Discrete Particle Swarm Optimization for Combinatorial Problems with Innovative Applications

UNIVERSITY OF
KWAZULU-NATAL

POPOOLA, PETER AYOKUNLE

(215079768)

A thesis submitted in fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the

School of Mathematics, Statistics and Computer Science
University of KwaZulu-Natal
Durban, South Africa

February, 2016

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 Prof. A.O. Adewumi and co-supervision of Dr. Martins A. Arasomwan. I hereby declare that all materials incorporated in this thesis are 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:

_____

Popoola Peter Ayokunle

Date: February, 2016

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

Signed:

_____

Date: February, 2016

As the candidate's co-supervisor, I have approved/disapproved the dissertation for submission

Signed:

_____

Date: February, 2016

Declaration 1 – Plagiarism

I, Popoola Peter Ayokunle, 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, their writing has been placed in italics and in quotation marks, and referenced.
5. This thesis does not contain text, graphics or tables which are copied and pasted from the internet, unless specifically acknowledged, and the source being detailed in the thesis and in References section.

Signed:

_____

Date: February, 2016

Dedication

I dedicate this research work to God the Father, Son and Holy Spirit, without whom this work would have been impossible.

Acknowledgements

First and foremost, I fall in complete adoration and awe before the God of the whole Universe, the Father and Guide of my youth and the Friend who has stuck closer to me than the closest brother. Your faithfulness to me has been always fresh as the morning dew and constant from day to day. You saw me through so many situations where I was stuck between a rock and a hard place, having no one to turn to. I owe You everything I am and will ever be, and I am nothing without You.

My loving parents, Dr. Samuel Funso Popoola and Mrs. Deborah Oluremi Popoola, you are the best I could ever have. God has used and continues to use you as the hands that shape my life and make me the person I am today. You have always been there for me, and you have sacrificed your comforts and convenience countless times to see to my well-being. May God continue to give you the deepest of your hearts desires. My dear siblings, Peace, Praise and Paul, and my cousin, Idowu, you are so special and dear to me! I love you with all my heart, and I pray that God grants you His best every day of your lives, here and hereafter.

My distinguished supervisor, Professor Aderemi Oluyinka Adewumi, you have been a strong guiding and motivating force for me during this period of my research, and you have always pushed me to achieve my potential. A big THANK YOU to you, sir. My able co-supervisor, Dr. Martins Akugbe Arasomwan, you have been a friend and kind adviser to me, encouraging me and giving me pithy tips at times when your guidance was most needed. Thank you so much, sir.

I want to express my deep appreciation to Dr. Mike Olusanya who has been a father to me here in South Africa. I cannot put into words all the sundry means through which you have been of great help to me. May God reward you for all your investments into my life and a multitude of others in exceeding abundant folds! I cannot also forget the indelible impact which Dr. Joseph Adesina has made upon my life. May your way ever shine brighter and brighter by the light of God! Professor Blamah V. Nachamada played such a significant role in my life that I can never forget. You always went out of your way to be of help to me, more

than most people would ever do. I say a heart-felt thank you to you, sir. May you never lose your reward.

I want to sincerely appreciate the contributions of all members of the Computer Science Discipline Optimization Group, Alex, Ayo, Ebenezer, Peter, Raphael, Seun, Stephen, and Valentine for their valuable contributions and camaraderie during the period of this research. Worthy of special mention is Ayo, whose brotherly love and concern I cannot forget. May God reward you mightily! I also want to specially thank my friend Mafunda for his wonderful sense of humor, and Shalin Singh for the times of stimulating conversation we had which helped me gain a deeper grasp of stochastic graphs.

My gratitude goes to all members of the big family of Deeper Life Bible Church, KZN Province for their love and support. Special thanks to my able G.C. Jeremiah Ogunniyi, and my beloved co-members of the DLBC KZN choir. I really enjoyed my time spent with you, and may God bless you all richly!

Finally, I want to sincerely thank all members of the UKZN community for the opportunity of working on this thesis within a conducive didactic environment, and for all the assistance you have offered to me in one way or the other. May you always have the best in life!

Abstract

Particle Swarm Optimization (PSO) is a swarm intelligence metaheuristic which has been widely applied to various real-world problems with great results. The original algorithm was designed to solve continuous optimization problems. However, a lot of real-world problems are inherently discrete in nature and are sometimes modelled as benchmark Combinatorial Optimization Problems (COP) such as Travelling Salesman Problem (TSP) and Shortest Path Problem (SPP). Therefore, various methods have been proffered towards discretizing it to handle COPs. This work proposes two new Discrete Particle Swarm Optimization (DPSO) algorithms based on insights from gaps noticed in literature and demonstrates that it compares favorably with existing DPSOs. The proposed algorithms were designed and applied specific to three COPs: the Travelling Salesman Problem, the Shortest Path Problem (SPP) and its two variants, the Stochastic Shortest Path problem (SSP) and the Multi-Objective Stochastic Shortest Path Problem (MOSSP), and the Submission Decision Process problem (SDP).

The proposed DPSO for TSP introduces a new concept of guiding particles' search during the PSO process by their collective experience on how 'good' an edge which constituted their previous positions was. Parameters were introduced for determining the intensity with which this 'experience' is followed. The proposed algorithm performed much better when compared with the classical DPSO for TSP. Its performance in comparison with a widely-known DPSO for TSP was also reported and analyzed.

Another DPSO with new ideas for particle position discretization was proposed for the SPP and exhibited very positive performance in comparison with the best-performing DPSO algorithm designed for SPP found in literature. Both algorithms were compared on instances of size ranging from 50 nodes and 166 edges to 100 nodes and 280 edges, and results show that the proposed algorithm out-performs its counterpart on all instances, with equal or lower number of iterations. The proposed DPSO algorithm was also extended to solve instances of size (150, 400) and (200, 450), achieving accuracy levels of 90 and 91 percent respectively.

