On the Performance of Recent Swarm based Metaheuristics for the Traveling Tournament Problem

by

Sandile Saul

Submitted in fulfilment of the academic requirements
for the degree of Master of Science in the
School of Mathematics, Statistics and Computer Science
University of KwaZulu-Natal

Durban, South Africa, 2013

UNIVERSITY OF KWAZULU-NATAL

COLLEGE OF AGRICULTURE, ENGINEERING AND SCIENCE

DECLARATION

The research described in this thesis was performed at the University of KwaZulu-Natal under the supervision of Dr. A. O. Adewumi. I hereby declare that all materials incorporated in this thesis is my own original work except where acknowledgement is made by name or in the form of a reference. The work contained herein has not been submitted in part or whole for a degree at any other university.

Signed:

Sandile Sinethemba Saul

Date: November 2013

As the candidate's supervisor, I have approved the dissertation for submission

Signed:

Dr. A. O. Adewumi

Date: November 2013

UNIVERSITY OF KWAZULU-NATAL

COLLEGE OF AGRICULTURE, ENGINEERING AND SCIENCE

DECLARATION – PLAGIARISM

I, Sandile Sinethemba Saul declare that

1. The research reported in this thesis, except where otherwise indicated, is my original research.
2. This thesis has not been submitted for any degree or examination at any other University.
3. This thesis does not contain other persons' data, pictures, graphs or other information, unless specifically acknowledged as being sourced from other persons.
4. This thesis does not contain other persons' writing, unless specifically acknowledged as being sourced from other researchers. Where other written sources have been quoted, then:
    a. Their words have been re-written but the general information attributed to them has been referenced
    b. Where their exact words have been used, then their writing has been placed in italics and inside quotation marks, and referenced.
5. This thesis does not contain text, graphics or tables copied and pasted from the Internet, unless specifically acknowledged, and the source being detailed in the thesis and in the References sections.

Signed:

Sandile Sinethemba Saul

Contents

List of Tables

List of Figures

Abstract

For a number of years, researches on optimization have investigated and explored different methods that can uncover or provide optimal solutions to a number of problems in the area of timetabling and scheduling. For decades the scheduling of sport tournaments has been done manually through years of experience, however the manual process takes a substantial amount of time and to automate this process, a different set of heuristics have been applied to produce the schedules in a significantly less amount of time. This study focuses on the performance of the Swarm Intelligence (SI) metaheuristics on the Traveling Tournament Problem (TTP).

TTP is a timetabling problem in sports which abstracts the issues that are important in creating timetables where traveling is a significant issue. TTP is a combinatorial optimization problem that involves devising a schedule that allows teams in a league or tournament play each other twice while minimizing the traveling distance. Scheduling of professional sports is one of many researched practical problems in combinatorial optimization. The scheduling of professional sports is a known NP-Hard problem which is very difficult to solve as it involves multiple constraints.

Various methods of solving the problem have been implemented. Some methods such as integer and linear programming are only effective for small instances of the problem, and metaheuristics have been successfully applied to the problem, even for larger instances of the problem. Five Swarm Intelligence (SI) metaheuristics are implemented in this study to address the problem and the algorithms are applied to two different data instances which are used frequently in literature, with a maximum of 16 teams. Results obtained are compared with previous results obtained in literature. The algorithms obtained good solutions for small instances, and some of them proved to be very competitive for large instances.

Acknowledgements

I would like to extend my greatest gratitude to my supervisor Dr A.O Adewumi for his guidance throughout this research study and for his genuine interest in this research study without which this study would have remained incomplete. To my mother, for the struggles you went through trying to provide a better life for my two sisters and I, and for teaching us the value of education. To my two sisters, for your support in every decision I have made regarding my education, and for providing me with the necessary resources to ensure that I perform well academically. To Mr S. Bangura, for taking time from your busy schedule to provide extra tutorials on computer programming,of which I benefited a great deal and that contributed significantly to this research study. And lastly to the University of KwaZulu-Natal for granting me the opportunity to further my studies.

Dedication

This research study is dedicated to my family, and to all the people who played a role in facilitating my education. This is especially dedicated to my late father who passed away when I was still very young and never got the chance to see me complete my tertiary education.

Outcome of Research

Conference Papers

S. Saul and A. Adewumi. "*An artificial bees colony algorithm for the Traveling Tournament Problem*", Proceedings of the 41st Annual Conference of the Operations Research Society of South Africa (ORSSA), pp. 10-19 , 16-19 September 2012, Aloe Ridge Hotel, Muldersdrift, South Africa.

Journal Papers

Our article titled "*An improved Cuckoo Search algorithm for the Traveling Tournament Problem"* was submitted for publication in the Orion journal which is the official journal of the Operations Research Society of South Africa (ORSSA), and it is still under review.

Chapter One

Introduction

Scheduling is one of the many researched practical problems in combinatorial optimization. Scheduling consists of developing a timetable of events within a specified time frame and given a set of resources such that no one of the events conflict with one another or violate any constraints [1]. Some real world scheduling problems include job shop scheduling, school examination timetable and scheduling processes in a computer operating system.

For many years the design of professional sports schedules have been done manually through a combination of adherence to tradition and years of experience [2]. However there have been several efforts to understand and examine the goals and constraints of sport scheduling, most of these constraints have focused on the difficulty of producing a feasible schedule. The manual process takes a substantial amount of time, and the final result is usually far from the expected optimal schedule [1]. To eliminate the inefficient practice of manual scheduling, researchers have tried to automate the task of scheduling by applying a set of heuristics that produce schedules in significantly less time than a manual approach.

The organization of sports is becoming more complex as professional sports leagues now consist of many teams playing long schedules, and thus the creation of sports league schedules have become a difficult task. Beyond the complexity of creating a schedule to be contained within a specified time period, there are also the issues

of venue availability, travel time and cost, fairness in schedules between the teams, along with a multitude of other constraints and desired goals [3]. Devising a schedule that meets all of a league administrator's criteria regarding when and where games should be played is an NP-Hard optimization problem *i.e.* there is no known polynomial time algorithm capable of finding a solution that combines the given resources in an optimal way without violating constraints, because as more teams are added to a league, the search space of possible schedules grows exponentially.

Sports scheduling has attracted a lot of interest from different research communities such as the Operations Research (OR) and Artificial Intelligence (AI) communities. There are various important or essential aspects to be considered in determining the best schedule, in some situations one seeks for a schedule that minimizes the total distance traveled and in some situations one seeks for a schedule that attempts to minimize the number of breaks [4]. The former will be the focus of this thesis. A solution to the Traveling Tournament Problem (TTP) is a Double Round-Robin Tournament (DRRT) that minimizes the total traveling distance amongst all the teams and satisfies feasibility constraints. Because the total traveling distance is an immense issue for every team taking part in the tournament, solving the TTP may be a base for the solution of real timetabling applications in sports. Schedules with minimum traveling times are of major interest to leagues, teams, fans and sponsors [4].

Several researchers have addressed the problem of scheduling tournaments in various sports and leagues using different approaches such as Constraint Programming (CP) or Integer Programming (IP), local search and population based algorithms. The TTP has a combination of common features from the Traveling Salesman Problem (TSP) [5] and tournament scheduling problem, thus it puts forward strong issues of feasibility together with a optimization part that is complex (distance traveled) [6]. From both the theoretical and practical side it is considered an interesting problem. Due to the complexity of the TTP, it has proved to be difficult to solve by means of exact methods even for very small problem instances, even

by employing advanced combinatorial optimization techniques. For instance, its solution by either CP or IP has so far been made possible for problem instances for up to only six teams [7].

1.1 Metaheuristics

Metaheuristics are designed and developed to solve optimization problems that are complex, where other relevant optimization techniques have either been inefficient or ineffective [8]. Metaheuristics are now recognized as one of the best approaches that are practical for tackling complex problems, especially for numerous real world problems which are combinatorial in nature. General applicability and effectiveness are the beneficial properties of metaheuristics. Previously, specific heuristics were usually developed to tackle complex optimization problems which are combinatorial. The need for a new or different approach to each problem arose as whatever was learned from one problem was not always generalized to a class of different problems. With the emergence of strategies that provide a general solution like metaheuristics such as Simulated Annealing (SA), Genetic Algorithm (GA) *etc.*, the only challenge one faces currently is adapting the metaheuristics to a specific problem, and this requires minimal effort compared to developing a specific heuristic for a certain application. A good implementation of a metaheuristic will probably provide a close to optimal solution in moderate computational time.

The application of metaheuristics as a chosen method over other optimization techniques/methods is mainly to enable researchers find good solutions to optimization problems that are complex having numerous local optima and little structure that is essential to lead the search. Using the metaheuristic approach to solve problems begins by obtaining an initial solution or sets of initial solutions and by introducing an improved search which is led by certain principles with the search structure having many elements which are common across different methods. There is always a set of solutions or a solution θ_k that denotes the current state in every step of

the search algorithm. Metaheuristics are classified into solution-to-solution search methods such as SA and Tabu Search (TS) and set based search methods such as GA *etc.* The former means that θ_k is a single solution or point $\theta_k \in \Theta$ in a solution space, and the latter means that in every step θ_k represents a set of solutions $\theta_k \subseteq \Theta$ [8]. Nevertheless, regardless of whether it is a solution-to-solution or set based metaheuristic, the fundamental structure of the search stays the same.

"*Given a neighborhood $N(\theta_k)$ of the solution (set), a candidate solution (set) $\theta^c \subset N(\theta_k)$ is selected and evaluated. This evaluation involves calculating or estimating the performance of the candidate solution(s) and comparing them with the performance of θ_k and sometimes with each other. Based on this evaluation, the candidate solution may either be accepted, in which case $\theta_{k+1} = \theta^c$, or rejected, in which case $\theta_{k+1} = \theta_k$ [8].*"

1.2 Swarm Intelligence

The main focus of this study is on Swarm Intelligence (SI) metaheuristics and their performance compared to other optimization algorithms in literature, in this section the background of SI is discussed.

For a number of years numerous biological systems have tackled complex problems by sharing information with other members of the group. By studying and understanding the fundamental individual behaviours and collective behaviour of insects, one will have an understanding of the fundamental mechanisms of SI. In this section insects will used to illustrate the formation of patterns, the making of collective decisions and how a large group of insects are able to move together as one.

1.2.1 Decision Making

"*The evolution of sociality, the phenomenon where individuals live together within a nest such as is found in many bees and wasps, and all ants and termites, has created the need for information transfer among group members. No longer can each individual simply behave as if solitary, but actions by different group members need to be carefully tuned to achieve adaptive behaviour at the level of the whole group [9]*". Collective decisions need to be made by the colony of insects, decisions such as where to forage, which nest to move to, the right time to reproduce and the delegation of tasks according to resource availability or work force. Individuals act mainly on information obtained locally by interacting with their peers and the immediate environment, and the group level decisions are the result of this behaviour. Ants and honeybees will be used to show how this is achieved by insect colonies.

1.2.1.1 Where to Forage?

A form of recruitment needs to be performed or undergone by social insects in order to organize foraging. This enables insects to efficiently forage in an environment with food sources that are distributed intermittently/sporadically or too large for a single individual to exploit. Social insects that can pass information about the location of good food sources have the ability to exploit an area much more than those that lack a mechanism for recruitment as sophisticated as this one. Honeybees are an example of insects with such recruitment mechanism, their dance language which is very sophisticated enables them to forage food sources which are far from their colony; as far as 10 kilometers. Among social insects, recruitment mechanisms are very different, but can be split into two classes: direct and indirect mechanisms. One good example of an indirect mechanism is the use of chemical trails for mass recruitment. There is no physical contact between the recruitee and recruits, instead communication is via the modulation of the environment; the trail [9]. On its way back from a good food source, the recruiter deposits pheromone and

this trail is simply followed by the recruits. One can think of this mechanism being similar to broadcasting in a way, broadcasts have no control over who receives the broadcasted information. The honeybees' dance language is the best known example of indirect recruitment. The dance encodes information about the distance and direction of the food source, and is performed by successful foragers (recruiters). Potential recruits will try to locate the food source being advertised by extracting the information passed to them via the dance.

A double bridge experiment, which a lot of computer scientists are quite familiar with, is an example of the organization of foraging in ant colonies. In this experiment, a colony of ants laying trails are offered two equal food sources which are located at the end of two paths with different lengths. The shorter path is eventually chosen by the majority of foragers after some time. This is due to the positive feedback process; on their way back to the nest, ants that find food mark the environment with pheromone trails, and probabilistically these trails are followed by ants searching for food. This trail following behaviour enables ant colony to select the best food source out of many possible food sources without the individual ants having to compare the quality of food sources available.

Several experiments performed on different species of ants have illustrated that ants modulate the amount of pheromone deposited depending on the quality of the food source, the more pheromone is left, the more likely other ants are to follow the trail to the best food source. "The success of the pheromone trail mechanism is likely to be due, at least in part, to the non-linear response of ants to pheromone trails where, for example, the distance at which an ant follows a trail before leaving it, is a saturating function of the concentration of the pheromone. In other words, the probability an ant will follow a trail is a function of the trail strength (expressed as concentration of pheromone), However ants never have a zero probability of losing a trail, irrespective of the strength of the trail [9]". Mathematically, non-linearity in response means an increase in the number and complexity of solutions of the model equations which may be thought of as underlying foraging.

