Data("Expression") = X(i, 9)
m10.Modules(m10.Modules.Find(smFindTag, "object.10a")).Data("Value") = "NQ(Station 1
Process.Queue) < " & X(i, 10)
m10.Modules(m10.Modules.Find(smFindTag, "object.10b")).Data("Expression") = X(i, 10)
m1.Modules(m1.Modules.Find(smFindTag, "object.56")).Data("Batch Size") = X(i, 0)
m2.Modules(m2.Modules.Find(smFindTag, "Clear Jam 2")).Data("Expression") =
"EXPO(36," & CInt(Rnd() * 10) & ")"
m4.Modules(m4.Modules.Find(smFindTag, "Clear Jam 4")).Data("Expression") =
"EXPO(36," & CInt(Rnd() * 10) & ")"
m8.Modules(m8.Modules.Find(smFindTag, "Clear Jam 8")).Data("Expression") =
"EXPO(36," & CInt(Rnd() * 10) & ")"

'runs the model

        m.Go

'Waits until the model is finished running

If m.SIMAN.RunMaximumReplications = m.SIMAN.RunCurrentReplication Then
        m.End

End If

'Collected output data from text files and sets as modelout

      Open "C:\ AASOutput.txt" For Input As #1
```

```
    Line Input #1, modelout
    Close #1
'Based on results from model calculates objective function

    Cp = 500 * CSng(X(i, 0)) * 0.1627 '.1627 is the A/P factor that spreads the cost of the pallets over 10 years
    ConveyorLength = CSng(CInt(X(i, 1)) + CInt(X(i, 2)) + CInt(X(i, 3)) + CInt(X(i, 4)) + CInt(X(i, 5)) + CInt(X(i, 6)) + CInt(X(i, 7)) + CInt(X(i, 8)) + CInt(X(i, 9)) + CInt(X(i, 10)))
    Cf = CSng(1500 * (ConveyorLength + 10) * (0.2259 + 0.0314 * (ConveyorLength + 10)))
    CC = CSng((ConveyorLength + 10) * 15000 * 0.1627) '.1627 is the A/P factor that spreads the cost of the pallets over 10 years
    Ch = 15000 + 100 * CInt(X(i, 0)) * 0.1
    Cg = 52 * 4 * 10 * (2200 - CSng(modelout)) 'demand is a 6 second cycle time, and $10 per unit

    F(i) = Cp + Cf + CC + Ch + Cg

'Since this is the first iteration sets the current location as the particles best location

    Pbest(i, 11) = F(i)
    Pbest(i, 0) = X(i, 0)
    Pbest(i, 1) = X(i, 1)
    Pbest(i, 2) = X(i, 2)
    Pbest(i, 3) = X(i, 3)
    Pbest(i, 4) = X(i, 4)
    Pbest(i, 5) = X(i, 5)
    Pbest(i, 6) = X(i, 6)
    Pbest(i, 7) = X(i, 7)
    Pbest(i, 8) = X(i, 8)
    Pbest(i, 9) = X(i, 9)
    Pbest(i, 10) = X(i, 10)
'Checks to see if this particles current location is better than the current global best, if it is better sets current particles location as gbest

    If Pbest(i, 11) < Gbest Then
      Gbest = Pbest(i, 11)
      Gbestpoint(0) = X(i, 0)
      Gbestpoint(1) = X(i, 1)
      Gbestpoint(2) = X(i, 2)
      Gbestpoint(3) = X(i, 3)
      Gbestpoint(4) = X(i, 4)
      Gbestpoint(5) = X(i, 5)
      Gbestpoint(6) = X(i, 6)
      Gbestpoint(7) = X(i, 7)
```

```
            Gbestpoint(8) = X(i, 8)
            Gbestpoint(9) = X(i, 9)
            Gbestpoint(10) = X(i, 10)
'Records in a test file when a new gbest is found

            Open "C:\ModelRun.txt" For Append As #1
               Print #1, "New G Best"; Tab; count; Tab; Gbest
            Close #1
         End If
      Next

'Sets the starting velocity for all particles as 0

      For i = 0 To N
         For j = 0 To 10
            V(i, j, 0) = 0
         Next
      Next
'This completes the start up of the algorithm