In further evaluating the performance of the proposed DPSO for SPP, the classical SSP was solved and the results compared with an existing work in literature. The proposed algorithm

was able to find the path with shortest expected length on instances of size ranging from 10 – 15 nodes with 100% accuracy on all instances. Instances of size ranging from (50, 150) to (200, 400), were also solved with 100% accuracy. The proposed DPSO was then extended to solve MOSSP. To evaluate its performance, three graphs, each of size (25, 50) were generated following exponential, triangular, uniform and normal distributions. The algorithm reports the accurate Pareto front for all generated instances. The results were validated by comparing each extreme of the Pareto front against the result produced by Dijkstra's algorithm on the corresponding single-objective problems. A problem instance of size (200, 1000) was also solved, showing a near-optimal Pareto front, and demonstrating the effect of varying values of ε on the algorithm. From experiments, values between 0.01 and 0.02 were recommended as optimal for ε, in line with other results reported in literature.

Finally, the proposed DPSO algorithm for SPP was applied to the recently introduced SDP. The algorithm's performance was compared to that of the Metropolis algorithm used in the paper introducing the problem, and drastic reductions in time complexity were reported while maintaining comparable accuracy. All objectives were then combined in a 3-objective problem of maximizing citation count while minimizing number of resubmissions and time from submission to publication. The algorithm gives a satisfactory Pareto front which highlights the optimal journal submission order to be followed depending on what compromises are desired within the objective space.

Table of Contents

Table of Figures

List of Tables

List of Abbreviations

ACO	Ant Colony Optimization
AIF	Author Impact Factor
ATSP	Asymmetric Travelling Salesman Problem
BPSO	Binary Particle Swarm Optimization
COP	Continuous Optimization Problem
CSP	Constrained Shortest Path Problem
DPSO	Discrete Particle Swarm Optimization
DSPP	Dynamic Shortest Path Problem
DTSP	Dynamic Travelling Salesman Problem
FSPP	Fuzzy Shortest Path Problem
GA	Genetic Algorithms
GTSP	Generalized Travelling Salesman Problem
IF	Impact Factor
JIF	Journal Impact Factor
JSSP	Job Shop Scheduling Problem
KP	Knapsack Problem
MDP	Markov Decision Process
MOO	Multi-Objective Optimization
MOOP	Multi-Objective Optimization Problem
MOPSO	Multi-Objective Particle Swarm Optimization
MOSSP	Multi-Objective Stochastic Shortest Path Problem
NI	Nature-Inspired
PSO	Particle Swarm Optimization
SAT	Satisfiability Problem
SCP	Set Covering Problem
SDP	Submission Decision Problem
SI	Swarm Intelligence
SPP	Shortest Path Problem

SSP	Stochastic Shortest Path Problem
SSP-RC	Shortest Path Problem with Resource Constraints
SSSPP	Single-Source Shortest Path Problem
STSP	Symmetric Travelling Salesman Problem
TSP	Travelling Salesman Problem

List of Articles

Articles under review:

1. P. A. Popoola, A. O. Adewumi, and M. O. Olusanya, "A Memetic Discrete Particle Swarm Optimization Method for the Shortest Path Problem", Submitted to the *South African Journal of Industrial Engineering (SAJIE)*
2. P.A. Popoola, N.V. Blamah, and A.O. Adewumi, "An Enhanced Communication Discrete Particle Swarm Optimizer for the Travelling Salesman Problem", Submitted to *The Transactions of the South African Institute of Electrical Engineers (SAIEE)*
3. P.A. Popoola and A.O. Adewumi, "A Multi-Objective Particle Swarm Optimization Approach to the Submission Decision Process", *International Journal of Systems Assurance Engineering and Management (IJSA)*
4. P.A. Popoola, A.O. Adewumi, and A.M. Arasomwan "A Discrete Particle Swarm Optimizer for the Multi-Objective Stochastic Shortest Path Problem", *International Journal of Advances in Soft Computing and its Application (IJASCA)*

Chapter One

Introduction

1.0 Introduction

Combinatorial Optimization Problems (COP) are among the most widely studied problems in Computer Science owing to their applicability in a myriad of real world scenarios. They are generally problems, which involve finding the best arrangement of a set of discrete objects, where an arrangement's fitness is dependent on the specific problem domain [25].

This work focusses on three COPs, the Travelling Salesman Problem (TSP), Shortest Path Problems (SPP) and its variants, Stochastic Shortest Path problem (SSP) and Multi-Objective Stochastic Shortest Path problem (MOSSP), and the Submission Decision Process (SDP). The TSP involves finding the cheapest arrangement of all nodes in a graph such that a tour/cycle which includes exactly one node each is formed [93]. It is a classical NP-complete COP which has been intensely studied over the years as a result of its ubiquitous real world applications in vehicle routing, circuit board design, x-ray crystallography, and many others.

The SPP has been applicable from the most primitive days of man when he needed to find the shortest path to various essential destinations (e.g. in search of food, shelter, and so on). Up to the early 1950's, SPP was applicable in computing the shortest and alternative shortest paths for freeway drivers and long distance telephone calls in the United States [199]. SPP applications then extended to applications such as network routing, scheduling, production planning, energy (electric power, petroleum and natural gas), decision making process and air, land and sea travel [77], and continue to extend modern applications at the cutting-edge of technological advancement such as shortest path queries on massive social networks [8], path planning in robotics [48], sequence alignment in molecular biology [156], Route Guidance Systems [83, 136] and so on. In addition, when constraints are added to SPPs, they form an important part of the well-known Column Generation algorithm for large-scale Linear Programming problems [112].

