2005

Evaluating Particle Swarm Intelligence Techniques for Solving University Examination Timetabling Problems

Daniel R. Fealko
Nova Southeastern University, daniel.fealko@gmail.com

Evaluating Particle Swarm Intelligence Techniques for Solving University Examination Timetabling Problems

by

Daniel R. Fealko

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy

Graduate School of Computer and Information Sciences
Nova Southeastern University

2005

We hereby certify that this dissertation, submitted by Daniel R. Fealko, conforms to acceptable standards and is fully adequate in scope and quality to fulfill the dissertation requirements for the degree of Doctor of Philosophy.

Sumitra Mukherjee, Ph.D. Chairperson of Dissertation Committee	Date
Junping Sun, Ph.D. Dissertation Committee Member	Date
James Cannady, Ph.D. Dissertation Committee Member	Date

Approved:

Edward Lieblein, Ph.D. Dean, Graduate School of Computer and Information Sciences	Date

Graduate School of Computer and Information Sciences
Nova Southeastern University

2005

An Abstract of a Dissertation Submitted to Nova Southeastern University
in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

Evaluating Particle Swarm Intelligence Techniques for Solving University Examination Timetabling Problems

by
Daniel R. Fealko

August 2005

The purpose of this thesis is to investigate the suitability and effectiveness of the Particle Swarm Optimization (PSO) technique when applied to the University Examination Timetabling problem. We accomplished this by analyzing experimentally the performance profile—the quality of the solution as a function of the execution time—of the standard form of the PSO algorithm when brought to bear against the University Examination Timetabling problem. This study systematically investigated the impact of problem and algorithm factors in solving this particular timetabling problem and determined the algorithm's performance profile under the specified test environment. Keys factors studied included problem size (i.e., number of enrollments), conflict matrix density, and swarm size. Testing used both real world and fabricated data sets of varying size and conflict densities. This research also provides insight into how well the PSO algorithm performs compared with other algorithms used to attack the same problem and data sets. Knowing the algorithm's strengths and limitations is useful in determining its utility, ability, and limitations in attacking timetabling problems in general and the University Examination Timetabling problem in particular. Finally, two additional contributions were made during the course of this research: a better way to fabricate examination timetabling data sets and the introduction of the PSO-NoConflicts optimization approach. Our new data set fabrication method produced data sets that were more representative of real world examination timetabling data sets and permitted us to construct data sets spanning a wide range of sizes and densities. The newly derived PSO-NoConflicts algorithm permitted the PSO algorithm to perform searches while still satisfying constraints.

Acknowledgements

First, I wish to thank my dissertation committee chairperson, Dr. Sumitra Mukherjee. I greatly appreciated his willingness to accept me as a Ph.D. student without hesitation, as well as his providing prompt and incisive guidance whenever called upon. To the other members of my dissertation committee, Dr. Junping Sun and Dr. James Cannady, I offer my sincerest gratitude for their assistance in this work.

I owe a great deal of thanks to my wife, Sue Ellen, who never wavered in her belief that I would someday finish this project, even when I had doubts. I cannot express the gratitude I have for the words of encouragement and confidence she has provided throughout the years.

A special word of thanks goes to my father-in-law, the late Norman L. Matthews, Ph.D., M.D., who is truly missed. Though no longer with us, he nevertheless provided motivation through his exemplary life. His wisdom in financial matters largely made possible the means to afford this long endeavor, and he instilled the importance of lifelong learning in my wife. For these things, I will forever be indebted to him.

Table of Contents

List of Tables

Tables

List of Figures

Figures

Chapter 1

Introduction

Problem Statement

The general timetabling problem consists of assigning resources to objects in space and time while satisfying a set of hard constraints and, as nearly as possible, a set of soft constraints. The three most common academic timetabling problems are the school timetabling, university timetabling, and examination timetabling problems. School timetabling and university timetabling differ primarily by how students are allocated to classes. School timetabling groups students in classes with similar academic objectives whereas university timetabling considers students as individuals with possibly differing academic objectives. The examination timetabling problem consists of assigning a set of examinations to a limited number of time slots while satisfying the hard constraints. The most commonly cited hard constraint is the prevention of double booking; i.e., no student should be required to take more than one exam during any single time slot. Many times this requirement is unobtainable and is therefore relaxed. When this happens, the objective is to minimize the number of double booking conflicts. (Carter & Laporte, 1996)

The sheer size of university student bodies makes examination timetabling a complex combinatorial problem. Even in moderately sized universities, the manual solution of timetabling usually requires many person-days of effort. As a result, much research has been conducted over the last forty years, starting with the work of Gotlieb (1962), and since then many papers related to the automation of academic timetabling

have been published. Numerous approaches to solving the timetabling problem have been proposed in the literature. The earliest approaches were heuristic in nature, next came more general methods (e.g., graph coloring), and then came the more recent attempts using simulated annealing and genetic algorithms. (Carter, 1986; Schaerf, 1999)

Despite enormous effort expended over the last four decades to uncover efficient methods for solving timetabling problems, these problems are nevertheless still the focus of intense research (Willemen, 2002). Nor is this research limited to just a few countries as finding solutions to the general set of school timetabling problems is of interest to educational institutes throughout the world (Burke, Elliman, Ford, & Weare, 1995; Hansen & Vidal, 1995; Junginger, 1986).

Problem Representation

There are a number of possible ways to represent the basic university examination timetabling search problem. We choose to take almost directly from Schaerf (1999, p. 109) as it is a relatively general representation.

Given that

- q is the number of courses,
- r is the number of exams,
- k is a time slot
- p is the number of time slots,
- l_k is the maximum number of exams that can be scheduled at time slot k,
- i is a course, having a single exam,

- S_l is a group of exams such that in each group there are students that take all exams in the group, giving us 2^r-1 possible non-empty sets, then

find y_{ik} $(i = 1..q; k = 1..p)$

such that

$$\sum_{k=1}^{p} y_{ik} = 1 \quad (i = 1..q) \qquad (1)$$

$$\sum_{i=1}^{q} y_{ik} \le l_k \quad (k = 1..p) \qquad (2)$$

$$\sum_{i \in S_l} y_{ik} \le 1 \quad (l = 1..2^r\text{-}1; k = 1..p) \qquad (3)$$

$$y_{ik} = 0 \text{ or } 1 \quad (i = 1..q; k = 1..p)$$

where $y_{ik} = 1$ if the exam of course K_i is scheduled at time slot k, and
$y_{ik} = 0$ otherwise.

Constraints (1) impose that each exam is scheduled only once. Constraints (2) enforce the maximum number of exams that can be scheduled for a given time slot. Constraints (3) prevent conflicting exams from being scheduled at the same time slot.

This search problem may be modeled as an optimization problem when we add soft constraints. A typical list of soft constraints is presented below:

- *Examination Spread Constraint* – examinations associated with a student should be spread out as much as possible to allow the student time to prepare for the next exam.
- *Consecutive Periods Constraint* – a student must not be required to take two exams within consecutive periods, or time slots. This constraint is a special case of the *Examination Spread Constraint* but universities do not necessarily implement both constraints. In addition, even if both

constraints are implemented, they can have different associated penalty values.

- *Room Capacity Constraint* – sometimes referred to as Venue Capacity Constraint, requires the availability of a room with sufficient capacity in which to hold the exam. Sometimes universities permit examination splits, which is the splitting of an exam across multiple rooms.
- *Exam Proximity Constraint* – a student must not be required to attend consecutive exams on different campuses.
- *Special Room Assignment Constraint* – an exam must be held in either a specific room or room type. For example, a laboratory room might be required for the exam.

Constraints and Objectives

Optimization problems have three basic features: an objective function, decision variables, and a set of constraints. The goal of optimization is to determine the decision - variable values that minimize (or maximize) the objective function, subject to the constraints (Rardin, 1998).

Optimization problems can have zero, one, or more objective functions. An objective function – or cost function, penalty function, loss function – is a function that assigns a quantitative measure of worth to a point in *D*-dimensional decision variable space indicating its contribution to the specific objective. An optimization problem having no objective function is a *feasibility problem* or a *constraint satisfaction problem*. In this case, the goal is not to optimize but to find decision variables that satisfy the constraints. A problem with more than one objective function is referred to as a multi-

objective optimization problem and is normally reformulated, by combining the separate functions, into a single objective function.

In addition, all optimization problems require decision variables in order to define the objective function and problem constraints. These are variables under the control of the decision maker and affect the value of the objective function.

Lastly, optimization problems may or may not have associated constraints, which identify the set of feasible points in the *D*-dimensional decision variable space and fall into two separate categories: hard and soft. Optimization problems having constraints are referred to as *constraint optimization problems* and those lacking constraints, which actually make up a large and well study class of problems, are known as *unconstrained optimization problems*.

In general, hard-constraints reflect timetable requirements such as preventing examination conflicts and enforcing seating capacity restrictions. That is, students must not have more than one examination scheduled during the same time slot and an examination cannot exceed a room's seating capacity, respectively. Hard-constraints define the set of feasible solutions to a constrained optimization problem. Infeasible solutions are those that violate one or more hard-constraints.

Soft-constraints, on the other hand, generally reflect student and teacher preferences and not strict requirements. The ability to satisfy all soft-constraints is rare for any real world situation because simultaneous satisfaction of two incompatible soft constraints, which is a common occurrence, is unachievable. For example, maximizing the time between student examinations, to allow for maximum study time, would be in opposition to a requirement to minimize the total time spanned by the exam period.

Therefore, optimization problems traditionally use an objective function to measure how well all soft-constraints are satisfied; in essence, it measures the quality of the solution. This objective function is customarily an aggregation of soft-constraint penalty functions; each in turn measuring how well their related constraint is satisfied. The ultimate goal in solving an optimization problem is to locate the objective function's global minimum while satisfying all hard-constraints.

The examination conflict constraint is a first order conflict, where an *n*th order conflict has *n*-1 time slots between two examinations. In like manner, a student that has to take examinations in two consecutive time slots has a second order conflict (Burke, Elliman, & Weare, 1993). In all but the most difficult situations, preventing first order conflicts is a hard-constraint, as a student cannot take more than one examination at a time. Even so, this constraint is sometimes relaxed as satisfying it may be impractical due to an insufficient number of allotted examination days, especially in large universities with thousands of students and hundreds of examinations. As Carter, Laporte, and Lee (1996) point out for the 1993 Purdue spring exam period, "the calendar limits the examination week to six days. Five two-hour exams can be scheduled each day. Direct conflicts (2 in 1) are unavoidable" (p. 382). By this, Carter et al. (1996) mean that two conflicting exams in one time slot are unavoidable for their scenario. In such cases, one should focus on reducing the amount of infeasibility rather than finding feasible solutions (Burke & Newall, 1999). This does not mean that first order conflicts are unimportant. On the contrary, permitting them requires the quarantining or cloistering of students after an examination so they may take the conflicting examination immediately after (Burke & Newall, 1999; Merlot, Boland, Hughes, & Stuckey, 2002).

For other institutions, “the most important criterion is to eliminate (or minimize) the number of student conflicts in a fixed number of periods” (Carter et al., p. 337). Because of this, one still must make a concerted, but not all consuming, effort at finding a conflict-free solution.

Although most agree that preventing examination conflicts represents a hard-constraint, how other restrictions are classified depends in large part on other factors. For example, some institutions do not consider seating capacity a hard-constraint because they have the option to obtain additional off-campus rooms when required (Merlot, Boland, Hughes, & Stuckey, 2002). In cases like these, the institution may instead have a hard-constraint on the total number of days spanned by the exam period. Other institutions do classify seating capacity as a hard-constraint (Burke, Newall, & Weare, 1998b).

Goal

The overriding reason the school timetabling problems have been the focus of such prolonged and intense research is the inescapable fact that, except for the most trivial non-real world cases, these problems are all NP-complete (Schaerf, 1999). Hence, it should come as no surprise then that attempts are made to apply each newly discovered and relevant algorithmic approach to this challenge. In like manner, the goal of this study was to investigate the suitability and effectiveness of the relatively new Particle Swarm Optimization techniques when applied to the University Examination Timetabling class of problems. We accomplished this by analyzing experimentally the performance profile of the PSO algorithm when brought to bear against the university examination timetabling problem using both real world and fabricated data sets of varying size and

conflict densities. The term Performance Profile here denotes the quality of the solution as a function of the execution time. Since there are many possible factors affecting the execution time, the performance profile was determined empirically, not analytically, by applying the PSO method to numerous data sets and examining factors across iterations of the algorithm.

Relevance and Significance

The university examination timetabling problem consists of assigning examinations to a limited number of time slots in such a way that ideally there are no hard constraint conflicts and the number of soft constraint conflicts is minimized (Carter, Laporte, & Lee, 1996). When one wants to determine the existence of a timetable that satisfies all constraints, the problem is formulated as a search or constraint satisfaction problem (CSP). If instead, one wants to find a solution that minimizes (or maximizes) an objective function, which embeds the soft constraints, while satisfying all hard constraints, then the problem is formulated as an optimization problem (Lim, Chin, Kit, & Oon, 2000; Schaerf, 1999).

All real world university examination timetabling problems have associated constraints. The number and definitions of the constraints varies from university to university because each institute embodies unique business rules. Regardless of the disparity of constraint requirements, all constraints fall into only one of the aforementioned categories, hard constraints, or soft constraints. Hard constraints are regarded as essential, in terms of producing a practical timetable, whereas soft constraints are deemed desirable but can be violated if necessary (Burke & Newall, 1999). Another way of thinking of soft constraints is that how well one satisfies soft constraints

determines the timetable's solution quality (Burke, Bykov, & Petrovic, 2001; Reis & Oliveira, 2001).

Numerous algorithmic approaches to solving the timetabling problem have been put forth in the literature. The more common approaches used are Tabu Search (Di Gaspero & Schaerf, 2000; Schaerf, 1996), Graph Coloring (Kiaer & Yellen, 1992), Genetic Algorithms (Corne, Fang, & Mellish, 1993; Ross, Hart, & Corne, 1998; Terashima-Marín, Ross, & Valenzuela-Rendón, 1999), Simulated Annealing (Elmohamed, Fox, & Coddington, 1997; Thomson & Dowsland, 1998), and Heuristic Approaches (Colorni, Dorigo, & Maniezzo, 1998; Zhaohui & Lim, 2000). With so many approaches already researched, one might well wonder if timetabling is still a problem and, if so, why? Even though a number of good approaches exist for solving school timetabling problems, Even, Itai, and Shamir (1976) proved that all common (i.e., real world) timetable problems are NP-complete. A number of other authors (Cooper & Kingston, 1995; Schaerf, 1999) have also shown that almost all variants of the timetabling problem are NP-complete. If one considers timetabling problem sets with non-trivial cardinality or even where a few constraints are considered, then the problem is computationally very demanding to solve.

Particle Swarm Optimization

Metaphors are frequently used as guides in modeling systems that solve problems. For example, Genetic Algorithms (GAs) use the metaphor of genetic and evolutionary principles of fitness selection for reproduction to search solution spaces. In a similar fashion, the collective behavior of insect colonies, bird flocks, and other animal societies

are the motivation for Swarm Intelligence algorithms, which use self-organization and division of labor for distributed problem solving.

The Particle Swarm Optimization (PSO) method is a relatively new stochastic Global Optimization (GO) member of the broader Swarm Intelligence field for solving GO problems (Kennedy & Eberhart, 2001; Parsopoulos & Vrahatis, 2002d). James Kennedy and Russell Eberhart first introduced this method in 1995 (Kennedy & Eberhart, 1995). Like GAs, the particle swarm approach is a population-based method. In contrast to GAs though, PSOs do not use evolutionary inspired operators to construct a new generation of candidate solutions. Instead, they simply modify the movement of the individuals, called particles, in its population, called a swarm, without the generation of a completely new population. In addition, GAs use mutation and selection operators, among other operators, to "evolve" better solutions over time in contrast to PSO's use of volume-less particles in multi-dimensional space, which update their velocities. Each particle's velocity, and thus trajectory, is modified based upon their own best previous position, determined with a fitness function, coupled with the previous best attained by members of their topological neighborhood. (Parsopoulos & Vrahatis, 2002a)

Particle Swarm Technique Advantages

The simplicity of the particle swarm algorithm, coupled with its demonstrated ability to deal with complex problems, makes the technique an ideal candidate for consideration in attacking the university examination timetabling problem. In addition to the algorithm's inherent simplicity, there are other motivations for utilizing the particle swarm algorithm. Some of those advantages include:

- Its robustness and speed for solving non-linear, non-differentiable, multi-modal problems (Shi & Eberhart, 1998).
- Its ability to optimize hard mathematical problems in continuous or binary space domains (Kennedy, 1998).
- Its consistent record of accomplishments in locating near optima solutions significantly faster than evolutionary optimization methods (Angeline, 1998; Kennedy & Spears, 1998).
- Its quick convergence rate, insensitivity to population size, and data set scalability (Shi & Eberhart, 1999).

According to Schaerf (1999), PSO gets one near the optima faster than GA, but GA eventually gets one closer. Many believe that getting near a solution faster is better than achieving an optimum solution for timetabling problems as in almost all cases users hand-tweak the result in the end anyway (Burke, Newall, & Weare, 1995). Many universities arrive at acceptable examination timetables through a process of iteration. Under these circumstances, providing a faster cycle time has advantages as it permits the university to review the results and modify the parameters of the algorithm in a more timely fashion.

Suitability of Particle Swarm Techniques for Timetabling Problems

The university examination timetabling problem involves constraints and the presence of constraints adversely affects the performance of all optimization algorithms, including evolutionary search methods. Regardless, research results indicate that the recent advances in the PSO algorithm not only make it capable of handling unconstrained and constrained multiobjective problems but it is also able to do this with continuous,

discrete or mixed variables without restrictions on the number of variables, constraints or objectives (Ray & Liew, 2002). In addition, the nature of the university examination timetabling problem is one of a constrained optimization (CO) problem. PSO not only is an alternative approach to solving CO problems but in most cases, it detects superior solutions than those achieved through other evolutionary algorithms (Parsopoulos & Vrahatis, 2002a).

Barriers and Issues

Even though timetabling problems have been around for decades, these problems have only lately been tackled by artificial intelligence techniques such as genetic algorithms and tabu search. These approaches generally outperform the more traditional operational research approaches for large timetabling problems (Schaerf, 1999). Even though the success of particle swarm techniques have been demonstrated recently (Parsopoulos & Vrahatis, 2002b), the vast majority of testing has been limited to well understood mathematical test functions. Research with real world problems (Robinson, Sinton, & Rahmat-Samii, 2002) is starting but, to date, the author is unaware of any attempt to apply particle swarm techniques to any timetabling problem, let alone the school timetabling problems. Other than the lack of any prior research on using the PSO algorithm against the general class of timetabling problems to provide insight or guidance for this research effort, the author knows of no other barriers or issues.

Research Questions

The PSO algorithm works well for a wide range of problems, dimensions, and problem sizes and its real advantage over other algorithms is its simplicity and

extensibility (Kennedy & Eberhart, 2001). Consequently, it may also prove valuable in solving the university examination timetabling problem, providing a different attack approach on the problem. This reasoning leads to the following research questions:

- How well does the PSO handle university examination timetabling problems of different sizes and conflict matrix densities?
- How does the PSO algorithm compare with other algorithms against this problem class?
- What are the advantages and disadvantages of using the PSO algorithm on this problem over other methods?

The purpose of this thesis was to investigate these basic questions. We accomplished this by analyzing experimentally the performance profile of the standard form of the PSO algorithm when brought to bear against the university examination timetabling problem. This study systematically investigated the impact of various factors in solving this particular class of timetabling problems and determined the algorithm's performance profile under the specified test environment.

Conflict Matrix Density

The terms *conflict matrix* and *conflict matrix density* are used throughout the timetabling literature as they are fundamental to the research involving timetabling problems and graph theory. (Diestel (2000) is an excellent reference for both graph theory and terminology.)

A *conflict matrix* is an $n \times n$ matrix with entries c_{ij}, where c_{ij} is the number of students taking both examinations i and j. The density of the conflict matrix is the average number of all other exams that each exam conflicts with, divided by the total

number of exams (Burke & Newall, 1999). Hence, a density of 0.1 denotes that, on average, each exam conflicts with 10% of all the exams.

We use a simplified timetabling example here in order to illustrate the definitions in a manner unencumbered by size. For our example, we use only seven courses, represented by the letters A through G, and seven students, represented by the numbers 1 through 7. This provides 49 (7×7) possible student/course combinations, which are enrollments. We then select a random subset of these enrollment possibilities. We show our selection results in Table 1. Dots located at the intersection of course rows and student columns designate a selected enrollment. We should also note here that we are assuming each course has one and only one examination. Given this assumption, the terms "course" and "examination" are synonymous in the discussion that follows.

Table 1. Enrollments for Conflict Matrix Example

Course	Student						
	1	2	3	4	5	6	7
A	•		•			•	
B		•		•	•	•	
C	•	•	•		•	•	
D			•				•
E		•		•		•	
F	•	•			•		
G			•			•	

From the enrollments identified in Table 1, we construct the conflict matrix using its definition and arrive at Table 2.

Table 2. Conflict Matrix for Enrollment Example

	Course						
Course	A	B	C	D	E	F	G
A	3	1	3	1	1	1	2
B	1	4	3		3	2	1
C	3	3	5	1	2	3	2
D	1		1	2			1
E	1	3	2		3	1	1
F	1	2	3		1	3	
G	2	1	2	1	1		2

Now, using the *conflict matrix density* definition from Carter, Laporte, and Lee (1996, p. 376), "the density represents the proportion of non-zero and non-diagonal entries of the matrix (c_{ij}), where c_{ij} is the number of students writing both examinations i and j," we construct Table 3. It is important to note that the conflict matrix density does not take into consideration the quantity of students involved between two conflicting exams, only that a conflict occurs.

Table 3. Matrix for Computing Density

	Course						
Course	A	B	C	D	E	F	G
A		1	1	1	1	1	1
B	1		1		1	1	1
C	1	1		1	1	1	1
D	1		1				1
E	1	1	1			1	1
F	1	1	1		1		
G	1	1	1	1	1		

From above, we have that the density of the conflict matrix is the average number of all other exams that each exam conflicts with, divided by the total number of exams, which is always less than or equal to $(1 - 1/n)$ and approaches this limit for dense graphs.

Yet another way of looking at the conflict matrix density is to view it from a graph perspective. Figure 1 is a pictorial representation of the graph for our example

timetabling problem. From a graph perspective, the conflict matrix density equals the ratio of the graph's *degree* to the *degree* of a *complete* graph of the same *order*. This graph definition should be apparent from the matrix examples shown above, as it is merely a graph representation of the matrix. This iconographic representation of the conflict matrix density may afford a more intuitive feel for the basic concepts.

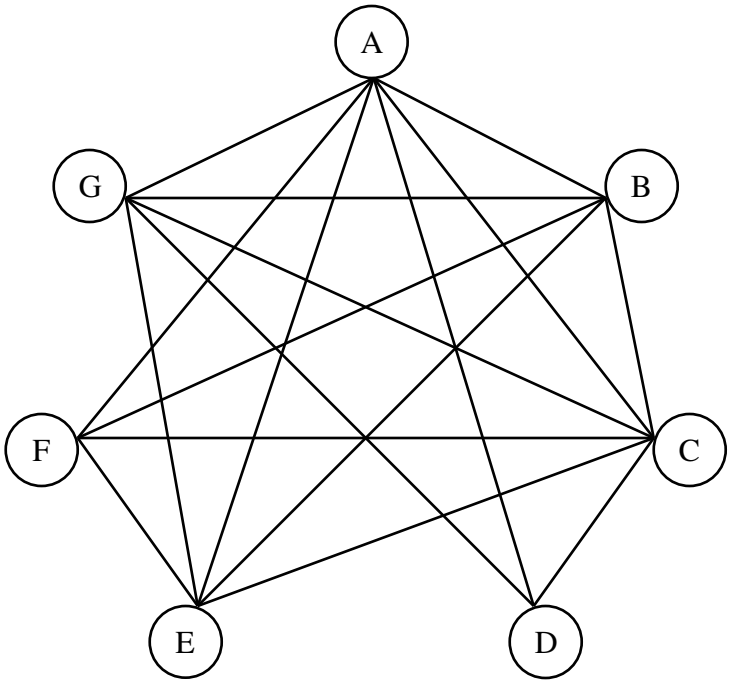

Figure 1. Graph of Example Timetabling Problem

Summary

University examination timetabling is a challenge routinely faced by university schedulers throughout the world, requiring many person-days of effort. This problem has resulted in decades of intense research due to its significance to universities and because of its NP-complete complexity.

This research investigated the efficacy of the PSO algorithm when applied to the university examination timetabling class of problems by analyzing the performance profile of the standard PSO algorithm. Both real world and fabricated data sets of varying size and conflict densities were used to provide insight into how well the PSO algorithm performs compared with other algorithms used to attack the same problem and data sets.

Chapter 2

Review of the Literature

University Examination Timetabling Theory and Research Literature

Numerous algorithmic techniques to solving the examination timetabling problem are contained within literature, which encompasses nearly four decades and a number of good surveys exist (Burke, Elliman, Ford, & Weare, 1995; Burke & Petrovic, 2002; Carter, 1986; Carter & Laporte, 1996; Schaerf, 1999). In addition to the surveys, the Ph.D. thesis of Bykov (2003) contains an overview of the chief algorithmic approaches to date. There are many more algorithmic approaches described throughout the literature than those covered by Bykov, as an exhaustive survey is unreasonable within the confines of a thesis. We discuss a few of the more important approaches in the paragraphs that follow and direct the reader to the surveys above for a more comprehensive overview.

Graph Coloring

It is well known that the examination timetabling problem, when considering only the examination conflicts constraint, maps into an equivalent graph coloring problem (Kiaer & Yellen, 1992), which is NP-complete (Burke, Elliman, & Weare, 1993; Willemen, 2002). The graph coloring problem is an assignment of colors to vertices in such a manner that no two adjacent vertices have the same color. Therefore, a solution to the graph coloring problem represents a solution to the core examination timetabling problem, where graph vertices correspond to exams, graph edges indicate that the connected vertices have an examination conflict, and colors represent unique time slots (Welsh & Powell, 1967). The graph coloring problem in turn is solved using one of the

graph coloring heuristics (e.g., Largest Degree), usually with backtracking (Burke, Newall, & Weare, 1998a; Carter, Laporte, & Chinneck, 1994).

Backtracking simply means that when no legal period is available for a particular examination, one removes already scheduled examinations in order to schedule that examination. Examinations removed due to this process are placed on a waiting list and rescheduled into new time slots. Carter, Laporte, and Lee (1996) describe in detail one algorithm used to perform backtracking. We chose instead to use the backtracking algorithm outlined by Burke, Newall, and Weare (1998a).

The complexity and richness of most soft-constraints makes it extremely difficult to incorporate them into the graph coloring model. This is why soft-constraints are not solved using the graph coloring approach but by other means, such as Genetic Algorithms, Tabu Search, Simulated Annealing, and others. Graph coloring techniques are typically limited to solving the first order conflicts. Nevertheless, Burke, Elliman, and Weare (1994) successfully used this method while considering pre-assignments, consecutive exams, specialist rooms, and room allocation constraints.

Genetic Algorithms

Evolutionary methods in general and GAs in particular, are probably the examination timetabling approach that has received the most attention over the last decade, and genetic algorithms in particular (Corne, Fang, & Mellish, 1993; Fang, 1994; Ross, Hart, & Corne, 1998; Terashima-Marín, Ross, & Valenzuela-Rendón, 1999). Genetic Algorithm techniques are a computational analogy of adaptive systems and are an approximate model of the principles of evolutionary pressure found in natural selection theory. The basic features of a genetic algorithm are that:

1. it consists of a population of individuals, each representing a solution, that undergo selection,
2. each individual (solution) has a genetic representation that is modeled as a genome (or chromosome),
3. it uses a fitness function to determine each individual's reproductive success, and
4. it uses genetic operators, such as mutation and crossover, to evolve new solution each successive generation in order to find the best ones.

Tabu Search

Tabu Search is another common heuristic found throughout the exam timetabling literature (Burke, Bykov, Newall, & Petrovic, 2003; Di Gaspero & Schaerf, 2000; Schaerf, 1996). The overall feature of this method is the management of a list of the last *n* solutions visited in order to avoid "re-finding" those solutions in subsequent iterations (thus, "tabu list"). Tabu search works under the assumption that one should not accept a new poor solution unless it is to avoid a solution already discovered. The objective is to avoid premature convergence on local minima and to force the algorithm to consider new regions of space. The basic steps, when assessing a neighborhood for a new solution, are:

1. evaluate each solution in the topological neighborhood, and
2. select the highest quality solution not already in the "tabu list" even if its quality is lower than the current one.

Simulated Annealing

Quite a few research efforts (Burke, Bykov, Newall, & Petrovic, 2003; Burke, Eckersley, McCollum, Petrovic, & Qu, 2003b; Elmohamed, Fox, & Coddington, 1997;

Salman, Ahmad, & Al-Madani, 2002; Thomson & Dowsland, 1996, 1998) have used Simulated Annealing as their heuristic in an attempt to solve the examination timetabling problem. This algorithm works in very much the same conceptual way as Tabu Search. That is, the algorithm occasionally accepts inferior candidate solutions in its attempt to discover better final solutions. Solutions with inferior objective function values than the current one are accepted with probability $P=exp(-\Delta f / T)$ where $-\Delta f$ is the change in the objective function and T is a control parameter. T is referred to as the "temperature" only because of its analogous nature to the real annealing process equation. The temperature of the system is gradually reduced throughout the search process and this method is referred to as the "cooling schedule" of the process.

Others

Numerous other approaches have been tried throughout the decades since the first attempts of automating the examination timetabling process. Three of the more common approaches are:

- Hybridization – (Burke, Petrovic, & Qu, 2004; Merlot, Boland, Hughes, & Stuckey, 2002; Newall, 1999) where several techniques are applied in combination in the hopes that there is a synergetic action among them.
- Memetic Algorithm – where the *meme* is analogous to the *gene* except it represents a modifiable cultural idea. The memetic approach of Burke, Newall, & Weare (1995) used a hybrid of the evolutionary algorithm with the hill-climbing algorithm.
- Case-Based Reasoning Approach – (Burke, Eckersley, McCollum, Petrovic, & Qu, 2003a; Burke, Petrovic, & Qu, 2002) which is a system

that selects a heuristic, from a collection of solutions, based on similarity of the problem, its data set, and objective functions. This approach works under the assumption that similar problems are solved most effectively by similar heuristic approaches.