Biologically, the distribution of ants between food sources corresponds to a differential equation solution, and an increase in solutions means more flexibility as the ants make a selection between different possible food sources or solutions. Provided that the food source has unlimited capacity, this allocation of workers among food sources, which basically assigns to the best food source all trail following foragers is optimal. If a food source has unlimited capacity, trail following ants will be directed to it and they cannot feed from it. The ant colony gets stuck in a suboptimal solution in a way and the only way to get out, is by introducing some layers of complexity which basically signals to the ants to avoid such a food source, this is known as negative pheromone in ant colony. Another drawback of pheromone reliance is that competing with an existing trail may be difficult, even if a good food source is found. If due to conditions that existed initially, a food source which is mediocre is found first, ants that have discovered a better food source after the establishment of the first trail will not be able to build up a trail that is strong enough to attract nest mates to the newly found food source and thus again the ants are stuck in a sub-optimal solution.

Honeybees do not get stuck in sub-optimal solutions because their recruitment mechanism is fundamentally different. The direct mechanism encodes two important pieces of information; the distance and the direction to the target. Both are necessary as honeybees have to deal with a three dimensional space unlike ants. The dancer shakes her body from side to side vigorously and strides forward about 1.5 times her length during the typical dance, and this is known as the waggle phase of the dance. The bee makes an abrupt turn to the right or left after this phase, circling back to start this phase again and this is known as the return phase. After doing the waggle phase for the second time, the bee turns in the opposite direction so that the figure eight pattern of the waggle dance will be traced with every second circuit of the dance [9]. The waggle phase is the phase that is the most information rich. In the duration of the waggle phase, distance information is encoded. For nearby targets, dances for the waggle phase are short and longer for distant targets. For dance followers to be able to decode the direction information

they need to be in close contact with the dancer and thus only limited individuals receive this information. Furthermore, more than one dance can take place simultaneously, and these dances can either be for the same or different sites. The assumption is that the number of followers that read a dance are infinite, so there is no direct competition between dances and these dances are only performed for very good food sources.

For a forager to determine the quality of the food source encountered, it uses an internal gauge to evaluate the profitability of the source, based nectar sugar content and the distance of the source from the colony, as well as how easy nectar can be collected. The nervous system of a bee is calibrated with a threshold which it uses to measure these variables when deciding whether it is worth foraging for a particular patch and whether it is worth advertising that patch to other workers. Dancers adjust both the vigour and duration of their dance depending on the profitability of their current food source. The number of waggle phases performed by the dancer in a certain dance, measures the duration of the dance and the time interval between the waggle phases measures the vigour. More nest mates are recruited to a more profitable food source which is indicated by a larger number of waggle phases and by a smaller return phase. This basically means that when two dances are performed at the same time, one for a good site and the other for a mediocre site, dance followers are more likely to be attracted to the good site, however the mediocre site will still attract some other followers, since dances are not weighed or evaluated by the followers before making a decision as to which one to choose/follow.

1.2.1.2 Exploration vs. Exploitation

In most studies done on the allocation of foragers to food sources, the forager sites or feeders are kept constant in a stable environment. In stable conditions, from the colony's point of view, the optimal solution is to focus only on the best or good food source and the ant species do exactly this when collecting stable food

sources like leaves or honeydew. They construct trails that last for a long time, and connect the foraging locations to the nest. Due to the workers changing the environment actively by removing vegetation, trails are more or less permanent in some species. When the conditions become unstable, there has to be some sort of mechanism that allows a change over to another profitable food source, when the initial food source has been exhausted. This basically means that for these species to do well in a dynamically changing environment, the colony of insects should have some sort of repository for storing information about existing food patches that are being exploited but also allow exploration for new sites at the same time. The trade off between exploitation and exploration is key in keeping track of changing conditions.

As mentioned earlier, there is never a zero probability that ants that follow a trail would not lose it, irrespective of the trail, resulting in some ants getting lost even when the trail is at its strongest [9]. Assuming, that the ants that got lost are capable of discovering new food sources and thus assist as explorers or scouts for the colony, depending on the profitability of the food source being/already exploited, this strategy allows the number of scouts to be fine tuned. This is because, as the trail gets weaker, the number of ants that get lost increases and thus become scouts. A stronger trail will result with a less number of ants getting lost and hence a smaller number of scouts. The honeybees' regulation of scouts ensures that there is a right balance between the number of workers/individuals assigned to exploration and exploitation.

A forager that does not know where to forage will try to locate a dance to follow first, if she fails to do so due to the number of dancers being small, she will search the surroundings and leave the colony, and thus become a scout. Because of this, if many good forage sites have not been discovered by the colony, the number of scouts becomes high. This mechanism affords the colony ways to immediately adjust the number of scouts based on the available information on forage sites that are profitable. Other profitable sites that need to be exploited may still exist even

if the colony is currently exploiting profitable patches. As soon as the number of dancers decrease in the colony, there is an increase in the probability that some foragers that are not employed will not be able to locate a dancer, and for that reason the colony sends out some more scouts.

1.2.1.3 Where to Live?

Even more astonishing is that the same communication methods are frequently used to achieve a very different goal; selecting a new nest. A new home needs to be selected under two conditions; either an old nest has been ruined and the whole colony needs to move, or a new nest site is required by some part of the colony for reproductive swarming reasons *i.e.* the initial/original nest has grown to a point that some part of the colony needs to be sent off to start a new colony, this part is sent off with two or three queens. Insects exhibit similar behaviour as humans when changing homes, asking the same kind of questions: what potential homes are alternatively available? How do they compare in terms of attributes? Has enough information been collected? Social insects use the same communication mechanism as above when house hunting [9].

1.3 Purpose of Study

A lot of research has been done on the TTP using various methodologies but only a few of these methodologies are based on Swarm Intelligence (SI). The purpose of this study is to tackle the TTP using SI algorithms, apply these algorithms to two data sets that have been used quite often in literature to address this problem, and then compare the performance of these algorithms with previous results found in literature. The algorithms are Cuckoo Search (CS), Artificial Bee Colony (ABC), Bacterial Foraging (BFO), Bat and an algorithm that combines the CS, BFO and Bat.

1.4 Scope of Study

The TTP is a broad problem, there are different relevant aspects to consider in determining the best schedule. Some people seek for a schedule that attempts to minimize the total number of breaks and some seek for a schedule that minimizes the total distance traveled. In this study the focus is on the latter (minimize the total distance traveled), usually there are breaks in sports leagues or tournaments but this is not taken into consideration in this study for simplicity purposes. This study limits testing to only two data sets with a maximum of 16 teams, since these are commonly used in literature and this study only focuses on the SI.

1.5 Thesis Structure

In Chapter Two the literature review is discussed and is split into two sections; Constraint and Integer Programming, and Metaheuristics. A comparative study of the results found in literature is also undergone. Other scheduling problems tackled with SI are also explored. In Chapter Three, the methodology adopted including an overview of the TTP and the mathematical model employed are discussed and finally in Chapter Four, the experimental results and findings from the study are represented.

Chapter Two

Literature Review

In this chapter, an overview of the various approaches that have been utilised in solving the TTP is given. In section 2.1, a few Integer Programming (IP) and Constraint Programming (CP) approaches that have been used to solve the problem are discussed, in section 2.2, some metaheuristics that have been used to solve the problem are discussed and in section 2.3 a review of other scheduling problems that have been solved using swarm intelligence algorithms is given.

2.1 Constraint and Integer Programming

CP is a programming paradigm in which constraints are used to state relations between variables [10]. Constraints determine the properties of a solution to be found rather than a sequence of steps to execute and this differentiates them from the common imperative programming languages [10]. A problem must first be formulated as a Constraint Satisfaction Problem (CSP) when solving it using CP. In order to achieve this, one first introduces variables that range over set domains and define constraints for the introduced variables, the constraints can be expressed in any chosen language. Finally one can use either general or domain methods [11] to solve the representation chosen. The modeling language offered by CP is very rich, it allows various constraint types and this makes it easier to use an intuitive model

to formulate a problem and people do not need to have an indepth understanding of the underlying mechanisms for them to use the solution technique.

An IP problem "*is a mathematical optimization or feasibility program in which some or all of the variables are restricted to be integers*" [12]. IP may sometimes be referred to as Integer Linear Programming (ILP), and can also be referred to as Mixed Integer Linear Programming (MILP) when some but not all the variables are restricted to be integers. Linear Programming (LP) problems contain an objective function that is linear and the objective is to minimize or maximize while satisfying several constraints, discrete problems are modelled as IP or MILP problems in the OR community and both problems are formulated as LP problems.

Easton *et al.* [7] proposed a branch and price algorithm which uses IP for the pricing problem, they used parallel implementation for the algorithm. In [7] they used CP in addition as a primary heuristic and also used their proposed approach to solve data sets with 8 teams ($n = 8$). They proposed the following IP model:

$$Minimize \sum_{i \in P} C_i X_i \tag{2.1}$$

Where:

P : is a small set of columns that consist of the best tours for each team, $P = 1, 2, ..., k$, k is the number of time slots.

C : distances affiliated with the tours.

X_i: is a binary for $i \in P$

Subject to the following constraints:

- A single tour must be chosen for every team in the tournament.
- In each timeslot every team must play only once.

Easton *et al.* [7] also proposed a CP model that runs once for each team. [7] used variables to represent venues where games for each team will be played in each time slot. The teams in the tournament are represented by the domains of the variables. The variables also represent the distances between venues which are in sequence, but these variables are strictly used in the calculation of the objective. The variable domains are reduced in accordance with prior branching before running the CP. In the model proposed in [7], constraints are set as follows:

- The venue for each opponent must appear only once.
- The home venue must appear exactly n-1 times.
- A team must not have more than 3 home or away games consecutively.
- Non-home venues must not appear more than 3 times in sequence.
- The dual values and the total distance affiliated with the games in the tour must be less than zero.

For the approach proposed in [7], data sets with 4 teams ($n = 4$) were not difficult to solve while data sets with 6 teams were more challenging to solve and data sets with 8 teams were left unsolved.

Juan *et al.* [13] presented a new hybrid algorithm which combines constraint-based algorithm and a neighbourhood search. This is an exploration into an improved and alternative methodology to sports tournament scheduling problems with a number of special constraints. In [13] the sports tournament scheduling problem is used at the University Utara Malaysia as a case problem and thus modelled as a constraint satisfaction problem. The algorithm proposed encompasses three stages; generation of possible combination of matchups based on the partial round-robin strategy, gathering and enumeration of all matchups and assignment of teams based on the relevant constraints through constraint-based scheduling algorithm for the preliminary round. In [13] the tournament scheduling problem is modelled as CSP, the CSP consists of Constraint Network (CN): (T, S, C), where:

- A set of teams, $T = [t_1, t_2, ..., t_n]$, is allocated as the set of variables.
- A set of teams domains, $S = [S(t_1), S(t_2), ..., S(t_n)]$, where $S(t_i)$ is a finite set of possible values for team t_i. Domains in this problem are the time slots.
- A set of constraints related to teams, $C = c_1, c_2, ..., c_k$.
- The objective is to assign pairs of teams into time slots such that all constraints are satisfied.

The algorithm proposed in [13] was evaluated in terms of computational time, venue assignment and duration of break and it managed to provide feasible, efficient and quick solutions.

Rasmussen [14] modelled the TTP using an IP formulation and CP formulation. [14] formulated the problem as an IP model using a variable (binary) h_{is} (for every $i \in T$ and $s \in S$), where T and S are set of teams and time slots respectively. H_{is} is equal to 1 when team i is playing a home game in slot s and is equal to 0 when team i is playing away. A variable d_{is} (for every $i \in T$ and $s \in S^0$) which is an integer is used to calculate the total distance, which is given by the distance travelled between slot s and $s+1$ by team i. Below is the IP model proposed in [14]:

$$Minimize \sum_{i \in T} \sum_{s \in S^0} d_{is} \tag{2.2}$$

Subject to the following constraints:

- Every team starts at home and ends at home.
- The home games for every team must be exactly $n-1$.
- Each time slot consists of a match between two teams (team i_1 and i_2), only one of the teams must be a home team and the other one an away team.
- Team i must play the same opponent two times, one home game and one away game in two different slots.

Rasmussen [14] used variables similar to the ones used in the IP model to formulate the problem as a CP model which allowed them redefine the constraints and this is why their CP model differs from their IP model. Rasmussen [14] also proposed a hybrid IP/CP technique that decomposed the problem into two different phases. The first is responsible for generating feasible patterns for every team using CP and phase 2 is responsible for the assignment of teams to the generated patterns to find the optimal pattern set using IP. In [14], CP solved instances with $n = 6$ teams better than IP but could not solve instances with $n = 8$, while IP solved all the data sets with $n \leq 10$ and only one data set with $n = 10$ teams. The proposed hybrid IP/CP technique in [14] outperformed the IP and CP models, it proved to be fastest on average, but the major drawback with this hybrid is that it consumes a lot of memory when generating initial patterns.