The SDP problem arises as a result of the dilemma which researchers face in deciding what journal to send their manuscript to such that the accumulated number of citations is maximized. Journal Impact Factors (JIF) usually give a general approximation of how many citations can be accumulated by papers published in them, and journals with high JIFs also tend to have high rejection rates and take generally longer periods in review. This is most probably as a result of the large number of submissions they receive. Therefore, authors need to know what schedule to follow in submitting their manuscripts, taking into cognizance the probability of getting rejected and having to re-submit to another journal. The SDP can be easily described as a COP, since it essentially involves finding the best arrangement of journals which maximizes/minimizes the desired objectives. The SDP is a newly introduced problem which has a more immediate practical application to every researcher and author who wishes to have their published work recognized.

As a result of this widespread interest in COPs, Nature-Inspired (NI) metaheuristics have been applied to approximate solutions to them. This work focusses on the Swarm Intelligence-based heuristic, Particle Swarm Optimization (PSO), which needs to be first discretized before it is applied to COPs. Various discretization schemes have been applied to PSO, one of which is the set-based approach where candidate solutions are represented as a set of nodes. Two new set-based Discrete Particle Swarm Optimization (DPSO) algorithms are presented in this work, each addressing TSP and SPP with new ideas which have not been used in literature. These algorithms are applied to the TSP, classical SPP, SSP, and MOSSP.

1.1 Problem Statement

It was discovered from reviewed literature that the existing DPSOs for TSP do not optimize their search by taking into account the cost of edges which constitute positions within the fitness landscape. This could lead to the particles spending time unnecessarily exploring positions constituted by relatively high-cost edges which stand a very low chance of being part of the optimal solution.

A key characteristic of existing set-based DPSO algorithms designed for SPP was also discovered. Since candidate paths are represented as sets of nodes, their position within the set is a critical and distinguishing feature which determines the order in which the path will

be traversed from source to destination. However, in the existing DPSO methods, this critical information is not taken into consideration when the various required set operations are carried out such as addition (set union) and subtraction (set difference), as well as in the representation of particles' velocity.

In most real-world situations, the cost/weight of networks are not deterministic as they are usually represented in the classical SPP [114]. They usually vary based on various factors such as road/link congestion, safety or reliability of the link, and so on. As a result, a variant of SPPs, known as Stochastic Shortest Path problem (SSP) is modelled to address them. In SSPs, edge costs are modelled as probability distributions rather than fixed values, and solutions to them usually give the expected shortest path. The PSO method which was found in existing literature for the SSP only reported solutions to SSPs of size 10-15 nodes.

Again, most users are not only concerned with the expected minimum-length path, but are also concerned about the degree of variance or 'sureness' of the path. This variance comes into play as a result of the stochastic nature of the networks under consideration. Therefore, multi-objective optimization is required, where both objectives are to minimize expected cost and variance. It was observed in literature that the application of DPSOs to this type of problem has not been well explored.

In introducing the SDP, Salinas and Munch [193] used a Metropolis algorithm to achieve the Pareto front (or efficiency frontier) showing the optimal levels of compromise available to the authors. Such compromises exist between maximizing citation count and minimizing number of submissions on one hand, or total time spent in review by the various journals on the other. A problem which was identified with this existing method is the huge computational burden entailed by the Metropolis algorithm which they used, where about 3,200,000 different submission schedules had to be tested before the optimal efficiency frontier was discovered. It was also noted that optimizing all three objectives at once has not been considered in literature.

1.2 Research Objectives

The aim of this research is therefore to propose two new DPSOs for the TSP and SPP which will be extensible to other problems such as SSP, MOSSP and the SDP. This will be achieved by considering the following objectives:

1. Develop a new DPSO heuristic for the TSP which is optimized through informing particles' search by the input graph's edge cost distribution, thereby causing them to rapidly avoid edges with relatively high cost and gravitate towards those with much lower costs.
2. Develop a new DPSO heuristic for the SPP which takes into consideration node position-specific information and leads to more accurate results.
3. Adapt and apply the new DPSO for SPP algorithm to SSP and MOSSP, reporting the results.
4. Adapt and apply the new DPSO for SPP algorithm to the Submission Decision Process (SDP) with reasonable time savings, extending it to a 3-objective multi-objective optimization of all objectives in the SDP.

1.3 Research Methodology

Two new DPSO algorithms for TSP and SPP are introduced, and their viability demonstrated by applying them to a range of problems. The results obtained are validated empirically. An empirical study uses a statistical tool to validate a hypothesis or result by experimentation [254]. Four broad classifications of research methods exist in Computer Science literature, as seen in [88]. They are scientific, engineering, empirical and analytical methods. Tichy et al. [219] identify five major categories of works published in the field of Computer Science. These categories are Formal Theory, Design and Modelling, Empirical Work, Hypothesis Testing, and Others. Wainer et al. [227] randomly selected 200 articles published in ACM in 2005, using them as a sample population to estimate the percentage of articles published in the categories identified in [219]. They found out that papers under the Empirical Work category formed 17% of the total works published, with Theory, Hypothesis, Design and Others forming 4, 4.7, 3.4 and 70 percent respectively. This shows that an empirical research

methodology is valid and is being used by a significant percentage of computer science researchers, especially in the area of heuristics and meta-heuristics.