These are but a small set of the plethora of techniques developed over the last four decades to tackle the examination timetabling problem.

Particle Swarm Optimization Theory and Research Literature

Metaphors are frequently used as guides in modeling systems that solve problems. For example, Genetic Algorithms (GAs) use the metaphor of genetic and evolutionary principles of fitness selection for reproduction to search solution spaces. In a similar fashion, the collective behavior of insect colonies, bird flocks, and other animal societies are the motivation for Swarm Intelligence algorithms, which use self-organization and division of labor for distributed problem solving.

The Particle Swarm Optimization (PSO) method is a comparatively new stochastic Global Optimization (GO) member of the broader Swarm Intelligence field for solving GO problems (Kennedy & Eberhart, 2001; Parsopoulos & Vrahatis, 2002d). James Kennedy and Russell Eberhart first introduced this method in 1995 (Kennedy & Eberhart, 1995). Like GAs, the particle swarm approach is a population-based method. In contrast to GAs though, PSOs do not use evolutionary inspired operators to construct a new generation of candidate solutions. Instead, they simply modify the movement of the individuals, called particles, in its population, called a swarm, without the generation of a completely new population. In addition, GAs use mutation and selection operators, among other operators, to "evolve" better solutions over time in contrast to PSO's use of

volume-less particles in multi-dimensional space, which update their velocities. Each particle's velocity, and thus trajectory, is modified based upon their own best previous position, determined with a fitness function, coupled with the previous best attained by members of their topological neighborhood. (Parsopoulos & Vrahatis, 2002a)

Particle Swarm Optimization Algorithm

The PSO algorithm iteratively manipulates the position and velocity of particles in its search for solutions. Using notation similar to that used by Parsopoulos, Laskari, and Vrahatis (2001), we have:

- D is the number of dimensions in the multi-dimensional space,
- N is the size of the swarm population,
- p_i is the i-th particle's best previous position and represented as $(p_{i1}, p_{i2}, \ldots, p_{iD})$,
- X_i is the i-th particle in the D-dimensional space and represented as $(x_{i1}, x_{i2}, \ldots, x_{iD})$,
- V_i is the i-th particle's velocity (spatial change) and represented as $(v_{i1}, v_{i2}, \ldots, v_{iD})$,
- χ is the constriction factor,
- w is the inertia weight,
- φ_1 and φ_2 are two positive constants, frequently referred to as the *cognitive* and *social* parameters respectively,
- r_1 and r_2 are two random numbers uniformly distributed over the range the range [0, 1], and
- g is the particle with the best function value,

then the particles are manipulated according to the following two equations:

$$v_{id} = \chi\,(w v_{id} + \varphi_1 r_1(p_{id} - x_{id}) + \varphi_2 r_2(p_{gd} - x_{id})) \quad (1)$$

$$x_{id} = x_{id} + v_{id} \quad (2)$$

where $d = 1, 2, \ldots, D$

and $i = 1, 2, \ldots, N$

and from Clerc (1999) we have

$$\chi = \frac{2}{\left|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}\right|} \quad (3)$$

and $\varphi = \varphi_1 + \varphi_2, \ \varphi > 4.$

Equation 1, the velocity equation, computes each particle's new velocity. This is achieved by using the particle's current velocity and adding to it the influence from its current best position plus the swarm's current best position. The three base terms of Equation 1 are multiplied by the constriction factor, which improves PSO's ability to constrain and control velocities.

The inertia weight *w* term is a parameter used to control the influence of previous velocities on the particle's current velocity. It regulates the trade-off between the global (exploration) and the local (exploitation) abilities of particles (Abido, 2001). In addition, it is common practice to start with a large inertia weight and linearly decrease its value to improve the PSO's performance (Shi & Eberhart, 1999). This method enhances the algorithm's exploration abilities early in the process and gradually transitions to an exploitation mode of local regions of interest. Though we briefly investigated this

method of linearly decreasing its value, we dismissed it early on in favor of other more promising parameters, though a more thorough future investigation may be warranted.

Equation 2, the position equation, uses the particle's updated velocity value computed in Equation 1 to determine its new position.

Equation 1 and 2 represent a slightly updated form of the original PSO algorithm put forth by Kennedy and Eberhart (1995). Some, like Peer, Engelbrecht, and van den Bergh (2003), refer to Kennedy and Eberhart's original algorithm as the canonical form of the algorithm and others, such as Carlisle and Dozier (2001), refer to their own as the canonical form. We follow Carlisle and Dozier's lead and refer to our set of equations above as the canonical form as it is not too different from theirs, it represents the generalized version utilized by most researchers, and it embodies recent advances.

Implementation

This research requires a programmatic implementation of the PSO algorithm. The following pseudo-code form of the algorithm is adapted from the version presented by Abido (2001). Each numbered step describes, in general terms, a core algorithmic procedure. This pseudo-code is neither an exhaustive enumeration of operations nor a thorough specification for those steps shown. The algorithm is as follows:

1. *Initialize algorithm* - Initialize the time interval to zero. Generate *m* particles, giving them initial positions and velocities in the *D*-dimensional space.
2. *Evaluate objective function* - Use objective function to evaluate each particle.

3. *Initialize individual best* - Set each particle's *individual best* equal to its current position.
4. *Initialize global best* - Set the *global best* equal to the position of the particle with the best objective function.
5. *Update time* - Increment the time interval by one unit.
6. *Update weight* - Update the inertia weight.
7. *Update velocity* - For each particle, use the following to compute their new velocities.

$$v_{id} = \chi\,(w v_{id} + \varphi_1 r_1(p_{id} - x_{id}) + \varphi_2 r_2(p_{gd} - x_{id}))$$

8. *Update position* - Using the updated velocity from step 7 and the following equation, compute each particle's new position.

$$x_{id} = x_{id} + v_{id}$$

9. *Evaluate objective function* - Use objective function to evaluate each particle.
10. *Update individual best* - If a particle's new objective function value is better than its previous best, then update the particle's *individual best* to its current position.
11. *Update global best* - If the best objective function value, from all of the current particles, is better than the current *global best*, then set the *global best* equal to that particle's position.
12. *Test for termination* - If at least one of the *stopping criteria* is satisfied, then terminate algorithm. Otherwise, go to step 5.

PSO – Initialization

The first step in the PSO algorithm is initialization of the swarm. As the particles of the swarm have no previous collective knowledge from which to determine their position or behavior, the algorithm must specify the starting values of each particle's position and velocity. Moreover, as Parsopoulos and Vrahatis (2002a) point out, proper initialization may help the algorithm to detect better solutions through more efficient exploration of the search space.

Initialization has been performed in the past using random (stochastic) methods. Recent research in GA and PSO indicates that there may be better ways to perform population initialization. (Burke, Newall, & Weare, 1998b; Parsopoulos & Vrahatis, 2002a)

Random Initialization Methods

Typically, one initializes each particle's position in the swarm by selecting random values from a uniform probability distribution over each dimension's domain. After the initial position of a particle is fixed, its initial velocity is randomly selected from among a uniform distribution between the allowable minimum and maximum values.

We took a slightly different tack. First, we created random positions and assigned these to each particle's best position, as that is what they were by definition. Next, we created a second set of random positions and assigned these to the particle's position values. From these two values, we computed the velocity. Lastly, we updated the swarm and particle's best function value positions, in keeping with the algorithm. This approach was not due to guidance from prior research. We initialized in this manner because it seemed more natural to derive the velocity from successive random positions.

Most published PSO algorithms generate an initial swarm using a Pseudo-Random Number Generator (PRNG). Unfortunately, this does not guarantee a uniformly distributed swarm over multidimensional search spaces. Research (Parsopoulos & Vrahatis, 2002a) indicates improved PSO efficacy if one guarantees multidimensional uniformity, not randomness (independence), for initialization. A Quasi-Random Number Generator (QRNG) ensures such multidimensional uniformity by producing number sequences guaranteed uniformly distributed over multiple dimension, unlike the PRNG used for random number sequences. Previous researchers used two different QRNGs to initialize the PSO swarm, the Sobol sequence generator (Parsopoulos & Vrahatis, 2002c) and the Nonlinear Simplex Method (Parsopoulos & Vrahatis, 2002a). Research results imply that using either one improves the convergence rate of the PSO. Regardless, for this study, we used a PRNG. Our results create a baseline upon which, in the future, one could use a QRNG method to make comparisons.

Heuristic Initialization Methods

A random number generator is sufficient for initializing the swarm but may not be the best method, in particular, for the examination timetabling problem. Burke, Bykov, Newall and Petrovic (in press) point out that specialized initialization strategies could be influential on evolutionary algorithm performance in disconnected search spaces, which is typical for examination timetabling problems, and Burke, Newall, and Weare (1998b) show performance improvement due to initialization strategies in a memetic examination timetabling approach.

A number of exam timetabling approaches employ heuristic ordering based on graph coloring (Carter and Laporte, 1996) for constructing conflict-free timetables. The

heuristic provides a means whereby one can estimate the difficulty of scheduling a particular exam (Burke & Petrovic, 2002). The idea behind this approach is that we would expect more latitude in scheduling the difficult exams first then by waiting until the end when the number of valid timeslots may be fewer.

The following four documented (Petrovic, Yang, & Dror, 2003; Burke, Newall, & Weare, 1998a) heuristic ordering methods are characteristic of this approach but do not represent an exhaustive list of ordering methods.

- ***Largest Degree First (LDF)***

  Schedule first the examinations having the largest number of conflicts (i.e., largest degree) with other examinations. The reasoning is that scheduling examinations having a large number of conflicts are more problematic and so should be dealt with first.

- ***Least Saturation Degree (lSD)***

  Schedule first the examinations having the least number of remaining valid timeslots available for scheduling. The reasoning is that scheduling these examinations, which have fewer timeslots available, may prove harder or impossible to schedule later as the number of options could easily be exhausted.

- ***Largest Color Degree (LCD)***

  Schedule first the examinations having the largest number of conflicts with other examinations already scheduled. This heuristic is a dynamic version of the LDF heuristic. The reasoning is that scheduling these examinations is harder than scheduling examinations having little or no conflicting examinations already in the timetable. For example, if we had an exam with a large number of conflicting

exams but where none of these conflicting exams are scheduled yet, then this is considered not currently as difficult as this could be scheduled anywhere.

- ***Largest Weighted Degree (LWD)***

  Schedule first the examinations having the largest sum of weighted conflicts, where the number of students enrolled in both examinations weights each conflict. This heuristic is a variation of the LDF ordering. The reasoning is that core exams, where large numbers of students have similar conflicts, should have priority.

PSO – Random Numbers

As PSO is a random (stochastic) optimization heuristic, the ability to generate quality random numbers is fundamental to its success. First, the PSO algorithm requires the generation of two separate random numbers for each particle's velocity equation per iteration. In addition, if the PSO uses a randomized initialization method, then one also randomly generates each particle's initial position and velocity values. Hence, as the number of iterations may well be in the thousands, the need exists for a large set of randomly generated numbers.

Programmatically one is never able to generate a truly random sequence of numbers, as any algorithmic approach is deterministic. The best one can do is use a Pseudo-Random Number Generator (PRNG), which produces number sequences that are statistically indistinguishable from truly random sources. That is, a number sequence whose components are mutually independent and uniformly distributed over an interval.

PSO – Control Parameters

The PSO algorithm contains a number of control parameters that require initial specification in order for success. These parameters include the swarm population size, topological neighborhood size, inertia weight, and cognitive/social ratio. A number of recent research efforts have examined these parameters and their effect on the overall performance of the algorithm (Beielstein, Parsopoulos, & Vrahatis, 2001; Carlisle & Dozier, 2001). Clerc's (2002b) research went so far as to investigate the possibilities of creating a parameter free PSO to eliminate the need to specify any values.

PSO – Convergence

The structure of the PSO algorithm is the main causal force in its convergence. The first contributing factor is a direct result of each particle's motion over time. According to the particle update equations, a particle's new velocity vector is always the resultant vector combination of the particle's previous velocity, a vector towards the particle's personal best position, and a vector towards the swarm's global best particle position. This resultant vector continually prods the particle back to a value lying between its personal best and the swarm's best particle position (van den Bergh, 2001). Over time, and with an appropriate choice of parameter values, each particle's personal best converges to the swarm's best particle position. If each particle's inertia weight and velocity are very close to zero at the time, the swarm ceases to move, regardless if the swarm has even discovered an optimum (van den Bergh & Engelbrecht, 2002).

Though not a strict rule, convergence can be considered as the point in an evolutionary algorithm where the population's average fitness is identical to its best fitness (Peram, Veeramachaneni, & Mohan, 2003). In the case of a swarm, each

particle's personal best position eventually converges the swarm's global best position. At this point, nothing exists in the algorithm to move the particles from this position. If it is the case that the swarm has not discovered the global optimum, then this represents premature convergence, sometimes also referred to as stagnation.

According to van den Bergh and Engelbrecht (2002), the standard PSO algorithm (Kennedy & Eberhart, 1995) exhibits premature convergence on sub-optimal solutions. In addition, van den Bergh and Engelbrecht point out that this algorithmic characteristic exists even in the later inertia weight and constriction factor versions. According to Angeline (1998), the PSO algorithm can quickly converge to an optimum in a relatively low number of iterations but it struggles with finding a near optimal solution. Shi and Eberhart (1998) introduced the use of a linear decreasing inertia weight parameter, reminiscent of the temperature parameter in simulated annealing, in an attempt to counter stagnation. Using the inertia weight parameter, one can influence the algorithm to remain explorative longer and gradually ease into the exploitative mode where convergence pressure is stronger. Thus, by preventing the swarm from prematurely converging to local sub-optima, the algorithm has a higher probability of finding the global optimum region of the search space. Though we looked briefly at this, we left this for possible future work, as initial testing did not seem promising at the time.

The reason for avoiding premature convergence is readily apparent. As mentioned, in the standard PSO, convergence does not imply the discovery of the global optimum or even local sub-optima necessarily (van den Bergh, 2001), making the value of the solution highly questionable. This problem is compounded in the case of university examination timetabling problems as they are multi-modal and, according to

Vesterstrøm and Riget (2002), the standard PSO has premature convergence problems with search spaces that are strongly multi-modal. Vesterstrøm and Riget also point out that high diversity is "crucial for preventing premature convergence in multi-modal optimization" (p. 101).

PSO – Constraints

The standard PSO algorithm does not have a mechanism for handling constraints, either hard or soft, and, like genetic algorithms in general, the algorithm does not respect feasibility. Unless special steps are taken, algorithmic transformations are not guaranteed to produce feasible candidate solutions even when the input to the transformation is within the feasible domain, that is, the feasible search space. Regardless, there are three main approaches (Paquet & Engelbrecht, 2003) taken in Evolutionary Computing for handling constraints in addition to an objective function; constraint-preserving, penalty, and repair methods.

Constraint-preserving methods only consider solutions contained within the feasible domain. This means that not only are initial solutions drawn from the feasible domain but also that the only permissible transformations of candidate solutions are those that result in a feasible solution. This method can be computationally wasteful as one ends up discarding computed infeasible solutions.

Penalty methods assign a cost with each type of undesirable condition, measuring the violation. These costs are aggregated to form a weighted sum of all the hard and soft constraint violations. One sums this aggregated quantity to the objective function to decrease the value or worth of infeasible solutions. By doing this, the constrained problem is transformed into an unconstrained problem where infeasible solutions are

permitted, thus forming a *constraint optimization problem*. While this method is frequently used, it nevertheless does not guarantee feasible solutions since the search is not limited to the feasible domain, making success highly dependent on the choice of penalty functions weights.

Repair methods attempt to fix infeasible solutions by replacing them with a nearby point within the feasible domain. This process can be computationally expensive as it is not even certain that one can find a feasible replacement.

Summary

The university examination timetabling problem has been shown to be a hard problem and one of significant importance. Additionally, it is an active research area even after decades of concentrated effort. We have also shown that the particle swarm optimization has shown promise in heuristically solving difficult problems while being relatively uncomplicated itself.

What is unknown is how well the particle swarm optimization algorithm works against the university examination timetabling problem. This approach is new. In fact, to the author's knowledge, this approach has never been tried on any timetabling problem. Therefore, not only is the PSO's efficacy unknown in this problem domain but the appropriate procedural steps are unknown at this point. Even more fundamentally, what are appropriate values for the control parameters and what initialization method should one consider using?

Contribution to the Field

This dissertation's contribution systematically investigates the impact of various factors in solving the university examination form of the timetabling class of problems and determines the algorithm's efficacy under the specified test environment. It provides insight into how well the PSO algorithm handles university examination timetabling problems and reveals the algorithm's utility, ability, and limitations in this domain.

Chapter 3

Methodology

Research Methods

Determining the suitability and effectiveness of the PSO algorithmic approach to the university examination timetabling problem is the purpose of this research effort. That determination will be reached by using an experimental design or Design of Experiments (DOE) approach. DOE is a well-accepted experimentation methodology that provides a means of determining which simulation runs to perform to obtain the essential information with the least number of experiments. An experimental design model consists of input control variables called *factors* and output values called *responses*, *measures*, or even *response measures*. The different values for a *factor* are *levels* and a *design* is a set of factor level pairs. In a heuristic experiment, common *factor* types to study are *Problem Factors*, *Algorithm Factors*, and *Test Environment Factors*. (Barr, Golden, Kelly, Resende, & Stewart, 1995; Beielstein, Parsopoulos, & Vrahatis, 2001; Hooker, 1996) The overall process of experimental design consists of:

- Identifying *factors* that may affect the result of an experiment,
- Designing the experiment so that effects of uncontrolled variables are minimized, and
- Use statistical analysis to separate the effects of the different *factors*.

This research focused on Problem and Algorithm Factors as the test environment in our case was fixed. For the problem factors, items such as number of students, number of rooms, seating capacity of rooms, number of exams, and the conflict matrix density

were all candidates. In the case of algorithmic factors, control variables such as neighborhood size and initial algorithmic control parameters were potential candidates. We did not study test environmental factors in the current research, as these have no direct relationship to this effort's stated purpose.

There are three actions one can take with factors; vary them, fix them, and ignore them. A researcher chooses to ignore factors assumed inefficacious concerning the experimental results. Other factors may be fixed because of testing assumptions, such as fixing the computer hardware and software choices for the testing environment. The factors used for study were chosen with the goals of the experiment in mind as the goals are only achieved through the analysis of these factors and measures. The following lists outline the factors investigated in each action group.

Problem Factors to Vary

- Number of exams, students, and enrollments
- Conflict Matrix Density
- Average number of exams per student
- Weighting value of clashing exams in objective function (see Equation 5)

Problem Factors to Fix

- Number of time slots
- Hard constraints
- Computing environment

- Enrollment distribution – i.e., characteristic of university examination timetables where a few exams have many students and a larger portion of exams have relatively few students

Problem Factors to Ignore

- Room capacities
- Resource (e.g., rooms and teachers) availabilities
- Exam requirements for special rooms (e.g., labs)
- Examination ordering (e.g., Exam A must precede Exam B)

Algorithm Factors to Vary

- Initialization method
- Number of particles relative to problem size
- Parameter settings

Algorithm Factors to Fix

- Particle's representation
- Particle's neighborhood topology
- Termination criteria

Algorithm Factors to Ignore

- None identified

Measures – Solution Quality

We define the *Average Particle Best Penalty* value as each particle's penalty value, at their respective previous best positions (p_{id} in Equation 1), averaged across all particles and replications for each iteration step. We used *Average Particle Best Penalty* as a measure of solution quality and plotted its value as a function of the iteration. By looking at each particle's best value, instead of the best swarm value, we got a truer sense of how the swarm's individual particles behaved on average, given the factors under investigation.

Burke, Eckersley, McCollum, Petrovic, and Qu (2003a) used average penalty per student on the Carter data set and defined it as the overall objective function penalty for the entire timetable divided by the number of students in the data set. This measure is really on an *effort* basis, however one measures effort. In our case, effort represents running the algorithm until the termination criteria is met.

Dividing our average particle best penalty by the total number of students gives the average penalty per student. This allowed us to compare our solution quality against published values, which we did on some of the real world examinations. We also computed average, minimum, maximum, and the population standard deviation for the last iteration.

Measures – Computational Effort

Our interest with computational effort was not in comparing this algorithm's effort with other algorithms given the same input factors, as results from comparisons of this nature can be misleading and there are too many uncontrollable variables with this approach. For example, if one uses published results for other algorithms, these

algorithms reflect results based on different computing hardware using possibly different software languages. Even if one were to use the exact same hardware and software environment, if the same source code is not used, then differences in algorithm implementation and code optimization can produce different results. Some researchers implement the alternate algorithms using their test software and hardware environment in an attempt to eliminate these differences. Nevertheless, even here, factors such as programming skills and tuning ability vary between implementations.

We limited ourselves to reporting on our implementation's average computational effort per iteration for a couple of the test suites. This provided us with a general sense of how the computational effort varies across problem size while permitting us to ignore such uncontrollable environmental variables as garbage collection, memory caching, and operating system background processing, all of which affect timing. We leave a more thorough analysis of computational effort for a future study.

Measures – Robustness

As Bartz-Beielstein (2003) points out, "Robustness can be defined in many ways, i.e. as a good performance over a wide range of instances of one test problem or even over a wide range of different test problems" (p. 11). Barr, Golden, Kelly, Resende, and Stewart (1995) state that, "Generally, robustness is based on the ability of a heuristic to perform well over a wide range of test problems and is usually captured through measures of variability. For example, the quality-versus-time graph in Figure 4 also includes standard deviation bars with the average quality points" (p. 16).

This study investigated the algorithm's robustness with respect to the university timetabling problem by studying solution quality for both a fixed set of factors and a

range of factors. These tests show us two things. Firstly, the "fixed set of factors" tests show how predictable (i.e., consistent) the solution quality is when starting multiple times from the same set of factors. If the solution quality varies widely for tests having the same starting factors, then one cannot have a high level of confidence in the solution's quality when given only a single run. Thus, for real world problems, this would require one to perform multiple runs in the hopes of finding a good solution, which is not a desirable scenario.

Secondly, the "range of factors" tests show how predictable the solution quality is over data sets having different sizes and conflict matrix densities. Ideally, one wants an algorithm that provides quality solutions over a wide range of starting factors. For example, an algorithm that provides high-quality solutions only for data set sizes or conflict matrix densities within a very narrow range is of limited use in the real world.

Specific Research Procedures

Introduction

The goal of this research was to determine the suitability and effectiveness of Particle Swarm Optimization techniques when applied to the University Examination Timetabling Problem. We used data sets, both fabricated and real, spread over a spectrum of sizes and densities to analyze PSO performance profile on this class of timetabling problems. In addition, the results were compared against those from other published studies that used alternate algorithms.

The following subparagraphs describe the approach used to perform this evaluation. Not only is the PSO algorithm covered but also topics such as Initialization,

choice of Objective Function, Termination Criterion, and Data Set properties. We discuss all major aspects of this research effort's investigation.

First, we describe some important concepts, methods, and data sets and then we present the research steps.

PSO Algorithm – Swarm Initialization Method

This research studies the effect of both random and heuristic ordering initialization of the swarm.

Random Initialization Methods

Instead of using a QRNG, our study will use the more common PRNG and compare its results against the heuristic ordering methods. Implementations of the PRNG method were readily available to us from within the SQL Server 2000 environment and we were more interested in how the more commonly used PRNG implementation faired against the heuristic approaches described later.

Heuristic Ordering Initialization Methods

Burke, Newall, and Weare (1998b) demonstrated that these heuristic ordering methods could provide good initial solutions for evolutionary algorithms used on timetabling problems. If this were true, then the PSO algorithm might make better use of its time in fine-tuning the solution. Therefore, our initial expectation was that the use of these heuristic ordering methods would provide us with a better quality initial swarm. Interestingly, our results revealed something quite different.

To perform this test, we chose the Least Saturation Degree heuristic ordering method, also referred to as just Saturation degree (Carter, Laporte, & Lee, 1996) and used by others for producing an initial timetabling solution (Bykov, 2003), as it generally

produces the best results of all the heuristic methods.

Heuristic Initialization Implementation

As specified, the above heuristic ordering methods do not stipulate how to resolve ties in the ordering. Other authors have used a number of methods to break ties. The approaches can vary between heuristics and even between authors. Additionally, as each initial swarm particle represents a different timetabling solution, one must generate multiple initial solutions. Here again, we took the approach used by Bykov (2003) and solved both these problems by randomly assigning timeslots from among the available ones. Procedurally, we used a random number, sorted in ascending order, as a secondary sort order on the generated list. That way, using the heuristic to specify the primary sort order, this secondary sort order automatically breaks any ties.

Though others used heuristic ordering methods to solve the examination timetabling problem, we used the heuristics to generate a suitable initialized swarm population.

PSO Algorithm – Random Number Generation

This study uses Microsoft's Visual Basic .NET 2002 (VB.NET) language to implement the algorithm. Although VB.NET contains a PRNG function, as mentioned above, it does not meet statistical requirements for randomness. So, instead, we used the RAND() function found in Microsoft's SQL Server Transact-SQL (T-SQL) data base language. Connolly (2004, March 1) establishes its validity as a random number generator and Novick (2003, April 8) provides a method that made this function useful for our algorithmic implementation.

PSO Algorithm – Particle Representation

One of the primary issues in designing an effective PSO algorithm is to identify an appropriate mapping between the problem solution and the PSO particle's position. A *direct* chromosome representation is the customary approach used with Genetic Algorithms for mapping the problem solution to the examination timetable. In the direct representation, each chromosome represents the timetable itself. An array of integers, indexed by exam, represents the chromosome whose value corresponds to a timeslot for the exam (Terashima-Marín, Ross, & Valenzuela-Rendón, 1999). We used a similar approach to form the mapping between problem solution and the PSO particle. In our model, the particle's position in *D*-dimensional space represents a timetable, where *D* is the number of exams. Each exam maps directly to a dimension and each dimension's domain is the permissible timeslots, which are discrete values.

PSO Algorithm – Particle Representation – Discrete Search Space

Though the PSO technique's effectiveness in solving real-valued optimization problems is well documented (Parsopoulos & Vrahatis, 2002b), literature detailing discrete PSO research is limited. The more notable works in this area are Kennedy and Eberhart's (1997) discrete binary PSO version, and Al-kazemi and Mohan's (2002a) multi-phase discrete PSO method. Both works use a discrete-binary encoded representation. Another notable effort is Laskari, Parsopoulos, and Vrahatis's (2002) research in Integer Programming. These papers notwithstanding, nearly every documented PSO implementation deals with continuous space, where changes in particle's position and velocity have a natural representation. Regardless, the previous research, involving discrete methods such as binary and integers, shows the ability of the

PSO method in efficiently handling high dimensional discrete domain optimization problems.

This research used an approach similar to the one adopted by Laskari, Parsopoulos, and Vrahatis's (2002). That is, we truncated each particle's position value to the nearest integer, or time slot in our case, after the PSO algorithm determines its new position. These same authors performed a second set of experiments where the truncation was progressive. For example, six decimal digits of precision were used for the first 50 iterations, four were used for the next 100, two were used for the next 100, and the rest were simply truncated. We briefly investigated this approach and found, as did the original authors, that there was essentially no difference between the two methods.

PSO Algorithm – Objective and Constraints

Our research followed a similar tact to the one taken by Carter, Laporte, and Lee (1996) in their constrained optimization approach using graph coloring heuristics to initialize our algorithm and limiting ourselves to only considering the Examination Spread cost function during the optimization portion. Carter et al. (1996) discuss research performed against real world data with the addition of soft-constraints. They incorporated soft-constraints into their algorithm through a simple method; that is, "whenever an attempt is made to schedule an exam, feasibility with respect to the side constraints has to be checked in addition to potential conflicts" (p. 379). Burke, Newall, and Weare (1998a) point out that spread constraints cause the greatest problems when solving exam timetabling challenges as these constraint types occur most often. Therefore, as Carter et al. did for their initial testing, we limited our research efforts to the examination spread cost function and our research did not look at soft-constraints.

Additionally, we did not use the clique-processing step they mention. Nevertheless, the study of soft-constraints is a natural extension of the current study though and its incorporation into the algorithm is straightforward, as evidenced by the approach mentioned above.

The Examination Spread cost function's purpose is to force the reduction of second order, and higher, conflicts. The function is formulated in such a manner that second order conflicts receive the greatest penalty and higher order conflicts receive proportionally smaller penalty values. This causes the dispersion of clashing examinations across the full exam period on a per student basis, where clashing examinations are any two examinations that share a common student. The intent is to provide students with ample preparation time between examinations and minimize the chances of back-to-back examinations.

Carter, Laporte, and Lee (1996) used the same cost function used by Laporte and Desroches (1984), based on the sum of proximity costs, and given as

$$w_s \equiv \frac{32}{2^s}, \quad s \in \{1, ..., 5\} \tag{4}$$

where w_s is the weight given to clashing examinations scheduled s periods apart. The penalty value is equal to this proximity cost weight multiplied by the number of students involved in the clash. Finally, the objective function value equals these penalty values aggregated over all clashes and divided by the total number of students, giving us the average penalty per student. Other examination spread functions and approaches exist in the literature (Burke & Newall, 2004; Zhaohui & Lim, 2000), all having this same basic

aim, but we did not consider these.

Although Carter, Laporte, and Lee (1996) refer to the function in Equation 4 as a cost and claim to use no side constraints in their initial set of tests, others refer to this function as a proximity constraint, spread constraint, or even a soft-constraint (Burke, Eckersley, McCollum, Petrovic, & Qu, 2003a; Burke & Petrovic, 2002). This variation in terminology is partly because soft-constraints are routinely handled by incorporating them directly into the objective function. In actuality, it is a cost function or objective function, as spreading the examinations evenly throughout the exam period is an objective of most examination timetabling problems and not an inviolable rule, as implied by a hard-constraint. In addition, some look upon the proximity constraint as reflecting a preference, which implies a soft-constraint when viewed this way. Regardless, for the examination spread objective, we use the two forms of reference interchangeably. That is, we refer to this objective as a cost function, objective function, or constraint in the context of the referenced work; otherwise, we refer to the spread cost function as the examination spread cost function or simply as the examination spread.