2.2 Related Research

A metaheuristic [15] is a high-level algorithm framework which is independent from the problem and heuristic optimization algorithms are developed using a set of tactics provided. There is no positive assurance that optimal solutions will be found with metaheuristics, and for this reason, they are specifically developed to find a suitable or competent solution in the smallest computational time possible. Metaheuristics can search large spaces of possible solutions and make few or no assumptions about the problem being optimized [16].The scientific community has shown that metaheuristics are the viable and often preferred alternative to exact methods, especially for large data sets or problem instances, most of the time they are able to offer a better compromise between computing time and the quality of the solution.

There are two important ways in which metaheuristics are more flexible than exact methods. Firstly, most optimization problems in real-life have requirements in terms of the expected quality of the solution and the computing time allowed, which

can differ depending on the problem, metaheuristic algorithms can be adjusted to fit those needs. Secondly, when it comes to the formulation of the problem, metaheuristic algorithms do not place any complex or unnecessary demands [15]. In this section some of the metaheuristic algorithms that have been used for the TTP are reviewed.

Anagnostopoulos *et al.* [17] presented a Simulated Annealing (SA) algorithm for the TTP which explores both the infeasible and feasible schedules. In [17] they use a large neighbourhood and advanced methods such as *reheats* for balancing the exploration of the infeasible and feasible regions and for escaping local minima. The other advanced method they use is the strategic oscillation. The results they obtained match the best known results on smaller instances and are much better on larger instances. Hung *et al.* [18] used a Genetic Algorithm (GA) to solve the constraint satisfaction problem for sport schedules and analysed the battle combination constraints of the National Basketball Association (NBA) and National Hockey League (NHL). They simulated the NBA schedules in 2009. In [18] they planned the schedules based on the shortest traveling cost. The system was able to arrange two or more sport schedules. The schedules satisfied all the expected characters of every league and the genetic algorithm used, solved the scheduling problems effectively.

Gaspero and Schaerf [6] proposed a Tabu Search (TS) approach to the solution of the TTP. The proposed approach uses a combination of two neighborhood relations. In [6], before applying a local search to the problem, they defined numerous features. These features include illustrating the cost function, the procedure for computing the initial state and the search space. Then finally, they define the structure of the neighbourhood and described the guiding metaheuristic. Their algorithm performed very well in terms of computational time and also obtained good quality solutions for some benchmark problems.

Wei *et al.* [19] tackled the mirrored Traveling Tournament Problem (mTTP). The only difference between mTTP and the general TTP is the mirrored double round-

robin concept, which is basically a double round-robin tournament in which the second half of the tournament is the mirror of the first half (*i.e.* the venues are reversed). They proposed a local search hybrid that combines TS and a Variable Neighbourhood Descent (VND) metaheuristic together with a procedure that is based on a greedy randomized adaptive search which helps in the exploration of large neighbourhoods. The approach proposed in [19] did not perform well on small benchmark instances due to the conciseness of the neighbourhood structure.

Lim *et al.* [20] used a combination of SA and Hill-Climbing (HC) algorithms or techniques to solve the problem. They divided the search space; for searching a timetable space they use SA and for exploring team assignment space they use HC. SA uses conditional jumps to mutate timetables that will lead to better schedules and the enhancement of HC by dynamic cost updating and pre-computation to produce efficient and fast searches [20]. Their approach gave better results compared to other results they used for benchmarking. Tajbakhsh *et al.* [21] proposed a different mathematical model with no-repeater constraints for the TTP and additionally proposed a hybrid algorithm that combines SA and Particle Swarm Optimization (PSO). In phase 1 of the proposed hybrid, PSO is used to generate many schedules very quickly. In phase 2, the best schedules are applied and improved by SA [21]. The results obtained in [21] were comparable to the best known solutions used for benchmarking and even performed better with standard (small) instances, in terms of computational time.

Chen *et al.* [22] proposed a framework in which ant algorithm is employed as a hyper-heursitic. The problem is modelled as a fully connected graph by the proposed algorithm. The pheromone volume on the edges of the graph is used by the ants when deciding on their next move (which vertex to visit next) from the current vertex. The vertices represent some visible feature of the problem. A heuristic applied to a current solution returns a new solution, each vertex represents a heuristic to be applied. The vertices of the network have many ants carrying initial solutions uniformly located with each ant representing a hyper-heuristic agent. When an ant

arrives at a vertex, a heuristic at that node is applied. The proposed algorithm in [22] performed very well (found optimal solutions) with smaller instances ($n = 4$ and $n = 6$ teams) and also for large instances, the algorithm obtained very good and high quality solutions.

2.2.1 Comparative Study

Tables 2.1 and 2.2 below show the solutions obtained for two different data sets by the metaheuristics discussed above. Some metaheuristics were only tested for only one of the instances and some were only tested for instances with $n \geq 8$ because most metaheuristics obtain very good solutions for smaller instances. The highlighted entries illustrate the best solution obtained for that data instance by comparing all the metaheuristics.

Table 2.1: Comparative Study of Metaheuristics in Literature

Author	Method	Circ4	Circ6	Circ8	Circ10	Circ12	Circ14	Circ16
Anagnostopoulos *et al.* [17]	SA	-	-	132	254	420	682	976
Gaspero and Schaerf [6]	TS	-	-	-	**244**	426	682	1004
Wei *et al.* [19]	TS, VND	-	-	132	288	456	714	1002
Lim *et al.* [20]	SA, HC	20	54	132	246	**408**	**654**	**928**

From Table 2.1 and 2.2 we can observe that in most Circ$n$ and NL10 instances good solutions are obtained by Lim *et al.* [20] and Anagnostopoulos *et al.* [17] respectively.

2.3 Other Scheduling Problems

In this section a few of the various approaches used to automate scheduling problems are explained with more focus on those that employ Swarm Intelligence (SI).

Table 2.2: Comparative Study of Metaheuristics in Literature

Author	Method	NL4	NL6	NL8	NL10	NL12	NL14	NL16
Anagnostopoulos *et al.* [17]	SA	-	-	39721	**59583**	**112298**	**190056**	**267194**
Wei *et al.* [19]	TS, VND	-	-	42802	64992	120635	208086	286532
Lim *et al.* [20]	SA, HC	8276	23916	39721	59821	115089	196363	274673
Chen *et al.* [22]	Ant	8276	23916	40361	65168	123752	225169	321037
Tajbakhsh *et al.* [21]	SA, PSO	8276	23916	39721	65002			

Finding the optimal solution for fairly large scheduling problems may not be practical and thus resorting to techniques that find reasonable solutions in an acceptable time frame is more feasible.

2.3.1 The School Timetabling Problem

In many respects, scheduling timetables for a university or high school is very similar to the sports scheduling problem and there are significant researches that have been done in this area. Timetabling entails "*assigning a set of events to a number of rooms and timeslots such that they satisfy a number of constraints*" [23]. The most common variants of the problem are the University Course Timetabling Problem (UCTP) and the Exam Timetabling Problem (ETP). The two have minor constraint related differences. For ETP, exams can take place in the same room and timeslot as long as all constraints are satisfied whereas in UCTP only one event can occupy a certain room at a specific time [23].

Sheafenho *et al.* [24] proposed a PSO approach to solve the UCTP. In [24] they use a standard PSO-based algorithm and apply it to three different UCTP datasets. Their proposed approach did not do so well compared to other algorithms used for benchmarking, but did well in terms of computational time. Sheafenho *et al.*

[25] also proposed a combination of PSO and local search to effectively search the solution space in solving the UCTP. PSO is applied to schedule subjects into the timetable at each iteration and whenever there are clashes in the timetable, the local search is applied to seek for the nearest available neighbourhood timeslot and room [25]. The local search algorithm used in [25] is not mentioned in their research study. The proposed algorithm was tested on three different datasets of different sizes and performed very well compared to other algorithms used for benchmarking.

Djamaras and Ku-Mahamud [26] presented an algorithm; an ant system based algorithm for solving the UCTP. They use a bipartite graph to model the problem. From the graph's characteristics, four heuristic factors are derived for directing ants as agents in finding elements of course timetabling. The first factor is more concerned with finding a destination vertex that has the shortest path leading to it from the source vertex, and the source vertex load is distributed by the second factor to destination vertices. The third factor gives high priority to courses that require more time to deliver, and finally, high priority is given to edges that represent expertise of the lectures in choosing courses and preferable time slots by the fourth factor and this leads to high quality schedules [26]. [26] applied the concept of negative pheromone which ensures that the paths that lead to a dead end are not selected. The proposed algorithm was tested on randomly generated data and the use of the four heuristic factors and negative pheromone concept improved the performance of the algorithm.

Khang *et al.* [27] proposed the bees algorithm to solve real-world UCTP in Vietnam, this was a highly constrained UCTP. The curriculum-based UCTP was considered in [27] mapping between a set of periods, devices, rooms and a set of courses in such a way that the requirements of the university are satisfied. The proposed algorithm was applied to 9 real-world data sets and obtained very promising results which were much better than handmade results.

2.3.2 Job Shop Scheduling Problem

In the Job Shop Scheduling Problem (JSSP) n jobs $j_1, j_2, j_3, ..., j_n$, which need to be scheduled on m machines are presented with the objective of minimizing the makespan (time required to complete the jobs) [28].

Chong *et al.* [29] presented an innovative method that uses the foraging model of the honey bees to solve the JSSP. The JSSP is presented as the disjunctive graph, each operation has a node and there are two extra nodes called the *source* which serves as an initial operation and the *sink* which serves as a final operation. The initial operation of each job is connected to the source and the sink is connected to the final operation. In the graph, the length of the longest directed path gives a solution's makespan and the total of the processing times of the problem in a path gives the length of that path [29]. The algorithm proposed in [29] was applied to 82 JSSP instances and did not obtain good solutions but was able to provide some solutions that were comparable to other algorithms used for benchmarking.

Sha and Lin [30] proposed a PSO for solving a multi-objective JSSP. In the approach suggested in [30], rather than moving particles towards the best solution, the focus is on preventing them from becoming stuck in local optima. The proposed algorithm was applied to benchmark problems and obtained very good results on all instances. Li *et al.* [31] presented an Artificial Bee Colony (ABC) hybrid algorithm to solve the flexible JSSP. In proposed algorithm, local search is performed using TS for the employed bee phase, scout bees and onlooker bees. The proposed algorithm was then applied to five well-known benchmark instances and proved to be superior to other algorithms in terms of computational complexity and search quality.

Li and Wang [32] proposed an improved version of an ant colony algorithm to solve the multi-objective flexible JSSP. The flexible JSSP is based on several objects; the first object is based on the time taken for production *e.g.* the makespan, the second object is based on delivery date *e.g.* cost of delay and the last one is based on cost *e.g.* cost of processing [32]. The approach proposed in [32] only focus on production time which includes makespan, total workload and machine workload.

The proposed approach was applied to practical instances and it proved to be efficient and feasible for the flexible multi-objective JSSP.

2.3.3 Airline Crew Scheduling

Crew scheduling involves the assignment of a number of workers to a set of tasks. The aim of crew scheduling is to create a set of pairings that are feasible and that also minimize the total cost of assigning crew while satisfying given government regulations, flight schedule, company's own policy and flight routes [33]. The problem has been getting a lot of attention from both the academic community and from industry. Lo and Deng [34] proposed an Ant Colony Optimization (ACO) algorithm when tackling the problem. They expressed the problem as a Traveling Salesman Problem (TSP), that makes use of a flight graph representation. The graph uses flights as nodes of paths and the connecting edges to comply to the constraints between sequential flights [34]. The proposed ACO algorithm was tested on real cases of a Taiwanese airline company, the algorithm was benchmarked with three other algorithms and was able to converge to better solutions more efficiently than the other algorithms.

2.4 Conclusion

From the literature survey, one would observe that the TTP is very difficult to solve using exact methods (IP and CP) especially for medium to large data instances. Local search algorithms such as SA have proven that they can obtain good quality solutions when they are applied to the problem and a number of these algorithms have been tested. Only few SI algorithms have been tested, one of them is the ant algorithm which performed well only on small datasets and the other one is the PSO which only performed well when hybridized with a local search algorithm. SI has been applied to other scheduling problems which are similar to the TTP and

it performed very well. In this research study more SI algorithms are explored as SI has a large subset of algorithms.

Chapter Three

Swarm Intelligence for the Traveling Tournament Problem

In this chapter we discuss all the different methods used in this research; they include all the SI algorithms that have been implemented in solving the problem. Also the TTP including the mathematical model used is used.