1.4 Scope and Limitations

This research is restricted to DPSO and its application to the three COPs earlier highlighted. The limitations encountered for each problem are as follows:

- **TSP:** The TSP instance size in number of cities attempted was limited to 76. Efforts will be made to increase the capability of the proposed algorithm in future.
- **SPP:** The number of papers which apply DPSO to the classical SPP is relatively low. Therefore, there was a limitation to the works with which the results of the proposed DPSO algorithm could be compared.
- **MOSSP:** It was observed that the application of PSO to Multi-Objective SSP has not been really explored in the existing literature. The only similar work which we were able to find, that of Zhang, et al. [262], did not report the Pareto front for their optimization results, making it difficult for us to compare our results with theirs. We were therefore constrained to randomly generate problem instances, though it is believed that this will be of service to other researchers in testing and reporting their results.

1.5 Thesis Contributions

The major contributions which this work makes are as follows:

1. A new DPSO algorithm for the TSP which takes advantage of edge cost information in directing particles' search.
2. A new DPSO algorithm which can be applied to the SPP. An extension of the DPSO to solve the SSP and MOSSP for graph sizes ranging from 10 nodes and 15 edges to 200 nodes and 1000 edges. The provision of randomly generated SSP graph data which follows four different probability distributions and can be used by other scientists in validating their SSP-related work, and recommendation of optimal ε-

value for ε-dominance (related to MOSSP) which agrees with existing literature. It is believed that the proposed DPSO for SPP can be applied to other similar COPs.

3. The first application of PSO to the SDP in literature (to our knowledge), leading to drastic reduction in processing time and computational burden. Extension of the SDP to a 3-objective optimization problem which considers all three intrinsic objectives: maximizing citation count, minimizing number of resubmissions and minimizing time spent in review. Optimal submission schedules are also reported for the implemented journal data set which validates existing results in [196] to the benefit of the scientific community.

1.6 Outline

The rest of this dissertation is structured as follows:

Chapter two gives some general information about COPs, focusing on the TSP and SPP, as well as the various Exact, Heuristic, and NI methods used in addressing them. It also gives some information needed for a good grasp of the SSP and Multi-Objective Optimization, as well as PSO, with the various discretization methods existing in literature. A brief background for the SDP is also given, together with the models developed to address it.

In Chapter three, the new DPSO algorithm for the TSP is presented and discussed, and empirical results presented. The impact of various introduced parameters such as *R* and *K* on its performance is also discussed.

Chapter four presents the DPSO designed for SPP with a memetic component used to enhance local search. The proposed discretization of PSO is done specifically with respect to weight-biased position encoding of paths in the SPP, the explorative nature of the algorithm being balanced by the local search capability of the memetic algorithm and an optimal selection of *c1* and *c2* parameters.

Chapter five presents the results of applying the proposed memetic DPSO to the classical SPP, comparing them with two PSO and one GA algorithm in literature.

Chapter six presents an adaptation of the new DPSO to SSP with various improvements on existing techniques, and applies it to both the single and multi-objective SSP with very favorable results. The effect of varying ε-dominance values on the proposed discrete MOPSO algorithm is also investigated and reported.

Chapter seven also presents an adapted version of the new DPSO for SDP, showing that it cuts down computational time by a significant factor. An improvement in the computation of the various objectives is also proposed which further reduces computational burden. The problem is extended beyond the usual bi-objective optimization to a 3-objective optimization and is solved with the proposed discrete PSO algorithm.

Finally, Chapter eight concludes and gives some recommendations for future studies.

Chapter Two

Literature Review

2.0 Introduction: Combinatorial Optimization Problems

Combinatorial Optimization is a field of study with roots in combinatorics, operations research, economics and theoretical computer science. It involves finding the best ordering, grouping, selection or arrangement of discrete objects which are usually finite in number [134]. Combinatorial Optimization is one of the most widely-studied fields in mathematics and computer science because it represents thousands of real-life problems such as investment planning and facility location, capital budgeting, gene sequencing, design of new molecules, design of fast and reliable communication networks, assignment of workers to jobs (e.g. airline crew scheduling) design and management of transportation systems and many others [95] [2, 5]. These types of problems can be formulated or modelled as abstract combinatorial problems and formally defined [210] as a tuple $I = (U, P, val, extr)$, where:

U is the solution space

P is the feasibility predicate

val is the objective function $val: U \rightarrow \mathbb{R}$,

extr is the extremum (min or max)

P induces a set, S of feasible solutions:

$$S = \{X \in U : X \text{ satisfies } P\} \tag{2.1}$$

The goal is to attain a solution, $s \in S$ which attains the desired extremum of *val*.

A few of these abstracted COPs are highlighted below:

- **Knapsack Problem (KP):** This is a family of problems which involves a set of items each having a profit, p_j and a weight, w_j to be packed in one or more knapsacks of capacity, c. A subset of these items is to be chosen in such a way that the accumulated profit is maximized without the accumulated weights exceeding c. A most popular member of this family is the 0-1 KP, which is mathematically formulated thus:

$$\begin{aligned} &maximize \sum_{j=1}^{n} p_j x_j \\ &subject\ to \sum_{j=1}^{n} w_j x_j \leq c, \\ &x_j \in \{0,1\}, \quad j = 1, \dots, n, \end{aligned} \tag{2.2}$$

where x_j is a binary variable which takes the value 1 if item j is to be included in the knapsack, and 0 otherwise [180]. Other variants of the Knapsack Problem include *Bounded Knapsack Problem*, *Multiple-choice Knapsack Problem*, *Multiple Knapsack Problem*, *Nested Knapsack Problem*, and a popular extension of the KP, the *Bin-Packing Problem* [149] [180].