Along with the single hard-constraint, preventing examination conflicts and the examination spread cost function, Carter, Laporte, and Lee (1996) made the following assumptions:

- Time gaps between consecutive time slots, such as overnight and weekends, were ignored when computing the examination spread.
- No limit was set on the total number of seats available during each time slot.

PSO Algorithm – Constraint Handling

The PSO algorithm has two main process sections: initialization and optimization. In either of these two sections, the solution points can either be restricted to the set of feasible solutions or drawn from the larger set that also includes infeasible solutions. These two algorithmic sections, when restricted to either the feasible or the infeasible domains, provide four possible scenario combinations. For example, the scenario where solutions are restricted to the feasible domain for both the initialization and optimization sections is a good match for the constraint-preserving method of handling constraints. Of course, the repair method of handling constraints is applicable to this scenario too. The penalty constraint handling method is well suited to handling the two scenarios having infeasible search space for the optimization section, where one uses feasible initial solutions and the other uses infeasible. Finally, we have the case where the initialization section considers the points within the infeasible search space and its optimization phase only considers those within the feasible domain. This last scenario requires the algorithm to consider only solutions from the highly restrictive feasible domain after generating initial points in infeasible domain. It is possible that the initial solutions in this case are topologically distant from any feasible solutions; putting a significant computational burden on the algorithm. For this reason, we did not consider this fourth and last case at this time.

We elaborate on these scenario combinations below, where each subparagraph's caption indicates the approach discussed. For example, the approach that considers only feasible solutions during initialization and infeasible solutions during the optimization

process has **Feasible/Infeasible Approach** as its caption. As mentioned above, we are not considering the Infeasible/Feasible Approach.

Feasible/Feasible Approach

This approach considers the case where both initialization and optimization processes draw only from the feasible search space for candidate solutions. This follows the general method for solving examination timetabling problems (Carter, Laporte, & Chinneck, 1994), which typifies the constraint-preserving method. The algorithm consists of two core steps: the use of a graph coloring heuristic to generate initial feasible solutions, and an optimization process where the PSO algorithm is used to optimize the objective function while only accepting feasible solutions.

1. *Create Initial Feasible Solutions*. For this step, we used the computationally expensive graph coloring heuristics with backtracking to create initial feasible timetables. As was done by Carter, Laporte, and Lee (1996), we only considered the examination conflicts hard-constraint during this step. Because we used the PSO algorithm, we required multiple initial particles for the swarm. Each particle represented a timetable in our encoding scheme, which in turn required the generation of multiple initial feasible solutions. We accomplished this using the following procedural steps:
    1.1. Execute the graph coloring heuristic for each required particle. The use of randomness when breaking ties produces multiple solutions, though this does not guarantee a solution each time, let alone a unique one. We want to stress that it is not always possible to find multiple feasible solutions. If the problem is over-constrained, then by definition, a single solution does not even exist.

In these cases, we may end up with fewer than the desired number of initial particles or even none. To resolve this situation when it arises, we perform the following two additional steps.

1.2. If step 1.1 produced an incomplete set of initial particles, then, for every missing particle, use a solution randomly selected from the set of successfully created ones.

1.3. If step 1.1 produced no feasible solution, then flag the data set with a "Failure" indicator.

2. *Perform Optimization Process*. This step incorporates optimization into the algorithm by way of the objective function. Normally, one would perform this step by repeatedly testing solution points in *D*-dimensional space for feasibility. This approach would then terminate only after finding a feasible solution, making its termination indeterminate. If the number of feasible to infeasible solutions is very small in the search vicinity, then this approach could take a very long time before discovering a feasible solution. For this reason, we devised an approach that requires only a slight modification to the canonical PSO algorithm and results in deterministic termination and still prohibits illegal assignments. The general steps are:

2.1. Use the canonical PSO algorithm to compute the particle's new position but do not move the particle to this position.

2.2. Until all of the particle's dimensions have been checked, do the following:

2.2.1. Randomly select one of the particle's unchecked dimensions.

2.2.2. If moving the particle along this dimension to the new position, while

holding the values of the other dimensions constant, results in a new feasible solution, then update this dimension to its new value. That is, move the particle along this dimension.

2.2.3. Otherwise, do not move the particle along this dimension.

The ability of this method to discover better-quality feasible solutions depends in large part on the density of alternate feasible solutions around the initial solution point. It was thought that this altered PSO algorithm, which we will referred to as the PSO-NoConflicts optimization algorithm, would be better able to maintain the explorative qualities of the algorithm well enough that finding improved feasible solutions would not be a problem. Moreover, in the case where there is a scarcity of feasible solutions, at least this method deterministically terminates.

Infeasible/Infeasible Approach

This approach, which more closely matches the original PSO approach for solving problems, considers the case where both initialization and optimization processes draw candidate solutions from the infeasible search space. The algorithm consists of two core steps: the random generation of initial, potentially infeasible, solutions and an optimization process, which uses the PSO algorithm to optimize the objective function. In this case, we used the penalty method, via the objective function, for handling constraints, both hard and soft.

This differs from the previous approach in that infeasible solutions are not restricted but instead just heavily penalized in much the same manner as Burke and Newall (2004). Minimization of the objective function, now encompassing both hard and soft constraints, guides the algorithm to search in ever more promising areas of the

search space. Our goal was that by suitably weighting hard and soft constraint violations through the penalty function, the PSO would find satisfactory solutions to the timetabling problem. The term "satisfactory" is of course subjective with respect to the one seeking a solution. What we are saying is that there are no infeasible solutions within this approach, only feasible solutions having varying degrees of quality or acceptability (i.e., all solutions are considered feasible by definition). Of course, this approach does not guarantee a feasible solution, as it is drawing from the infeasible search space.

Construction of the objective function for this approach requires penalty functions for all hard and soft constraints. As there may be a large range in the domains among the penalty functions, typically, one normalizes and weights the function values so each more accurately represents its significance in the overall objective. The process of determining the weights is typically empirical and therefore iterative by nature. As we were only interested in hard-constraints and the examination spread cost function for this study, we started with a reformulation of the Carter, Laporte, and Lee (1996) Equation 4 and used the following:

$$w_s \equiv \frac{32}{2^s}, \quad s \in \{0, ..., 5\} \tag{5}$$

The difference between Equation 4 and 5 is that Equation 5 now considers the case where two exams share the same time slot. This occurs when *s* equals zero and produces a penalty value of 32. Now this altered objective function not only provides a penalty for the examination spread but it also penalizes, to a proportionally larger degree, exam clashes. That is, first-order through sixth-order conflicts provide values for the objective function. This reformulation was our starting position for the penalty

approach. Additionally, we ran tests with larger weights to investigate how these affected the occurrence of examination clashes.

Feasible/Infeasible Approach

This final approach is merely a hybrid of the previous two approaches. In this case, initialization draws candidate solutions from the feasible search space and the optimization portion draws them from the infeasible search space. The algorithm consists of two core steps: the use of a graph coloring heuristic to generate initial feasible solutions, and an optimization process where the PSO algorithm optimizes the objective function.

PSO Algorithm – Control Parameters

Carlisle and Dozier (2001) performed a study and established a set of default values for the control parameters, which performed well in the majority of scenarios tested. Our original intention was to use these default control parameters unless preliminary testing indicated the need for a reassessment of these values. Pilot testing with these published default values suggested the existence of parameter values more suitable to the timetabling problem domain. Therefore, we designed our first test suites to uncover more appropriately tuned parameter values.

PSO Algorithm – Premature Convergence

We used an initialization method designed to get the swarm very near feasible solutions or right on them. Our hope was that, by starting the particles off very close to feasible solutions, we would be in close proximity to a good solution during the optimization phase and bypass many local sub-optimal locations in the search space.

PSO Algorithm – Algorithm Termination

Algorithms are normally designed to terminate when at least one of the *stopping criteria* is satisfied. The *stopping criteria* are specified conditions under which the algorithm terminates. Examples of termination conditions commonly found in the literature are:

- Number of iterations reaches a preset maximum number.
- Preset maximum length of clock time passes.
- Preset maximum amount of CPU time consumed.
- Preset maximum number of iterations passes since last change to the best solution.
- Best solution comes within a delta of a predetermined fitness function value.

According to Barr, Golden, Kelly, Resende, and Stewart (1995), unlike straightforward algorithms, which usually have well-defined termination criteria, complex algorithms tend not to have standard termination rules; instead, simply searching for improved solutions until reaching an arbitrary stopping point. Though longer searches tend to produce better results, in the case of the standard PSO algorithm, premature convergence to local minimum usually occurs. At this point, no amount of additional time will improve the result. Nevertheless, this study used a preset maximum number of iterations criterion, specified at the outset of each experiment, as the algorithm requires a deterministic method of termination.

Testing

Ensuring that necessary and sufficient testing takes place requires a methodology. Greenberg (1990) identifies two methods of computational testing, which are:

- *statistical analysis* – presumes random generation over a problem space and collects performance values of replicated trials.
- *library analysis* – uses a fixed library generally available to the professional community and whose properties are already known or are reported along with the computational test results. (p. 95)

The former method corresponds to our use of stochastically generated data sets and the latter to our use of the publicly available university examination timetabling data sets. Both testing methods were used to analyze the PSO algorithm's suitability in solving the university examination timetabling problem.

Testing – Data Sets

Nearly all timetabling research results are based on studies performed against artificial data sets that were programmatically fabricated (Carter, Laporte, & Lee, 1996). The reason for this is twofold. First, one invariably wants more control and variation over the data set than provided by actual data. Secondly, there is an underlying assumption that controlled sets of input data offer better opportunities to exercise the algorithm than is obtainable through a snapshot of real world data. This study used both fabricated data and real world university examination timetabling data.

Testing – Data Sets – Real World

Barr, Golden, Kelly, Resende, and Stewart (1995, p. 20) state, "Real-world problems reflect the ultimate purpose of heuristic methods, and are important for assessing the effectiveness of a given approach." As such, real world data for this study came from the University of Melbourne, Department of Mathematics and Statistics Web site, Operations Research Group (13 October 2003), under the "Public Data" section. In

particular, we used the University of Toronto data set. This data set contains 13 university examination timetable sets from 11 different institutions and it afforded us many opportunities to make comparisons against other university examination timetabling research efforts. The extensive use of this university examination data set, first used by Carter (1986) and commonly referred to as the "Carter data set," makes comparison with other heuristic research heuristic methods straightforward. In addition, the set's public availability allowed us to know the precise input for these alternate methods.

The Operations Research Group web site also contains two other data sets: one from the University of Nottingham and the other one from the University of Melbourne used by Merlot, Boland, Hughes, and Stuckey (2002). These sets were not included in this study.

The following is only a partial listing of previous research efforts that used the Carter data sets:

- Burke, Eckersley, McCollum, Petrovic, & Qu (2003a, 2003b)
- Burke & Newall (1999)
- Burke & Newall (2004)
- Burke, Newall, & Weare (1995)
- Carter & Laporte (1996)
- Carter, Laporte, & Lee (1996)
- Di Gaspero & Schaerf (2000)
- Ross, Hart, & Corne (1998)

Testing - Data Sets - Real World - Other Attributes

Table 4 through Table 6 present many other attributes related to the 13 examination data sets found in the Carter data set. The meanings should be self-explanatory based upon column headings. The features were extracted from the data sets to provide a detailed view of the test data attributes for use during the analysis phase of this research effort. These tables represent, to the author's knowledge, the first comprehensive feature listing across all data sets. Some of the listed features are found throughout various reported research articles. For example, the "Maximum Clique" and "# of periods" columns listed in Table 5 come from Carter and Johnson (2001) and Burke, Eckersley, McCollum, Petrovic, and Qu (2003a), respectively.

Table 4. Attributes of Carter Data Sets – 1 of 3

Data Set	Institution	# of exams	# of students	enrollment	# of exams having a clash	conflict matrix density	# of conflict edges	students with ≤ 1 exam
CAR-F-92	Carleton University, Ottawa	543	18419	55522	542	0.138	20305	3969
CAR-S-91	Carleton University, Ottawa	682	16925	56877	678	0.128	29814	3409
EAR-F-83	Earl Haig Collegiate Institute, Toronto	190	1125	8109	190	0.266	4793	1
HEC-S-92	École des Hautes Études Commerciales, Montreal	81	2823	10632	81	0.415	1363	321
KFU-S-93	King Fahd University of Petroleum and Minerals, Dharan	461	5349	25113	444	0.055	5893	276
LSE-F-91	London School of Economics	381	2726	10918	379	0.062	4531	99
PUR-S-93	Purdue University, Indiana	2419	30032	120681	2413	0.029	86261	2630
RYE-S-93	Ryerson University, Toronto	486	11483	45051	485	0.075	8872	2025
STA-F-83	St. Andrew's Junior High School, Toronto	139	611	5751	139	0.143	1381	0
TRE-S-92	Trent University, Peterborough, Ontario	261	4360	14901	260	0.180	6131	667
UTA-S-92	Faculty of Arts and Sciences, University of Toronto	622	21266	58979	622	0.125	24249	6180
UTE-S-92	Faculty of Engineering, University of Toronto	184	2750	11793	184	0.084	1430	79
YOR-F-83	York Mills Collegiate Institute, Toronto	181	941	6034	181	0.287	4706	1

Table 5. Attributes of Carter Data Sets – 2 of 3

Data Set	Institution	Maximum Clique		# of periods	avg. (max) # of clashes per student	enrollments per student			
		Size	Number			average	standard deviation	max	(count)
CAR-F-92	Carleton University, Ottawa	24	3	32	2.57 (6)	3.01	1.46	7	(29)
CAR-S-91	Carleton University, Ottawa	23	156	35	2.96 (8)	3.36	1.57	9	(1)
EAR-F-83	Earl Haig Collegiate Institute, Toronto	21	7	24	6.21 (9)	7.21	1.20	10	(9)
HEC-S-92	École des Hautes Études Commerciales, Montreal	17	1	18	3.12 (6)	3.77	1.44	7	(1)
KFU-S-93	King Fahd University of Petroleum and Minerals, Dharan	19	2	20	3.90 (7)	4.69	1.36	8	(11)
LSE-F-91	London School of Economics	17	2	18	3.12 (7)	4.01	0.99	8	(3)
PUR-S-93	Purdue University, Indiana				3.31 (8)	4.02	1.42	9	(1)
RYE-S-93	Ryerson University, Toronto				3.55 (9)	3.92	2.08	10	(1)
STA-F-83	St. Andrew's Junior High School, Toronto	13	60	13	8.41 (10)	9.41	1.22	11	(209)
TRE-S-92	Trent University, Peterborough, Ontario	20	4	23	2.85 (5)	3.42	1.41	6	(20)
UTA-S-92	Faculty of Arts and Sciences, University of Toronto	26	128	35	2.50 (6)	2.77	1.50	7	(23)
UTE-S-92	Faculty of Engineering, University of Toronto	10	4	10	3.39 (5)	4.29	1.01	6	(20)
YOR-F-83	York Mills Collegiate Institute, Toronto	18	32	21	5.42 (13)	6.41	1.80	14	(1)

Table 6. Attributes of Carter Data Sets – 3 of 3

Data Set	Institution	avg. (max) # edge weight per exam	avg. (max) # of clashes per exam	avg. (max) edge weight per clash	enrollments per exam		
					average	standard deviation	maximum
CAR-F-92	Carleton University, Ottawa	278.60 (4730)	74.93 (381)	3.72 (290)	102.25	142.02	1566
CAR-S-91	Carleton University, Ottawa	259.39 (4718)	87.95 (472)	2.95 (321)	83.40	101.18	1385
EAR-F-83	Earl Haig Collegiate Institute, Toronto	273.49 (1665)	50.45 (134)	5.42 (192)	42.68	44.37	232
HEC-S-92	École des Hautes Études Commerciales, Montreal	435.26 (2315)	33.65 (62)	12.93 (535)	131.26	131.72	634
KFU-S-93	King Fahd University of Petroleum and Minerals, Dharan	231.14 (5089)	26.55 (247)	8.71 (997)	54.48	135.87	1280
LSE-F-91	London School of Economics	93.67 (1229)	23.91 (134)	3.92 (204)	28.66	48.95	382
PUR-S-93	Purdue University, Indiana	176.23 (6789)	71.50 (857)	2.46 (966)	49.89	105.75	1961
RYE-S-93	Ryerson University, Toronto	373.89 (5118)	36.59 (274)	10.22 (617)	92.70	121.98	943
STA-F-83	St. Andrew's Junior High School, Toronto	354.60 (2090)	19.87 (61)	17.85 (209)	41.37	45.22	237
TRE-S-92	Trent University, Peterborough, Ontario	171.86 (1267)	47.16 (145)	3.64 (150)	57.09	67.93	407
UTA-S-92	Faculty of Arts and Sciences, University of Toronto	244.70 (4382)	77.97 (303)	3.14 (824)	94.82	149.92	1314
UTE-S-92	Faculty of Engineering, University of Toronto	226.09 (1847)	15.54 (58)	14.55 (301)	64.09	71.39	482
YOR-F-83	York Mills Collegiate Institute, Toronto	197.26 (779)	52.00 (117)	3.79 (80)	33.34	26.53	175

Testing – Data Sets – Real World – Data Set Structure

In mathematical terminology, a *graph* is a set of lines that connect a possibly empty set of points. These graph points are commonly known as *vertices* or *nodes* and the lines as *arcs* or *edges*. A *clique* is any subset of graph nodes having every node pair in the subset connected by an edge. A *maximal clique* is a clique that is not contained in any other clique and a *maximum clique* is a graph's largest maximal clique.

As mentioned elsewhere in this document, the basic examination timetabling problem can be mapped to a graph coloring problem where a graph node corresponds to an exam and an edge indicates that at least one student is taking both connected exams. In this case, a clique corresponds to a set of examinations that must be scheduled in distinct time slots. Given this, it is readily apparent that the number of nodes (exams), within a timetable graph's maximum clique, correspond to the minimum number of time slots required in preventing a clash constraint violation. Hence, the maximum clique provides us with a lower bound in the number of time slots required to schedule the exams, regardless of the number of maximum cliques. Carter and Johnson (2001) point out that the "graphs tend to have fairly large cliques compared with random graphs with the same density" (p. 538). Using a graph's maximum clique to initialize examination timetabling problems is common and this approach has been studied by a number of authors (Carter, Laporte, and Lee, 1996; Carter & Johnson, 2001; Petrovic, Yang, & Dror, 2003). Unfortunately, the determination of a graph's maximum clique is an NP-Complete problem (Papadimitriou & Steiglitz, 1982/1998).

We use the aforementioned clique information to show that timetabling problems exhibit a definite non-random structure. This structure is largely due to course selection

biases caused by degree curriculum requirements. That is, the probability of a student selecting a particular course has a built in bias towards courses within a student's chosen degree program. This results in dense clusters—i.e., a set of nodes, or subgraph, within the graph having higher probability of connectivity with other nodes within the same subgraph—of university examinations in the graph with relatively sparse edge connectivity between the clusters (Carter & Johnson, 2001; Erben, 2001; Newall, 1999). As an example, Burke, Elliman, Ford, and Weare (1995) observed that the average probability of an exam conflicting with an exam from a different department at the University of Nottingham is 0.023 while the probability is 0.281 between exams from the same department.

Carter, Laporte, and Lee (1996) devised a method, adopted by others (Petrovic, Yang, & Dror, 2003; Burke, Petrovic, & Qu, 2004), which fabricates university examination test data sets with specific conflict matrix densities. Carter et al. (1996) constructed test data by "creating students one at a time, and then assigning them to r randomly selected courses, where r follows a discrete uniform distribution in the interval [2, 6]. This process ends when a specified density d is reached" (p. 376). These studies generated data sets by altering the number n of examinations and the density d of a conflict matrix. In the Carter et al. study, "ten different instances were generated for each combination of n = 200, 400 and d = 0.05, 0.15, 0.25" (p. 376).

Unfortunately, for test purposes, data sets generated using the Carter, Laporte, and Lee (1996) do not exhibit the fundamental non-random structure of actual university examination timetabling data sets described earlier. To see this deficiency, we plot the

number of weighted examination conflicts (i.e., conflicts weighted by the number of students enrolled in conflicting exams) for each exam in descending order of magnitude.

The data plotted in Figure 2 was generated using the method of Carter, Laporte, and Lee (1996). We randomly picked examinations, as prescribed by the algorithm, from the Carter data set's London School of Economics (LSE-F-91) data. Thus, the number of examinations, 381, matches the number found in the LSE-F-91 data set. Additionally, the conflict matrix density matches the LSE-F-91 density, but the number of students and enrollments differ.

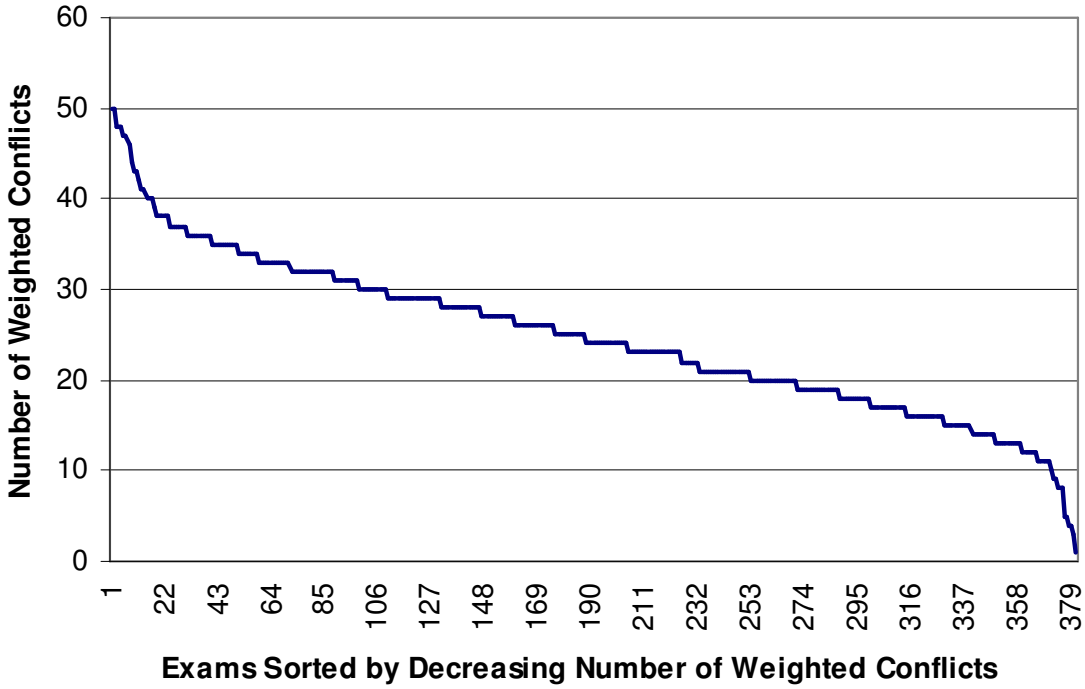

Figure 2. 381 LSE-F-91 exams, 2731 enrollments, 693 students, 0.062 density

Contrast this plot with the one found in Figure 3, which also shows the number of weighted examination conflicts for each exam in descending order of magnitude, but for the actual and complete LSE-F-91 data set. (The abscissa dimension ends at 379, instead

of 381, because two of the exams have no corresponding enrollment in this data set.) It is readily apparent that the structures of these two plots differ considerably. In fact, if one were to plot the other 12 data sets found in the Carter data set, representing other universities, one would obtain plots having forms nearly identical to that of Figure 3, irrespective of their conflict matrix density or data set sizes.

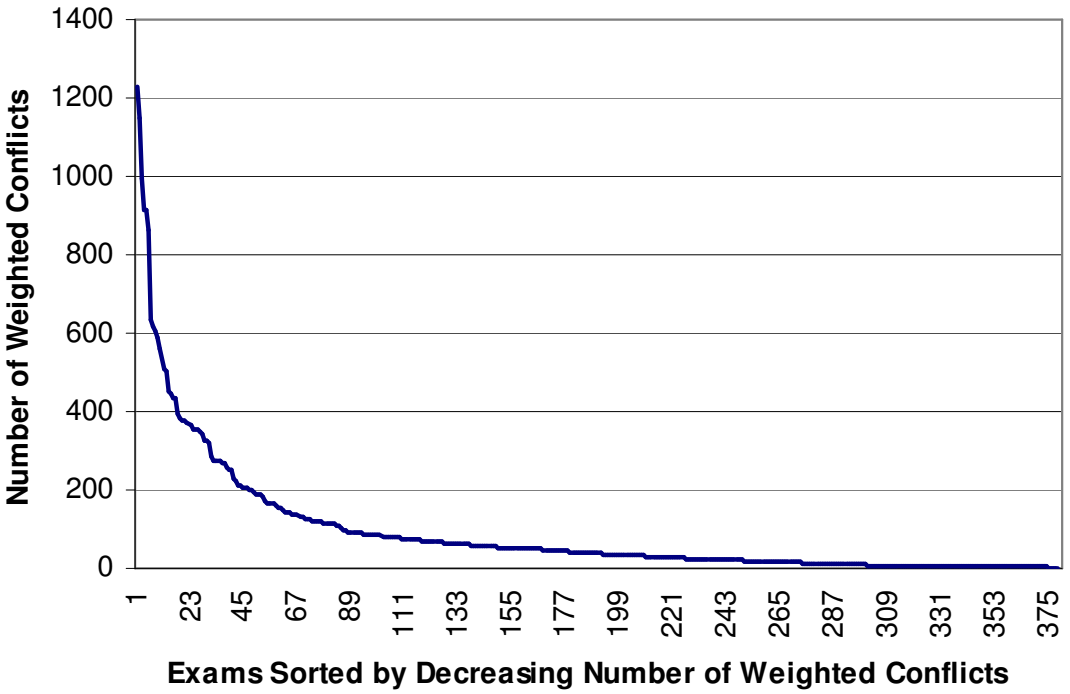

Figure 3. 381 LSE-F-91 exams, 10918 enrollments, 2726 students, 0.062 density

It is because of this disparity in data structures between that used by Carter, Laporte, and Lee (1996) and that found in real world data sets that we additionally used fabricated and reduced size real world data sets for testing; both describe more fully below. Moreover, although real-world data sets provide realistic input for exercising an algorithm, in the case of university examination timetabling at least, their sheer size tends to make them unwieldy for analysis; that is, their size makes testing the effects of

variations in factors cumbersome because the testing cycle-time is so long. Providing structurally similar data sets of varying sizes not only increased the number of available test sets but also increased the range of sizes over which we exercised the algorithm, which permitted us to see how measures change with variations in problem size.

Maintaining Structure while Removing Non-Conflicting Enrollments

An enrollment (i.e., a student registered in a course) takes part in an examination conflict only if the related student is registered in at least one other course. Hence, an enrollment participates in a conflict only if the student has other courses. To restate, for our timetabling problem under consideration, every course has one and only one examination; so, in the contexts of our discussion, *course* and *examination* are synonymous. From this, it is apparent that removing enrollments not participating in conflicts is equivalent, with respect to conflict generation, to removing students having only one examination. This action does not alter the conflict matrix density of the problem, as all entries within the conflict matrix are a result of students having more than one exam.

Burke, Eckersley, McCollum, Petrovic, and Qu (2003a) observed that some of the Carter data sets contained significant numbers of students who had only one enrollment. For example, data sets UTA-S-92 and CAR-S-91 in Table 4 have greater than 29% and 20% of the students respectively with this quality. This is readily seen by comparing the values in the "# of students" column against those in the "students with ≤ 1 exam" column for these particular data sets. A quick comparison of the other data sets listed show similar reductions possible. Removing these students from the data set will reduce the computation overhead without altering the conflict matrix density or the structural

qualities of the graph problem that are of interest to us. As these authors point out, students having only a single enrollment are able to take their examination any time without possibility of an examination clash. Of course, this ignores soft-constraints that might apply, such as the room-capacity constraint, but, like the authors, this study ignored these constraints.

This study used a "reduced" real world data set for test. The set, referred to throughout the rest of this dissertation as the Reduced Carter Data Set, was produced by removing the students having only one exam, and does not represent a reduction in exams. This reduction in students leads to a subsequent reduction in enrollments. The attributes of this reduced data set are shown in Table 7. The suffix notation of "-R" on the Data Set name indicates that it represents the reduced version of the full Carter set.

Table 7. Attributes of Reduced Carter Data Sets

Data Set	Graph Density	# of Exams	# of Students	# of Enrollments	Enrollments / Student	Enrollments / Exam	Exams w/ Clash	Conflict Edges
CAR-F-92-R	0.138	543	14450	51553	3.57	94.94	542	40610
CAR-S-91-R	0.128	682	13516	53468	3.96	78.40	678	59628
EAR-F-83-R	0.266	190	1124	8108	7.21	42.67	190	9586
HEC-S-92-R	0.415	81	2502	10311	4.12	127.30	81	2726
KFU-S-93-R	0.055	461	5073	24837	4.90	53.88	444	11786
LSE-F-91-R	0.062	381	2627	10819	4.12	28.40	379	9062
PUR-S-93-R	0.029	2419	27405	118054	4.31	48.80	2413	172522
RYE-S-93-R	0.075	486	9458	43026	4.55	88.53	485	17744
STA-F-83-R	0.143	139	611	5751	9.41	41.37	139	2762
TRE-S-92-R	0.180	261	3693	14234	3.85	54.54	260	12262
UTA-S-92-R	0.125	622	15086	52799	3.50	84.89	622	48498
UTE-S-92-R	0.084	184	2672	11715	4.38	63.67	184	2860
YOR-F-83-R	0.287	181	940	6033	6.42	33.33	181	9412

Testing – Data Sets – Atypical Fabricated Data Sets

Carter, Laporte, and Lee (1996) devised a method, adopted by others (Petrovic, Yang, & Dror, 2003; Burke, Petrovic, & Qu, 2004), which fabricates university examination test data sets with specific conflict matrix densities. Carter et al. (1996) constructed test data by "creating students one at a time, and then assigning them to *r* randomly selected courses, where *r* follows a discrete uniform distribution in the interval [2, 6]. This process ends when a specified density *d* is reached" (p. 376). These studies generated data sets by altering the number *n* of examinations and the density *d* of a conflict matrix. In the Carter et al. study, "ten different instances were generated for each combination of $n = 200, 400$ and $d = 0.05, 0.15, 0.25$" (p. 376).