'Algorithm then continues until ConCrit number of iterations pass without a
change in the objective function value

      Do Until c > ConCrit
'Updates iterations counters

         count = count + 1
         c = c + 1
'Updates the velocity vector

            For j = 0 To 10
               V(i, j, 1) = w * V(i, j, 0) + c1 * Rnd() * (Pbest(i, 1) - X(i, j)) + c2 * Rnd() *
(Gbestpoint(j) - X(i, j))
'If velocity is greater than the variable range * RLF, the velocity is set to the
variable range * RLF

               If V(i, j, 1) > (Xmax(j) - Xmin(j) * RLF) Then
                  V(i, j, 1) = (Xmax(j) - Xmin(j) * RLF)
               End If
'If velocity is less than the variable range * RLF, the velocity is set to the
variable range * RLF

               If V(i, j, 1) < -(Xmax(j) - Xmin(j) * RLF) Then
                  V(i, j, 1) = -(Xmax(j) - Xmin(j) * RLF)
               End If
               V(i, j, 0) = V(i, j, 1)
```

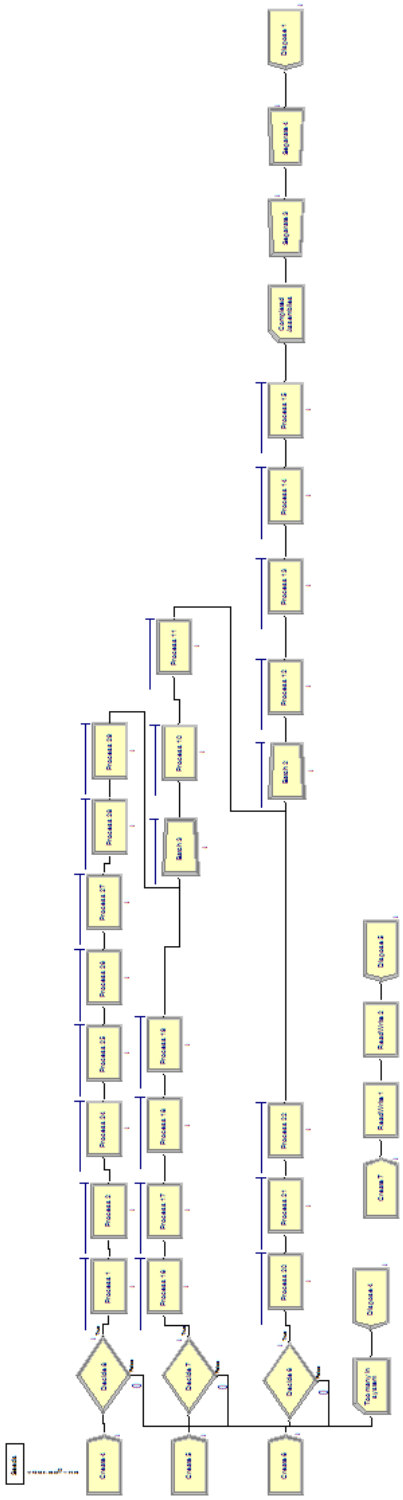
```
        Next
    Next
'Adjusts particles position based on velocity

    For i = 0 To N
        For j = 0 To 10
            X(i, j) = X(i, j) + V(i, j, 1)
            If X(i, j) < Xmin(j) Then
                X(i, j) = Xmin(j)
            End If
        Next
    Next