- **Set Covering Problem (SCP)**: Given a universal set U of n elements which has a family of subsets $S = \{S_1, \dots, S_k\}$, and a cost function $c: S \rightarrow Q^+$, the SCP seeks a minimum-cost sub-collection of S which covers all elements of U [224]. The SCP has major applications in airline crew scheduling, location of emergency facilities, assembly line balancing and information retrieval [20].
- **Job Shop Scheduling Problem (JSSP)**: The JSSP is a very challenging and difficult problem which has prompted intense studies by scientists over the years. Its fame for difficulty is in part attributed to the fact that an instance with 10 machines and 10 jobs which was proposed in 1963 remained unsolved for over 20 years [11]. It involves a set of jobs to be scheduled on a set of machines, each having a predetermined order in which they must be executed on each machine. Each machine can only process one job at a time, and each job can be processed by at most one machine at a time. Given that each processing operation has a fixed duration, the JSSP involves scheduling jobs such that their maximum completion time is minimized [257]. Obviously, this problem has numerous applications, with different variations, in the manufacturing industry.
- **Satisfiability Problem (SAT):** The Satisfiability problem is among the six basic core NP-complete problems which have been identified as intrinsic to many areas in computing theory and engineering [133]. NP-complete problems are those for which

no proven polynomial-time algorithms exist over all instances. Apart from its indirect applications from the reformulation of other problems, SAT has direct applications in Boolean circuit synthesis, test pattern generation, VLSI engineering, design of asynchronous circuits, sports planning, and so on [92, 96, 133]. It is defined [96] thus: Given a Boolean formula with three components:

- A set of *n* variables, $x_1, x_2, \ldots, x_n$
- A set of variables or their negated equivalents
- A set of *m* distinct clauses: $C_1, C_2, \ldots, C_m$, each consisting of literals combined by solely logical OR ($\vee$) connectives,

Determine whether there exists an assignment of truth values to the variables which makes the Conjunctive Normal Form:

$$C_1 \wedge C_2 \wedge \ldots \wedge C_m \tag{2.3}$$

true (or satisfiable), where $\wedge$ is a logical AND connective.

The remainder of this review focuses on the three COPs which are pertinent to this work, TSP, SPP and the Submission Decision Problem, with a greater emphasis on SPP since it constitutes a larger portion of the work. As for the SDP which was very recently introduced (in 2015) by Salinas and Munch [196], we were not able to find any articles in the literature which explicitly handles it apart from the introducing paper. However, closely-related issues such as citation rate/count, Impact Factor and other publication-related issues have been researched, discussed and debated for decades. A preliminary review of these issues is therefore given before the models for the SDP are presented. Also, we discuss the solution approaches existing in literature for these problems in two broad groupings as exact and heuristic. Heuristic algorithms are further discussed as non NI-based and NI-based methods, as PSO, which is this work is restricted to, is an NI-based heuristic.

2.1 The Travelling Salesman Problem

The Travelling Salesman Problem (TSP) is a well-known combinatorial optimization problem which was officially stated by the Austrian mathematician Karl Menger as follows [94]: "A traveling salesman has to visit exactly once, each of one of a list of *x* cities and then

return to the home city. He knows the cost of traveling from any city *i* to any other city *j*. Which of the possible tours of cities can he take at least possible total cost?" Such a tour which includes all nodes exactly once is referred to as a Hamiltonian cycle [93]. Various approaches exist for approximating solutions to the TSP which is a classical NP-complete problem, and it is widely used as a test benchmark for these optimization algorithms [31]. An Integer Linear Programming formulation of the TSP is given thus [132]:

$$\text{Minimize} \sum_{i \neq j} c_{ij} x_{ij} \tag{2.4}$$

Subject to

$$\sum_{j=1} x_{ij} = 1\,, \quad i = 1, \dots, n \tag{2.5}$$

$$\sum_{i=1}^{n} x_{ij} = 1, \qquad j = 1, \dots, n. \tag{2.6}$$

$$\sum_{i,j \in S} x_{ij} \leq |S| - 1, \quad S \subset V, 2 \leq |S| \leq n - 2, \quad x_{ij} \in \{0,1\} \tag{2.7}$$

$$i, j = 1, \dots, n, \quad i \neq j. \tag{2.8}$$

Where c_{ij} represents the cost of the edge (i,j), S represents the set of nodes constituting a feasible Hamiltonian cycle, and V represents the set of all nodes in the graph. Constraints (2.5) and (2.6) describe the cost of the optimal tour, constraints (2.7) prevent the formation of subtours (tours/cycles with less than *n* nodes), and constraint (2.8) imposes binary conditions on the variables.

If $c_{ij} = c_{ji}$ for all edges in the graph, the problem is known as the Symmetric TSP (STSP). If this is not the case, it is known as the Asymmetric TSP (ATSP). A generalization of the TSP known as Generalized TSP (GTSP) partitions all nodes into clusters and seeks a minimum cost cycle which visits at least one node in each cluster [78]. A variant of the TSP is the Dynamic TSP (DTSP) where the input graph changes over time either with the deletion and

inclusion of new nodes and their connecting edges [97], or changes to edge costs [151]. Another is the Probabilistic TSP (PTSP) where each city in the input graph has an associated probability of being visited [239].

The TSP has a wide range of applications including, but not at all limited to, computer wiring, vehicle routing, data clustering, job shop scheduling, circuit board design, x-ray crystallography, and so on [150].

2.1.1 Solution Approaches to the TSP: Exact Methods

One of the first exact methods for solving the TSP follows from the Integer Programming model earlier presented, which was formulated by Dantzig, et al. [56]. It involved a linear relaxation of some of the constraints, and they were able to optimally solve a 42-city instance using this method. However, the most basic and general exact method for solving the TSP is the branch-and-bound method which involves exhaustive search and is applicable in solving all COPs. It directly originated from research work on TSP, being first introduced in the paper by Little, et al. [142] in 1963.