The Carter, Laporte, and Lee (1996) method for constructing timetabling data sets, described earlier, is "typically" used by others. We chose to refer to the data set produced via this method as "atypically" due to the fact the its structure, as shown earlier, is in fact atypical of real world data sets. As such, we also chose not to use this data set in our testing as any conclusions drawn from its use would be of questionable value. Instead, we chose to devise an algorithmic approach that produces fabricated data sets exhibiting structure similar to the real world data sets. We refer to these data sets throughout the rest of this thesis as the Typical Fabricated Data Sets as their structure typifies real world data set structures.

Testing – Data Sets – Typical Fabricated Data Sets

Our goal in constructing a fabricated data set that typified the real world data was to construct data sets that emulate the structure found in Figure 3. Specially, a data set, which when plotted in the same manner, produced a graph of similar form; unlike the

Carter, Laporte, and Lee (1996) method, which, though easily constructed, does not produce a truly representative data set. That is to say, we generated test data sets that more accurately mirror the associations between students and exams, expressed through the enrollment relationship, found in real world data sets.

We used this algorithm to generate a number of examination sets. We chose the values 100, 200, and 400 for the number of exams in the data sets, and 0.04, 0.08, 0.16, and 0.32 for the conflict density values. The combination of these factors produced 12 distinct examination data sets, which we designated using the notation *nnndd*, where *nnn* was a value from the set {100, 200, 400} and *dd* a value from the set {04, 08, 16, 32}. For example, the designation "20016" refers to the data set having 200 exams and a conflict matrix density of 0.16.

We now give an outline of the algorithm.

Given an objective to create a data set having *nnn* exams with *dd* density, do the follow:

1. Create *nnn* exams.
2. Create a mapping between a real world examination data set and a data set containing *nnn* exams. For this step, we used the PUR-S-93 data set, which has 2419 exams, 30032 students, 120681 enrollments, and a density of 0.029. We chose this set due to its large number of exams, even though it also has the lowest density of the Carter set. Any of the Carter data sets could have been used for this purpose.
   2.1. Create a mapping table having the four columns as shown Table 8 below. The Ordinal Position and Weighted Conflicts columns correspond to the abscissa and ordinate axes respectively in Figure 3. The Real Exam # is the real world exam

number that gives rise to the associated weighted conflicts. Finally, the Fabricated Exam # was determined in the following way:

2.1.1. Partition the Ordinal Positions into *nnn* equal groups.

2.1.2. Map each of these *nnn* Ordinal Position groups to one of the *nnn* exams created in step 1. Though not a requirement, we mapped the first 2419/*nnn* real world exams to the first fabricated exam, the second 2419/*nnn* real world exams to the second fabricated exam, and so forth.

Table 8. Fabricated Data Set Mapping Table Example

Ordinal Position	Weighted Conflicts	Real Exam #	Fabricated Exam #
1	1961	637	1
2	1717	575	1
3	1439	1678	1
4	1382	1979	1
…	…	…	…
24	481	2106	1
25	478	541	1
26	471	2068	2
27	451	2207	2
28	446	1986	2
…	…	…	…
2419	1	889	100

3. Create a student record for the fabricated data set.
4. Determine the number of exams for this student by selecting a value from a discrete uniform distribution in the interval [2, 6].
5. For each exam in this set, do the following:

   5.1. Randomly select an enrollment record from the PUR-S-93 data set.

   5.2. Use Table 8 to look up the fabricated exam associated with this real world exam.

5.3. If the fabricated exam determined in step 5.2 is already associated with this student, then go to step 5.1.

5.4. Using this fabricated exam and student combination, create an enrollment record.

5.5. If conflict matrix density is greater than or equal to *dd*, then quit.

6. Otherwise, go to step 3.

Table 9 lists attributes for each of our constructed Typical Fabricated Data Sets. The meanings of the column values should be self-explanatory.

Table 9. Attributes of Typical Fabricated Data Sets

Data Set	Graph Density	# of Exams	# of Students	# of Enrollments	Enrollments / Student	Enrollments / Exam	Exams w/ Clash	Conflict Edges
				100 Exams				
10004	0.04	100	33	136	4.12	1.36	54	402
10008	0.08	100	88	341	3.88	3.41	78	804
10016	0.16	100	192	774	4.03	7.74	92	1600
10032	0.32	100	575	2266	3.94	22.66	100	3200
				200 Exams				
20004	0.04	200	154	599	3.89	3.00	146	1604
20008	0.08	200	319	1289	4.04	6.45	182	3202
20016	0.16	200	842	3342	3.97	16.71	196	6400
20032	0.32	200	2515	10061	4.00	50.31	200	12800
				400 Exams				
40004	0.04	400	604	2369	3.92	5.92	347	6400
40008	0.08	400	1376	5503	4.00	13.76	387	12802
40016	0.16	400	3556	14294	4.02	35.74	398	25600
40032	0.32	400	10738	42761	3.98	106.90	400	51200

Testing – Environment

Hardware

The author's personal computer, used for this study, contains of a 1.8 GHz Intel Pentium 4 processor with 512MB of memory running on Microsoft Windows XP SP2 Home Edition.

Software

This study used Microsoft's Visual Basic .NET 2002. Microsoft's Visual Basic .NET is highly readable, generates optimized executable code, and is expressive enough for the algorithms under consideration. Additionally, other software products, such as Microsoft Access, were used when and where deemed appropriate for the analysis of the algorithm.

Persistent Storage

A database management system was be used for persistent storage (e.g., to store exam, student and room information). Though any number of the current relational database management systems (RDBMS) would have sufficed for this purpose, we used Microsoft's SQL Server 2000.

Technically speaking, a RDBMS was not required for persistent storage as this was doable entirely within Visual Basic in conjunction with XML, for example. Regardless, it was felt that a database management system was better suited to handling the large data sets efficiently.

Research Steps

The following subparagraphs call out the major steps performed during our research effort. These are high-level steps and not an exhaustive listing.

Research Steps – Setup Environment

A software implementation requires both a hardware and software environment to operate. The hardware platform was the author's personal desktop computer running Microsoft Windows XP SP2 Home Edition. Different languages were used depending on the needs of the testing or analysis and included Microsoft's VB.NET for the algorithm's overall structure and testing environment, Microsoft SQL Server T-SQL for data base procedures, and Microsoft Visual Basic for Applications within the Microsoft Access 2000 environment.

Research Steps – Established Tests

We settled on three groups of experiments, designed to ascertain the canonical PSO algorithm's ability in handling real world university examination sets. The objectives of the three groups were:

Selecting PSO Parameters

Experiments were designed to determine appropriate PSO algorithm values for the Cognitive/Social Ratio, Inertia Weight, Swarm Size, and 1st Order Conflict Weight parameters.

Constraint Handling

Experiments were constructed to investigate how the handling of constraints influenced the algorithm's ability during the optimization phase.

Real World Data

Experiments were fashioned to assess the algorithm's effectiveness against real world examination data sets.

Research Steps – Exercised Tests

After establishment of the tests, the testing portion of this research proceeded in a methodically controlled and documented manner.

We constructed test matrices, where each row in the matrix defined a test case and the columns defined parameters. Each cell in the matrix contained a test case parameter's argument value where these values normally correspond to algorithmic and problematic factors. We then created a test harness around the algorithm with the problem's factors acting as the harness input parameters; thus, providing a well-defined interface for each test. Finally, we placed each test's initial harness arguments, defined in the test matrix, into a database table, and used this table to direct the algorithm. This method allowed us to document the primary inputs for each test and, additionally, permitted unattended test execution.

Besides capturing the usual output from the algorithm, we recorded additional relevant information, such as timing, into the database for use during the analysis phase.

Performed Analysis and Interpretation

According to Barr, Golden, Kelly, Resende, and Stewart (1995), "*Data Analysis* refers to evaluating the recorded empirical data with statistical and nonstatistical techniques with respect to the experiment's purpose and goals" (p. 21). In particular, we

looked for correlations between factors or factor combinations and measures such as solution quality.

Besides using the work of Barr, Golden, Kelly, Resende, and Stewart (1995) as a guide, our research also used the publications by Bartz-Beielstein (2003), Dolan and Moré (2002), and Beielstein, Parsopoulos, and Vrahatis (2001) as standard for directing the analysis and interpretation phases.

Report Results

This research effort uses the guidelines put forward by Barr, Golden, Kelly, Resende, and Stewart (1995) for reporting computational results. As pointed out by these authors, the research report provides necessary and sufficient information to convince a reader the experiment has scientific merit, is reproducible, and addresses the goal. The guidelines are:

Reproducibility – Real world data sets are accessible from the Internet and methods used to generate artificial data sets are documented.

Specify influential factors in detail – The algorithm, all algorithm parameters, random number generation, and test environment parameters are fully documented.

Precise timing – Testing factored out timing as much as possible so timing precision was not a problem. Reporting instead focused on progress of the algorithm per iteration. Using time as a metric introduces many uncontrollable variables that tend to obfuscate the truth rather than illuminate it. For example, one may not have direct control over items such as background processing for operating system processes or garbage collection events in a programming environment. In addition, with the rate of increase in machine processing speed, one can easily reduce the processing time by

simply using a computer with a faster processor or, in some cases, by just adding more RAM memory. Though we did perform some reporting on time, algorithms that have distinct generations or iterations cycles most often report based on iterations.

Show how algorithm parameters are set – The reasoning beyond parameter settings was documented. For example, if generally accepted parameter values from peer-reviewed literature are chosen, then the source was cited and rationale given to justify the decision. If instead, parameters are empirically derived, then the method used to produce them was documented and justification given.

Use of statistical experimental design techniques – Techniques similar to those used by Burke, Eckersley, McCollum, Petrovic, and Qu (2003b) are used for reporting purposes.

Compare the heuristic with other methods – The results of our experiments, using the Carter data set, are compared against the results from other heuristic techniques applied to the same data set.

Reduce variability of results – This was accomplished by using the same testing environment throughout and by factoring out uncontrollable environmental influenced variables such as timing.

Produce a comprehensive report of the results – Experimental test results are reported in tabular form and show items similar to those reported in Burke, Eckersley, McCollum, Petrovic, and Qu (2003b). Performance profiles are shown in graphical form similar to those shown in Peram, Veeramachaneni, and Mohan (2003).

In addition to the tabular and graphical reports, we performed a narrative analysis the observations by comparing and contrasting results from the range of test cases. From

this, we drew conclusions, with respect to the research's goal and, finally, provided recommendations for further research and areas of study.

Resource Requirements

As is the case with almost all timetabling investigations (Schaerf, 1999), this study required a software implementation in order to perform tests of the algorithms. Hence, hardware and software components were required to support the implementation and their descriptions follow.

Reliability and Validity

Reliability refers to the consistency of test results in different experiments or statistical trials. That is, by using the same experimental setup, one should consistently arrive at similar results regardless of who performs them or when. This study implemented a number of measures to guarantee the reliability and repeatability of the test results. These include the following research conventions:

- *Real World Data Sets* – This study used the Carter benchmark data set, which is a publicly available collection of thirteen real world examination timetabling data sets.
- *Fabricated Data Sets* -- Methods used for fabrication of examination timetabling data sets are documented, permitting others to construct similar data sets.
- *Algorithms* -- All algorithms and methods used during the study are documented and source code will available from the author.

- *Algorithmic Parameters* -- All input parameters used for algorithms are documented.
- *Configuration* -- A complete list of all software, hardware, and third-party utilities used by the algorithm or during data analysis is documented.
- *Environmental Consistency* -- The same software and hardware environment was used throughout all test runs.
- *Multiple Runs* -- Multiple test runs were performed using the same input parameters to obtain typical algorithm behavior. Variations in test results are a byproduct of the algorithm's stochastic nature.

Validity refers to the degree to which the study's conclusions are logically deduced from its premises. This study followed steps to guarantee that its experimental procedures actually tested the efficacy of the PSO algorithm when applied to the university examination timetabling problem.

- *Real World Data Sets* -- Use of real world examination timetabling data sets employed in other timetabling research permitted us the opportunity to verify the legitimacy of our results through comparison.
- *Fabricated Data Sets* -- Fabricated data sets were constructed to mimic the structure of real world data sets whenever possible.
- *Established Procedures* -- Valid test methods, established by other university examination timetabling problem researchers, were used as guides.
- *Causality* -- The relationship between independent variables (factors) and the dependent variables (measures) was analyzed to determine causality.

- *Generalizability* -- This means how well do the results of the experiment generalize beyond the suite of test cases studied to the rest of the possible scenarios of the real world. This study tested this quality through its use of thirteen real world data sets and fabricated data sets spanning a wide range of sizes and conflict densities.

Summary

In this chapter, we outlined the research methodology, relevant problem factors, and measures. Additionally, we detailed the research aspects for each of the PSO algorithm features. These features included, among others, initialization methods, objectives, constraints, and control parameters. Testing was covered, with reasoning and justification provided for real world data set choices, data set reduction methods, artificial data set construction procedures, and testing environment specifications. Of particular significance, we covered details and importance of maintaining the underlying real world data sets' structural similarity. Finally, individual research steps, analysis and interpretation techniques, report results prescriptive methods, and testing reliability and validity processes were formalized.

Chapter 4

Results

Data Analysis

We performed three basis suites of experiments, each in turn designed for a specific task. These high-level groups consisted of Selecting PSO parameters, constraint handling, and testing with real world data sets.

Selecting PSO Parameters

These tests PSO parameter experiments to determine appropriate PSO algorithm values for the Cognitive/Social Ratio, Inertia Weight, Swarm Size, and 1st Order Conflict Weight parameters.

Selecting PSO Parameters – Cognitive/Social Ratio Experiment

We looked first at the effect of variations in the cognitive/social ratio, $\varphi_1 : \varphi_2$. To do this we varied the relationship between the cognitive and social ratios across three values: (1.3:2.8), (2.05:2.05), and (2.8:1.3). We chose these ratios based on the findings of Carlisle and Dozier (2001) that discovered, even though the (2.05:2.05) ratio is commonly used, a better value appears to be (2.8:1.3). We included the additional (1.3:2.8) ratio for comparison.

Selecting PSO Parameters – "Best" Inertia Weight Experiment

Having determined a cognitive/social ratio, we used it in determination of an Inertia Weight value. We were interested in finding one that worked well across the wide range of conflict densities and data set sizes presented by the Typical Fabricated Data

Sets. To do this we varied the inertia weight value across five values: 1, 0.825, 0.75, 0.625, and 0.5. We chose these values based on initial trials performed by the author, which suggested that the commonly used value of one might not be the most appropriate one in the majority of cases.

Selecting PSO Parameters – "Best" Swarm Size Experiment

Having decided on the cognitive/social ratio and inertia weight values, we turned our attention to arriving at a good swarm size based on the Typical Fabricated Data Sets. To do this, we experimented with four different swarm size values: 5, 10, 20, and 40. We chose these values for a variety of reasons. The value of 10 was selected next as it is the value commonly used in the literature when exercising the PSO algorithm's abilities. We chose the two values of 20 and 40 to encompass the value of 30 chosen by Carlisle and Dozier (2001). These authors suggested 30 particles as a good all-around value based on their research. It was felt that these 20 and 40 particles provided a better range and, furthermore, the algorithm's behavior for 30 particles would be evident through simple interpolation of the results. Finally, we decided on using a value of five to determine how well the PSO would perform in this problem domain with very few swarm particles.

Selecting PSO Parameters – 1st Order Conflict Weight Experiment

The 1st Order Conflict Weight (i.e., w_s, where s = 0) was the last factor investigated. Here our goal was to measure the 1st Order Conflict Weight (see Equation 5) value's effect on the number of first order conflicts. One might assume that a larger value for the weight would produce a lower number of first order conflicts due to the resulting larger penalty value pressuring particles away from less desirable solutions. The design of this experiment was meant to shed some light on this assumption.

Once more, we used the Typical Fabricated Data Sets and experimented with four different weight values: 32, 64, 2048, and 131072. These particular values were selected for a couple of reasons. We chose the 2048 and 131072 values in keeping with the power of two nature of our selected objective function, these values equaling 211 and 217 respectively. The value of 32 is the one specified by our specific objective function and, finally, the value of 64 was chosen because it was the next higher power of two; permitting us to see how a slightly higher value than 32 affects the results.

Constraint Handling

Experiments were constructed to investigate how the handling of constraints influenced the algorithm's ability during the optimization phase.

Constraint Handling – Feasible/Infeasible Experiment

We then use the parameter values determined above to investigate the abilities of the Feasible/Infeasible hybrid approach. As detailed earlier in this thesis, this approach first extracts candidate solutions from the feasible search space for the initialization phase and then follows with the canonical PSO algorithm, whose solutions are not limited to the feasible search space, in order to optimize the objective function.

Constraint Handling – Feasible/Feasible Experiment

We next performed tests using the Feasible/Feasible hybrid approach. To reiterate from earlier, this approach first extracts candidate solutions from the feasible search space for the initialization phase and then we used a modified PSO algorithm to optimize the objective function while only accepting feasible solutions. We described this algorithm more fully in the *Perform Optimization Process* subparagraph found on page

50. For the purposes of our discussion here, we will refer to this optimization process as the PSO-NoConflicts approach.

We used the same initial swarms as those used in the previous Feasible/Infeasible Experiment in order to permit comparisons between the optimization methods of these two approaches. This was possible because these two approaches use the exact same Least Saturation Degree method for initialization.

Real World Data

Experiments were fashioned to assess the algorithm's effectiveness against real world examination data sets.

Real World Data – Full PSO Random Experiment

We then turned our attention to real data sets, looking first at the Full Carter Data Set. This experiment used the canonical PSO algorithm with random swarm initialization.

Real World Data – Reduced PSO Random Experiment

We next ran the PSO algorithm using the Reduced Carter Data Set to provide data for comparison against the previous full set. In the same fashion as above, this experiment used the canonical PSO algorithm with random swarm initialization. This set is identical to the full Carter set except it contains no students having only a single enrollment. A solution to this data set is also a solution to the full set, but we performed this test to see if its convergence differed significantly from the full data set.

Real World Data – Reduced LSD No Conflicts Experiment

This experiment used the Least Saturation Degree swarm initialization method combined with the PSO-NoConflicts optimization algorithm. This hybridization showed

some promise earlier when applied against the fabricated data and its capability against real world data was therefore of interest.

Unlike our tests using fabricated data, we did not consider the combination of Least Saturation Degree swarm initialization followed by the canonical PSO algorithm, as this coupling did not show promise during earlier tests.

Findings

The following subparagraphs report the findings from each experiment.

Selecting PSO Parameters – Cognitive/Social Ratio Experiment

Table 10 lists the values used for each experimental attribute. We ran the experiment for each of the 12 data sets found in the Typical Fabricated Data Sets, described earlier in this document. Additionally, we limited each run to 300 iterations. This was done to reduce the experimental time required and because we only needed to get a sense of the factor's effect. As for the other parameter values (e.g., Inertia Weight), we chose their settings based on preliminary testing.

In order to see the effect each ratio has on the algorithm, each experiment was repeated (i.e., replicated) 10 times and the *Average Particle Best Penalty* value plotted across iterations. Here we define the *Average Particle Best Penalty* value as each particle's penalty value, at their respective previous best positions (p_{id} in Equation 1), averaged across all particles and replications for each iteration step.

Figure 4 through Figure 15 present the results.

We see from the results that the (2.8:1.3) ratio:

- generally does not converge as quickly as the (2.05:2.05) ratio and about the same rate as the (1.3:2.8) ratio choice, and
- produces the minimum more often than the other choices, within the 300 iterations used.

Table 11 shows the results from the last iteration grouped into sections according to the number of exams. This table alone does not give a feel for how things change over time, thus the graphs, but the table does give us some hard numbers to consider. From this we see that the (2.8:1.3) ratio provides the best results, at this point, more often than the other two ratios, having a slight edge of 7 out of 12 best values (indicated by the bold and italicized font).

As these tests were designed to find an appropriate, not necessarily an optimal choice, we will use the (2.8:1.3) ratio throughout the rest of the experiments. As this is in agreement with Carlisle and Dozier (2001), the choice is a reasonable one.

Table 10. Cognitive/Social Ratio ($\varphi_1 : \varphi_2$) Experiment Attributes

Attribute	Value(s)
Initialization Method	Random
Optimization Algorithm	Canonical PSO
Swarm Size	10 particles
Max Iterations	300
Cognitive/Social Ratios ($\varphi_1 : \varphi_2$)	1.3 : 2.8, 2.05 : 2.05, 2.8 : 1.3
Inertia Weight (w)	0.75
1st Order Conflict Weight	32
Replications	10
Dataset Used	Typical Fabricated Data Sets

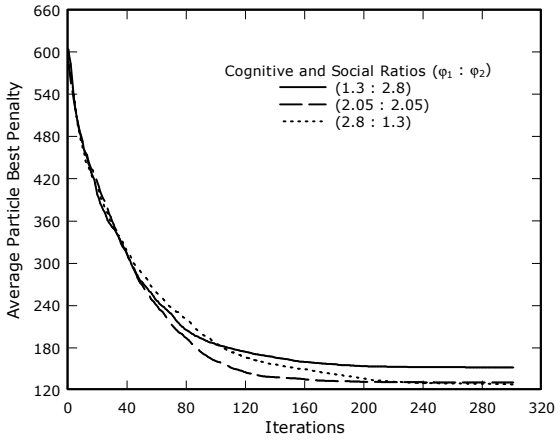

Figure 4. Cognitive/Social Ratio Results for Fabricated Data Set having $n = 100$, $d = 0.04$, and $r = 10$

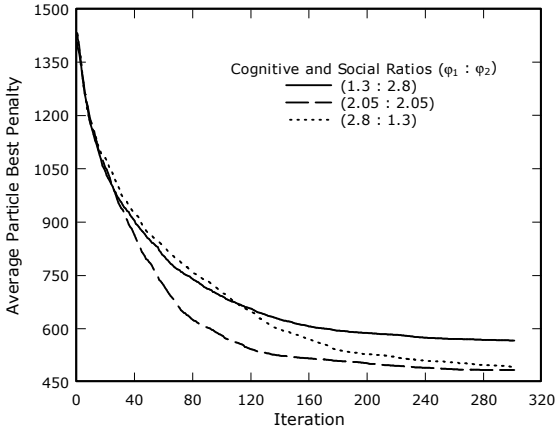

Figure 5. Cognitive/Social Ratio Results for Fabricated Data Set having $n = 100$, $d = 0.08$, and $r = 10$

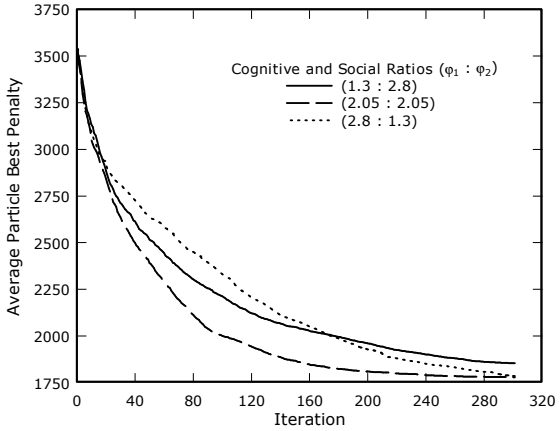

Figure 6. Cognitive/Social Ratio Results for Fabricated Data Set having $n = 100$, $d = 0.16$, and $r = 10$

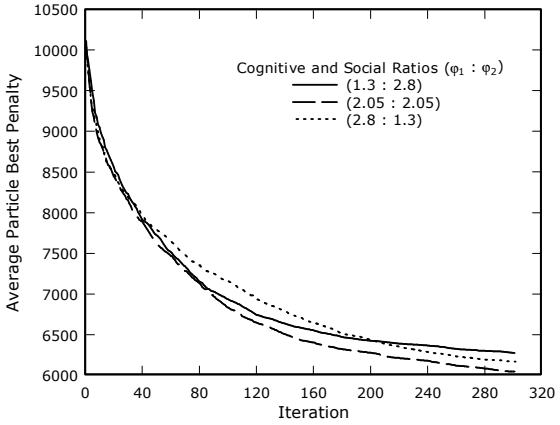

Figure 7. Cognitive/Social Ratio Results for Fabricated Data Set having $n = 100$, $d = 0.32$, and $r = 10$

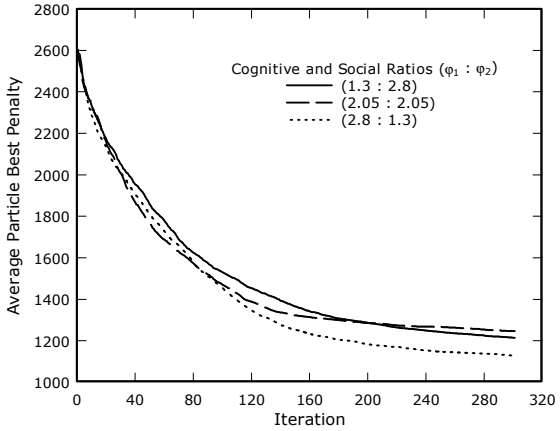

Figure 8. Cognitive/Social Ratio Results for Fabricated Data Set having $n = 200$, $d = 0.04$, and $r = 10$

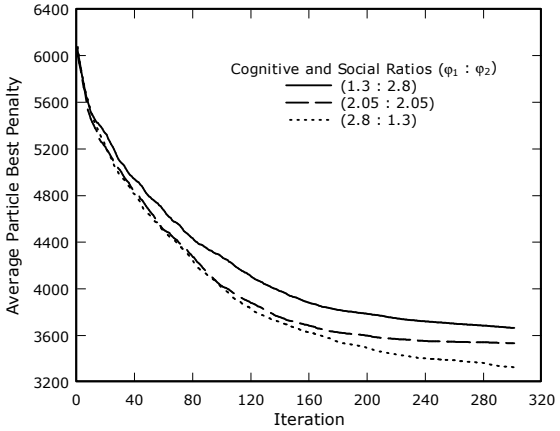

Figure 9. Cognitive/Social Ratio Results for Fabricated Data Set having $n = 200$, $d = 0.08$, and $r = 10$

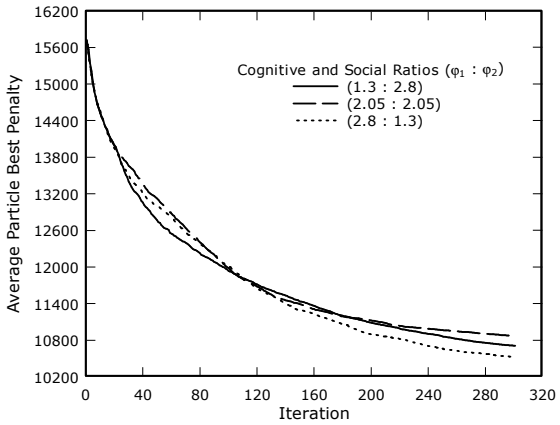

Figure 10. Cognitive/Social Ratio Results for Fabricated Data Set having n = 200, d = 0.16, and r = 10

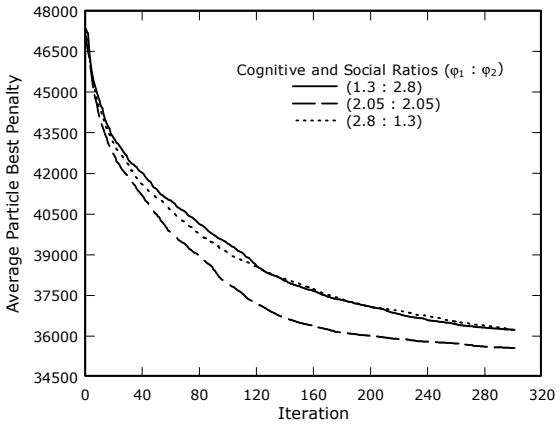

Figure 11. Cognitive/Social Ratio Results for Fabricated Data Set having n = 200, d = 0.32, and r = 10

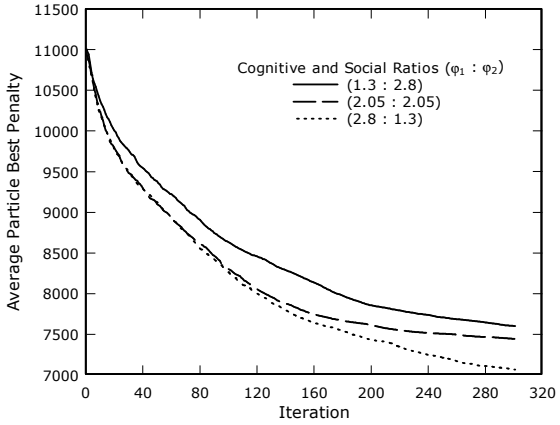

Figure 12. Cognitive/Social Ratio Results for Fabricated Data Set having n = 400, d = 0.04, and r = 10

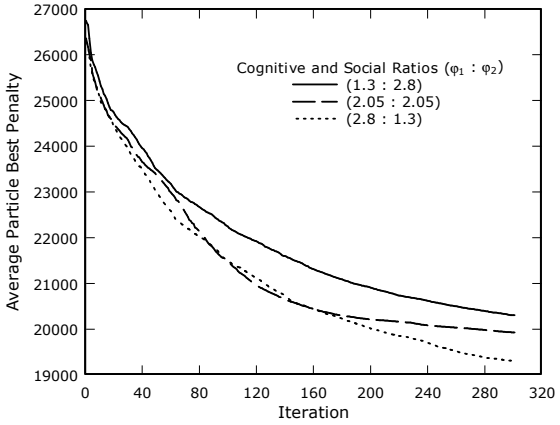

Figure 13. Cognitive/Social Ratio Results for Fabricated Data Set having n = 400, d = 0.08, and r = 10

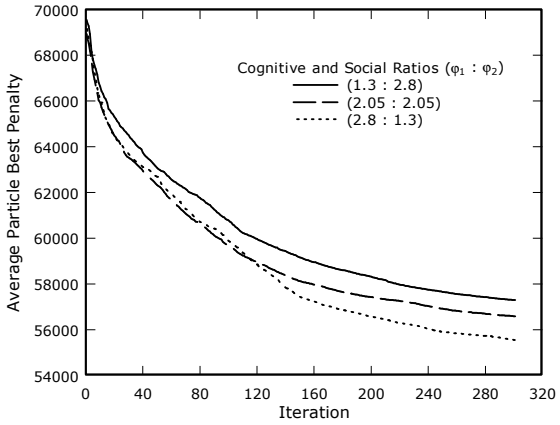

Figure 14. Cognitive/Social Ratio Results for Fabricated Data Set having n = 400, d = 0.16, and r = 10

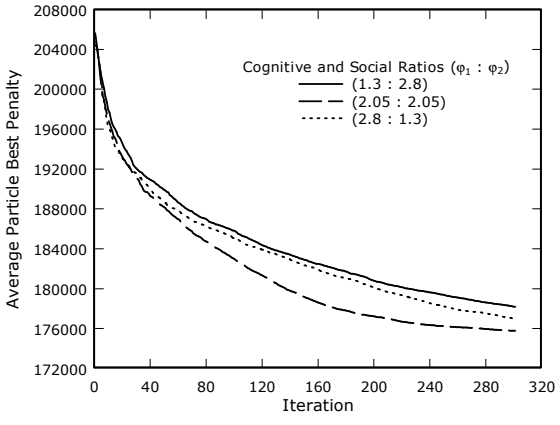

Figure 15. Cognitive/Social Ratio Results for Fabricated Data Set having n = 400, d = 0.32, and r = 10

Table 11. Cognitive/Social Ratio ($\varphi_1 : \varphi_2$) Experiment Results

	Conflict Matrix Density			
$\varphi_1 : \varphi_2$	0.04	0.08	0.16	0.32
		100 Exams		
1.3 : 2.8	152	567	1854	6276
2.05 : 2.05	131	***484***	***1779***	***6046***
2.8 : 1.3	***128***	492	1786	6168
		200 Exams		
1.3 : 2.8	1215	3666	10710	36231
2.05 : 2.05	1246	3534	10873	***35564***
2.8 : 1.3	***1129***	***3330***	***10525***	36231
		400 Exams		
1.3 : 2.8	7603	20305	57307	178212
2.05 : 2.05	7444	19928	56592	***175786***
2.8 : 1.3	***7069***	***19308***	***55557***	176976

Selecting PSO Parameters – "Best" Inertia Weight Experiment

Table 12 lists the values used for each experimental attribute. We again ran the experiment for each of the 12 data sets found in the Typical Fabricated Data Sets; this time limiting the number of iterations to 200 to reduce the experimental time required.