3.1 Problem Definition

Easton *et al.* [7] introduced the TTP. The TTP takes in an even set of n teams and an $n \times n$ symmetric matrix D, whose entries d_{ij} denotes the distance between the home cities of team T_i and T_j. A desired solution for the TTP is a DRRT that minimizes the total travelling distance amongst all the teams and satisfies feasibility constraints. A DRRT is a tournament in which every team plays its opponent twice, one home game and one away game. A DRRT has $2(n-1)$ timeslots or rounds, where n is the number of teams. Each team starts the tournament at its home site and must return home after its last away game. When a team plays away games consecutively, it travels from one away venue directly to the next one. The main constraints of the TTP are the *double round-robin* constraints which must be satisfied at all times, *at-most* constraints and *no-repeat* constraints which may or

may not be satisfied at times. The at-most constraints ensure that a team cannot play more than three home or away games in sequence and the no-repeat constraints ensure that a game between team T_i and T_j, at the home venue of team T_i, cannot be immediately followed by a game between the two teams at the home venue of team T_j. In this study the mathematical model proposed in [21] which proved to work for instances as large as $n = 16$ is used.

$$\text{Minimize} \sum_{i=1}^{n} \sum_{k=1}^{2(n-1)} d_{ij} x_{ijk} \tag{3.1}$$

Subject to the following constraints:

$$\sum_{j=1}^{n} (x_{i,j,k} + x_{j,i,k}) = 1 \quad \forall 1 \leq i \leq n, \ \forall 1 \leq k \leq 2(n-1) \tag{3.2}$$

$$\sum_{k=1}^{2(n-1)} x_{i,j,k} = 1 \quad \forall 1 \leq i \leq n, \ \forall 1 \leq j \leq n, \ i \neq j \tag{3.3}$$

$$x_{i,j,k} + x_{j,i,k+1} \leq 1 \quad \forall 1 \leq i \neq j \leq n, \ \forall 1 \leq k \leq 2n-3 \tag{3.4}$$

$$x_{j,i,k} + x_{i,j,k+1} \leq 1 \quad \forall 1 \leq i \neq j \leq n, \ \forall 1 \leq k \leq 2n-3 \tag{3.5}$$

$$\sum_{j=1}^{n} \sum_{s=k}^{k+3} x_{i,j,s} \leq 3 \quad \forall 1 \leq i \leq n, \ \forall 1 \leq k \leq 2n-5 \tag{3.6}$$

$$\sum_{j=1}^{n} \sum_{s=k}^{k+3} x_{j,i,s} \leq 3 \quad \forall 1 \leq i \leq n, \ \forall 1 \leq k \leq 2n-5 \tag{3.7}$$

Where:

n : represents the number of teams

i : a team index

j : a team index

k : represents a timeslot or a round

d_{ij} : is the distance between team *i's* home venue and team *j's* home.

$$x_{ijk} = \left\{ \begin{array}{ll} 0, & \text{if team } i \text{ is playing at home in slot or round } k, \\ 1, & \text{Otherwise.} \end{array} \right\} \tag{3.8}$$

Constraint (3.2) ensures that each team plays against exactly one team in each timeslot. Constrainst (3.3) ensures that every team plays one home game with other teams. Constraints (3.4) and (3.5) prevent the no-repeat constraints while constraints (3.6) and (3.7) prevent the at-most constraint.

3.1.1 New Objective Function

When applying the algorithm, it is difficult to satisfy all the TTP constraints in all iterations, especially the no-repeat and at-most constraints. Exploring the infeasible regions can help one find solutions which are of high quality and hence the need to modify the objective function. The new objective function $C(S)$ is given below [17]:

$$C(S) = \left\{ \begin{array}{ll} d, & \text{If schedule } S \text{ is feasible} \\ \sqrt{d^2 + (w * f(n(S)))^2}, & \text{Otherwise} \end{array} \right\} \tag{3.9}$$

Where d is the cost of a schedule (total distance travelled amongst all the teams), w is the penalty weight incurred by a non-feasible schedule and $n(S)$ represents the number of violations of the at-most and no-repeat constraints. Anagnostopoulos *et al.* [17] proved that a suitable function to control the violations of the constraints is given by: $f(v) = 1 + \sqrt{v} \times \ln(v/2)$, v is the number of violations.

3.2 Creating Initial Schedules

A polygon method as in [4] was used to create a schedule with n abstract teams for a single round-robin tournament without assigning venues. The abstract teams are randomized rather than having a simple increasing order from 1 to n. The abstract teams arranged from 1 to $n-1$ are placed initially at clockwise sequentially numbered nodes of a regular polygon with $n-1$ nodes, the abstract team n is not placed in the polygon. At every round, the team placed in the node on one side of the symmetric axis plays against the team on the other side (opposite side). The

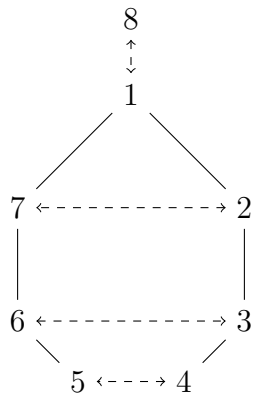


Figure 3.1: Polygon Method for $n = 8$.

Table 3.1: Polygon Method Output

$T \backslash K$	1	2	3	4	5	6	7
1	8	6	4	2	7	5	3
2	7	5	3	1	6	4	8
3	6	4	2	7	5	8	1
4	5	3	1	6	8	2	7
5	4	2	7	8	3	1	6
6	3	1	8	4	2	7	5
7	2	8	5	3	1	6	4
8	1	7	6	5	4	3	2

abstract team n plays against the team placed in node 1. After every round, every abstract team is moved right away to the next node in a clockwise direction until all $n-1$ assignments are accomplished. Figure 3.1 depicts how the polygon method is executed for 8 teams and Table 3.1 shows the results from the execution of the polygon method.

Table 3.2: Final Schedule Produced after Initialization

$T \backslash K$	1	2	3	4	5	6	7
1	−8	−6	4	−2	−7	−5	3
2	7	−5	3	1	−6	−4	−8
3	6	4	−2	7	5	−8	1
4	5	−3	−1	6	−8	2	7
5	−4	2	7	−8	−3	1	6
6	−3	1	−8	−4	2	−7	−5
7	−2	8	−5	−3	1	6	4
8	1	−7	6	5	4	3	2

The next stage in creating an initial schedule is assigning real teams to the abstract teams. The initial schedule is duplicated and used to produce an $n \times n$ matrix of consecutive opponents [19]. Each entry (i,j) is equal to the number of times the two teams are consecutive opponents of other teams. Real teams with smaller distances between their home cities are likely to be assigned to abstract teams that are played more times, consecutively, so as to reduce the total travelling distance.

The final stage is to assign venues to each game, venues are assigned randomly and feasibility constraints must be satisfied, if the schedule is not feasible, the assignments are repeated until a feasible schedule is obtained. Table 3.2 depicts how the final schedule should look after initialization. Home and away games are represented by different signs, $-t_i$ is an away game and t_i is a home game. The schedule is then duplicated, and the venues are reversed to create the second half of the tournament, and thus, a DRRT is produced.

3.3 Neighbourhoods

In this study four different types of moves are defined and explored by the algorithms. The neighbourhood of a solution is basically a set of possible schedules obtained by applying these moves. These are the four moves that yielded better results after experimenting with different combination of moves.

Home-Away Swap: This move swaps a venue for a game between teams T_i and T_j. If team T_i initially plays at home against T_j at round K_l and away at T_j's home at round K_m, where l and m are round indexes, then in the solution obtained by this neighbourhood, T_i will play away at T_j's home at round K_l and home at round K_m. Consider the schedule with $n = 6$ and $2(n-1)$ rounds. Below the appplication of home-away swap to teams T_2 and T_5 is shown. The tables simply give an overview of how this particular move works, some constraints might be violated. The highlighted entries are the entries affected by the move.

Table 3.3: Schedule before Home-Away Swap

$T \backslash K$	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2	7	3	−4	−6	5	−8	−2	−7	−3	4	6	−5	8
2	−1	8	7	−3	4	−6	**−5**	1	−8	−7	3	−4	6	**5**
3	−6	5	−1	2	7	−8	4	6	−5	1	−2	−7	8	−4
4	8	−6	5	1	−2	7	−3	−8	6	−5	−1	2	−7	3
5	7	−3	−4	−6	8	−1	**2**	−7	3	4	6	−8	1	**−2**
6	3	4	−8	5	1	2	−7	−3	−4	8	−5	−1	−2	7
7	−5	−1	−2	8	−3	−4	6	5	1	2	−8	3	4	−6
8	−4	−2	6	−7	−5	3	1	4	2	−6	7	5	−3	−1

Table 3.4: Schedule after Home-Away Swap

$T \backslash K$	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2	7	3	−4	−6	5	−8	−2	−7	−3	4	6	−5	8
2	−1	8	7	−3	4	−6	**5**	1	−8	−7	3	−4	6	**−5**
3	−6	5	−1	2	7	−8	4	6	−5	1	−2	−7	8	−4
4	8	−6	5	1	−2	7	−3	−8	6	−5	−1	2	−7	3
5	7	−3	−4	−6	8	−1	**−2**	−7	3	4	6	−8	1	**2**
6	3	4	−8	5	1	2	−7	−3	−4	8	−5	−1	−2	7
7	−5	−1	−2	8	−3	−4	6	5	1	2	−8	3	4	−6
8	−4	−2	6	−7	−5	3	1	4	2	−6	7	5	−3	−1

Team Swap: This move selects two teams T_i and T_j randomly and swaps the schedule of the two teams except when they play against each other. The corresponding lines of the opponents of T_i and T_j must be changed as well *i.e.* the rest of the schedule must be updated to produce a DRRT. Below the application of team swap to teams T_3 and T_6 is shown.

Table 3.5: Schedule before Team Swap

$T \backslash K$	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2	7	3	−4	−6	5	−8	−2	−7	−3	4	6	−5	8
2	−1	8	7	−3	4	−6	−5	1	−8	−7	3	−4	6	5
3	−6	**5**	**−1**	**2**	**7**	**−8**	**4**	6	**−5**	**1**	**−2**	**−7**	**8**	**−4**
4	8	−6	5	1	−2	7	−3	−8	6	−5	−1	2	−7	3
5	7	−3	−4	−6	8	−1	2	−7	3	4	6	−8	1	−2
6	3	**4**	**−8**	**5**	**1**	**2**	**−7**	−3	**−4**	**8**	**−5**	**−1**	**−2**	**7**
7	−5	−1	−2	8	−3	−4	6	5	1	2	−8	3	4	−6
8	−4	−2	6	−7	−5	3	1	4	2	−6	7	5	−3	−1

Table 3.6: Schedule after Team Swap

$T \backslash K$	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2	7	**6**	−4	**−3**	5	−8	−2	−7	**−6**	4	**3**	−5	8
2	−1	8	7	**−6**	4	**−3**	−5	1	−8	−7	**6**	−4	**3**	5
3	−6	**5**	**−1**	**2**	**7**	**−8**	**4**	6	**−5**	**1**	**−2**	**−7**	**8**	**−4**
4	8	**−3**	5	1	−2	7	**−6**	−8	**6**	−5	−1	2	−7	**3**
5	7	**−6**	−4	**−3**	8	−1	2	−7	**6**	4	**3**	−8	1	−2
6	3	**4**	**−8**	**5**	**1**	**2**	**−7**	−3	**−4**	**8**	**−5**	**−1**	**−2**	**7**
7	−5	−1	−2	8	**−6**	−4	**3**	5	1	2	−8	**6**	4	**−3**
8	−4	−2	**3**	−7	−5	**6**	1	4	2	**−3**	7	5	**−6**	−1

Round Swap: This move selects two rounds randomly K_l and K_m and then simply swaps all the games between the two rounds. In the table below, the application of round swap to rounds K_2 and K_5 is shown.

Table 3.7: Schedule before Round Swap

$T \backslash K$	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2	**7**	3	−4	**−6**	5	−8	−2	−7	−3	4	6	−5	8
2	−1	**8**	7	−3	**4**	−6	−5	1	−8	−7	3	−4	6	5
3	−6	**5**	−1	2	**7**	−8	4	6	−5	1	−2	−7	8	−4
4	8	**−6**	5	1	**−2**	7	−3	−8	6	−5	−1	2	−7	3
5	7	**−3**	−4	−6	**8**	−1	2	−7	3	4	6	−8	1	−2
6	3	**4**	−8	5	**1**	2	−7	−3	−4	8	−5	−1	−2	7
7	−5	**−1**	−2	8	**−3**	−4	6	5	1	2	−8	3	4	−6
8	−4	**−2**	6	−7	**−5**	3	1	4	2	−6	7	5	−3	−1

Table 3.8: Schedule after Round Swap

$T \backslash K$	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2	**−6**	3	−4	**7**	5	−8	−2	−7	−3	4	6	−5	8
2	−1	**4**	7	−3	**8**	−6	−5	1	−8	−7	3	−4	6	5
3	−6	**7**	−1	2	**5**	−8	4	6	−5	1	−2	−7	8	−4
4	8	**−2**	5	1	**−6**	7	−3	−8	6	−5	−1	2	−7	3
5	7	**8**	−4	−6	**−3**	−1	2	−7	3	4	6	−8	1	−2
6	3	**1**	−8	5	**4**	2	−7	−3	−4	8	−5	−1	−2	7
7	−5	**−3**	−2	8	**−1**	−4	6	5	1	2	−8	3	4	−6
8	−4	**−5**	6	−7	**−2**	3	1	4	2	−6	7	5	−3	−1

Partial Round Swap: This move selects a random team T_i and two rounds K_l and K_m, and swaps T_i's games at these two rounds. The rest of the schedule is updated in order to produce a DRRT. The table below shows the application of partial round swap to K_1 and K_8 and team T_4. After applying the move, note that it is also necessary to do a swap for team T_8 in order to obtain a DRRT.