For i = 0 To N
'Based on particles location, adjusts controls in Arena

m1.Modules(m1.Modules.Find(smFindTag, "object.1a")).Data("Value") = "NQ(Station 2
Process.Queue) < " & X(i, 1)
m1.Modules(m1.Modules.Find(smFindTag, "object.1b")).Data("Expression") = X(i, 1)
m2.Modules(m2.Modules.Find(smFindTag, "object.2a")).Data("Value") = "NQ(Station 3
Process.Queue) < " & X(i, 2)
m2.Modules(m2.Modules.Find(smFindTag, "object.2b")).Data("Expression") = X(i, 2)
m3.Modules(m3.Modules.Find(smFindTag, "object.3a")).Data("Value") = "NQ(Station 4
Process.Queue) < " & X(i, 3)
m3.Modules(m3.Modules.Find(smFindTag, "object.3b")).Data("Expression") = X(i, 3)
m4.Modules(m4.Modules.Find(smFindTag, "object.4a")).Data("Value") = "NQ(Station 5
Process.Queue) < " & X(i, 4)
m4.Modules(m4.Modules.Find(smFindTag, "object.4b")).Data("Expression") = X(i, 4)
m5.Modules(m5.Modules.Find(smFindTag, "object.5a")).Data("Value") = "NQ(Station 6
Process.Queue) < " & X(i, 5)
 m5.Modules(m5.Modules.Find(smFindTag, "object.5b")).Data("Expression") = X(i, 5)
m6.Modules(m6.Modules.Find(smFindTag, "object.6a")).Data("Value") = "NQ(Station 7
Process.Queue) < " & X(i, 6)
m6.Modules(m6.Modules.Find(smFindTag, "object.6b")).Data("Expression") = X(i, 6)
m7.Modules(m7.Modules.Find(smFindTag, "object.7a")).Data("Value") = "NQ(Station 8
Process.Queue) < " & X(i, 7)
m7.Modules(m7.Modules.Find(smFindTag, "object.7b")).Data("Expression") = X(i, 7)
m8.Modules(m8.Modules.Find(smFindTag, "object.8a")).Data("Value") = "NQ(Station 9
Process.Queue) < " & X(i, 8)
m8.Modules(m8.Modules.Find(smFindTag, "object.8b")).Data("Expression") = X(i, 8)
m9.Modules(m9.Modules.Find(smFindTag, "object.9a")).Data("Value") = "NQ(Station 10
Process.Queue) < " & X(i, 9)
m9.Modules(m9.Modules.Find(smFindTag, "object.9b")).Data("Expression") = X(i, 9)
m10.Modules(m10.Modules.Find(smFindTag, "object.10a")).Data("Value") = "NQ(Station 1
Process.Queue) < " & X(i, 10)
m10.Modules(m10.Modules.Find(smFindTag, "object.10b")).Data("Expression") = X(i, 10)
```

```
m1.Modules(m1.Modules.Find(smFindTag, "object.56")).Data("Batch Size") = X(i, 0)
m2.Modules(m2.Modules.Find(smFindTag, "Clear Jam 2")).Data("Expression") =
"EXPO(36," & CInt(Rnd() * 10) & ")"
m4.Modules(m4.Modules.Find(smFindTag, "Clear Jam 4")).Data("Expression") =
"EXPO(36," & CInt(Rnd() * 10) & ")"
m8.Modules(m8.Modules.Find(smFindTag, "Clear Jam 8")).Data("Expression") =
"EXPO(36," & CInt(Rnd() * 10) & ")"
'Runs the model

        m.Go
'Waits until the model is finished running

 If m.SIMAN.RunMaximumReplications = m.SIMAN.RunCurrentReplication Then
        m.End
End If
'Collected output data from text files and sets as modelout

      Open "C:\Thesis\AASOutput.txt" For Input As #1
        Line Input #1, modelout
      Close #1
'Based on results from model calculates objective function

      Cp = 500 * CSng(X(i, 0)) * 0.1627 '.1627 is the A/P factor that spreads the cost of the
pallets over 10 years
      ConveyorLength = CSng(CInt(X(i, 1)) + CInt(X(i, 2)) + CInt(X(i, 3)) + CInt(X(i, 4)) +
CInt(X(i, 5)) + CInt(X(i, 6)) + CInt(X(i, 7)) + CInt(X(i, 8)) + CInt(X(i, 9)) + CInt(X(i, 10)))
      Cf = CSng(1500 * (ConveyorLength + 10) * (0.2259 + 0.0314 * (ConveyorLength +
10)))
      CC = CSng((ConveyorLength + 10) * 15000 * 0.1627) '.1627 is the A/P factor that
spreads the cost of the pallets over 10 years
      Ch = 15000 + 100 * CInt(X(i, 0)) * 0.1
      Cg = 52 * 4 * 10 * (2200 - CSng(modelout)) 'demand is an 6 second cycle time, and $10
per unit