Each experimental run was repeated 10 times and the results were then used to compute the Average Particle Best Penalty. Figure 16 through Figure 27 present the results.

We observe from these figures that an inertia weight of 0.75 provides swarm convergence that is not too overly aggressive and yet aggressive enough that the swarms'

average overtakes the other inertia weight values for the majority of experiments; and that within only 200 iterations.

Table 13 shows the results from the last iteration. From this, we see that the 0.75 value provides the best results, at this point, more often than the other inertia weights chosen, 10 out of the 12 experiments (indicated by the bold and italicized font).

As the 0.75 value performs satisfactorily across all data set sizes and for all conflict densities of the Typical Fabricated Data Sets, this is the value used throughout the rest of the testing.

Table 12. "Best" Inertia Weight Experiment Attributes

Attribute	Value(s)
Initialization Method	Random
Optimization Algorithm	Canonical PSO
Swarm Size	10 particles
Max Iterations	200
Cognitive/Social Ratios ($\varphi_1 : \varphi_2$)	2.8:1.3
Inertia Weight (w)	1, 0.825, 0.75, 0.625, 0.5
1st Order Conflict Weight	32
Replications	10
Dataset Used	Typical Fabricated Data Sets

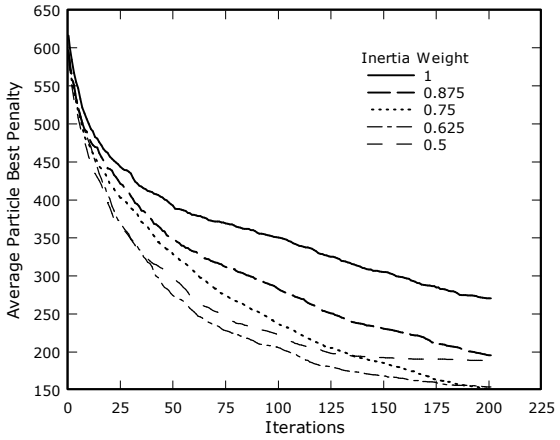

Figure 16. Best Inertia Weight Results for Fabricated Data Set having n = 100, d = 0.04, and r = 10

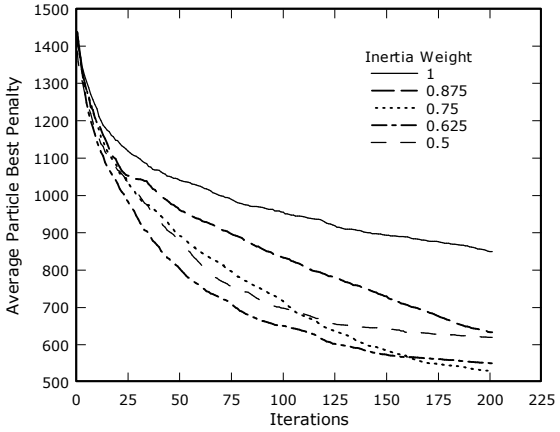

Figure 17. Best Inertia Weight Results for Fabricated Data Set having n = 100, d = 0.08, and r = 10

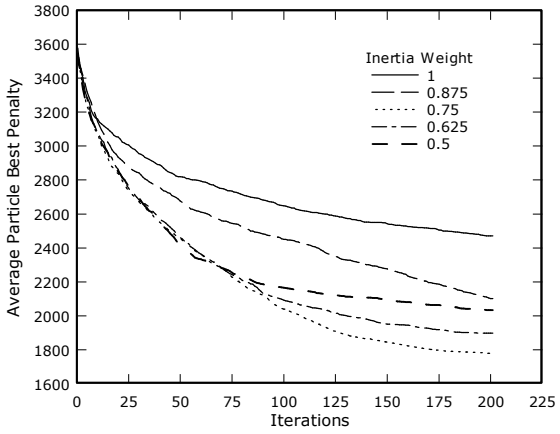

Figure 18. Best Inertia Weight Results for Fabricated Data Set having n = 100, d = 0.16, and r = 10

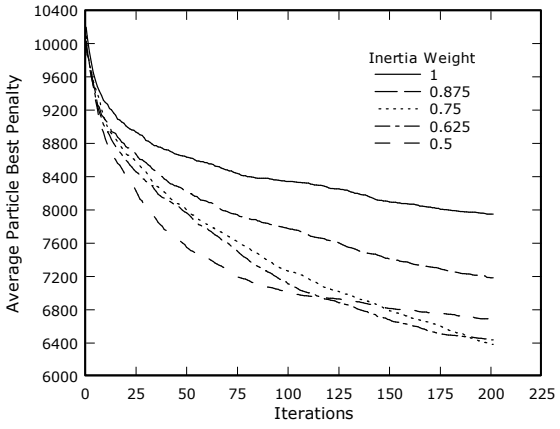

Figure 19. Best Inertia Weight Results for Fabricated Data Set having n = 100, d = 0.32, and r = 10

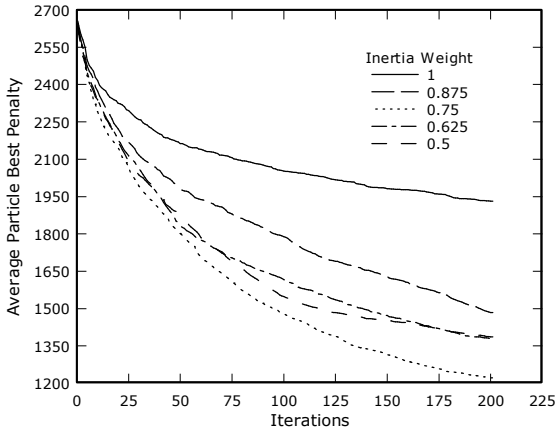

Figure 20. Best Inertia Weight Results for Fabricated Data Set having n = 200, d = 0.04, and r = 10

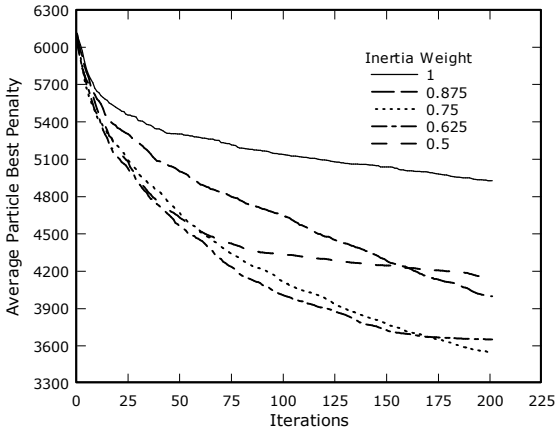

Figure 21. Best Inertia Weight Results for Fabricated Data Set having n = 200, d = 0.08, and r = 10

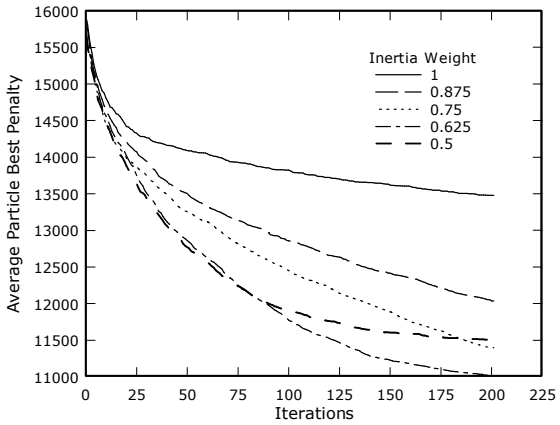

Figure 22. Best Inertia Weight Results for Fabricated Data Set having n = 200, d = 0.16, and r = 10

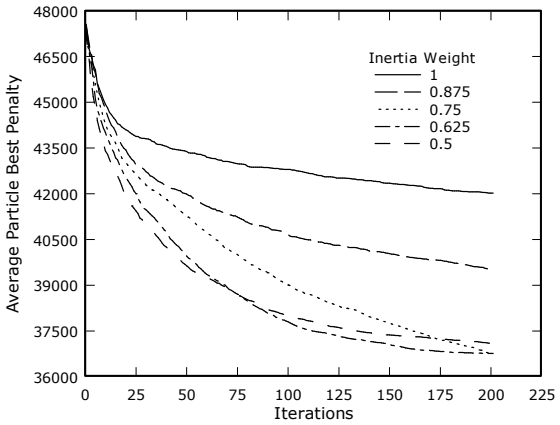

Figure 23. Best Inertia Weight Results for Fabricated Data Set having n = 200, d = 0.32, and r = 10

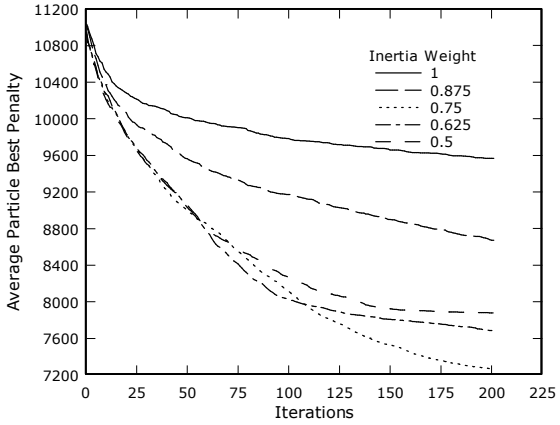

Figure 24. Best Inertia Weight Results for Fabricated Data Set having n = 400, d = 0.04, and r = 10

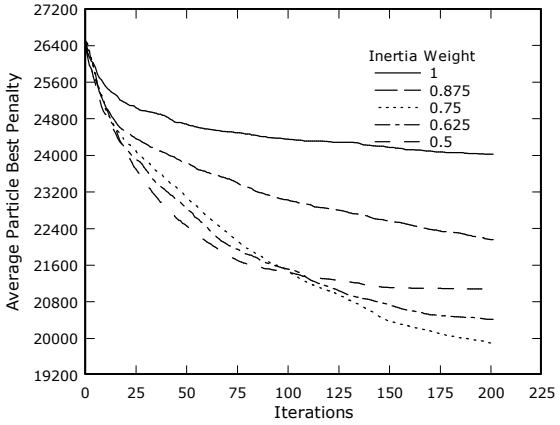

Figure 25. Best Inertia Weight Results for Fabricated Data Set having n = 400, d = 0.08, and r = 10

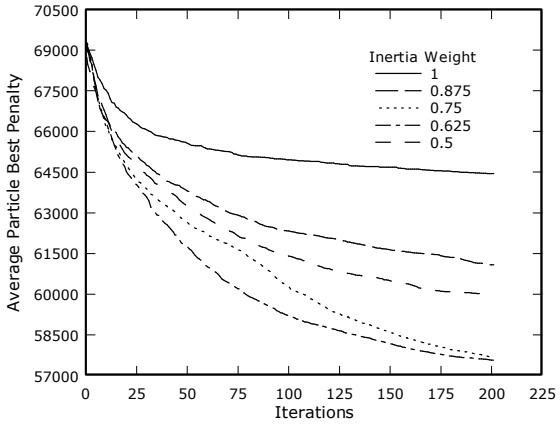

Figure 26. Best Inertia Weight Results for Fabricated Data Set having n = 400, d = 0.16, and r = 10

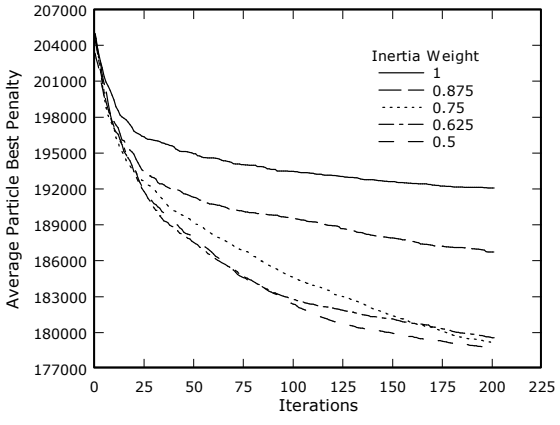

Figure 27. Best Inertia Weight Results for Fabricated Data Set having n = 400, d = 0.32, and r = 10

Table 13. "Best" Inertia Weight Experiment Results

	Conflict Matrix Density			
w	0.04	0.08	0.16	0.32
	100 Exams			
1	271	850	2471	7949
0.875	196	634	2101	7184
0.75	***152***	***530***	***1779***	***6386***
0.625	154	551	1897	6439
0.5	189	620	2034	6689
	200 Exams			
1	1931	4928	13479	42017
0.875	1484	3999	12038	39504
0.75	***1222***	***3546***	11398	***36763***
0.625	1378	3652	***11012***	36769
0.5	1387	4154	11505	37102
	400 Exams			
1	9568	24029	64449	192077
0.875	8673	22161	61080	186737
0.75	***7269***	***19896***	57686	***179193***
0.625	7687	20419	***57573***	179569
0.5	7879	21072	59972	178752

Selecting PSO Parameters – "Best" Swarm Size Experiment

Table 14 lists the values used for each experimental attribute. As before, we ran the experiment for each of the 12 data sets found in the Typical Fabricated Data Sets; using a value of 200 iterations as the limit. Each experimental run was repeated 10 times and the results were then used to compute the Average Particle Best Penalty. Figure 28 through Figure 39 present the results.

The experimental results are in keeping with our expectations. That is, the greater the number of particles available for searching the problem domain, the greater the probability of finding a better solution sooner. What was interesting is the observation that doubling the number of particles from 20 to 40 did not necessarily lead to an appreciably better solution. This is evident in Figure 37 through Figure 39 where the results are similar for these two values. This close tracking between these two values also occurs in Figure 29, Figure 30, and Figure 33. Yet, from Figure 34 and Figure 36 we see that this is not always the case.

Table 15 shows the results from the last iteration. In this case, there is no "best" parameter value, as one would expect, because a greater number of particles tends to lead to better solutions faster. As this is a computational cost versus solution value trade-off, it is a largely subjective decision.

Given this, we decided to use a swarm size of 20 throughout the rest of the testing. This is due to this swarm size's perceived ability to handle the larger data sets, as seen in Figure 37 through Figure 39. Our choice of swarm size is less than the 30 particles suggested by Carlisle and Dozier (2001) but, from our experiments, it appears that the

additional computational effort required by a 50% increase in swarm size is not satisfactorily justified by solution quality.

Table 14. "Best" Swarm Size Experiment Attributes

Attribute	Value(s)
Initialization Method	Random
Optimization Algorithm	Canonical PSO
Swarm Size (particles)	5, 10, 20, 40
Max Iterations	300
Cognitive/Social Ratios ($\varphi_1 : \varphi_2$)	2.8:1.3
Inertia Weight (w)	0.75
1st Order Conflict Weight	32
Replications	10
Dataset Used	Typical Fabricated Data Sets

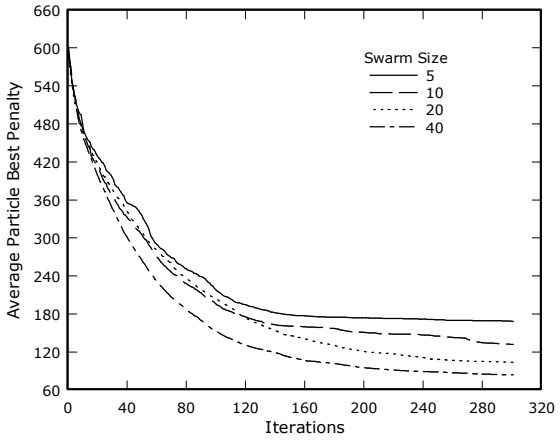

Figure 28. Best Swarm Size Results for Fabricated Data Set having n = 100, d = 0.04, and r = 10

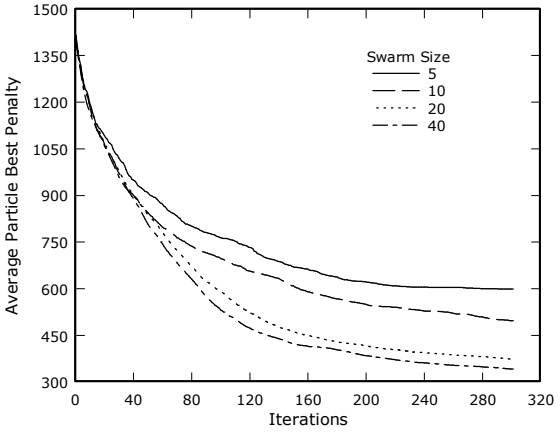

Figure 29. Best Swarm Size Results for Fabricated Data Set having n = 100, d = 0.08, and r = 10

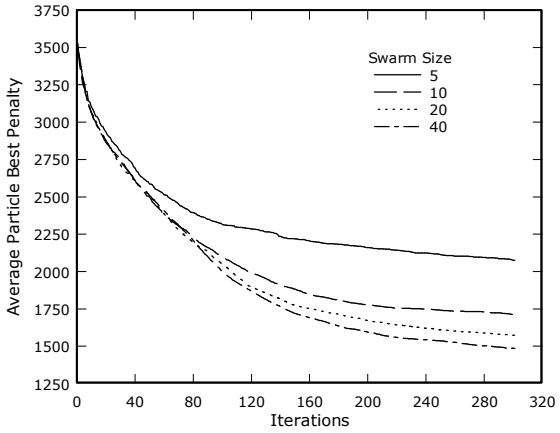

Figure 30. Best Swarm Size Results for Fabricated Data Set having n = 100, d = 0.16, and r = 10

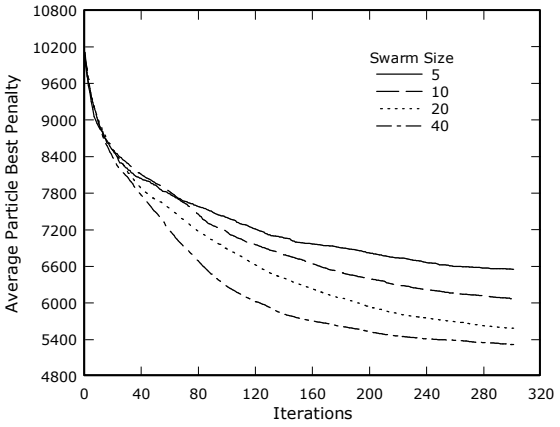

Figure 31. Best Swarm Size Results for Fabricated Data Set having n = 100, d = 0.32, and r = 10

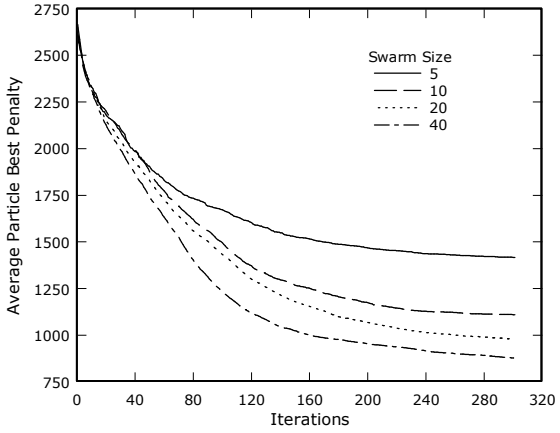

Figure 32. Best Swarm Size Results for Fabricated Data Set having n = 200, d = 0.04, and r = 10

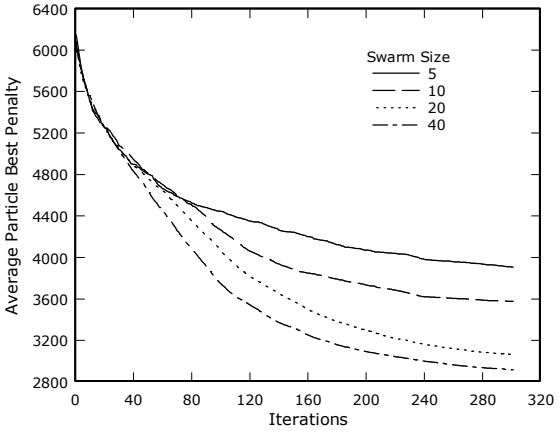

Figure 33. Best Swarm Size Results for Fabricated Data Set having n = 200, d = 0.08, and r = 10

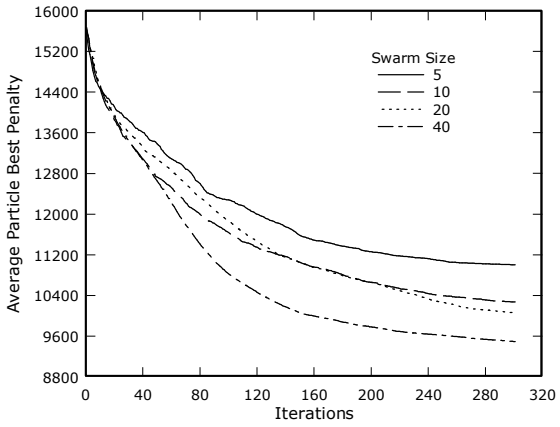

Figure 34. Best Swarm Size Results for Fabricated Data Set having n = 200, d = 0.16, and r = 10

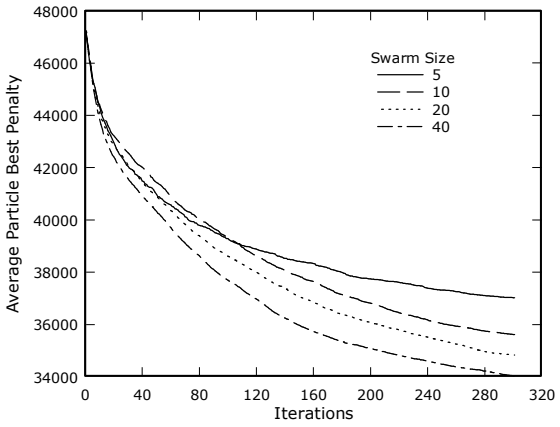

Figure 35. Best Swarm Size Results for Fabricated Data Set having n = 200, d = 0.32, and r = 10

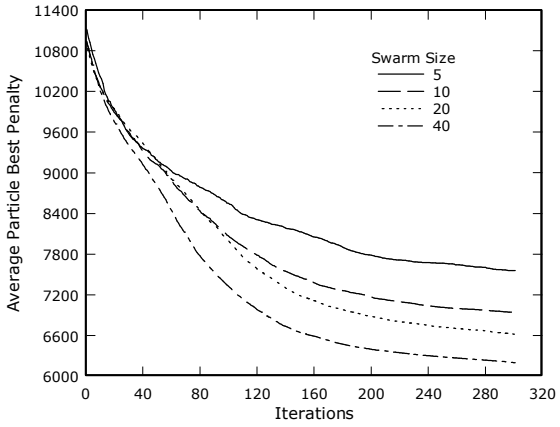

Figure 36. Best Swarm Size Results for Fabricated Data Set having n = 400, d = 0.04, and r = 10

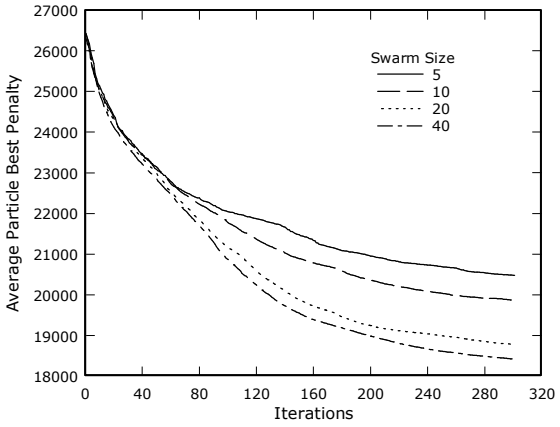

Figure 37. Best Swarm Size Results for Fabricated Data Set having n = 400, d = 0.08, and r = 10

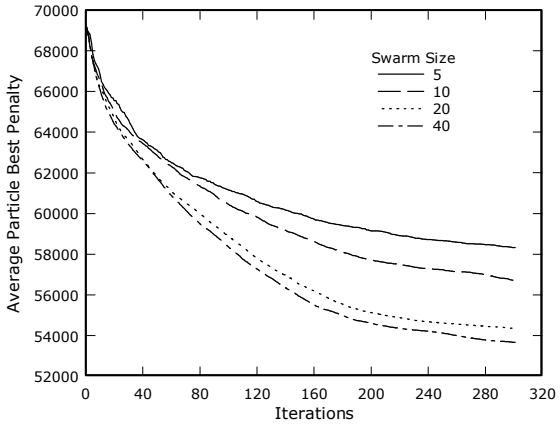

Figure 38. Best Swarm Size Results for Fabricated Data Set having n = 400, d = 0.16, and r = 10

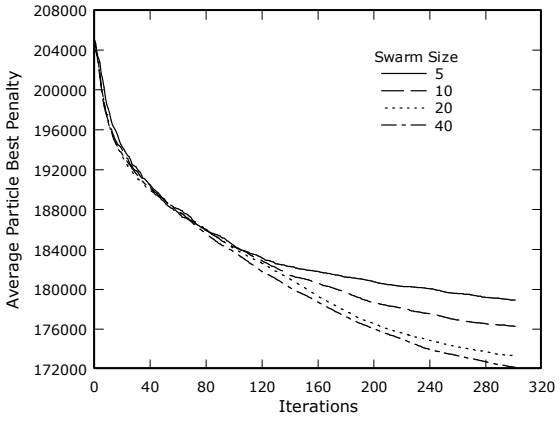

Figure 39. Best Swarm Size Results for Fabricated Data Set having n = 400, d = 0.32, and r = 10

Table 15. "Best" Swarm Size Experiment Results

	Conflict Matrix Density			
N	0.04	0.08	0.16	0.32
	100 Exams			
5	169	599	2075	6554
10	132	497	1711	6069
20	104	374	1574	5588
40	84	342	1485	5318
	200 Exams			
5	1418	3909	11006	37029
10	1110	3577	10272	35619
20	978	3066	10063	34835
40	879	2917	9495	34043
	400 Exams			
5	7556	20481	58324	178928
10	6943	19869	56717	176310
20	6621	18784	54352	173361
40	6200	18422	53666	172196

Selecting PSO Parameters – 1st Order Conflict Weight Experiment

Table 16 lists the values used for each experimental attribute. We ran this experiment for each of the 12 data sets found in the Typical Fabricated Data Sets; again, using a value of 300 iterations as the limit. Each experimental run was repeated 10 times and the results were then used to compute the *Average 1st Order Conflicts*. Here we define the *Average 1st Order Conflicts* value as each particle's total number of first order conflicts averaged across all particles and replications for each iteration step.

The results of these twelve experiments are shown in Figure 40 through Figure 51. In all cases, the value of 32 performed well, producing the least number of first order conflicts in the majority of experiments. Of course, these experiments only looks out 300 iterations and the value of 64 may very well produce fewer conflicts if permitted more iterations. From just the visual appearance of these plots, it looks as if the value of 64 would eventually surpass the quality of the 32 value. We look at this later.

As mentioned earlier, the goal of this set of experiments was not to find an optimal value, as our objective function already has a value defined, but to investigate how the value of 32 faired compared to some larger values. If during this experiment we discovered better values, then this would indicate that better results might be possible using different weights; in which case, this would warrant future experimentation, if one desired to find the optimal weight.

One thing we do notice immediately from these graphs is that a much larger 1st Order Conflict Weight does not automatically produce fewer first order conflicts. In fact, the values 2048 and 131072 consistently produce very similar results and, in some cases, results significantly higher in first order conflicts. Therefore, an assumption that a larger weight would lead to fewer conflicts is not valid.