Table 3.9: Schedule before Partial Round Swap

$T \backslash K$	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2	7	3	−4	−6	5	−8	−2	−7	−3	4	6	−5	8
2	−1	8	7	−3	4	−6	−5	1	−8	−7	3	−4	6	5
3	−6	5	−1	2	7	−8	4	6	−5	1	−2	−7	8	−4
4	**8**	−6	5	1	−2	7	−3	**−8**	6	−5	−1	2	−7	3
5	7	−3	−4	−6	8	−1	2	−7	3	4	6	−8	1	−2
6	3	4	−8	5	1	2	−7	−3	−4	8	−5	−1	−2	7
7	−5	−1	−2	8	−3	−4	6	5	1	2	−8	3	4	−6
8	−4	−2	6	−7	−5	3	1	4	2	−6	7	5	−3	−1

Table 3.10: Schedule after Partial Round Swap

$T \backslash K$	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2	7	3	−4	−6	5	−8	−2	−7	−3	4	6	−5	8
2	−1	8	7	−3	4	−6	−5	1	−8	−7	3	−4	6	5
3	−6	5	−1	2	7	−8	4	6	−5	1	−2	−7	8	−4
4	**−8**	−6	5	1	−2	7	−3	**8**	6	−5	−1	2	−7	3
5	7	−3	−4	−6	8	−1	2	−7	3	4	6	−8	1	−2
6	3	4	−8	5	1	2	−7	−3	−4	8	−5	−1	−2	7
7	−5	−1	−2	8	−3	−4	6	5	1	2	−8	3	4	−6
8	**4**	−2	6	−7	−5	3	1	**−4**	2	−6	7	5	−3	−1

3.4 Algorithms

In this section all the SI algorithms that have been implemented in trying to solve the problem are discussed.

3.4.1 Artificial Bee Colony (ABC) Algorithm

ABC [35, 36] is an optimization algorithm that is based on the foraging behaviour of the honey bee swarm. The algorithm starts with n food sources (solutions). A fitness function $f(x_i)$ is used to evaluate the fitness of each solution. The principal aim of the bees is to discover solutions with good fitness values [35]. If a newly generated solution has a better fitness value than the previous solution, the previous solution is forgotten and the new position is updated. The ABC has three different groups of bees; the employed bees, onlooker bees and scout bees. The employed bees produce modifications on the source positions in their memory and produce new positions of good solutions. If the fitness of the new solution is better than that of the prior solution, an employed bee forgets the previous solution and memorizes the new one. Employed bees share the information of the solutions with the onlooker bees that then choose a solution depending on the fitness values of the solutions. The scout bees abandon all the solutions that are not beneficial for search progress and insert new solutions. The algorithm has a well balanced exploration ability achieved by using employed and onlooker bees, and exploitation ability achieved by scout bees [35]. In this work, the food sources are all the possible schedules to the problem and the fitness is the cost (travelling distance) of each schedule. The parameter *limit* controls the number of trials given to a solution to improve, if it does not improve then it is abandoned. In this work the new solution v is obtained by exploring the different neighborhoods defined. Selection probability used in the onlooker phase is given by:

$$prob_i = \frac{fitness(i)}{\sum_{i=1}^{n} fitness(i)}$$

Where n is the population size divide by 2. Algorithm 1 below is the pseudo-code for ABC adapted from [35], $colonySize$ is the population size, $limit$ is the number of pre-determined trials before a food source is abandoned and $maxCycle$ is the number of iterations.

Algorithm 1 Artificial Bee Colony

Require: Parameters: $colonySize$, $limit$ and $maxCycle$.

Ensure: Initialize food sources randomly.

Ensure: Evaluate the fitness of the population.

for $i = 0$ to $maxCycle$ **do**

for $j = 0$ to $colonySize/2$ **do**

{Employed Bee phase}

Execute the defined neighbourhoods to get new solution v.

if $fitness(v) > fitness(x_i)$ **then**

Greedy selection

else

$count_i \leftarrow count_i + 1$

end if

end for

for $j = colonySize/2 + 1$ to $colonySize$ **do**

{Onlooker phase}

Calculate selection probability, select a bee using selection probability

Execute the defined neighbourhoods to get new solution v

if $fitness(v) > fitness(x_i)$ **then**

Greedy selection

else

$count_i \leftarrow count_i + 1$

end if

end for

for $j = 1$ to $colonySize/2$ **do**

{Scout phase}

if $count_i > limit$ **then**

$x_i \leftarrow init()$

end if

end for

Memorize best solution

end for

3.4.2 Cuckoo Search (CS) Algorithm

CS [35, 37] mimics the breeding behaviour of some cuckoo species, which hatch their eggs and raise their chicks using host birds. Most of the time parasitic cuckoos choose a nest where a host bird has just laid its eggs. When the host bird discovers that the eggs are not its own, it either throws them away or simply leaves the nest and builds a new one. From the cuckoo's strategy, three simple principles emerge and the design search is based on these principles:

- Each cuckoo lays a single egg, and choose an arbitrary nest to dump the egg.
- The nest with the best quality of eggs (i.e. nests with best solutions) proceeds to the next generation. The quality of a solution also known as the fitness of a solution (f_i) , is the total travelling distance.
- The number of host nests available is fixed and the probability of a cuckoo egg being discovered by a host is $p \in [0, 1]$. In this work, the host nests are all the possible solutions (schedules) to the problem. The probability of a host discovering an alien egg is given by:

$$p(i) = p_{\max} - i/n_{\max}[p_{\max} - p_{\min}]$$

Where i is the current iteration and n_{max} is the total number of iterations, $populationSize$ used later in algorithm 2 is the number of possible solutions. Below is the pseudo-code for CS adapted from [37].

Algorithm 2 Cuckoo Search

Require: Parameters: $populationSize$, n_{max}, p_{max} and p_{min}.

Generate initial solution/schedule

Calculate the fitness of the solution.

for $i = 0$ to $maxIterations$ **do**

Apply the moves discussed in 4.2 to get a new solution

Calculate the fitness of the new solution (f_i)

Select a random solution (say f_j)

if $f(i) < f(j)$ **then**

replace f_j with the new solution

end if

Remove worst solutions based on prob. p and build new solutions

Find the best solution by ranking the solutions

Keep track of the best solution

end for

3.4.3 Bacterial Foraging Optimization (BFO) Algorithm

The BFO [38, 35] algorithm gets its inspiration from the group foraging behaviour of bacteria such as M. xarithus and E. coli. The BFO is specifically inspired by the bacteria's behaviour called the chemotaxis that percieves chemical gradients in the environment and moves away or towards particular signals. The chemical gradients in the bacteria's environment helps them see the direction to the food source. Bacteria also secretes chemicals into the environment that may either repel or attract and this is how they notice each other [38]. All possible solutions to a problem are represented by a colony of n bacteria. A fitness function fx_i is used to evaluate the fitness of each solution. There are three main routines in the BFO [35]:

- Chemotaxis: In this routine a tumble is represented by a bacterium that has arbitrary direction and a run is represented by a bacterium that has the same

direction as its prior step.

- Reproduction: The fitness value of each bacterium is represented by its health. The health status of bacteria is used for sorting all bacteria and the only population that survives is the one in the first half. The surviving bacteria is duplicated to make a new population and thus the bacteria population is kept constant.

- Elimination-dispersal: The population's diversity is increased by this routine.

Chemotaxis and reproduction steps help to achieve the exploitation of the search space and exploration is accomplished by elimation-dispersal. Algorithm 3 below depicts the BFO algorithm, N_c is the number of chemotactic steps, N_{re} is the number of reproduction steps, N_s is the number runs steps and N_{ed} is the number of elimination-dispersal steps.

Algorithm 3 Bacterial Foraging algorithm

Require: Parameters: $n, N_c, N_{re}, N_s, N_{ed}$.

Ensure: Initialize randomly the bacterial colony

Ensure: Evaluate the fitness of the population.

for $i = 1$ to N_{ed} **do**

{Elimination dispersal}

for $j = 1$ to N_{re} **do**

{Reproduction loop}

for $j = 1$ to N_c **do**

{Chemotaxis loop}

for $k = 0$ to n **do**

Explore the neighbourhoods defined to get new solution

Compute its fitness

$m \leftarrow 0$

while $m < N_s$ **do**

if $newsolution < f(x_i)$ **then**

Update solution

Move again

else

$m \leftarrow N_s$

end if

end while

end for

end for

for $x = 1$ to n **do**

Sort bacteria

Best half of the colony duplicates and replaces the worst part

end for

end for

end for

3.4.4 Bat Algorithm

The Bat algorithm [35] has a population of n possible solutions (bats), that use biosonar for sensing the distance and for updating its velocities and positions by randomly flying through a search space. The fitness function $f(x_i)$ is used to evaluate each solution (the fitness of each solution is the total travelling distance). The aim of the flights is to find good solutions. The main parameters of the algorithm are the loudness decay factor ($loudness$) and the pulse rate ($pulseRate$). The loudness parameter works like the cooling schedule in the Simulated Annealing (SA) algorithm [39] and the pulse rate is responsible for regulating the frequency of the pulse. Updating both the pulse and loudness parameters properly helps to balance both the exploration and exploitation behaviour of each individual bat [35]. Once a bat finds its solution, the loudness decreases and the accuracy of the attack is raised by increasing the emission of the pulse rate. Algorithm 4 below shows the pseudocode of the Bat algorithm, $maxIterations$ is the number iterations.

Algorithm 4 Bat algorithm

Require: Parameters: n, $pulseRate$, $loudness$, $maxIterations$

Ensure: Initialize population

Ensure: Evaluate the fitness of each solution.

while $i < maxIterations$ **do**

$newSol \leftarrow$ Generate new solution by exploring defined neighbourhoods

$ftns \leftarrow Evaluate fitness of newSol$

$rand \leftarrow random[0, 2]$

if $rand < pulseRate$ **then**

if $ftns < f(x_i)$ **then**

Replace i with new solution

end if

end if

Generate new solution by randomly exploring the neighbourhood

$ftns \leftarrow$ Evaluate fitness of the new solution

$rand \leftarrow random[0, 1]$

if $rand < loudnes$ and $ftns < f(x_i)$ **then**

Accept new solution

Increase $pulseRate$

Decrease $loudness$

end if

Memorize best solution

$i++$

end while

3.4.5 Bacterial Foraging, Bat and Cuckoo Search (BBC)

This algorithm is the combination of BFO, Bat and CS. The parameter settings and the design of the algorithms are still the same. An algorithm is selected randomly, runs for n times and no algorithm is selected consecutively; they alternate for every run. Algorithm 5 below depicts the method that drives the running of this algorithm.

Algorithm 5 BBC algorithm

```
selected ← 1
algos ← array[1, 2, 3]
for i = 0 to numRuns do
    random ← [0, 2]
    while Same algorithm selected do
        random ← [0, 2]
    end while
    selected ← rand(random)
    switch (selected) do
    Case 1: run BFO()
    Case 2: run Bat()
    Case 3: run CS()
    end switch
end for
```

Chapter Four

Experimental Results and Discussion

All the algorithms were coded in Java and the experiments were performed on a machine running with the Ubuntu operating system, with 2.00 GHz $\times$ 2 processor and 1 GB of RAM.

4.1 Data Set

Two data sets are used in this study. The first data set is formed by artificially generated circle instances for testing whether the TTP problems are easier when the associated travelling salesman instances are trivial [4]. The instance is denoted by Circ$n$, where n is the number of teams and $8 \leq n \leq 16$. The second data set is formed by the National League (NL) instances which were generated by measuring the distances between the home cities of the teams participating in the league. The name of the instance is denoted by NLn, where n is the number of teams playing in the league and $8 \leq n \leq 16$. The NL instances are based on the Major League Baseball (MLB) in U.S.A.

4.2 Results

In this section the parameter settings and the results obtained by each of the implemented algorithms are discussed.

4.2.1 Cuckoo Search (CS)

Experiments were performed with the CS algorithm and the selected parameter values that led to better solutions: $populationSize$ was set to 100, n_{max} was set to 100, p_{max} was set to 1 and p_{min} was set to 0.05 and the algorithm was run 20 times in order to view or test its robustness. Table 4.1 shows the best, and worst solutions obtained for each instance over 20 runs, as well as the computational time measured in seconds. Table 4.2 illustrates the performance of the CS compared to other previous results found in literature. The three algorithms used for benchmarking are: the Simulated Annealing for the TTP (TTSA) [17] which uses advanced techniques like reheating for balancing the exploitation and exploration of both the infeasible and feasible regions, the GRILS_mTTP [4] which is based on iterated local search metaheuristics and the ejection chain in the neighbourhood to improve the quality of the solution and the last one is a hybrid heuristic which is a combination of the Genetic Algorithm (GA) and the SA (GA-SA) [40].

TTSA had the best known results as extracted from this study, especially for the NL instances. The last column of Table 4.2 contains the relative gap in percentage between the values of the best known solution and the best solution found by our algorithm.