      F(i) = Cp + Cf + CC + Ch + Cg

        'If the current particles location is greater than that particles personal
best and particle is feasible, then pbest is updated

        If F(i) < Pbest(i, 11) Then
          For j = 0 To 10

            'Checks if current particle is feasible

            If X(i, j) > Xmax(j) Then
              Exit For
            End If
```

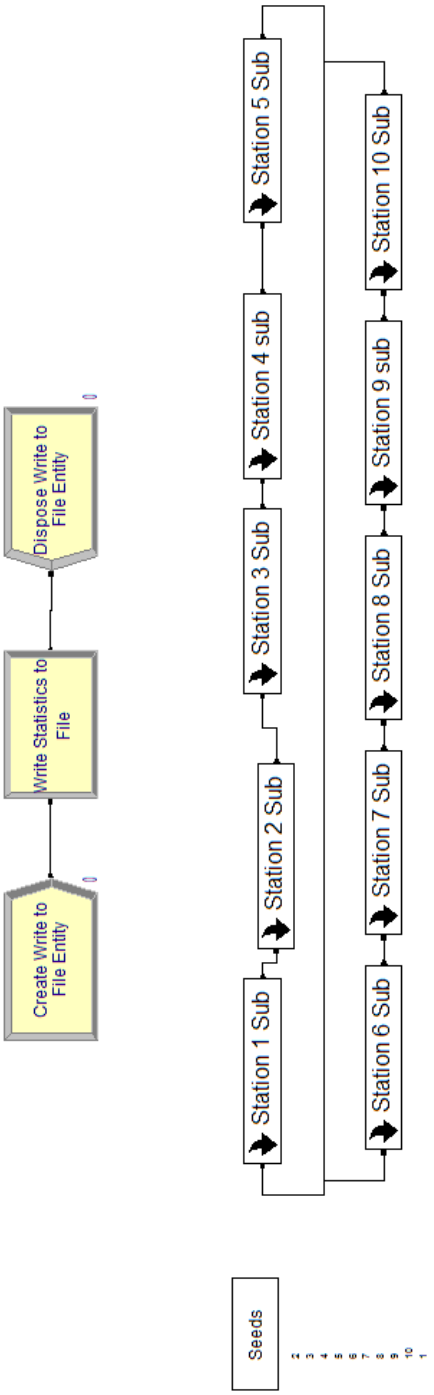
```
If X(i, j) < Xmin(j) Then
   Exit For
End If
If j = 10 Then
   Pbest(i, 11) = F(i)
   Pbest(i, 0) = X(i, 0)
   Pbest(i, 1) = X(i, 1)
   Pbest(i, 2) = X(i, 2)
   Pbest(i, 3) = X(i, 3)
   Pbest(i, 4) = X(i, 4)
   Pbest(i, 5) = X(i, 5)
   Pbest(i, 6) = X(i, 6)
   Pbest(i, 7) = X(i, 7)
   Pbest(i, 8) = X(i, 8)
   Pbest(i, 9) = X(i, 9)
   Pbest(i, 10) = X(i, 10)


        'If the current location is also better then Global best, gbest is
updated

      If Pbest(i, 11) < Gbest Then

            'Convergence criterion counter is updated

         c = 1
         Gbest = Pbest(i, 11)
         Gbestpoint(0) = X(i, 0)
         Gbestpoint(1) = X(i, 1)
         Gbestpoint(2) = X(i, 2)
         Gbestpoint(3) = X(i, 3)
         Gbestpoint(4) = X(i, 4)
         Gbestpoint(5) = X(i, 5)
         Gbestpoint(6) = X(i, 6)
         Gbestpoint(7) = X(i, 7)
         Gbestpoint(8) = X(i, 8)
         Gbestpoint(9) = X(i, 9)
         Gbestpoint(10) = X(i, 10)

            'Prints current solution to file

         Open "C:\ModelRun.txt" For Append As #1
            Print #1, "New G Best"; Tab; count; Tab; Gbest
         Close #1
      End If
End If
```