The branch-and-bound method is typically described as a traversal through a search tree, where the problem is progressively broken down into sub-problems as the tree grows outwards from the root. Each child of a node is obtained by imposing a single new constraint on the original relaxation of the problem. The search usually starts with the root node, and at each iteration, a selection strategy is used to pick a node from a pool of unexplored feasible subproblems. Branching is then performed through the construction of two or more child nodes by the addition of constraints to the node's subproblem. The bound for each child node is calculated and evaluated. If it corresponds to the optimal solution, the algorithm is terminated. If it is a feasible solution, it is compared to an earlier-generated initial solution which acts as the upper bound, and discarded if it is not better. It is also discarded if no feasible solution to this subproblem exists. Otherwise, the node is added to the pool of unexplored subproblems, and the algorithm proceeds to the next iteration. This process continues until the optimal solution is found or there are no more unexplored feasible

solutions. A summary of the branch-and-bound algorithm for a search tree, *T*, is presented in **Algorithm 2-I** [49].

Algorithm 2-I: Branch-and-Bound

1. Initialize $UB = \infty$, $LB(P_0) := g(P_0)$; $Live := \{(P_0, LB(P_0))\}$
2. Repeat until $Live = \emptyset$
    a. Select the node P from $Live$ to be processed; $Live := Live \setminus \{P\}$;
    b. Branch on P generating $P_1, ..., P_k$;
    c. For $1 \leq i \leq k$ do
        i. Bound P_i : $LB(P_i) := g(P_i)$;
        ii. If $LB(P_i) = f(X)$ for a feasible solution X and $f(X) < UB$ then
1. $UB := f(X)$; $Solution := X$;
2. go to EndBound;
        iii. If $LB(P_i) \geq UB$ then discard P_i
        iv. else $Live := Live \cup \{(P_i, LB(P_i))\}$
    d. EndBound
3. $OptimalSolution := Solution$; $OptimumValue := UB$

In Algorithm 2-I, *UB* is the upper bound computed so far by the algorithm, $LB(P_i)$ is the lower bound of the tree node P_i, *Live* is the list of unexplored feasible solutions, and g is a function which maps real numbers (*bounds*) to each node in the branch-and-bound search tree. For each internal node within the tree, this value is a lower bound/limit on any value within the subspace corresponding to that node (that is, that can be achieved for all subproblems which are children of the node). The function, g is required to satisfy these three conditions:

- $g(P_i) \leq f(P_i)$ for all nodes P_i in T
- $g(P_i) = f(P_i)$ for all leaves in T
- $g(P_i) \geq g(P_j)$ if P_j is the parent of Pi

Other exact methods include cutting-plane [56], branch-and-cut [171], and the *Concorde* solver [13].

2.1.2 Solution Approaches to the TSP: Non-Nature-Inspired Heuristics

As is to be expected, the exact methods discussed in section 2.1.1 increase exponentially in complexity with increase in number of cities for the TSP. This effect is exacerbated as a result of the search space for the ATSP being $(N-1)!/2$ large. Therefore, various heuristics have been proposed in literature to approximate solutions which are accurate to an acceptable level in the real world. A few of them are highlighted.

2.1.2.1 Nearest Neighbor Heuristic

The Nearest Neighbor heuristic is a greedy approach which simply builds a tour by starting from a randomly picked city and progressively adding the adjacent city with the minimal cost, stopping when all cities have been added to the tour. It was described by Flood [80] in 1956, but it weakness was quickly discovered as the possibility of getting into 'corners' which would require the addition of relatively high-length edges to be able to link with unvisited cities [13]. The same applies to its close counterpart, the proper ***greedy heuristic*** which sorts all edges by cost and adds them to the tour from lowest to highest until the tour is completed.

2.1.2.2 Insertion Heuristic

In the insertion heuristic, an initial subtour is formed with the shortest edge. Then a city which is not part of the subtour, but has the minimal distance to any of the cities in the tour, is repetitively selected and inserted between the adjacent cities of a selected edge, *(i,j).* Edge (*i,j)* is selected such that the increase in total cost of the new tour is minimal [93, 168].

2.1.2.3 K-opt Heuristics

K-opt heuristics are primarily tour improvement heuristics which seek the best way of reducing the cost of a pre-existing tour. They are the most widely-used heuristics for the TSP [106], consisting of making *k-opt* moves which involve removing *k* edges from the tour, and reconnecting the disconnected nodes with a different set of *k* edges such that the new total tour cost is reduced. This process is repeated with a fixed value of *k* until no more improvements can be made with a *k-opt* move. The tour is then said to be *k-optimal*. A *k-optimal* tour is also *i-optimal* for all $i < k$. The 2- and 3-opt moves are very popular as a result

of their simplicity to compute, speed, and near-optimal results [168]. An example of a 2-opt move is shown in Figure 2.1.

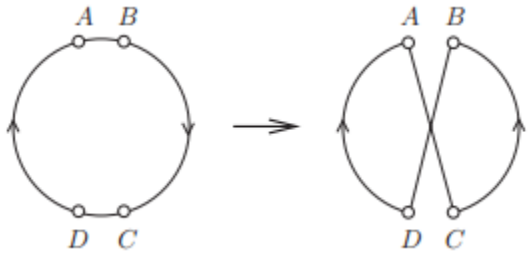


Figure 2.1: A sample 2-opt move [13]

2.1.2.4 Lin-Kernighan Heuristic

The Lin-Kernighan (LK) heuristic, which lies at the heart of the most-accurate solutions to the TSP today [13], was originally proposed by Lin and Kernighan in 1973 [141]. It is based on the *k-opt* heuristic, but uses a variable, rather than the usual static, value for *k.* It involves deciding what value of *k* is most suitable at each iteration of the algorithm. This comes at the cost of a much heightened level of complexity, however, making it quite difficult to implement or improve. Moreso, its accuracy depends largely on how well it is implemented [168]. In the year 2000, however, it was successfully implemented and improved upon by Helsgaun [104], who continued to propose successful improvements in 2006 [105] and 2009 [106]. This has led to the Lin-Kernighan-Helsgaun (LKH) heuristic being the virtual state-of-the-art algorithm for the TSP, and it has been used to solve very large instances of the TSP, the most recently proven one having 85,000 cities [12]. It has also been used to solve the largest TSP instance in the TSPLIB [191] known as *World*, having 1,904,711 cities, to within 0.049% of the optimal tour [105].