Table 4.1: Computational Results of CS Algorithm

Instance	Best	Worst	Time(sec)
Circ8	130	174	27.751
Circ10	249	313	46.632
Circ12	418	527	69.313
Circ14	673	814	149.054
Circ16	1001	1178	444.484
NL8	39044	57589	32.399
NL10	58112	80654	61.675
NL12	113769	151234	83.539
NL14	188678	239893	145.528
NL16	277579	345542	388.297

Table 4.2: CS Results with Three Benchmark Algorithms

Instance	TTSA	GRILS-mTTP	GA-SA	CS	%
Circ8	132	140	142	130	−1.5
Circ10	254	276	282	249	−2.0
Circ12	420	456	458	418	−0.5
Circ14	682	714	714	673	−1.3
Circ16	976	1004	1014	1001	2.5
NL8	39721	41928	43112	39044	−1.7
NL10	59583	63832	66264	58112	−2.5
NL12	112298	120655	120981	113769	1.3
NL14	190056	208086	208086	188678	−0.7
NL16	267194	285614	290188	277579	3.9

4.2.2 Artificial Bee Colony (ABC)

The colony size ($colonySize$) was set to 100, the maximun number of cycles ($maxCycle$) for foraging was also set to 100 and the algorithm was run 50 times in order to view or test its robustness. Table 4.3 shows the best, and worst solutions obtained for each instance over the 50 runs, as well as the computational time measured in seconds. Table 4.4 illustrates the performance of the ABC compared to other previous results found in literature.

Table 4.3: Computational Results of ABC Algorithm

Instance	Best	Worst	Time(sec)
Circ8	131	161	66.582
Circ10	238	270	101.016
Circ12	411	477	190.098
Circ14	715	813	384.025
Circ16	1114	1256	708.897
NL8	35651	49877	76.938
NL10	58675	70313	142.627
NL12	113140	131863	377.181
NL14	237691	268846	435.602
NL16	300429	338618	608.976

Table 4.4: ABC Results with Three Benchmark Algorithms

Instance	TTSA	GRILS-mTTP	GA-SA	ABC	%
Circ8	132	140	142	131	−0.7
Circ10	254	276	282	238	−6.3
Circ12	420	456	458	411	−2.1
Circ14	682	714	714	715	4.8
Circ16	976	1004	1014	1114	14.1
NL8	39721	41928	43112	35651	−10.3
NL10	59583	63832	66264	58675	−1.5
NL12	112298	120655	120981	113140	0.8
NL14	190056	208086	208086	237691	25.1
NL16	267194	285614	290188	300429	12.4

4.2.3 Bat

The population size (*populatonSize*) was set to 100, the *loudness* to 0.95 and the pulse rate (*pulseRate*) to 1, the maximum number of iterations was set to 100 and the number of runs to 50. Table 4.5 shows the results obtained by the algorithm for each data instance over 50 runs, it shows the best and the worst solutions obtained as well as the running time measured in seconds. Table 4.6 illustrates the performance of Bat compared with previous results found in literature.

Table 4.5: Computational Results of Bat Algorithm

Instance	Best	Worst	Time(sec)
Circ8	129	226	4.394
Circ10	296	450	8.334
Circ12	516	744	13.923
Circ14	896	1188	35.621
Circ16	1396	1802	133.589
NL8	33804	60756	9.855
NL10	60630	97122	14.597
NL12	117746	187016	17.149
NL14	255460	350832	72.631
NL16	364498	489766	93.196

Table 4.6: Bat Results with Three Benchmark Algorithms

Instance	TTSA	GRILS-mTTP	GA-SA	Bat	%
Circ8	132	140	142	129	−2.3
Circ10	254	276	282	296	16.5
Circ12	420	456	458	516	22.9
Circ14	682	714	714	896	31.4
Circ16	976	1004	1014	1296	32.8
NL8	39721	41928	43112	33804	−14.9
NL10	59583	63832	66264	60630	1.8
NL12	112298	120655	120981	117746	4.9
NL14	190056	208086	208086	255460	34.4
NL16	267194	285614	290188	364498	36.4

4.2.4 Bacterial Foraging (BFO)

The population size (colonySize) was set to 100, $N_c = 70$, $N_{re} = 4$, $N_s = 4$,$N_{ed} = 1$ and the number of runs was set to 50. N_c, N_{re}, N_s, N_{ed}, represent the number of chemotactic, reproductive, run and elimination dispersal steps respectively. Table 4.7 depicts the best and the worst solutions obtained by the algorithm over 50 runs, as well as the running time(seconds) of the algorithm. Table 4.8 illustrates the performance of BFO compared with previous results found in literature.

Table 4.7: Computational Results of BFO Algorithm

Instance	Best	Worst	Time(sec)
Circ8	130	206	277.66
Circ10	252	430	523.712
Circ12	411	686	999.573
Circ14	657	1070	2940.59
Circ16	1087	1652	6459.09
NL8	35546	56928	281.147
NL10	56443	90305	491.81
NL12	119036	173853	925.041
NL14	191362	194549	1585.665
NL16	296067	296438	4781.973

Table 4.8: BFO Results with Three Benchmark Algorithms

Instance	TTSA	GRILS-mTTP	GA-SA	BFO	%
Circ8	132	140	142	130	−1.5
Circ10	254	276	282	252	−0.8
Circ12	420	456	458	411	−2.1
Circ14	682	714	714	657	−3.7
Circ16	976	1004	1014	1087	11.4
NL8	39721	41928	43112	35546	−10.5
NL10	59583	63832	66264	56443	−5.3
NL12	112298	120655	120981	119036	6.0
NL14	190056	208086	208086	191362	0.7
NL16	267194	285614	290188	296067	10.9

4.2.5 Bacterial Foraging, Bat and Cuckoo Search (BBC)

This algorithm is the combination of BFO, Bat and CS algorithms. The same parameter settings are utilised for each of these algorithms. The algorithm was run 50 times. Table 4.9 depicts the best and the worst solutions obtained by the algorithm over 50 runs, as well as the running time(seconds) of the algorithm. Table 4.10 illustrates the performance of the BBC compared to previous results found in literature.

Table 4.9: Computational Results of BBC Algorithm

Instance	Best	Worst	Time(sec)
Circ8	125	234	109.64
Circ10	252	462	249.632
Circ12	421	754	366.251
Circ14	678	1210	916.678
Circ16	1077	1776	1549.274
NL8	31208	61308	119.406
NL10	56987	104581	258.769
NL12	111157	203303	417.678
NL14	191265	345765	774.965
NL16	292472	497854	1497.357

Table 4.10: BBC Results with Three Benchmark Algorithms

Instance	TTSA	GRILS-mTTP	GA-SA	BBC	%
Circ8	132	140	142	125	−5.3
Circ10	254	276	282	252	−0.8
Circ12	420	456	458	421	0.2
Circ14	682	714	714	678	−0.59
Circ16	976	1004	1014	1077	10.4
NL8	39721	41928	43112	31208	−21.4
NL10	59583	63832	66264	56987	−4.4
NL12	112298	120655	120981	111157	−1.0
NL14	190056	208086	208086	191265	0.6
NL16	267194	285614	290188	292472	9.5

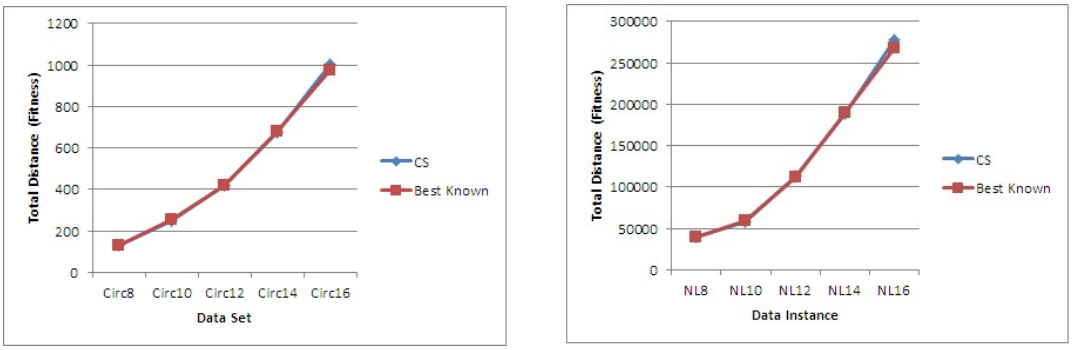
4.3 Discussion

The algorithms utilised in this study were very competitive on small instances ($n \leq 14$) and in some instances obtained solutions that are far better than the best known solution. From Table 4.2 and Figure 4.1, one can observe that for the circle instances; Circ8, Circ10, Circ12 and Circ14, CS improved the best solution values by 1.5%, 2.0%, 0.5% and 1.3% respectively. For the national league instances; NL8, NL10 and NL14, the algorithm improved the best known solution values by 1.7%, 2.5% and 0.7% respectively. CS outperformed GRILS-mTTP and GA-SA on all instances. ABC improved the best known solution values by 0.7%, 6.3%, 2.1%, 10.3%, 1.5% on Circ8, Circ10, Circ12, NL8 and NL10 respectively, and obtained better solutions than GRILS-mTTP, GA-SA on most instances except Circ14, Circ16, NL14 and NL16, this can be seen from Table 4.4 and Figure 4.2. From Table 4.6 and Figure 4.3, it can be observed the Bat algorithm only obtained better solutions on two instances Circ8 and NL8 and improved the best known solution values by 2.3% and 14.9% respectively and also did not perform well compared to the other two alogrithms on all instances except when $n = 8$.

From Table 4.8 and Figure 4.4, one can observe that the BFO algorithm improved the best known solution values on Circ8 to Circ14 by 1.5%, 0.8%, 2.1% and 3.7% respectively and improved NL8 and NL10 by 10.5% and 5.3% respectively, the algorithm obtained better solutions compared to the other two algorithms (GRILS-mTTP and GA-SA) on all instances except when $n = 16$. BBC obtained better solutions than the best known solution on Circ8, Circ10, Circ14, NL8, NL10, NL12 and improved the best known solution values by 5.3%, 0.8%, 0.6%, 21.4%, 4.4% and 1.0% respectively, this can be observed from Table 4.10 and Figure 4.5. From the results obtained, it is observed that CS is the best performing algorithm since it was able to improve the best known solution values for all seven instances, followed by BFO, BBC and ABC respectively. CS performs local search more efficiently, this is one of the key contributors to its good performance.

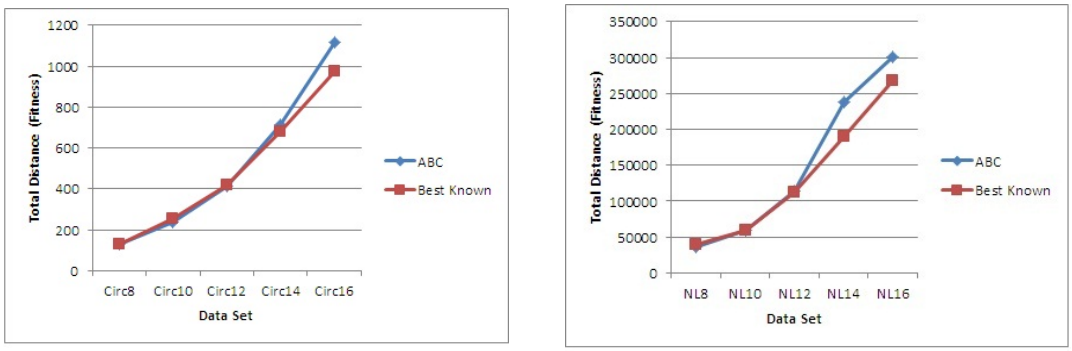
Bat is the worst performing algorithm as depicted in Figures 4.6 and 4.7 but proved

to be the best in terms of computational time as illustrated in Figures 4.8 and 4.9, this is because of its fast convergence speed. This inspired the implementation of BBC, which basically combines the best performing algorithms (CS and BFO) but quite slow in terms of computational time, compared to the algorithm that performed better and faster. From Figures 4.8 and 4.9, one can observe that BFO is the worst performing algorithm in terms of computational time due its slow convergence. All implemented algorithms performed poorly when $n = 16$ for both instances, but CS was able to retain/maintain the gap below 4% (1.9% on Circ16 and 3.9% on NL16) compared to the best known solution (TTSA).