Table 16. 1st Order Conflict Weight Experiment Attributes

Attribute	Value(s)
Initialization Method	Random
Optimization Algorithm	Canonical PSO
Swarm Size (particles)	20
Max Iterations	300
Cognitive/Social Ratios ($\varphi_1 : \varphi_2$)	2.8:1.3
Inertia Weight (w)	0.75
1st Order Conflict Weight	32 (2^5), 64 (2^6), 2048 (2^{11}), 131072 (2^{17})
Replications	10
Dataset Used	Typical Fabricated Data Sets

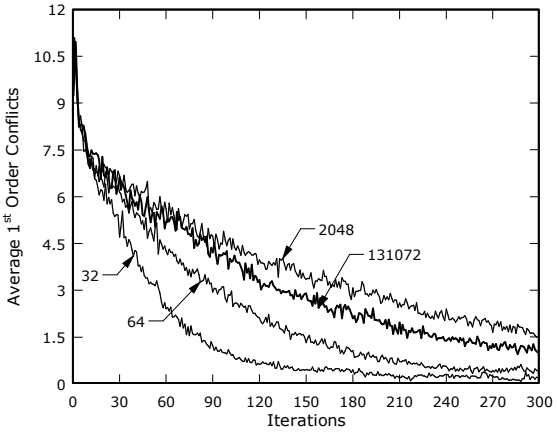

Figure 40. 1st Order Conflict Weight Results for Fabricated Data Set having n = 100, d = 0.04, and r = 10

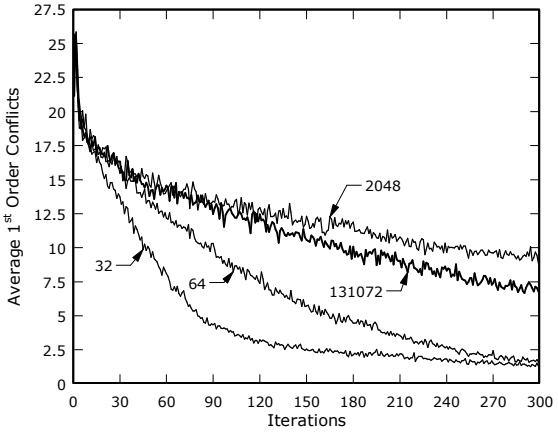

Figure 41. 1st Order Conflict Weight Results for Fabricated Data Set having n = 100, d = 0.08, and r = 10

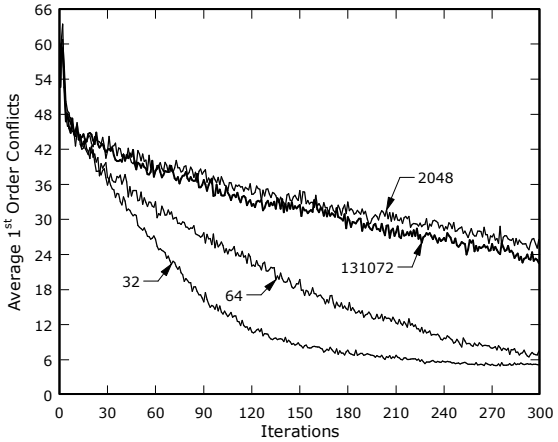

Figure 42. 1st Order Conflict Weight Results for Fabricated Data Set having n = 100, d = 0.16, and r = 10

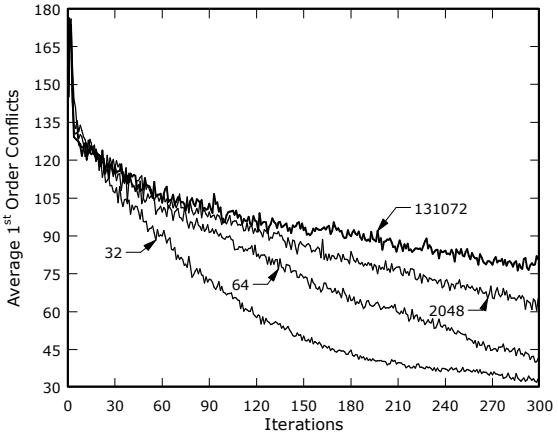

Figure 43. 1st Order Conflict Weight Results for Fabricated Data Set having n = 100, d = 0.32, and r = 10

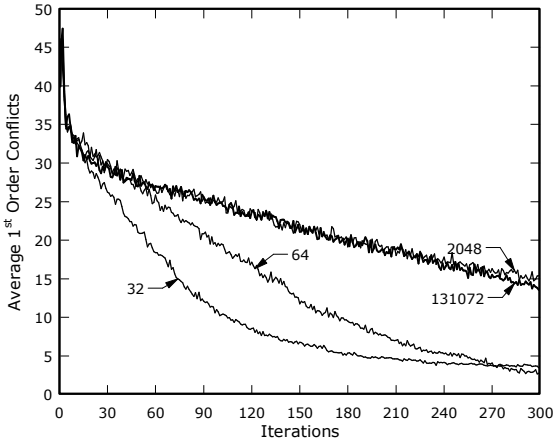

Figure 44. 1st Order Conflict Weight Results for Fabricated Data Set having n = 200, d = 0.04, and r = 10

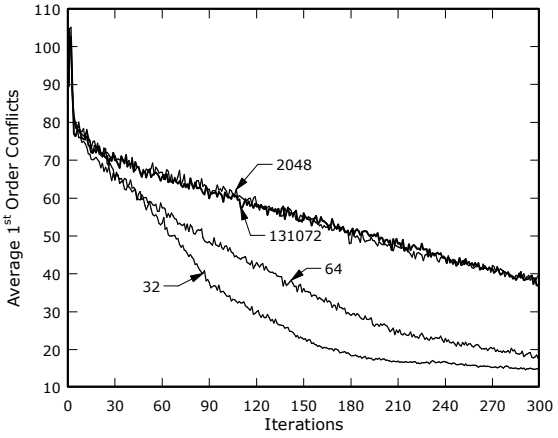

Figure 45. 1st Order Conflict Weight Results for Fabricated Data Set having n = 200, d = 0.08, and r = 10

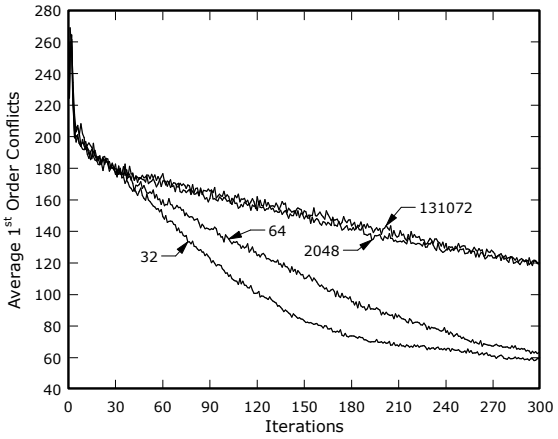

Figure 46. 1st Order Conflict Weight Results for Fabricated Data Set having n = 200, d = 0.16, and r = 10

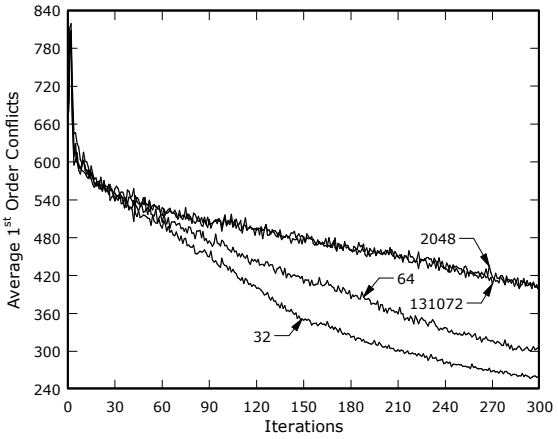

Figure 47. 1st Order Conflict Weight Results for Fabricated Data Set having n = 200, d = 0.32, and r = 10

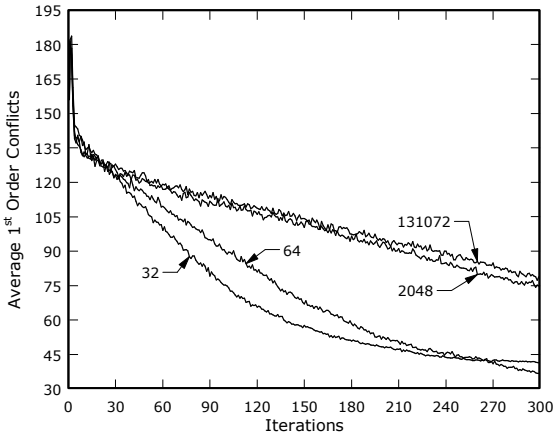

Figure 48. 1st Order Conflict Weight Results for Fabricated Data Set having n = 400, d = 0.04, and r = 10

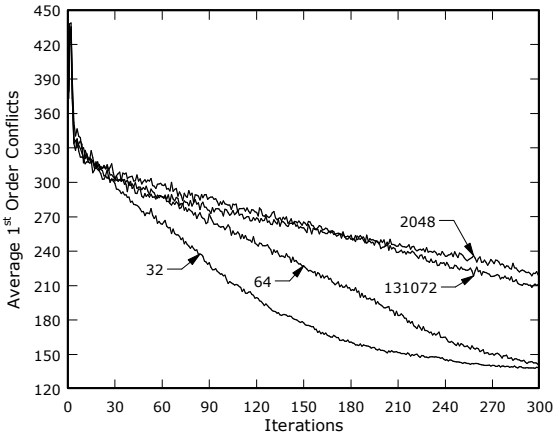

Figure 49. 1st Order Conflict Weight Results for Fabricated Data Set having n = 400, d = 0.08, and r = 10

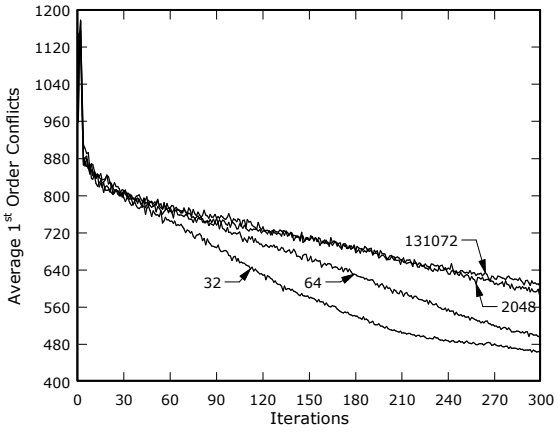

Figure 50. 1st Order Conflict Weight Results for Fabricated Data Set having n = 400, d = 0.16, and r = 10

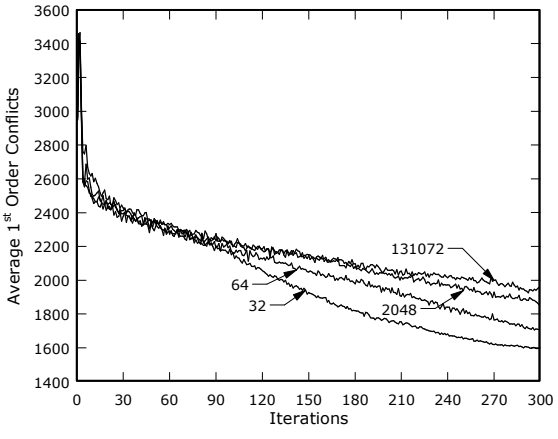

Figure 51. 1st Order Conflict Weight Results for Fabricated Data Set having n = 400, d = 0.32, and r = 10

Selecting PSO Parameters – 1st Order Conflict Weight Experiment – 1200 Iterations

A number of additional experiments were run using the 1st Order Conflict Weights of 32 and 64. We reran the previous experiments out to 1200 iterations but limited ourselves to those weights and data sets with a conflict density value of 0.16. The results are displayed in Figure 52 through Figure 54.

A value of 64 did produce fewer first order conflicts by the last iteration in all cases. The effect is more prominent as the number of exams increases. This outcome is not significant enough for us to consider altering our objective function definition but it may warrant future research.

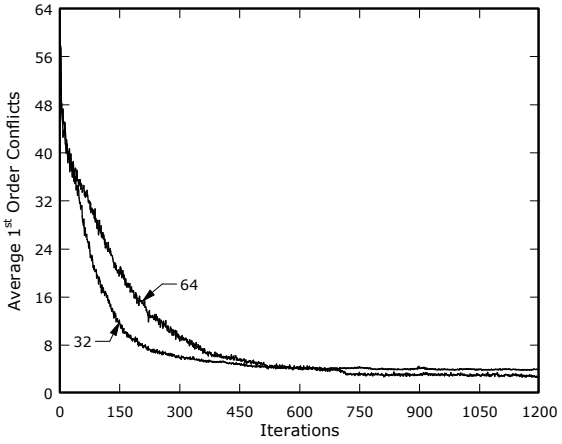

Figure 52. 1st Order Conflict Weight Results for Fabricated Data Set having 1200 Iterations, n = 100, d = 0.16, and r = 10

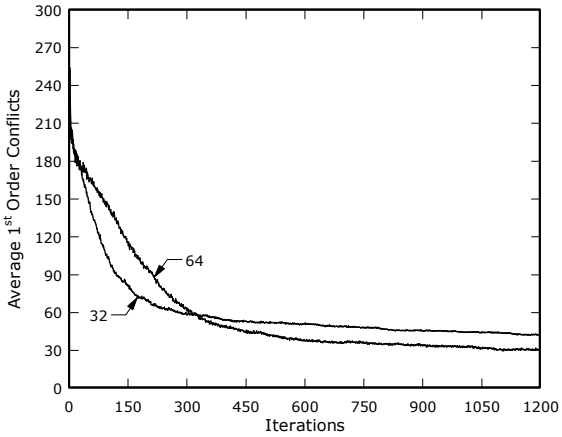

Figure 53. 1st Order Conflict Weight Results for Fabricated Data Set having 1200 Iterations, n = 200, d = 0.16, and r = 10

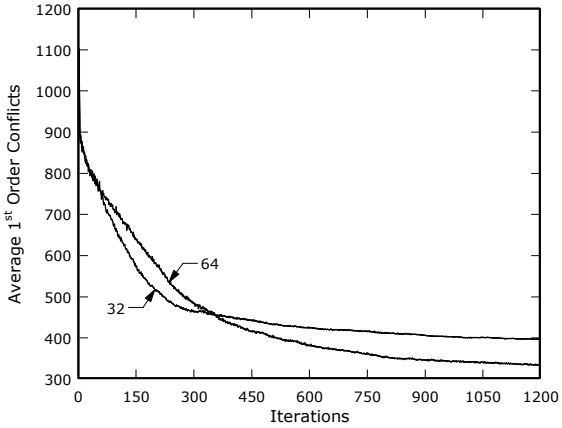

Figure 54. 1st Order Conflict Weight Results for Fabricated Data Set having 1200 Iterations, n = 400, d = 0.16, and r = 10

Constraint Handling – Feasible/Infeasible Experiment

Table 17 lists the values used for each experimental attribute. We ran the experiment for the 12 data sets found in the Typical Fabricated Data Sets, limiting the number of iterations to 50. This limitation was more than sufficient to observe the characteristics of this approach. In addition, we limited the replications to five due to the computationally higher cost the of Least Saturation Degree initialization method.

Figure 55 through Figure 57 plot the *Average Particle Best Penalty* value, which is the average value across all replicas. We only show 11 of the 12 data sets because our Least Saturation Degree implementation was unable to find even a single feasible solution for the 40032 data set.

This first set of graphs appear to have no visually perceptible change in the best value over the entire set of iterations, regardless of the number of exams or conflict density used. We did not plot the data set having 400 exams and conflict density equal to 0.32 due to initialization failure. That is, the Least Saturation Degree algorithm was unable to find a single solution for this data set given our restriction of 32 time slots.

Figure 58 through Figure 60 plot the *Average 1st Order Conflict Penalty* value for each iteration averaged across all particles and replicas. We compute the *Average 1st Order Conflict Penalty* by multiplying the average number of 1st Order Conflicts by the 1st Order Conflict Weight value, which equals 32.

This second set of graphs visually indicate no substantive change in the number of 1st Order Conflicts over the entire set of iterations, regardless of the number of exams or conflict density used. Like the previous set of graphs, and for the same reason, this set is missing the data set having 400 exams and conflict density equal to 0.32.

Table 18 presents some results for the Feasible/Infeasible experiment. Values under the 1st Order Conflicts subsection correspond to the experiment's last iteration and are represents averages across all particles and replicas. For example, the 1st Order Conflicts' "avg." column displays, for each data set, the average number of first order conflicts averaged across all 20 particles and 5 replications of this experiment. Some of the other columns under the 1st Order Conflicts section have the following meanings:

- avg % – what percent the average value represents of the enrollment value
- max. – the maximum number of 1st Order Conflicts
- min. – the minimum number of 1st Order Conflicts
- stdevp – the population standard deviation value

The Average Penalties subsection indicates the overall penalty value for the initial and final iterations averaged across all particles and replicas. The "reduced by" column shows the percent reduction in overall penalty between the initial and final values. The last two columns have the following meanings:

- 1st order – penalty value due solely to first order conflicts. This value equals the "avg." value multiplied by the first order penalty weight, which equals 32.
- % 1st of final – what percent the 1st order value represents of the final. In other words, it indicates how much first order conflicts contribute to the overall penalty value.

These tabular results show, with the slight exception in the case of the 10004 data set, no improvement in the average best value over the set of iterations.

It is obvious from the results that the initial swarm is unable to break free from the optimal point discovered through the Least Saturation Degree method. The hope was that the swarm would be able to fine-tune this initial solution, but this is not the case. Even though the swarm is still searching the solution space, evident from the changes in the 1st Order Conflict Penalty graphs, it never discovers a better solution. Consequently, it never updates its best solution value. As this occurred across all replicas, particles, exams, and conflict densities, it strongly indicates that using the LSD method to initialize the canonical form of the PSO does not assist it in its search. Quite the contrary, it locks its search to the initial solution.

Table 17. Feasible/Infeasible Experiment Attributes

Attribute	Value(s)
Initialization Method	Least Saturation Degree
Optimization Algorithm	Canonical PSO
Swarm Size (particles)	20
Max Iterations	50
Cognitive/Social Ratios ($\varphi_1 : \varphi_2$)	2.8:1.3
Inertia Weight (w)	0.75
1st Order Conflict Weight	32
Replications	5
Dataset Used	Typical Fabricated Data Set

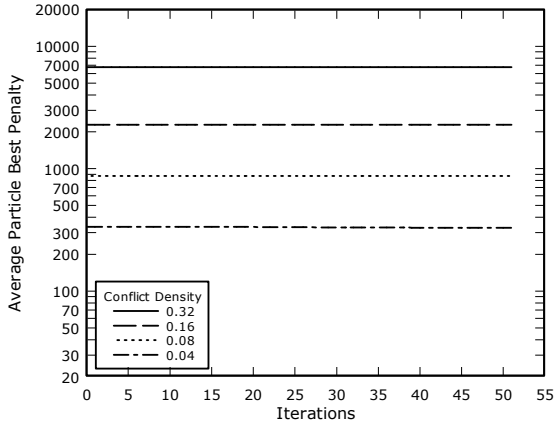

Figure 55. Feasible/Infeasible Experimental Results for Fabricated Data Set having n = 100 and r = 5

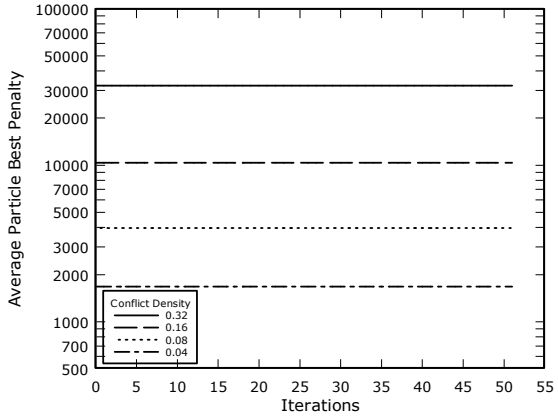

Figure 56. Feasible/Infeasible Experimental Results for Fabricated Data Set having n = 200 and r = 5

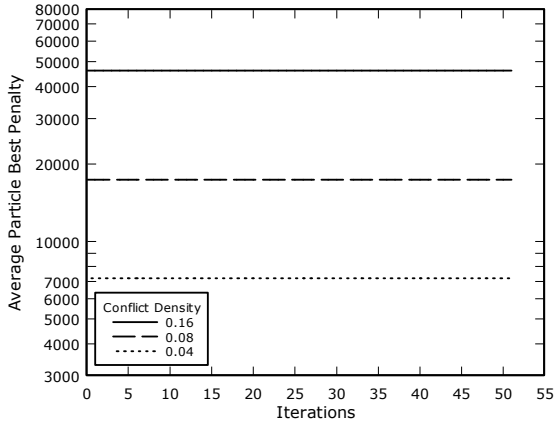

Figure 57. Feasible/Infeasible Experimental Results for Fabricated Data Set having n = 400 and r = 5

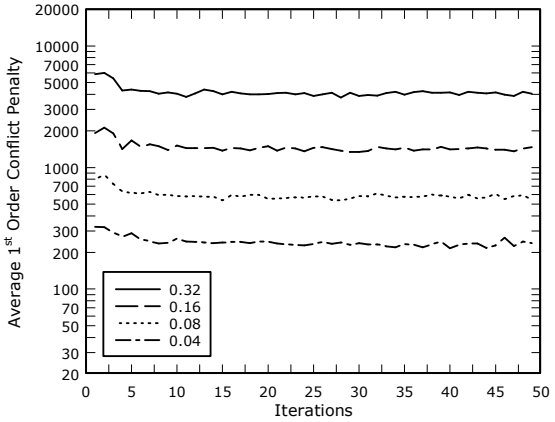

Figure 58. 1st Order Conflict Penalty for Feasible/Infeasible Experiment, using Fabricated Data Set with n = 100 and r = 5

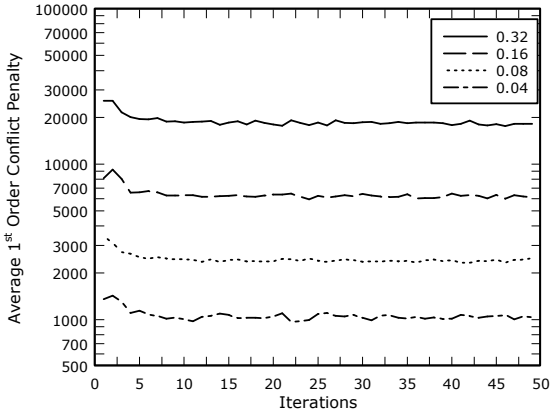

Figure 59. 1st Order Conflict Penalty for Feasible/Infeasible Experiment, using Fabricated Data Set with n = 200 and r = 5

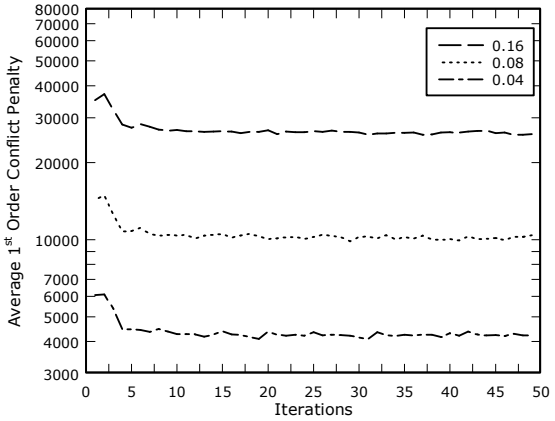

Figure 60. 1st Order Conflict Penalty for Feasible/Infeasible Experiment, using Fabricated Data Set with n = 400 and r = 5

Table 18. Feasible/Infeasible Experiment Results

data set	enroll.	1st Order Conflicts avg.	avg %	max.	min.	stdevp	Average Penalties initial	final	reduced by	1st order	% 1st of final
						100 Exams					
10004	136	7.9	5.77%	22	1	3.69	336	330	2%	251	76.2%
10008	341	17.2	5.04%	32	3	6.47	874	873	0%	550	63.0%
10016	774	45	5.79%	94	10	15.6	2290	2290	0%	1434	62.6%
10032	2266	130	5.75%	257	45	37.7	6757	6757	0%	4170	61.7%
						200 Exams					
20004	599	32.0	5.34%	59	8	9.14	1677	1677	0%	1024	61.1%
20008	1289	73.8	5.73%	128	19	20.6	3974	3974	0%	2362	59.4%
20016	3342	191	5.72%	310	79	44.5	10392	10392	0%	6112	58.8%
20032	10061	592	5.88%	1736	195	179.4	32205	32205	0%	18944	58.8%
						400 Exams					
40004	2369	132	5.57%	191	54	26.1	7193	7193	0%	4224	58.7%
40008	5503	318	5.78%	540	127	62.6	17381	17381	0%	10176	58.5%
40016	14294	818	5.72%	1173	326	147.1	46170	46170	0%	26176	56.7%
40032	42761	*INITIALIZATION FAILURE*									

Constraint Handling – Feasible/Feasible Experiment

Table 19 lists the values used for each experimental attribute. We ran the experiment for the 12 data sets found in the Typical Fabricated Data Sets, limiting the number of iterations to 50. As in the case with the Feasible/Infeasible approach, this limitation was sufficient to observe the characteristics of this approach. In like manner, we also limited the replications to five in order to make comparisons between these two methods possible.

Figure 61 through Figure 63 plot the *Average Particle Best Penalty* value against the iteration for all replicas and particles for 11 of the 12 data sets. These sets of graphs appear to show a slight change in the best value for a number of test parameter combinations. We did not plot the data set having 400 exams and conflict density equal to 0.32 due to the same initialization failure as reported under the previous Feasible/Infeasible section.

Table 20 presents some results for the Feasible/Feasible experiment. The columns having the same title as those in the Feasible/Infeasible experiment's result table also have the same meaning. These tabular results show, unlike the previous experiment, definite improvements in the average penalty values due to our use of the PSO-NoConflicts algorithm.

These results are significant seeing as how we used the same initial swarms as were used in the previous Feasible/Infeasible experiment. In contrast though to the previous experiment, not only do we realize improved solutions here, but also all solutions discovered are feasible. This experiment demonstrates that the PSO-

NoConflicts optimization algorithm is able to move the particle between feasible solution points, finding better solution along the way.

The Least Saturation Degree heuristic, by itself, is commonly used to solve the examination timetable problem as it produces near optimal feasible solutions. This experiment suggests that the hybridization of LSD with PSO-NoConflicts optimization produce better results without a substantial increase in computational time. Of course, it also demonstrates that the optimization phase is less successful as the data sets become more constrained, as it is less able to find better solutions. This makes sense, as even the LSD, with our implementation of backtracking, was unable to find a solution to the highly constrained 40032 data set. By definition, fewer available feasible solutions exist for the optimization algorithm to discover as the data set becomes more constrained.

Table 19. Feasible/Feasible Experiment Attributes

Attribute	Value(s)
Initialization Method	Least Saturation Degree
Optimization Algorithm	PSO-NoConflicts
Swarm Size (particles)	20
Max Iterations	50
Cognitive/Social Ratios ($\varphi_1 : \varphi_2$)	2.8:1.3
Inertia Weight (w)	0.75
1st Order Conflict Weight	32
Replications	5
Dataset Used	Typical Fabricated Data Sets

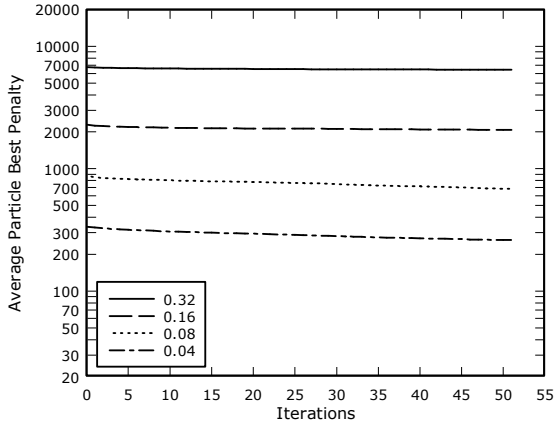

Figure 61. Feasible/Feasible Experimental Results for Fabricated Data Set having n = 100 and r = 5

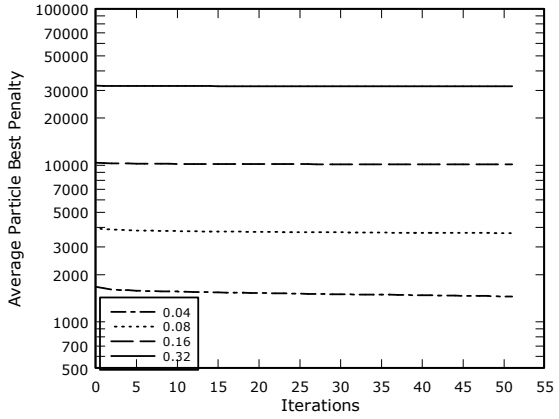

Figure 62. Feasible/Feasible Experimental Results for Fabricated Data Set having n = 200 and r = 5

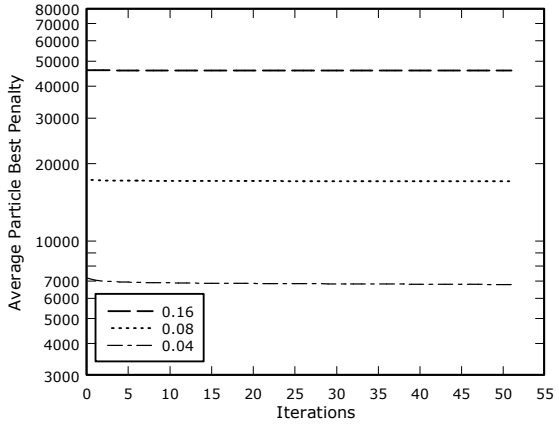

Figure 63. Feasible/Feasible Experimental Results for Fabricated Data Set having n = 400 and r = 5

Table 20. Feasible/Feasible Experiment Results

		Average Penalties			avg. secs.
dataset	enroll.	initial	final	reduced by	per pass
		100 Exams			
10004	136	336	261	22.3%	3.31
10008	341	874	685	21.6%	3.66
10016	774	2290	2074	9.4%	4.51
10032	2266	6757	6436	4.7%	4.78
		200 Exams			
20004	599	1677	1450	13.6%	7.75
20008	1289	3974	3688	7.2%	9.13
20016	3342	10392	10113	2.7%	10.02
20032	10061	32205	31983	0.7%	12.02
		400 Exams			
40004	2369	7193	6788	5.6%	19.00
40008	5503	17381	17069	1.8%	22.60
40016	14294	46170	46011	0.3%	24.52
40032	42761	*INITIALIZATION FAILURE*			

Real World Data – Full PSO Random Experiment

Table 21 lists the values used for each experimental attribute. We ran the experiment for the 13 data sets found in the Full Carter Data Sets, limiting the number of iterations to 300 and 10 replications.