2.2 The Shortest Path Problem

The Shortest Path Problem (SPP) also a ubiquitous problem which has triggered widespread, intensive and on-going research in the scientific community over the years. This is as a result of its wide applicability in the fields of routing [38, 83, 204], transportation [45, 246], robotic motion planning [35, 137], terrain navigation [128, 129], communication [156, 169] and so on. It can be mathematically modelled as an integer programming problem thus [10]: Given

a graph, G, with edge set E and vertex set V, specified source and destination nodes s and t, and an associated cost c_{ij} for each edge $(i, j) \in E$,

$$\begin{gathered} \text{Minimize} \sum_{i,j \in E} c_{ij} x_{ij} \\ subject\ to \\ x_{ij} \geq 0, \\ \forall i, \sum_j x_{ij} - \sum_j x_{ji} = \begin{cases} 1, & if\ i = s; \\ -1, & if\ i = t; \\ 0. & otherwise \end{cases} \end{gathered} \tag{2.9}$$

With the introduction of various constraints as well as dynamic properties, its range of application grows even wider [68, 185, 202].

The elementary SPP with single source and single destination nodes was solved by Dijkstra in his 1959 paper [65], and since then, various types and variants of the shortest path problem have been identified. A review of these problems is presented. It is important to note that the definition of cost is not restricted to length of edge or distance between nodes, but can also be expressed in other terms such as time taken to arrive at a particular node and so on.

2.2.1 Variants of the Shortest Path Problem

The form in which the input graph is presented has given rise to about four variants of the SPP, viz: static, dynamic, stochastic and fuzzy shortest path problems. The elementary case is the static SPP where the costs, nodes or constraints of the problem are predetermined and remain fixed throughout the duration of the problem being solved. This case has already been discussed earlier.

2.2.1.1 Dynamic Shortest Path Problem (DSPP)

In the Dynamic SPP (DSPP) which is hugely applicable to Mobile Ad-Hoc Communication Networks, nodes in the graph dynamically disappear (sleep) and appear (awake) during the course of time. In this case, algorithms used for static SPPs would have to be restarted whenever there is a change in network topology, and this would lead to excessive overhead

and computational costs. The need therefore arises for algorithms which can easily adapt to changing environment while maintaining optimal source-destination paths [40, 202].

2.2.1.2 Fuzzy Shortest Path Problem (FSPP)

Usually, edge weights or costs are represented as precise real numbers. However, as a result of the fact that in real-world scenarios such as decision making [63], risk assessment [261], and pattern recognition [251], these weights are naturally imprecise/vague, the Fuzzy Shortest Path Problem (FSPP) represents them as fuzzy numbers, and hence, the summing operation of fuzzy numbers as well as their comparison have to be redefined, as can be seen in [215] and [233].

2.2.1.3 Stochastic Shortest Path Problem (SSP)

In the stochastic SPP (SSP), edge costs are not deterministically known ahead of time. This arises in real-world applications as a result of the possibility of link failure, variable travelling time due to traffic congestions, and so on. Path costs are hence modelled as continuously or discretely distributed random variables, and at each node a probability distribution must be selected over all possible successor nodes which leads to the destination node and has minimum expected cost [42, 46, 157].

The SSP is one which has been studied over a long period of time since Frank [81] proposed a solution to it in 1969. However, it was first formulated in 1962 by Eaton and Zadeh [69] who termed it a "problem of pursuit", and introduced the conditions for optimal solution. In 1991, Bertsekas and Tsitsiklis [26] introduced the term "Stochastic Shortest Path", allowing negative costs contrary to the positivity condition earlier introduced by Eaton and Zadeh [69]. This condition had already been weakened to cost non-negativity in Bertsekas [27].

A generalization of SSPs are Markov Decision Processes (MDPs), which are models that evolve in both stochastic and non-deterministic ways. An MDP consists of *states,* and is executed in rounds. At each round, the MDP is in a given state, each state having a set of possible associated actions. A *control/action strategy* selects an action, and each action is associated with a probability distribution over all associated subsequent states. Based on the probability distribution, transition is made into a selected next state, and with each state

transition, an associated cost is incurred [30, 189]. The goal is therefore finding a control strategy (or policy) which minimizes the total incurred cost from the initial to final state with a probability 1 of reaching the final state. [26, 30]. A formal definition of an MDP is presented in [30] as:

1. A finite state space $S = \{1,\ldots,n\}$
2. A finite set of controls (actions) $U(i)$ for each state $i \in S$
3. Transition probabilities $p(i,u,j)$ associated with all $u \in U(i)$ where $p(i,u,j)$ is the probability of transitioning to state j from i after selecting action u.
4. A cost $g(i,u)$ associated with $u \in U(i)$ and $i \in S$