(a) Circ$n$ instances (b) NLn instances

Figure 4.1: CS Benchmarked with the Best known Solution



(a) Circ$n$ instances (b) NLn instances

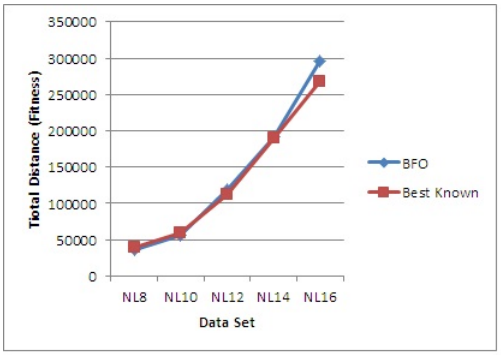
Figure 4.2: ABC Benchmarked with the Best known Solution

(a) Circ$n$ instances

(b) NLn instances

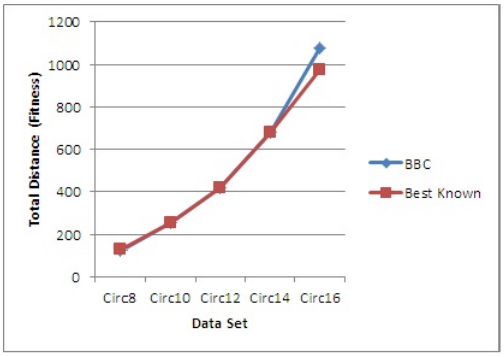
Figure 4.3: Bat Benchmarked with the Best known Solution

(a) Circ$n$ instances



(b) NLn instances

Figure 4.4: BFO Benchmarked with the Best known Solution



(a) Circ$n$ instances

(b) NLn instances

Figure 4.5: BBC Benchmarked with the Best Known Solution

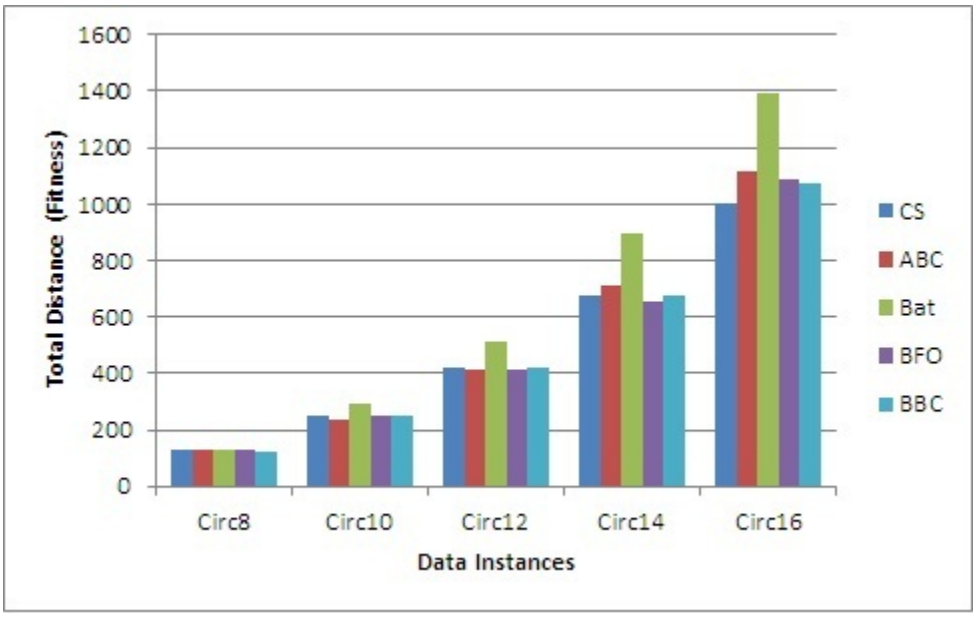


Figure 4.6: Swarm Intelligence Algorithms Performance on Circ$n$ Instances

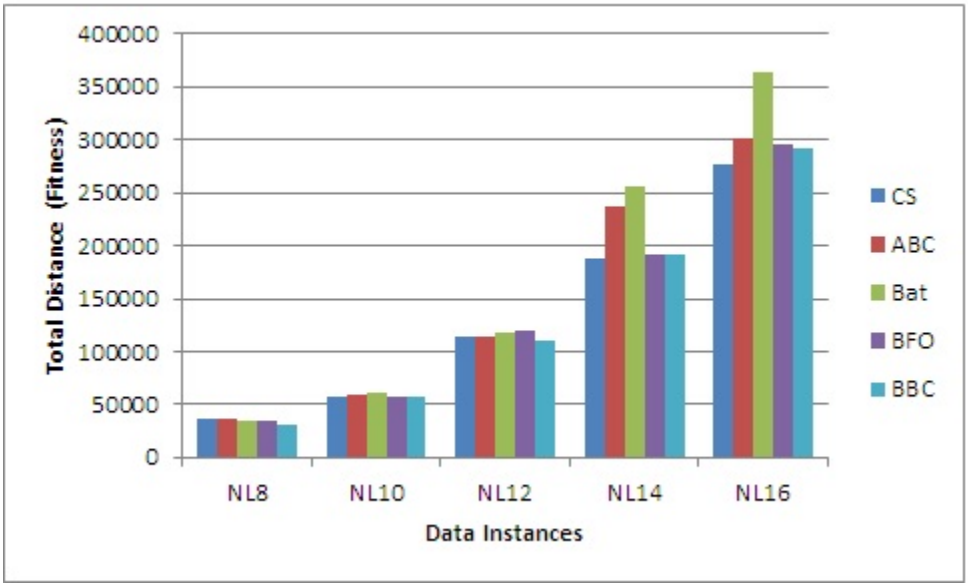


Figure 4.7: Swarm Intelligence Algorithms Performance on NLn Instances

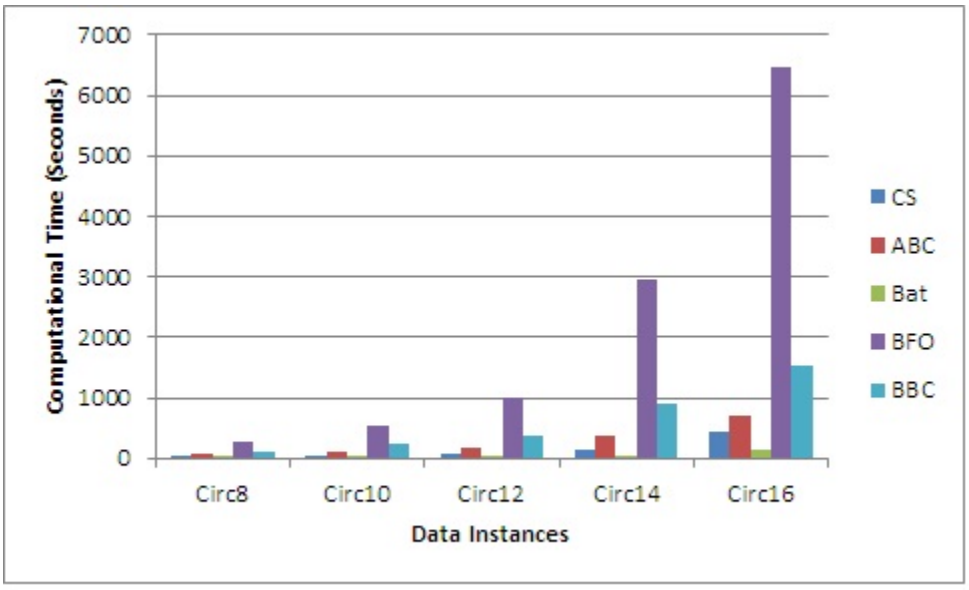


Figure 4.8: Swarm Intelligence Algorithms Computational Times on Circ$n$ Instances

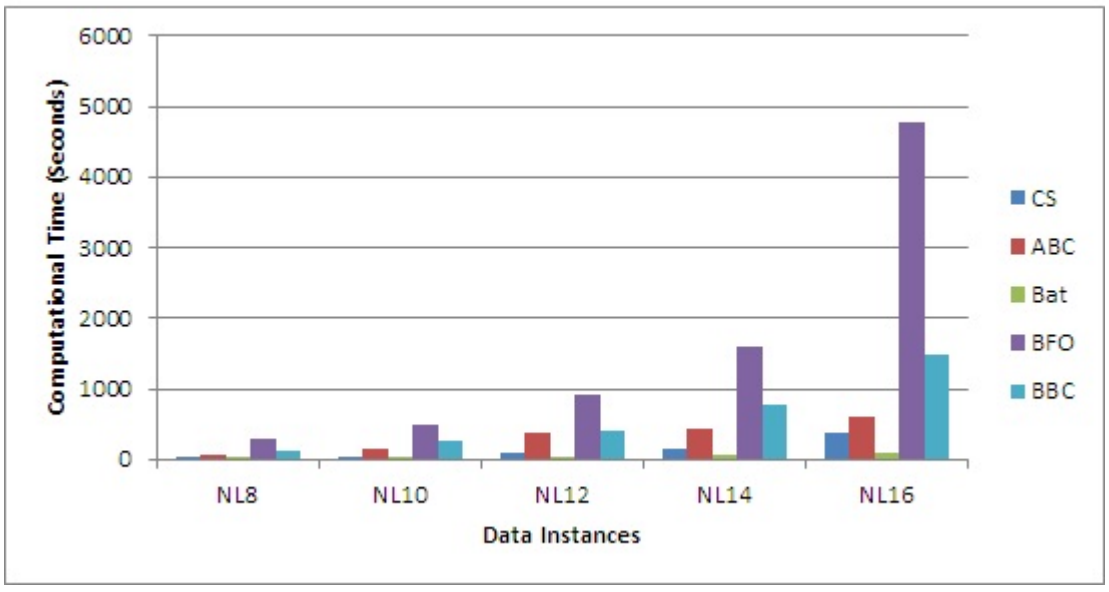


Figure 4.9: Swarm Intelligence Algorithms Computatinal Times on NLn Instances

Chapter Five

Conclusion and Future Research

In this study, the TTP was addressed. Five algorithms were developed and applied to the problem. Four neighbourhoods were defined and explored or executed by the algorithms with the goal of improving the solution available. The results obtained by some of the algorithms used in this study are very promising, providing better solutions than the algorithms used for benchmarking, in some instances and obtaining very competitive results in other instances especially for very large data sets ($n = 16$). Though some of the implemented algorithms proved to be very competitive, one of them (Bat) did not perform well for most instances, but was very competitive in terms of computational time.

From all the algorithms implemented, CS performed very well on most instances, the algorithm was able to improve 4 out of 5 best known solutions of Circ$n$ instances and 3 out of 5 of NLn instances. None of the algorithms managed to improve the best known solution for the largest instance ($n = 16$), but they did obtain competitive solutions. From this study, one can observe that from SI family, there are algorithms that can obtain near optimal solutions for scheduling problems such as the TTP, though they are often overlooked for this problem. One direction that future research studies can focus on, will be trying to improve the mathematical model or using different combinations of swaps with the aim of improving the solution.

Appendix One

User Manual

A.1 Starting the Application

We have developed a very simple Java GUI application. The name of the application file is **"Swarm.jar"**, to run the the file in Ubuntu, open the terminal, go to the directory where the jar file is located and then type **java -jar Swarm.jar**. The application will start and the main window will pop up as seen in figure A.1.

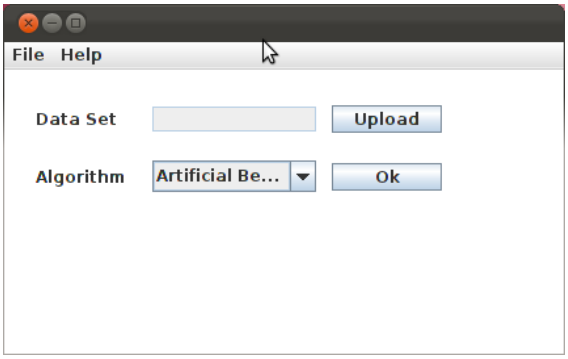


Figure A.1: Main Window

A.2 Running Algorithms

Select data set: To select a data set, click the upload button, the open dialog shown in figure A.2 will pop up, go to the directory where the data set is located, then click open once you have selected the file you want.

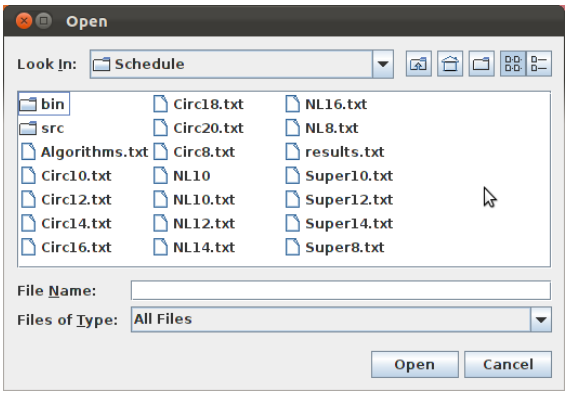


Figure A.2: Data Set

Selecting Algorithm: Select the algorithm that you wish to run from the list as seen in figure A.3, then click ok.

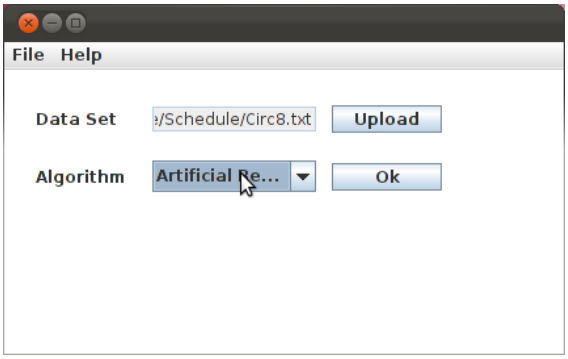
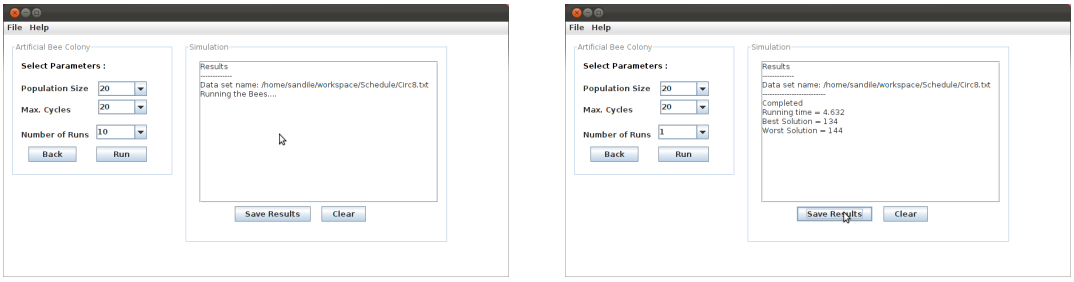


Figure A.3: Selecting Algorithm

This will take you to a new panel, select the parameter values that you want to run the algorithm with, then click run. The output will be displayed in the text area on the right hand side of the panel as seen in figure A.4.