Figure 64 through Figure 66 plot the *Average Particle Best Penalty* value against the iteration for all replicas and particles for the 13 data sets. The results are broken up across these three plots to better show the results. Because of the magnitude differences of results, placing them all on a single plot would obscure the changes over time.

Table 22 presents some results for the Full PSO Random Experiment. The meanings of the individual columns were covered earlier in this thesis.

Table 23 displays the minimum, maximum, average, and population standard deviation of the values from four columns within Table 22: “avg %”, “reduced by”, “% 1st of total”, and “avg. secs. per pass”.

It is visually apparent from the figures that the PSO algorithm was able to reduce the penalty across all examination data sets and, according to Table 23, this reduction ranged from 18.6% to 75.4% with an average reduction of 42.9% within the allotted 300 iterations. We also see from Table 23 that the 1st Order Conflicts range from 0.5% to 4.1%, having an average of 2.3% and a standard deviation of 1.0%.

Let us now look at the data set CAR-F-92 as a representative example set. According to Burke and Newall (2004), the solution to this data set takes 32 periods (i.e., time slots), which is not the case for the other data sets. Therefore, as we also used 32 time slots for our testing, this test provides realistic results.

From Table 4, we see that the CAR-F-92 data set has 543 exams, 18419 students, 55522 enrollments, 3.01 enrollments per student. Given, from Table 22, that this exam had an 1st Order Conflicts “avg %” value 2.3 percent, one can compute that there are, on average, 6.9 (2.3 x 3.01) 1st order conflicts per every 100 students, or about seven percent of the students are involved in a conflict. Given that there are 18419 students in this data set, on average, approximately 1275 students are involved in a 1st order conflict.

Figure 50, from a previous experiment, displays how the number of 1st order conflicts decrease over iterations for an data set with 400 exams and a density of 0.16; closely matching this case. In fact, the 1st order conflicts are reduced to the point where they only contribute to 28.1% of the total penalty value in this examination’s case. Even

though, as we can see from Figure 50, there is a substantial decrease in 1st order conflicts, most universities would probably consider seven percent far too high to be acceptable.

If we look at the rest of the "avg %" values in Table 22, we do notice examinations having lower percentages such as the UTE-S-92 examination with a value of only 0.5%. Unfortunately, this does not truly represent a real world case as we statically set our number of exam time slots to 32 regardless of the exam. According to Burke and Newall (2004), this particular exam's realistic number of time slots is 10.

In the end, we have three conclusions:

1. The PSO algorithm is able to consistently reduce the value of the penalty function over a wide range of real world examinations; an average of 42.9% according to Table 23, given our parameters.
2. Though there is good convergence, as is typical of the canonical PSO algorithm, it unfortunately converges too quickly to suboptimal solutions. Other, more modern, versions of the PSO may prevent this, but the simple canonical form obviously struggles.
3. The penalty function used for testing is incapable of guiding the algorithm enough to significantly reduce 1st order conflicts. As was shown earlier, increasing the 1st Order Conflict Weight may help some, but a different penalty function would be a more realistic line of attack for improvement as the choice of penalty function can have a significant impact on the PSO algorithm's success rate.

Table 21. Full PSO Random Experiment Attributes

Attribute	Value(s)
Initialization Method	Random
Optimization Algorithm	PSO
Swarm Size (particles)	20
Max Iterations	300
Cognitive/Social Ratios ($\varphi_1 : \varphi_2$)	2.8:1.3
Inertia Weight (w)	0.75
1st Order Conflict Weight	32
Replications	10
Dataset Used	Full Carter Data Set

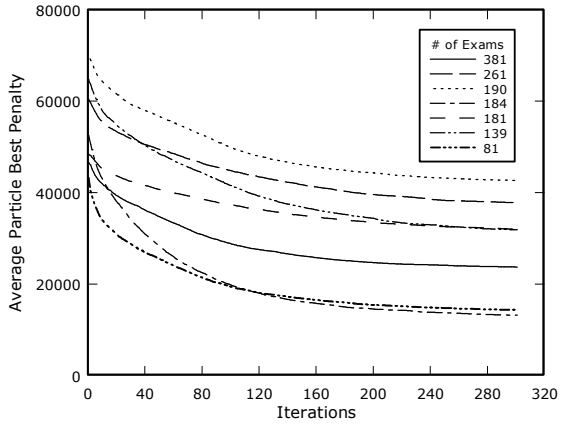

Figure 64. Full PSO Random Results with *n* between 81 and 381, and *r* = 10

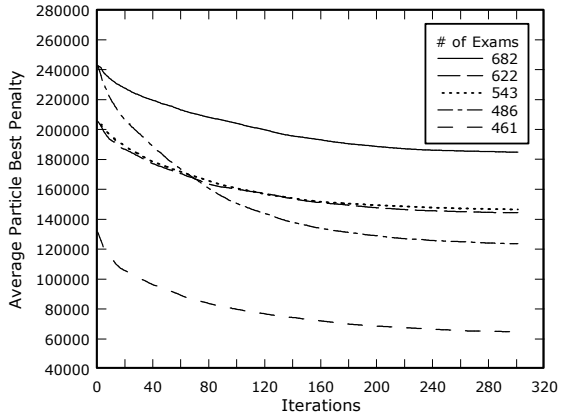

Figure 65. Full PSO Random Results with *n* between 461 and 682, and *r* = 10

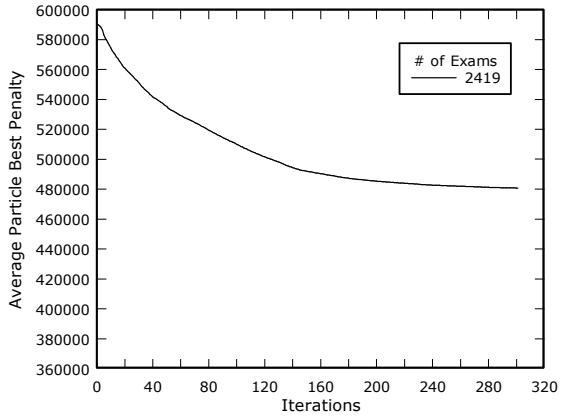

Figure 66. Full PSO Random Results with *n* = 2419 and *r* = 10

Table 22. Full PSO Random Experiment Results

		1st Order Conflicts					Average Penalties					
data set	enroll.	avg.	avg %	max.	min.	stdevp	initial pass	final pass	reduced by	1st at final	% 1st of total	avg. secs. per pass
CAR-F-92	55522	1286	2.3%	3034	1128	203	206264	146613	28.9%	41152	28.1%	8.16
CAR-S-91	56877	1694	3.0%	3007	1532	170	243002	184765	24.0%	54208	29.3%	11.90
EAR-F-83	8109	290	3.6%	1089	198	114	70433	42684	39.4%	9280	21.7%	2.52
HEC-S-92	10632	96	0.9%	1161	38	134	44634	14379	67.8%	3072	21.4%	1.15
KFU-S-93	25113	441	1.8%	2121	260	258	133699	64824	51.5%	14112	21.8%	5.02
LSE-F-91	10918	159	1.5%	587	108	50	47245	23744	49.7%	5088	21.4%	4.10
PUR-S-93	120681	4889	4.1%	6322	4556	237	590458	480740	18.6%	156854	32.6%	39.87
STA-F-83	45051	896	2.0%	1217	593	175	244683	123571	49.5%	30319	24.5%	5.78
STA-F-83	5751	132	2.3%	958	37	100	65503	31859	51.4%	4224	13.3%	1.84
TRE-S-92	14901	272	1.8%	810	203	90	61046	37826	38.0%	8704	23.0%	3.28
UTA-S-92	58979	1276	2.2%	2574	1135	153	205751	144337	29.8%	41507	28.8%	9.30
UTE-S-92	11793	61	0.5%	397	13	57	53670	13211	75.4%	1952	14.8%	2.06
YOR-F-83	6034	221	3.7%	624	166	62	48555	31994	34.1%	7072	22.1%	2.39

Table 23. Some Full PSO Random Result Statistics

	avg %	reduced by	% 1st of total	avg. secs. per pass
min.	0.5%	18.6%	13.3%	1.15
max.	4.1%	75.4%	32.5%	39.87
avg.	2.3%	42.9%	23.1%	7.49
stdevp	1.0%	16.0%	5.2%	--

Real World Data – Reduced PSO Random Experiment

Table 24 lists the values used for each experimental attribute. We ran the experiment for the 13 data sets found in the Reduced Carter Data Sets, limiting the number of iterations to 300 and 10 replications.

Figure 67 through Figure 69 plot the *Average Particle Best Penalty* value against the iteration for all replicas and particles for the 13 data sets. The results are broken up across the same three plots as the previous experiment to make comparisons easier.

Table 25 presents some results for the Reduced PSO Random Experiment. Table 26 displays the minimum, maximum, average, and population standard deviation of the values from four columns within Table 25. The format of these two tables is identical to the previous experiment for comparison purposes.

A comparison of the figures and tables of this experiment against those of the previous experiment make it quite clear that there is no appreciable difference in results between these two data sets. This is readily apparent by comparing the values of Table 23 against Table 26. Therefore, any tests performed using the reduced data set should be considered equivalent to the full Carter data set, as we original hypothesized.

Table 24. Reduced PSO Random Experiment Attributes

Attribute	Value(s)
Initialization Method	Random
Optimization Algorithm	PSO
Swarm Size (particles)	20
Max Iterations	300
Cognitive/Social Ratios ($\varphi_1 : \varphi_2$)	2.8:1.3
Inertia Weight (w)	0.75
1st Order Conflict Weight	32
Replications	10
Dataset Used	Reduced Carter Data Set

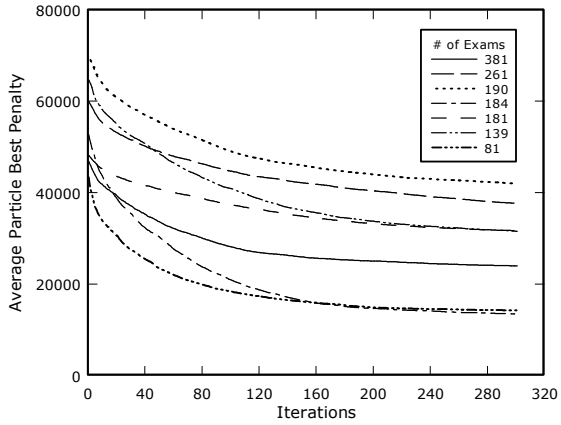

Figure 67. Reduced PSO Random Results with *n* between 81 and 381, and *r* = 10

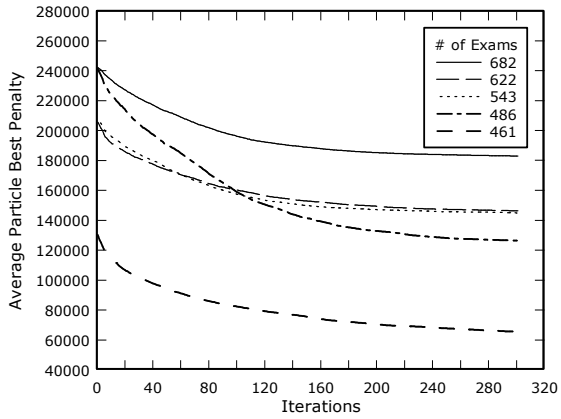

Figure 68. Reduced PSO Random Results with *n* between 461 and 682, and *r* = 10

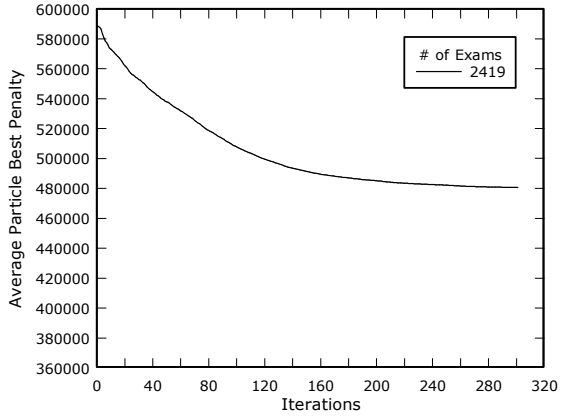

Figure 69. Reduced PSO Random Results with *n* = 2419 and *r* = 10

Table 25. Reduced PSO Random Experiment Results

		1st Order Conflicts					Average Penalties					
data set	enroll.	avg.	avg %	max.	min.	stdevp	initial pass	final pass	reduced by	1st at final	% 1st of total	avg. secs. per pass
CAR-F-92-R	51553	1227	2.4%	2586	1028	138	208497	144930	30.5%	39264	27.1%	8.12
CAR-S-91-R	53468	1625	3.0%	2735	1506	127	242728	182918	24.6%	52000	28.4%	10.97
EAR-F-83-R	8108	281	3.5%	1040	207	95	70321	42006	40.3%	8992	21.4%	2.66
HEC-S-92-R	10311	94	0.9%	758	31	95	44675	14259	68.1%	3008	21.1%	1.09
KFU-S-93-R	24837	441	1.8%	2863	303	216	132677	65570	50.6%	14112	21.5%	4.94
LSE-F-91-R	10819	181	1.7%	825	141	79	47359	23966	49.4%	5792	24.2%	4.09
PUR-S-93-R	118054	4876	4.1%	7090	4687	271	588699	480580	18.4%	156032	32.5%	38.63
RYE-S-93-R	43026	880	2.0%	3097	664	215	243526	126463	48.1%	28160	22.3%	5.62
STA-F-83-R	5751	135	2.3%	1042	35	120	65053	31578	51.5%	4320	13.7%	1.63
TRE-S-92-R	14234	289	2.0%	837	199	111	60554	37685	37.8%	9248	24.5%	3.28
UTA-S-92-R	52799	1361	2.6%	2905	1181	179	206623	146326	29.2%	43552	29.8%	9.75
UTE-S-92-R	11715	67	0.6%	712	20	82	53531	13492	74.8%	2144	15.9%	2.43
YOR-F-83-R	6033	201	3.3%	745	160	73	48313	31674	34.4%	6432	20.3%	2.41

Table 26. Some Reduced PSO Random Result Statistics

	avg %	reduced by	% 1st of total	avg. secs. per pass
min.	0.6%	18.4%	13.7%	1.09
max.	4.1%	74.8%	32.5%	38.63
avg.	2.3%	42.9%	23.3%	7.36
stdevp	1.0%	15.8%	5.1%	--

Real World Data – Reduced LSD No-Conflicts Experiment

Finding no significant difference in results between the Reduced and Full Carter Data Sets, we settled on using the reduced set for this experiment as it had slightly less data to handle.

Table 27 lists the values used for each experimental attribute. We ran the experiment for the 13 data sets found in the Reduced Carter Data Sets, limiting the number of iterations to 50. We again limited the replications to five due to the computationally higher cost the of Least Saturation Degree initialization method.

Figure 70 and Figure 71 plot the *Average Particle Best Penalty* value against the iteration for all replicas and particles for 12 of the 13 data sets. These sets of graphs appear to show a slight change in the best value for a number of test parameter combinations, which is more evident if one looks at the "reduced by" column of Table 28.

Table 28 presents some results for this experiment and includes columns found in Table 20 with the exception of the "Average Penalty per Student" section. The "Average Penalty per Student" metric is the one originally used by Carter, M. W., Laporte, & Lee, S. Y. (1996).

Table 27. Reduced LSD No-Conflicts Experiment Attributes

Attribute	Value(s)
Initialization Method	Least Saturation Degree
Optimization Algorithm	PSO-NoConflicts
Swarm Size (particles)	20
Max Iterations	50
Cognitive/Social Ratios ($\varphi_1 : \varphi_2$)	2.8:1.3
Inertia Weight (w)	0.75
1st Order Conflict Weight	32
Replications	5
Dataset Used	Reduced Carter Data Set

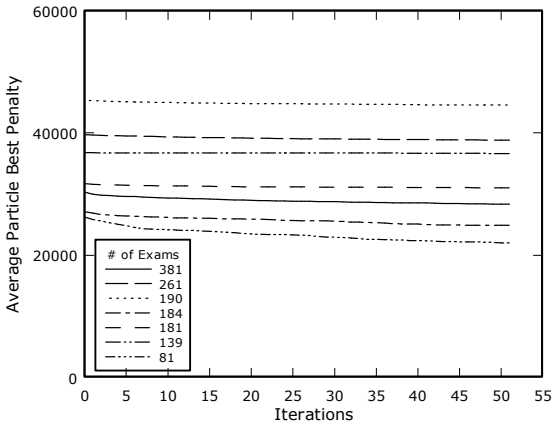

Figure 70. Reduced LSD No-Conflicts Results with *n* between 81 and 381, and *r* = 5

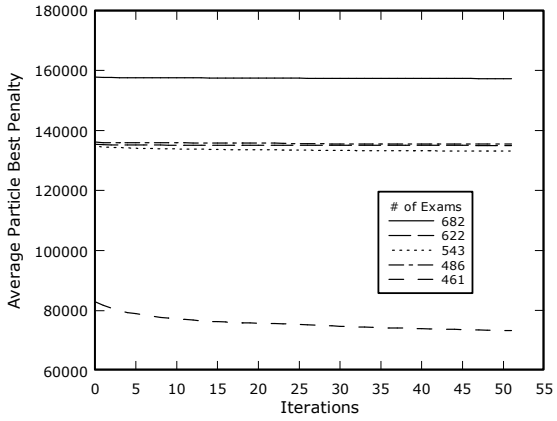

Figure 71. Reduced LSD No-Conflicts Results with *n* between 461 and 682, and *r* = 5

Table 28. Reduced LSD No-Conflicts Experiment Results

			Average Penalties			Average Penalty per Student			
data set	enroll.	density	initial	final	reduced by	students	initial	final	avg. secs. per pass
CAR-F-92-R	51553	0.138	134752	133137	1.2%	18419	7.32	7.23	33.53
CAR-S-91-R	53468	0.128	157817	157278	0.3%	16925	9.32	9.29	46.90
EAR-F-83-R	8108	0.266	45332	44531	1.8%	1125	40.30	39.58	9.88
HEC-S-92-R	10311	0.415	26238	22039	16.0%	2823	9.29	7.81	3.83
KFU-S-93-R	24837	0.055	82944	73315	11.6%	5349	15.51	13.71	24.04
LSE-F-91-R	10819	0.062	30309	28351	6.5%	2726	11.12	10.40	18.29
PUR-S-93-R	118054	0.029	*INITIALIZATION FAILURE*						
RYE-S-93-R	43026	0.075	136121	135456	0.5%	11483	11.85	11.80	26.62
STA-F-83-R	5751	0.143	36767	36615	0.4%	611	60.18	59.93	5.77
TRE-S-92-R	14234	0.180	39729	38796	2.3%	4360	9.11	8.90	14.56
UTA-S-92-R	52799	0.125	135430	135019	0.3%	21266	6.37	6.35	40.15
UTE-S-92-R	11715	0.084	27122	24880	8.3%	2750	9.86	9.05	8.45
YOR-F-83-R	6033	0.287	31689	31016	2.1%	941	33.68	32.96	9.61

Summary of Results

Discovering PSO Parameters

As this is the first time the PSO algorithm has been applied to any timetabling problem, we had to discover appropriate algorithmic parameters for this problem domain in order to achieve reasonable and acceptable experimental results. The values we derived are shown in Table 29 and the arguments for the final values are detailed in the Findings section of the Results chapter.

Table 29. Final PSO Algorithmic Parameter Choices

Algorithmic Parameter	Value for Experiments
Cognitive/Social Ratio	(2.8 : 1.3)
"Best" Inertia Weight	0.75
"Best" Swarm Size	20
1st Order Conflict Weight	32

We also discovered that a much larger 1st Order Conflict Weight does not automatically produce fewer first order conflicts. In fact, the values 2048 and 131072 consistently produced very similar results and, in some cases, results significantly higher ones. Therefore, an assumption that a larger weight would lead to fewer conflicts was not valid.

Constraint Handling

We then used the parameter values determined above to investigate the abilities of the Feasible/Infeasible and Feasible/Feasible approaches at handling constraints.

The Feasible/Infeasible results demonstrated that the Least Saturation Degree initial swarm was unable to break free from the optimal point it found. The experiment

strongly implied that using the LSD method to initialize the canonical form of the PSO does not assist it in its search. On the contrary, it locked its search to the initial solution. However, these results do not tell us how well some of the newer versions of the PSO algorithm, specifically designed to prevent premature stagnation, would fare given this same scenario. It does tell us though that, published assertions to the contrary, better initial solutions do not necessarily lead to better results.

The results from the Feasible/Feasible method established that we were able to discover solutions exceeding the quality of those found by the Least Saturation Degree initialization phase. This was a direct result of our original PSO-NoConflicts optimization approach. Unlike the Feasible/Infeasible approach, not only were we able to find improved solutions, but these solutions were feasible. This experiment suggests that the hybridization of LSD with PSO-NoConflicts optimization produce better results. Of course, it also demonstrated that the optimization phase is less successful as the data sets become more constrained, as it is less able to find better solutions.

Real World Data Sets

Our last section of experiments tested the algorithm against real world data sets. We first tested it against the full Carter data set and, in the end, came to the conclusion that:

1. The PSO algorithm is able to consistently reduce the value of the penalty function over a wide range of real world examinations.
2. Though there is good convergence, as is typical of the canonical PSO algorithm, it unfortunately converges too quickly to suboptimal solutions.

3. The penalty function used for testing is incapable of guiding the algorithm enough to significantly reduce 1st order conflicts.

Our next experiment was designed to compare the results from the reduced Carter data set against the previous results from the full Carter data set. A comparison of the results made it clear that there were no appreciable differences between these two data sets. Therefore, any tests performed using the reduced data set should be considered equivalent to the full Carter data set.

Finally, we performed an experiment using Least Saturation Degree for initialization, the PSO-NoConflicts algorithm for optimization, and the reduced Carter data set, with the intent of comparing these results against those of other researchers.

Table 30 lists a number of published Average Penalty per Student values for the CAR-F-92 examination data set and our result for the same set. We only show the values for this particular examination data set, as it is the only one of the thirteen real world data sets having 32 periods (i.e., timeslots) as we used. Others have either more or fewer than this number, according to Burke and Newall (2004).

Table 30. Average Penalty per Student Comparisons for CAR-F-92

Average Penalty per Student	Reference
7.23	"final" value from Table 28
5.6	Di Gaspero, L., & Schaerf, A. (2000)
6.2 – 8.2	Carter, M. W., Laporte, & Lee, S. Y. (1996)
4.86 – 6	Burke, E. K., & Newall, J. P. (2004)
6.82	Burke, E. K., Eckersley, A., McCollum, B., Petrovic, S., & Qu, R. (2003a)
5.77 – 5.86	Computed from penalty values given in Burke, E. K., Eckersley, A., McCollum, B., Petrovic, S., & Qu, R. (2003b)

Our final value of 7.23 is only slightly better than the initial value of 7.32 computed through our Least Saturation Degree initialization phase, which represents a reduction of just a little over 1.2%. We do see data sets in Table 28 having better improvements, but these correspond to real world data sets that would normally map to far fewer timeslots. For example, according to Burke and Newall (2004), the HEC-S-92 and KFU-S-93 data sets really only require 18 and 20 timeslots respectively. Therefore, these are nowhere nearly as constrained as the CAR-F-92 data set and our PSO-NoConflicts optimization algorithm is easily able to find valid solutions, as there are more available in the vicinity.

We should also point out here that the Least Saturation Degree method, by itself, found better solutions than even our Reduced PSO Random experiment did in the case of the CAR-F-92 data set. From Table 25 we see that this data set had a final penalty of 144930, which translates to an Average Penalty per Student value of approximately 7.87. This is not only greater, though not by much, but 27.1% of this penalty value represents

first order conflicts. Therefore, the PSO algorithm does get one close to existing published values, but at a cost.

Chapter 5

Conclusions, Implications, Recommendations, and Summary

Conclusions

The goal of this study was to investigate the suitability and effectiveness of the relatively new Particle Swarm Optimization techniques when applied to the University Examination Timetabling class of problems.

We limited this study to determining the efficacy of the canonical PSO algorithm. Though additional research has evolved the PSO algorithm far beyond this form, it was thought best to start with this model of the particle swarm as no published research exists even for this simplified model. Analyzing the capability of the particle swarm with the standard model kept the research unencumbered by extraneous and potentially superfluous algorithmic features.

This work also ignored soft-constraints even though these are important in any real world university examination timetabling scenario. Though easily added to the algorithm's objective function, the complexity they add to the analysis would not have been compensated for by an equivalent amount of insight into the basic workings of the PSO in this problem domain.

This study demonstrated that, even though the algorithm was able to reduce penalty function values of examination timetabling problems having relatively few examinations and enrollments, it did not satisfactorily scale up into timetabling problems beyond modest proportions. We believe this to be partly due to the PSO algorithm's known premature convergence tendency, which would hamper its ability to find the

global minimum in a strongly multi-modal search space. Other researchers have recently put forth PSO variants in an attempt to tackle this problem, but this work did not investigate this issue directly.

Another reason the algorithm may have failed to perform well deals with the mapping between the objective and parameter spaces. As Fieldsend and Singh (2002) point out, if there is little or no correlation between 'closeness' in objective space and 'closeness' in parameter space, PSO methods may experience problems. We assumed that there was some degree of correlation between the two spaces so that the objective function could adequately direct the swarm's movement. A mismatch can occur when the model—direct-representation in our case—turns out to be an unsatisfactory representation. Our tests showed that convergence did indeed occur with our representation but its efficacy appears limited by premature converge.

A further reason this algorithm may have struggled with the timetabling problem comes from the research of Tsou and MacNish (2003). They studied the PSO algorithm's efficacy in high-dimensional highly convex search spaces and found that the algorithm breaks down in this problem space. Our sense is that variants of the university examination timetabling problem may fall under the classification of "highly convex" but this is undoubtedly dependent on the set of soft and hard constraints. Fortunately, even in this special problem case, the authors found a modified PSO algorithm, referred to as the Adaptive Particle Swarm Optimizer (APSO), which contains an adaptive learning rate function and produced higher success rates than the original PSO. It is the ease with which one can extend the PSO that makes it appealing to researchers. Regardless, our

current research effort did not investigate PSO extensions. Extensions to the canonical form of the PSO are left for future studies.

We knew from the outset that finding solutions to the Carter data sets was a difficult problem. As pointed out by Burke, Newall, and Weare (1998b), the data sets "offer a challenge from an optimization point of view" (p. 95). They go on to state that "the data used here present a challenge merely to find a feasible solution" (p. 98). We therefore did not expect this study to show that the PSO algorithm was best or better than other approaches, but instead, the goal was to provide insight into its ability to tackle the truly large, highly multi-modal, and disconnected search spaces of the single-objective university examination timetabling problem, which is what we accomplished.

Implications

Contributions to the Field of Study

With respect to contributions to the field of study and advancement of knowledge, our research systematically investigated the influence of problem and algorithm factors in solving this particular timetabling problem and determined the algorithm's performance profile under the specified test environment. Keys factors employed for discovering the algorithm's timetabling utility, ability, and limitations included problem size (i.e., number of enrollments), conflict matrix density, and swarm size.

Additionally, this work presented insight into how well the PSO algorithm performs compared with other algorithms used to attack the same problem and data sets, providing understanding on the PSO algorithm's efficacy on university examination timetabling problems.

There were two additional contributions made during this research: a better way to fabricate examination timetabling data sets and the introduction of the PSO-NoConflicts optimization approach.

Firstly, the algorithm under the *Typical Fabricated Data Set* section (see page 68) describes a new method that produces data sets more representative of real world examination timetabling data sets. This approach not only permitted us to construct data sets spanning a wide range of sizes and densities, but it allowed us to have confidence in the results of experiments using these sets.

Secondly, our derived PSO-NoConflicts algorithm, described under the *Feasible/Feasible Approach* section of this thesis (see page 49), permits the PSO

algorithm to perform searches while still satisfying constraints. This approach shows promise and its ability to handle other constraint satisfaction problems is worth investigating further.

Benefits

According to Angeline (1998), the "Particle swarm often locates near optima significantly faster than evolutionary optimization but can not dynamically adjust its velocity step size to continue optimization at a finer grain of search once in the general region of the optima. This causes its performance to flatten out dramatically in almost every case tested" (pp. 608, 610). Angeline goes on to state that even though evolutionary optimization surpassed the performance of particle swarm on nearly every function in their test suite, when the gradient of the search space indicated the direction to the optima, performance of PSO algorithm was exceptional. Regardless, as pointed out by Burke and Newall (2004), one could consider the PSO approach a success if it just gets into the "ballpark" of the other heuristics. The PSO's real strength lies in it being more general, much quicker, and much easier to implement.

Determining the simple or canonical PSO's performance profile—the quality of the solution as a function of the execution time—on the university examination timetabling problem also provides a benefit. Knowing the algorithm's strengths and limitations is useful in determining its value in attacking timetabling problems in general and the university examination timetabling problem in particular. As far as the author can determine, the PSO algorithm has never been applied to any kind of timetabling problem before. This research provides insight into its suitability as an approach against

university examination timetabling problems. To this end, we employed the following problem and algorithm factors to probe the algorithm's efficacy:

- Conflict Matrix Density – measured the impact conflict matrix density has on the algorithm's ability over a range of problem sizes.
- Problem Size – measured the relationship between the size of the problem, as determined by enrollment, and the algorithm's effectiveness.
- Swarm Size – measured the relationship between the size of the swarm (i.e., number of particles), the size of the problem, and the algorithm's effectiveness.
- Objective Function Weight – measured the affect of altering the 1st order conflict weight value (i.e., w_s, where $s = 0$) in the spread examination objective function (see Equation 5).