```
            Next
          End If
        Next

    'Reduced weighted inertia factor and then optimization is continued until
convergence

        w = w * WIF
    Loop
'Once the convergence criterion, the finish time is recorded

Finishtime = Format(Time, "Long Time")
'Run statistics and best solution found is recorded to a file

Open "C:\OptimalOut.txt" For Append As #1
    Print #1, "Run length of " & count
    Print #1, Gbest
    Print #1, Gbestpoint(0)
    Print #1, Gbestpoint(1)
    Print #1, Gbestpoint(2)
    Print #1, Gbestpoint(3)
    Print #1, Gbestpoint(4)
    Print #1, Gbestpoint(5)
    Print #1, Gbestpoint(6)
    Print #1, Gbestpoint(7)
    Print #1, Gbestpoint(8)
    Print #1, Gbestpoint(9)
    Print #1, Gbestpoint(10)
    Print #1, Starttime
    Print #1, Finishtime
Close #1

End Sub
```

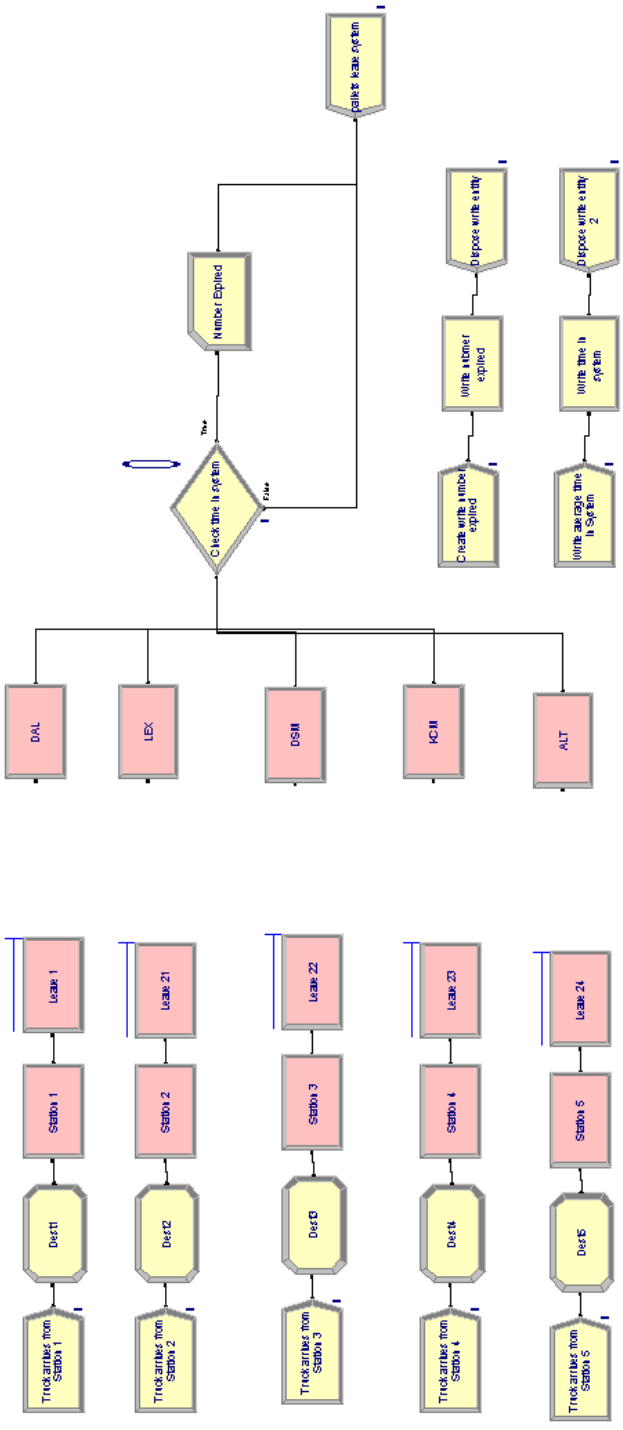
APPENDIX IV – LAMP MODEL



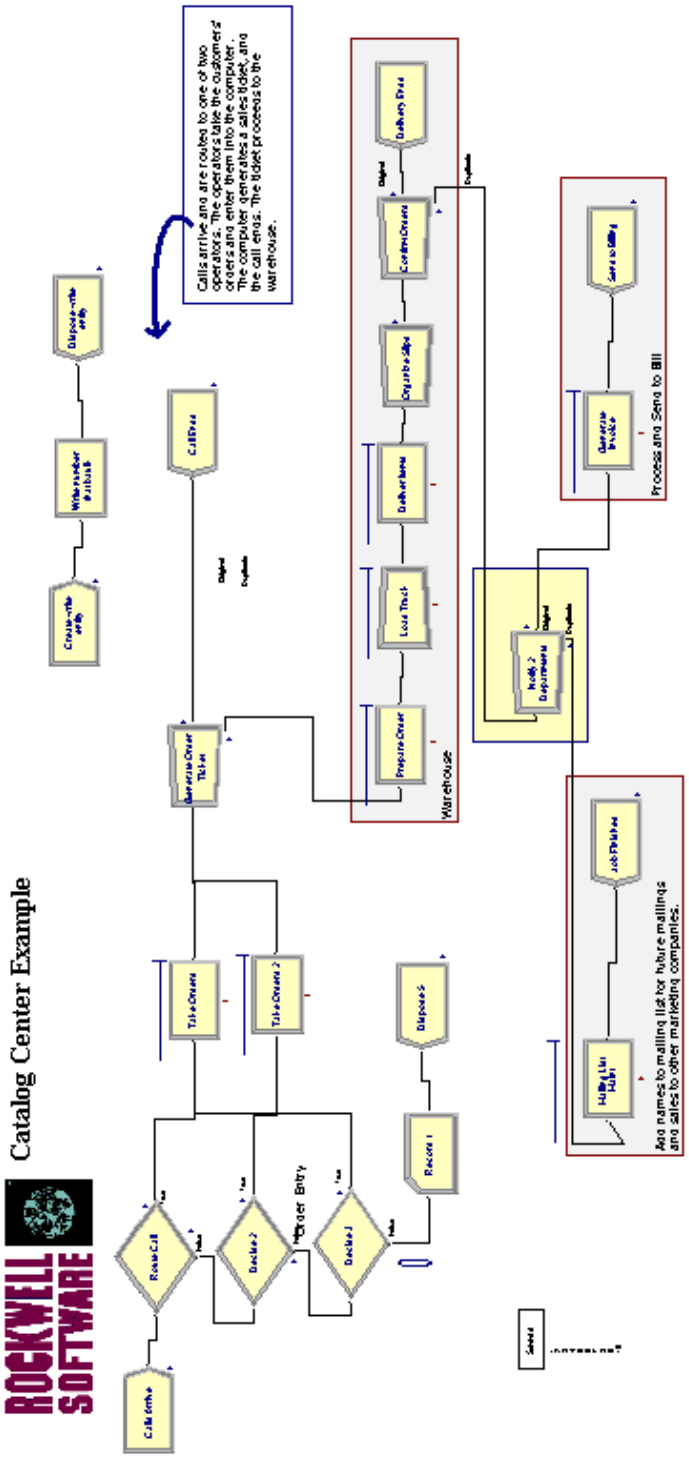
APPENDIX V – AAAS MODEL



APPENDIX VI – DISTRIBUTION CENTER MODEL



APPENDIX VII – CATALOG CENTER MODEL



APPENDIX VIII – DIFFERENCE IN MEANS

Example Test for Statistical Difference in Means:

$$H_0: \mu_1 - \mu_2 = 0$$

$$H_a: \mu_1 - \mu_2 \neq 0$$

$$T.S.: t' = \frac{(\bar{y}_1 - \bar{y}_2)}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}}$$

$$T.S.: t' = \frac{(202{,}332 - 279353)}{\sqrt{\frac{73052^2}{10} + \frac{134864^2}{13}}} = -1.75$$

Since $|t'|$ is not greater then $t_{\alpha/2}$=2.080 the two numbers are not statistically different.

APPENDIX IX – PARAMETER ADJUSTMENT