(a) Parameters (b) Output

Figure A.4: Algorithm Parameters and Output

To save the output, click the save results button and a save dialog will pop up as seen in figure A.5, select the directory that you wish to save results in, then click save.

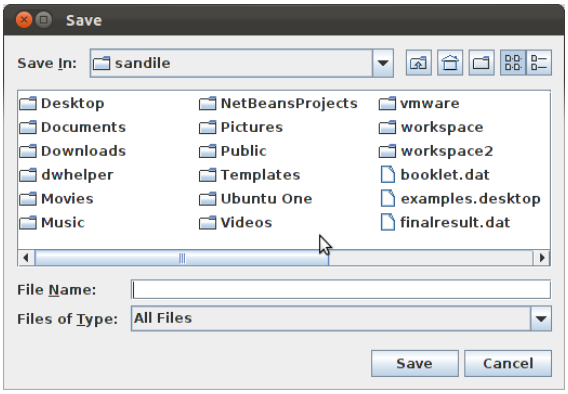
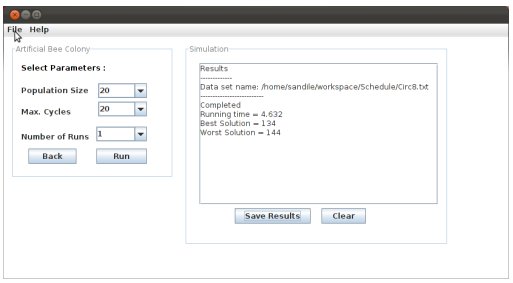
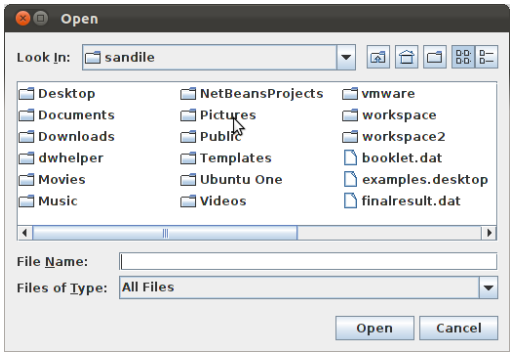


Figure A.5: Save Output

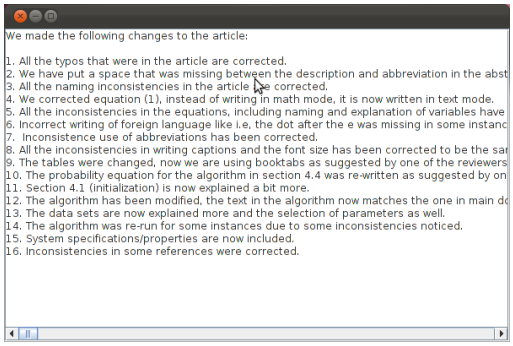
Open existing file: To open an existing file, go to file as shown in figure A.6, select the the first option file, then select the file you wish to open. The file will be displayed in a separate window.



(a) Open existing file



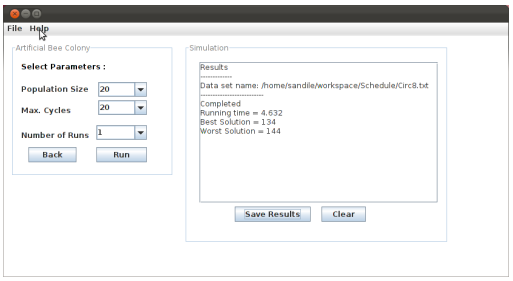
(b) Select file



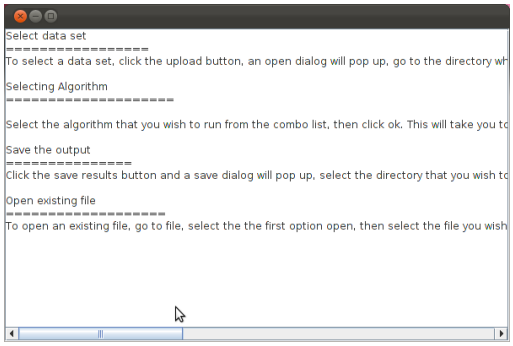
(c) Selected file

Figure A.6: Opening already Existing File

Help: To open the help file, go to the help tab, select manual, and a new window will pop up displaying the help file.



(a) Parameters



(b) Help file

Figure A.7: Help File

References

[1] A. Goldberg. Highly Constrained Sports Scheduling with Genetic Algorithms. Department of mathematics and computer science. Bachelors thesis, Amherst College, 2003.

[2] S. Young. Alternative Aspects of Sports Scheduling. *Preprint*, 2004.

[3] D. Uthus. *Sports Scheduling: An Artificial Intelligence Approach.* PhD thesis, University of Auckland, 2010.

[4] C. Ribeiro and S. Urrutia. Heuristics for the mirrored traveling tournament problem. Department of Computer Science, Universidale Federal Fluminese, Brazil. Department of Computer Science, Catholic University of Rio de Janeiro, Rio de Janeiro., 2004.

[5] M. Ringer, O. Reinelt, and O. Rinaldi. *The traveling salesman problem.* In M. Ball, T. Magnati, C. Monma, and G. Nemhauser, editors, Handbooks in Operations Research and Management Science, Chapter 7, pages 225-330, North-Holland, 1995.

[6] L. Gaspero and A. Schaerf. A tabu search approach to the traveling tournament problem. Dipartimento di Ingegneria Elettrica, Gestionale e Meccanica Universita di Udine, 2005.

[7] K. Easton, G. Nemhauser, and M. Trick. Solving the traveling tournament problem: A combined integer programming and constraint programming approach. School of Industrial and Systems Engineering, Georgia Institute of

Technology, Atlanta, Georgia USA. Graduate School of Industrial Administration, Carnegie Mellon, Pittsburgh, PA, USA, 2003.

[8] S. Olafsson. *Metaheuristics*. Nelson and Henderson (eds.). Handbook on Simulation, 2006.

[9] M. Beekman, G. Sword, and S. Simpson. Biological foundations of swarm intelligence. Behaviour and Genetics of Social Insects Lab, School of Biological Sciences, University of Sydney, Sydney, Australia. Behaviour and Physiology Research Group, School of Biological Sciences, University of Sydney, Sydney, Australia, 2008.

[10] Wikipedia. `http://en.wikipedia.org/wiki/Constraint_programming`. Accessed November 2012.

[11] R. Apt. Principles of constraint programming. Centrum Wiskunde and Informatica (CWI), Amsterdam, the Netherlands, 2003.

[12] Wikipedia. `http://en.wikipedia.org/wiki/integer_programming`. Accessed November 2012.

[13] S. Juan, R. Razamin, and I. Haslinda. A hybrid constraint-based programming approach to design a sports tournament scheduling. European Journal of Scientific Research, 49 (1). pp. 39-48., 2011.

[14] R. Rasmussen. Hybrid ip/cp methods for solving sports scheduling problems. Department of Operations Research, University of Aarhus, Denmark, 2006.

[15] K. Sorensen and F. Glover. `www.scholarpedia.org/article/Metaheurics`. Accessed November 2012.

[16] Wikipedia. `http://en.wikipedia.org/wiki/Metaheuristics`. Accessed November 2012.

[17] A. Anagnostopoulos, L. Michel, P. Van Hentenryck, and Y. Vergados. A simulated annealing approach to the traveling tournament problem. Brown University, 2005.

[18] J. Hung, Y. Yen, and K. Chein. Professional sport scheduling optimization system based on the shortest traveling cost. Department of Computer Science and Information Engineering, Tamkang University, 2005.

[19] W. Wei, S. Fujimura, X. Wei, and C. Ding. A hybrid local search approach in solving the mirrored traveling tournament problem. Graduate School of Information, Production and System, Waseda University, Kitakyasya, Japan. School of Electronic, Information and Electrical Enginnering, Shangai Jia Tong University, Shangai, China, 2010.

[20] A. Lim, B. Rodrigues, and X. Zhang. A simulated annealing and hill-climbing algorithm for the traveling tournament problem. Department of IEEM, Hong Kong University of Science and Technology, Clearwater Bay, Kowloon, Hong Kong. School of Business, Singapore Management University, 2003.

[21] A. Tajbakhsh, K. Eshghi, and A. Shamsi. A hybrid pso-sa algorithm for the traveling tournament problem. Faculty of Industrial Engineering, Sharif University of Technology, Tehran, Iran, 2009.

[22] P. Chen, G. Kendall, and G. Berghe. An ant based hyper-heuristic for the traveling tournament problem. School of Computer Science, University of Nottingham, UK. KaHo Sint-Lieven, Belgium, 2007.

[23] D. Qarouni-Fard, A. Najafi-Ardabali, M. Moeinzadeh, S. Sharifian-R, E. Asgarian, and J. Mohammadzadeh. Finding feasible timetables with particle swarm optimization. Department of Computer Science, Ferdowsi University, Mashad, Iran. Department of Computer Science, University of Tehran, Tehran, Iran. Department of Computer Engineering, Sharif University of Technology, Tehran, Iran, 2007.

[24] I. Sheafenho, D. Safaai, and M. Sitizaiton. A study on pso-based university course timetabling problem. Faculty of Computer Science and Information System, University of Technology Malaysia, 2009.

[25] I. Sheafenho, D. Safaai, and M. Sitizaiton. A combination of pso and local search in university course timetabling problem. Faculty of Computer Science and Information System, University of Technology Malaysia, 2008.

[26] D. Djamarus and K. Ku-Mahamud. Heuristic factors in ant system algorithm for course timetabling problem. College of Arts and Sciences, Universiti Utara Malaysia, 2008.

[27] N. Khang, N. Phuc, and T. Nuong. The bees algorithm for a practical university timetabling problem in vietnam. Faculty of Information Technology and Faculty of Mathematics, University of Science Ho Chi Minh city, Vietnam, 2010.

[28] J. Leornad. Interactive game scheduling with genetic algorithms. Department of Computer Science, Royal Melbourne Institute of Technology University, 1998.

[29] C. Chong, A. Sivakumar, M. Low, and K. Gay. A bee colony optimization algorithm to job shop scheduling. Singapore Institute of Manufacturing Technology. School of Mechanical and Production Engineering, School of Computer Engineering and School of Electrical and Electronics Engineering, Nanyang Technological University, 2006.

[30] D. Sha and H. Lin. A multi-objective pso for job-shop scheduling problems. Department of Industrial Engineering and System Management, Chung Hua University, Hsin Chu. Department of Industrial Engineering and Management, National Chia Tung University, Hsin Chu, 2009.

[31] J. Li, Q. Pan, and S. Xie. Flexible job shop scheduling problems by a hybrid

artificial bee colony algorithm. School of Computer, Liaocheng University, Liaocheng, China, 2011.

[32] L. Li and K. Wang. An improved ant colony algorithm for multi-objective flexible job shop scheduling problem. Information and Computer Engineering College, and Forestry Engineering Automation discipline, Northeast Forestry University, Harbin, Heilongjiang Province, China, 2008.

[33] S. Yan and Y. Tu. Case study a network model for airline cabin crew scheduling. European Journal of Operations Research, 140, pp. 531-540, 2002.

[34] C. Lo and G. Deng. Using ant colony optimization algorithm to solve airline crew scheduling problems. Department of Informatics, Fo Guang University. Department of Management Information Systems, National Chengchi University, 2007.

[35] R. Parpinelli and H. Lopes. New inspirations in swarm intelligence: a survey. Bioinformatics Laboratory, Federal University of Technology Paran (UTFPR), Curitiba (PR), Brazil and Applied Cognitive Computing Group, Santa Catarina State University (UDESC), Brazil. Bioinformatics Laboratory, Federal University of Technology Paran (UTFPR), Curitiba (PR), Brazil., 2011.

[36] Wikipedia. `http://en.wikipedia.org/wiki/Artificial_bee_colony_algorithm`. Accessed March 2012.

[37] E. Valian, S. Mohanna, and S. Tavakoli. Improved cuckoo search algorithm for global optimization. University of Sistan and Baluchestan, 2011.

[38] J. Brownlee. Clever algorithms: Nature-inspired programming recipes. `http://www.cleveralgorithms.com/nature-inspired/swarm/bfoa.html`. Accessed October 2012.

[39] D. Bertsimas and J. Tsitsilis. *Simulated Annealing*. Statistical Science Vol. 8, No.1, 10-15, Sloan School of Management, School of Electrical Engineering

and Computer Science Department, Massachusetts Institute of Technology, Cambridge, Massachussetts, 1993.

[40] F. Biajoli and L. Lorena. Mirrored traveling tournament problem: An evolutionary approach. Lecture Notes in Computer Science,Vol 4140, springer, pp. 208-217, 2006.