Besides the benefit of now knowing how well the canonical form of the algorithm handles the university examination timetabling problem, the information garnered provides insight into the algorithm's applicability to timetabling problems.

Recommendations

Recommendations for Additional Studies

This research could take a number of different future directions, given the results established by this effort, and we mention but a few here.

PSO variants – There have been many variants and extensions to the standard PSO algorithm proposed since the algorithm's original introduction. They include attempts to mitigate premature convergence (Blackwell, 2003a; Blackwell, 2003b; Peram, Veeramachaneni, & Mohan, 2003; Riget & Vesterstrøm, 2002; Silva, Silva, & Costa,

2002), handle dynamic environments (Carlisle & Dozier, 2002; Hu & Eberhart, 2002), provide self-adapting parameters (Clerc, 2002a; Clerc, 2002b), use multiple swarms (Parsopoulos & Vrahatis, 2002c; Al-kazemi & Mohan, 2002a; Al-kazemi & Mohan, 2002b), and enlist PSO/GA hybrids (Robinson, Sinton, & Rahmat-Samii, 2002). Studying one or more of these modified algorithms on this problem domain would provide insight into how each performs within the problem class.

Encoded representation – Using a non-direct representation for the particles would no doubt produce different results. Terashima-Marín, Ross, and Valenzuela-Rendón (1999) use a non-direct chromosome representation in their genetic algorithm approach to the timetabling problem, instead of the direct chromosome, and found performance improvement. Instead of the chromosome representing a timetable solution to the problem, the non-direct representation encoded instructions and parameters for guiding the algorithm's search for a timetable solution. This approach may be a good match for the particle swarm algorithm as well and is worth investigating.

Penalty function – The form of penalty functions, used in the objective function, appear to have a significant impact on the outcome, based on preliminary tests performed by the author. Studying the correlation between the form of the penalty function and the results would give guidance to others on how best to formulate the functions. (Burke, Eckersley, McCollum, Petrovic, & Qu, 2003b)

Soft-Constraints – Any future work should incorporate a number of typical university examination timetabling soft-constraints. Their inclusion would require extending the objective function using our approach. Other PSO variants handle soft-constraints in different fashions and this too would be insightful.

Premature Convergence – A possible reason our implementation of the PSO algorithm struggled with the timetabling problem comes from the research of Tsou and MacNish (2003). They studied the PSO algorithm's efficacy in high-dimensional highly convex search spaces and found that the algorithm breaks down in this problem space. Our sense is that variants of the university examination timetabling problem may fall under the classification of "highly convex" but this is undoubtedly dependent on the set of soft and hard constraints. Fortunately, even in this special problem case, the authors found a modified PSO algorithm, referred to as the Adaptive Particle Swarm Optimizer (APSO), which contains an adaptive learning rate function and produced higher success rates than the original PSO. It is the ease with which one can extend the PSO that makes it appealing to researchers. Extensions to the canonical form of the PSO that slow down converge would be ideal for future studies.

PSO-NoConflicts Approach – It would be useful to determine the value of using our PSO-NoConflicts optimization approach in other problem domains requiring constraint satisfaction. Al-kazemi and Mohan (2002b) incorporated a similar idea into their version of the PSO. They calculated new potential positions by considering a subset of the dimensions at the same time and "the fitness change resulting from updating the position along these dimensions is calculated, and then a decision is made whether to implement the corresponding change of position" (p. 490). Their approach does not guarantee feasible solutions as it was not designed to handle constraint satisfaction in addition to optimization as is the case with our approach.

Summary

The general timetabling problem consists of assigning resources to objects in space and time while satisfying a set of hard constraints and, as nearly as possible, a set of soft constraints. The three most common academic timetabling problems are the school timetabling, university timetabling, and examination timetabling problems. The examination timetabling problem consists of assigning a set of examinations to a limited number of time slots while satisfying the hard constraints. The most commonly cited hard constraint is the prevention of double booking; i.e., no student should be required to take more than one exam during any single time slot. Many times this requirement is unobtainable and is therefore relaxed. When this happens, the objective is to minimize the number of double booking conflicts.

The sheer size of university student bodies makes examination timetabling a complex combinatorial problem. Even in moderately sized universities, the manual solution of timetabling usually requires many person-days of effort. As a result, much research has been conducted over the last forty years and many papers related to the automation of academic timetabling have been published. Numerous approaches to solving the timetabling problem have been proposed in the literature. The earliest approaches were heuristic in nature, next came more general methods such as graph coloring, and then came the more recent attempts using simulated annealing and genetic algorithms. Despite enormous effort expended over the last four decades to uncover efficient methods for solving timetabling problems, these problems are nevertheless still the focus of intense research.

The overriding reason the school timetabling problems have been the focus of such prolonged and intense research is the inescapable fact that, except for the most trivial non-real world cases, these problems are all NP-complete. Hence, it should come as no surprise then that attempts are made to apply each newly discovered and relevant algorithmic approach to this challenge. In like manner, the goal of this study was to investigate the suitability and effectiveness of the Particle Swarm Optimization techniques when applied to the University Examination Timetabling class of problems.

We accomplished this by analyzing experimentally the performance profile of the standard form of the PSO algorithm when brought to bear against the university examination timetabling problem. This study systematically investigated the impact of various factors in solving this particular class of timetabling problems and determined the algorithm's performance profile under the specified test environment.

As this was the first time the PSO algorithm was applied to any timetabling problem, we had to discover appropriate algorithmic parameters for this problem domain in order to achieve reasonable and acceptable experimental results. Values were experimentally determined for the PSO algorithm factors such as the cognitive/social ratio, “best” inertia weight, “best” swarm size, and 1st order conflict weight.

These parameters were then used to investigate the abilities of the canonical PSO algorithm’s ability in handling constraints through two main methods we called the Feasible/Infeasible and Feasible/Feasible approaches.

The Feasible/Infeasible results demonstrated that the Least Saturation Degree initial swarm was unable to break free from the optimal point it found. The experiment

strongly implied that using the LSD method to initialize the canonical form of the PSO does not assist it in its search. On the contrary, it locked its search to the initial solution.

The results from the Feasible/Feasible method established that we were able to discover solutions exceeding the quality of those found by the Least Saturation Degree initialization phase. This was a direct result of our original PSO-NoConflicts optimization approach.

Our last set of experiments tested the algorithm against real world data sets. We first tested it against the full Carter data set and, in the end, came to the conclusion that, though the PSO is able to consistently reduce the value of the penalty function over a wide range of real world examinations, it unfortunately converges too quickly to suboptimal solutions. Additionally, the experiments implied that the penalty function used for testing is incapable of guiding the algorithm enough to significantly reduce 1^{st} order conflicts.

Our next experiment compared the results from the reduced Carter data set against the previous results from the full Carter data set. A comparison of the results made it clear that there were no appreciable differences between these two data sets. Therefore, any tests performed using the reduced data set should be considered equivalent to the full Carter data set.

Finally, we performed an experiment using Least Saturation Degree for initialization, the PSO-NoConflicts algorithm for optimization, and the reduced Carter data set, with the intent of comparing these results against those of other researchers. Our results were near those of other published works, but only because of our use of the Least Saturation Degree for initialization. This experiment suggested that the hybridization of

LSD with PSO-NoConflicts optimization produce better results. Of course, it also demonstrated that the optimization phase is less successful as the data sets become more constrained, as it is less able to find better solutions.

We also discovered through our set of experiments that the Least Saturation Degree method, by itself, found better solutions than even our reduced Carter data set that used random initialization experiment did in the case of the CAR-F-92 data set, though not by much, but 27.1% of the penalty value was due to first order conflicts. Therefore, the PSO algorithm does get one close to existing published values, but at a cost.

In addition to the above findings, we derived a better way to fabricate examination timetabling data sets and introduced a new optimization approach referred to as PSO-NoConflicts method.

Our new method for fabricating examination timetabling data sets produces values more representative of real world examination timetabling data sets. This approach not only permitted us to construct data sets spanning a wide range of sizes and densities, but it allowed us to have confidence in the results of experiments using these sets.

The newly derived PSO-NoConflicts algorithm permits the PSO algorithm to perform searches while still satisfying constraints. This approach shows promise and its ability to handle other constraint satisfaction problems is worth investigating further.

Finally, we knew from the outset that finding solutions to the Carter data sets was a difficult problem. We therefore did not expect this study to show that the PSO algorithm was best or better than other approaches, but instead, the goal was to provide insight into its ability to tackle the truly large, highly multi-modal, and disconnected

search spaces of the single-objective university examination timetabling problem, which is what we accomplished.

Reference List

Abido, M. A. (2001). Particle Swarm Optimization for Multimachine Power System Stabilizer Design. *Power Engineering Society Summer Meeting, 3,* 1346-2001.

Al-kazemi, B., & Mohan, C. K. (2002a). Multi-Phase Discrete Particle Swarm Optimization. *JCIS 2002 Proceedings of the 6th Joint Conference on Information Science* (622-625). Association for Intelligent Machinery, Inc.

Al-kazemi, B., & Mohan, C. K. (2002b). Multi-Phase Generalization of the Particle Swarm Optimization Algorithm. *CEC2002 Proceedings of the 2002 Congress on Evolutionary Computation* (489-494). Piscataway, NJ, USA: IEEE Press.

Angeline, P. J. (1998). Evolutionary Optimization Versus Particle Swarm Optimization: Philosophy and Performance Differences. *Evolutionary Programming VII: Proceedings of the Seventh Annual Conference on Evolutionary Programming* 601-610. San Diego: Springer.

Barr, R. S., Golden, B. L., Kelly, J. P., Resende, M. G., & Stewart, W. R. (1995). Designing and Reporting on Computational Experiments with Heuristic Methods. *Journal of Heuristics, 1,* 9-32.

Bartz-Beielstein, T. (2003). Experimental Analysis of Evolution Strategies - Overview and Comprehensive Introduction. (Reihe CI 157/03, SFB 531), Universität Dortmund.

Beielstein, T., Parsopoulos, K. E., & Vrahatis, M. N. (2001). Tuning PSO Parameters Through Sensitivity Analysis. *Reihe Computational Intelligence CI 124/02, Collaborative Research Center* (CI-124/01), University of Dortmund.

Blackwell, T. M. (2003a). Particle Swarms and Population Diversity I Analysis. *GECCO 2003: Proceedings of the Bird of a Feather Workshops, Genetic and Evolutionary Computation Conference* 103-107.

Blackwell, T. M. (2003b). Particle Swarms and Population Diversity II Experiments. *GECCO 2003: Proceedings of the Bird of a Feather Workshops, Genetic and Evolutionary Computation Conference* 108-112.

Burke, E. K., Bykov, Y., Newall, J. P., & Petrovic, S. (in press). A Time-Predefined Local Search Approach to Exam Timetabling Problems. *Accepted for publication in IIE Transactions on Operations Engineering.*

Burke, E. K., Bykov, Y., & Petrovic, S. (2001). A Multicriteria Approach to Examination Timetabling. *The Practice and Theory of Automated Timetabling III, Lecture Notes in Computer Science: 2079. Lecture Notes in Computer Science* (118-131). Berlin: Springer-Verlag.

Burke, E. K., Eckersley, A., McCollum, B., Petrovic, S., & Qu, R. (2003a). Similarity Measures for Exam Timetabling Problems. *1st Multidisciplinary Intl. Conf. on Scheduling: Theory and Applications (MISTA 2003)* Nottingham, UK:.

Burke, E. K., Eckersley, A., McCollum, B., Petrovic, S., & Qu, R. (2003b). Using Simulated Annealing to Study Behaviour of Various Exam Timetabling Data Sets. *Accepted by The 4th Metaheurisitcs International Conference (MIC2003)* Kyoto, Japan:.

Burke, E. K., Elliman, D. G., Ford, P., & Weare, R. F. (1995). Examination Timetabling in British Universities - A Survey. *Proc. of the 1st Int. Conf. on the Practice and Theory of Automated Timetabling* 76-90. Napier University, Edinburgh: ICPTAT'95.

Burke, E. K., Elliman, D. G., & Weare, R. F. (1993). Extensions to a University Exam Timetabling System. *Proceedings of the IJCAI-93 workshop on Knowledge-Based Production, Planning, Scheduling and Control* Chambery, France:.

Burke, E. K., Elliman, D. G., & Weare, R. F. (1994). A University Timetabling System based on Graph Colouring and Constraint Manipulation. *Journal of Research on Computing in Education, 27*(1), 1-18.

Burke, E. K., & Newall, J. P. (1999). A Multi-Stage Evolutionary Algorithm for the Timetable Problem. *IEEE Transactions on Evolutionary Computation, 3*(1), 63-74.

Burke, E. K., & Newall, J. P. (2004). Solving Examination Timetabling Problems through Adaptation of Heuristic Orderings. *Annals of Operations Research, 129*(1-4), 107-134.

Burke, E. K., Newall, J. P., & Weare, R. F. (1995). A Memetic Algorithm for University Exam Timetabling. *Proceedings of the First International Conference on the Practice and Theory of Automated Timetabling (ICPTAT-95): 1153. Lecture Notes in Computer Science* (241-250). Napier University, Edinburgh: Springer 1996.

Burke, E. K., Newall, J. P., & Weare, R. F. (1998a). A Simple Heuristically Guided Search for the Timetable Problem. *International ICSC Symposium on Engineering of Intelligent Systems (EIS '98)* 574-579. Canada/Switzerland: Academic Press.

Burke, E. K., Newall, J. P., & Weare, R. F. (1998b). Initialization Strategies and Diversity in Evolutionary Timetabling. *Evolutionary Computation, 6*(1), 81-103.

Burke, E. K., & Petrovic, S. (2002). Recent Research Trends in Automated Timetabling. *European Journal of Operational Research (EJOR), 140*(2), 266-280.

Burke, E. K., Petrovic, S., & Qu, R. (2002). Case-Based Heuristic Selection for Exam Timetabling. *Proceedings of ICONIP-SEAL-FSKD'02*.

Burke, E. K., Petrovic, S., & Qu, R. (2004). *Hybrid Graph Heuristics in Hyper-Heuristics Applied to Exam Timetabling Problems*. (Technical Report NOTTCS-TR-2004-1), University of Nottingham: School of CSiT.

Bykov, Y. (2003). *Time-Predefined and Trajectory-Based Search: Single and Multiobjective Approaches to Exam Timetabling.* Ph.D., University of Nottingham, UK.

Carlisle, A., & Dozier, G. (2001). An Off-The-Shelf PSO. *Proceedings of the Workshop on Particle Swarm Optimization* Indianapolis, IN: Purdue School of Engineering and Technology, IUPUI.

Carlisle, A., & Dozier, G. (2002). Tracking Changing Extrema with Adaptive Particle Swarm Optimizer. *ISSCI, 2002 World Automation Congress* Orlando, Florida, USA:.

Carter, M. W. (1986). A Survey of Practical Applications of Examination Timetabling Algorithms. *Operations Research, 34*(2), 193-202.

Carter, M. W., & Johnson, D. G. (2001). Extended clique initialisation in examination timetabling. *Journal of the Operational Research Society, 52*(5), 538-544.

Carter, M. W., & Laporte, G. (1996). Recent Developments in Practical Examination Timetabling. In Edmund Burke & Mike Carter (Eds.), *Proceedings of the Second International Conference on the Practice and Theory of Automated Timetabling (PATAT-95): 1153. Lecture Notes in Computer Science* (3-21). Berlin Heidelberg New York: Springer-Verlag.

Carter, M. W., Laporte, G., & Chinneck, J. W. (1994). A General Examination Scheduling System. *Interfaces, 24*(3), 109-120.

Carter, M. W., Laporte, & Lee, S. Y. (1996). Examination Timetabling: Algorithmic Strategies and Applications. *Journal of Operational Research Society, 47*(3), 373-383.

Clerc, M. (1999). The Swarm and the Queen: Towards a Deterministic and Adaptive Particle Swarm Optimization. *Proc. Congress on Evolutionary Computation* 1951-1957. Washington D.C:.

Clerc, M. (2002a). *Think Locally, Act Locally -- A Framework for Adaptive Particle Swarm Optimizers*. (Manuscript submitted for publication)

Clerc, M. (2002b). *TRIBES -- A Parameter Free Particle Swarm Optimizer*. Manuscript in preparation.

Colorni, A., Dorigo, M., & Maniezzo, V. (1998). Metaheuristics for High-School Timetabling. *Computational Optimization and Applications Journal, 9*(3), 275-298.

Connolly, B. (2004, March 1). *Random Sampling in T-SQL*. Last Retrieved June 21, 2005, from http://msdn.microsoft.com/ library/en-us/dnsqlpro04/html/sp04c1.asp

Cooper, T. B., & Kingston, J. H. (1995). The Complexity of Timetable Construction Problems. In E. Burke & P. Ross (Eds.), *Proceedings of the 1st International Conference on the Practice and Theory of Automated Timetabling PATAT '95: 1153. Lecture Notes in Computer Science* (283-295). Springer-Verlag.

Corne, D., Fang, H. L., & Mellish, C. (1993). Solving the Modular Exam Scheduling Problem with Genetic Algorithms. In Chung, Lovegrove, & Ali (Eds.), *Proceedings of the Sixth International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems* 370-373. Gordon and Breach Science Publishers.

Di Gaspero, L., & Schaerf, A. (2000). Tabu Search Techniques for Examination Timetabling. *Proc. of the 3rd Int. Conf. on the Practice and Theory of Automated Timetabling* 176-179.

Diestel, R. (2000). *Graph Theory*. New York: Springer-Verlag.

Dolan, E. D., & Moré, J. J. (2002). Benchmarking Optimization Software with Performance Profiles. *Mathematical Programming, 91*(2), 201-213.

Elmohamed, S., Fox, G., & Coddington, P. (1997). A Comparison of Annealing Techniques for Academic Course Scheduling. In Edmund Burke & Mike Carter (Eds.), *Practice and Theory of Automated Timetabling II, Selected Papers from the 2nd International Conference, PATAT'97* 146-166. University of Toronto: Springer.

Erben, W. (2001). A Grouping Genetic Algorithm for Graph Colouring and Exam Timetabling. *The Practice and Theory of Automated Timetabling III: 2079. Lecture Notes in Computer Science* (132-158). Berlin: Springer-Verlag.

Even, S., Itai, A., & Shamir, A. (1976). On the Complexity of Timetable and Multicommodity Flow Problems. *SIAM Journal on Computing, 5*(4), 691-703.

Fang, H. L. (1994). Genetic Algorithms in Timetabling and Scheduling. (PhD Thesis, Department of Artificial Intelligence. University of Edinburgh, Scotland, 1994). *PT9410*.

Fieldsend, J. E., & Singh, S. (2002). A Multi-Objective Algorithm based upon Particle Swarm Optimisation, an Efficient Data Structure and Turbulence. *Proceedings of the 2002 U.K. Workshop on Computational Intelligence* 37-44. Birmingham, UK:.

Gotlieb, C. (1962). The construction of class-teacher timetables. *Proc. IFIP Congress 62*, Munich, North Holland, Amsterdam, 73-77.

Greenberg, H. J. (1990). Computational Testing: Why, How and How Much. *ORSA Journal on Computing, 2*(1), 7-11.

Hansen, M. P., & Vidal, R. V. (1995). Planning of High School Examinations in Denmark. *European Journal of Operational Research, 87,* 519-534.

Hooker, J. N. (1996). Testing Heuristics: We Have It All Wrong. *Journal of Heuristics,* 32-42.

Hu, X., & Eberhart, R. C. (2002). Adaptive Particle Swarm Optimization: Detection and Response to Dynamic Systems. *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2002)* 1666-1670. Hawaii, USA:.

Junginger, W. (1986). Timetabling in Germany -- a Survey. *Interfaces, 16,* 66-74.

Kennedy, J. (1998). The Behavior of Particles. In V. W. Porto, N. Saravanan, D. Waagen, & A. E. Eiben (Eds.), *: 1447. Lecture Notes in Computer Science* (581-589). Berlin: Springer.

Kennedy, J., & Eberhart, R. C. (1995). Particle Swarm Optimization. *Proceedings of the 1995 IEEE International Conference on Neural Networks* 1942-1948. Piscataway, NJ: IEEE Service Center.

Kennedy, J., & Eberhart, R. C. (1997). A Discrete Binary Version of the Particle Swarm Algorithm. *Proceedings of the 1997 Conference on Systems, Man, and Cybernetics* 4104-4108. Piscataway, NJ: IEEE Service Center.

Kennedy, J., & Eberhart, R. C. (2001). *Swarm Intelligence* (Denise E.M. Penrose). San Francisco: Morgan Kaufmann Publishers.

Kennedy, J., & Spears, W. M. (1998). Matching Algorithms to Problems: An Experimental Test of the Particle Swarm and Some Genetic Algorithms on the Multimodal Problem Generator. *Proc. 1998 IEEE World Congress on Computational Intelligence* 74-77. Anchorage: Alaska: IEEE.

Kiaer, L., & Yellen, J. (1992). Weighted Graphs and University Course Timetabling. *Computers and Operations Research, 19*(1), 59-67.

Laporte, G., & Desroches, S. (1984). Examination Timetabling by Computer. *Computers & Operations Research, 11*(4), 351-360.

Laskari, E. C., Parsopoulos, K. E., & Vrahatis, M. N. (2002). Particle Swarm Optimization for Integer Programming. *Proceedings of the 2002 IEEE Congress on Evolutionary Computation* 1582-1587.

Lim, A., Chin, A. J., Kit, H. W., & Oon, W. C. (2000). A Campus-Wide University Examination Timetabling Application. *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence* 1020-1025. Austin, Texas, USA: AAAI Press / The MIT Press.

Merlot, L., Boland, N., Hughes, B., & Stuckey, P. J. (2002). A Hybrid Algorithm for the Examination Timetabling Problem. *Proceedings of the 4th international conference on the Practice and Theory of Automated Timetabling (PATAT2002) Lecture Notes in Computer Science*. Gent, Belgium: Springer-Verlag.

Newall, J. P. (1999). *Hybrid Methods for Automated Timetabling.* PhD Thesis, Department of Computer Science, University of Nottingham, UK.

Novick, A. (2003, April 8). *Retrieving a Random Record in SQL Server using udf_Num_RanInt.* Last Retrieved june 21, 2005, from http://www.novicksoftware.com/UDFofWeek/Vol1/T-SQL-UDF-Volume-1-Number-21-udf_Num_RandInt.htm

Operations Research Group. (13 October 2003). Timetabling Problem Database. Retrieved March 11, 2004, from The University of Melbourne, Department of Mathematics and Statistics Web site: http://www.or.ms.unimelb.edu.au/timetabling/

Papadimitriou, C. H., & Steiglitz, K. (1998). *Combinatorial Optimization: Algorithms and Complexity* (Unabridged). Mineola, New York: Dover Publications, Inc. (1982, Prentice Hall, Inc., New Jersey)

Paquet, U., & Engelbrecht, A. P. (2003). A New Particle Swarm Optimiser for Linearly Constrained Optimisation. *IEEE Congress on Evolutionary Computation* 227-233. Canberra, Australia: IEEE.

Parsopoulos, K. E., Laskari, E. C., & Vrahatis M.N. (2001). Solving L1 norm errors-in-variables problems using Particle Swarm Optimizer. In M.H. Hamza (Ed.), *Artificial Intelligence and Applications* 185-190. Anaheim, CA, USA: IASTED/ACTA Press.

Parsopoulos, K. E., & Vrahatis, M. N. (2002a). Initializing the Particle Swarm Optimizer Using the Nonlinear Simplex Method. *Advances in Intelligent Systems, Fuzzy Systems, Evolutionary Computation,* 216-221.

Parsopoulos, K. E., & Vrahatis, M. N. (2002b). Particle Swarm Optimization Method for Constrained Optimization Problems. In P. Sincak V. Kvasnicka, & J. Vascak (Eds.), *Intelligent Technologies - Theory and Applications: New Trends in Intelligent Technologies* (Frontiers in Artificial Intelligence and Applications, 76, 214-220). IOS Press.

Parsopoulos, K. E., & Vrahatis, M. N. (2002c). Particle Swarm Optimization Method in Multiobjective Problems. *Proceedings of the 2002 ACM Symposium on Applied Computing (SAC 2002)* 603-607. Madrid, Spain: SAC 2002.

Parsopoulos, K. E., & Vrahatis, M. N. (2002d). Recent approaches to global optimization problems through Particle Swarm Optimization. *Natural Computing, 1*(2-3), 235-306.

Peer, E. S., Engelbrecht, A. P., & van den Bergh, F. (2003). CIRG@UP OptiBench: A Statistically Sound Framework for Benchmarking Optimisation Algorithms. *IEEE Congress on Evolutionary Computation* 2386-2392. Canberra, Australia: IEEE.

Peram, T., Veeramachaneni, K., & Mohan, C. (2003). Fitness-Distance Ratio Based Particle Swarm Optimization. *IEEE Swarm Intelligence Symposium* Indianapolis, Indiana: IEEE.

Petrovic, S., Yang, Y., & Dror, M. (2003). Case-Based Initialisation of Metaheuristics for Examination Timetabling. *1st Multidisci-plinary Intl. Conf. on Scheduling: Theory and Applications (MISTA 2003)* Nottingham, UK:.

Rardin, R. L. (1998). *Optimization in Operations Research*. Upper Saddle River, New Jersey: Prentice Hall.

Ray, T., & Liew, K. M. (2002). A Swarm Metaphor for Multiobjective Design Optimization. *Engineering Optimization, 34*(2), 141-153.

Reis, L. P., & Oliveira, E. (2001). A Language for Specifying Complete Timetabling Problems. *Practice and Theory of Automated Timetabling III, Third International Conference, PATAT 2000: 2079. Lecture Notes in Computer Science* (322-341). Konstanz, Germany: Springer.

Riget, J., & Vesterstrøm, J. S. (2002). A Diversity-Guided Particle Swarm Optimizer -- the ARPSO. (EVALife Technical report no. 2002-02), Dept. of Computer Science, University of Aarhus, Denmark: EVALife Project Group.

Robinson, J., Sinton, S., & Rahmat-Samii, Y. (2002). Particle Swarm, Genetic Algorithms, and their Hybrids: Optimization of a Profiled Corrugated Horn Antenna. *IEEE International Symposium on Antennas & Propagation* San Antonio, Texas:.

Ross, P. M., Hart, E., & Corne, D. (1998). Some Observations about GA-Based Exam Timetabling. In E. Burke & M. Carter (Eds.), *Practice and Theory of Automated Timetabling II: 1408. Lecture Notes in Computer Science* (115-129). Springer-Verlag, Berlin.

Salman, A., Ahmad, I., & Al-Madani, S. (2002). Particle Swarm Optimization for Task Assignment Problem. *Microprocessors and Microsystems, 26,* 363-371.

Schaerf, A. (1996). Tabu Search Techniques for Large High-School Timetabling Problems. *13th National Conference on Artificial Intelligence* 363-368. Portland, USA: AAAI Press/MIT Press.

Schaerf, A. (1999). A Survey of Automated Timetabling. *Artificial Intelligence Review, 13,* 87-127.

Shi, Y., & Eberhart, R. C. (1998). A Modified Particle Swarm Optimizer. *Proceedings of the IEEE International Conference on Evolutionary Computation* 69-73. Piscataway, NJ: IEEE Press.

Shi, Y., & Eberhart, R. C. (1999). Empirical Study of Particle Swarm Optimization. *Proceedings of the 1999 Congress on Evolutionary Computation* 1945-1950. Piscataway, NJ: IEEE Service Center.

Silva, A. F., Silva, A. P., & Costa, E. (2002). Chasing the Swarm: A Predator-Prey Approach to Function Optimisation. *Proc. of the Mendel 2002 - 8th International Conference on Soft Computing* 103-110. Brno, Czech Republic: Mendel 2002.

Terashima-Marín, H., Ross, P., & Valenzuela-Rendón, M. (1999). Evolution of Constraint Satisfaction Strategies in Examination Timetabling. *Proceedings of the Genetic and Evolutionary Conference* 635-642. Orlando, FL:.

Thomson, J. M., & Dowsland, K. A. (1996). Variants of simulated annealing for the examination timetabling problem. *Annals of Operations Research, 63,* 105-128.

Thomson, J. M., & Dowsland, K. A. (1998). A Robust Simulated Annealing Based Examination Timetabling System. *Computers Operational Research, 25*(7/8), 637-648.

Tsou, D., & MacNish, C. (2003). Adaptive Particle Swarm Optimisation for High-Dimensional Highly Convex Search Spaces. *Proceedings of IEEE Congress on Evolutionary Computation* (CEC 2003) 783-789, Canbella, Australia.

Welsh, D. J., & Powell, M. B. (1967). An upper bound for the chromatic number of a graph and its application to timetabling problems. *The Computer Journal, 10*(1), 85-86.

Willemen, R. J. (2002). *School timetable construction algorithms and complexity.* PhD Thesis, Technische Universiteit Eindhoven.

van den Bergh, F. (2001, November). *An Analysis of Particle Swarm Optimizers.* PhD Thesis, University of Pretoria.

van den Bergh, F., & Engelbrecht, A. P. (2002). A New Locally Convergent Particle Swarm Optimizer. *IEEE Conference on Systems, Man, and Cybernetics*.

Veeramachaneni, K., Peram, T., Mohan, C. K., & Osadciw, L. A. (2003). Optimization Using Particle Swarms with Near Neighbor Interactions. *Lecture Notes in Computer Science, 2723,* 110-121. Heidelberg: Springer-Verlag.

Vesterstrøm, J. S., & Riget, J. (2002). *Particle Swarms: Extensions for Improved Local, Multi-modal, and Dynamic Search in Numerical Optimization.* Masters Thesis.

Zhaohui, F., & Lim, A. (2000). Heuristics for the Exam Scheduling Problem. Proceedings of the 12th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'00) 172-175. IEEE.