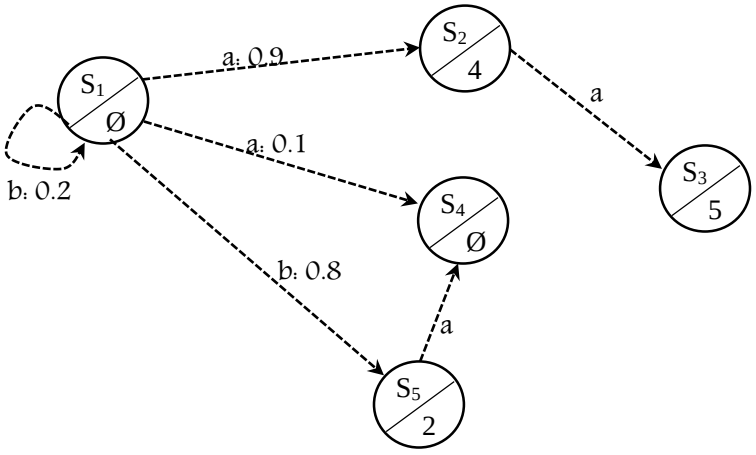


Figure 2.2: Simple Markov Decision Process

Figure 2.3 briefly illustrates an MDP. In the figure, S_1 is a state with associated actions *a* and *b*. Each associated action has a probability distribution over the states it leads to. For example, from state S_1 with action *a*, two states can be transitioned to, S_2 and S_4, with probabilities of 0.9 and 0.1 respectively. Also, a transition from S_1 to S_5 has an associated cost of 2.

A policy is a sequence of functions, $(\mu_0, \mu_1, \ldots)$, with μ_k mapping states to actions such that action $\mu_k(i)$ will be executed in state $x_k = i$ at time *k*, as long as $\mu_k(i) \in U(i)$. That is, a policy is a sequence which defines actions (mapped to states) which are to be performed at any given point in the decision process. The policy is said to be *stationary* when all μ_k are the

same, i.e. a sequence of functions of the form $(\mu, \mu, \ldots)$. The total cost incurred by a given policy, π starting from state x_0, is given by

$$J_\pi(x_0) = \lim_{N\to\infty} E\left\{\sum_{k=0}^{N-1} \alpha^k g\big(x_k, \mu_k(x_k)\big)\right\} \tag{2.10}$$

where $\alpha \in [0,1]$ is known as the discount factor which discounts future costs at a geometric rate. The aim in an MDP problem is therefore to find an optimal policy π^* satisfying

$$J^*(i) \stackrel{\text{def}}{=} J_{\pi^*}(i) \leq J_\pi(i), \qquad i = 1, \ldots, n \tag{2.11}$$

for every other policy [30].

Two exact (dynamic programming) algorithms exist which are used to solve the MDP problem. Value Iteration (VI), proposed by Bellman [23], iteratively computes an optimal cost vector *J** which satisfies the *Bellman Optimality* equation when $\alpha < 1$ (which implies that there always exists an optimal stationary policy) as an update rule:

$$J^*(i) = \min_{u\in U(i)} g(i,u) + \alpha \sum_{j=1}^{n} p(i,u,j) J^*(j) \tag{2.12}$$

Once the optimal cost vector is obtained, an optimal policy is determined thus:

$$\pi^*(i) = \arg\min_{u\in U(i)} \left[g(i,u) + \alpha \sum_{j=1}^{n} p(i,u,j) J^*(j) \right] \tag{2.13}$$

The algorithm is said to have converged when successive values of *J* are the same, or when the difference between them is sufficiently small [130].

Policy Iteration (PI) was also proposed by Bellman [23]. It begins with an arbitrary policy and computes its value function thus:

$$J_\pi(i) = \left[g\big(i, \pi(i)\big) + \sum_{j=1}^{n} p_{i,j}\big(\pi(i)\big) J^\pi(j) \right] \tag{2.14}$$

It proceeds to iteratively improve on π until convergence or an acceptable negligible error is obtained.

An SSP therefore specializes an MDP with the following properties [30]:

- Non-discounted (i.e. $\alpha = 1$) rewards
- A designated goal/target self-absorbing state which represents the destination and is cost-free
- A finite state and action space
- The state space maps to the node set of a graph
- Set of actions and their associated probability distributions maps to edge set
- A policy maps to a path through the graph from source to destination node

The SSP therefore seeks an optimal policy (path), π with minimum total associated cost which reaches the destination node with a probability of 1. To guarantee the existence of such an optimal policy, the following conditions have to be met [26]:

1. There exists a *proper policy* which is guaranteed to reach the destination node from any source node.
2. All costs are positive.

The exact methods discussed earlier (VI and PI) are dynamic programming methods, which involve a lot of computational burden. For example, the VI method has to update the value function for the entire state space at each iteration [130]. As a result, heuristics have been successfully developed to solve the SSP and its variants.

In [40], a multi-criteria A* algorithm is proposed to compute the least expected time path in Stochastic Time-Dependent networks (STDs), which are basically SSPs with costs represented as time chunks. They worked specifically on STD networks which satisfy the stochastic first-in-first-out property (S-FIFO), testing their results on large scale road networks. They demonstrated that their proposed algorithm performs significantly better on problems which have the S-FIFO property over algorithms which do not exploit that property. In [240], a new admissible heuristic search-based family of algorithms called FRET (Find, Revise, Eliminate Traps) was introduced to solve a newly introduced class of MDPs, the Generalized SSP (GSSP), which relaxes some of the conditions stated in [26] for SSPs to allow for negative costs and discounted-reward models. They investigated the performance

of their method against VI on the GSSP and found theirs to perform better in terms of speed and convergence, even in cases where VI could not converge. However, the proposed algorithm suffers the same problem of having to go through the entire state space, hence running out of memory on some problem instances.

As per maximizing path reliability, Chen, et al. [41] carried out a study which proposed two algorithms, a multi-criteria label-setting and an A* algorithm to maximize path reliability in cases where link travel times follow a normal distribution. Based on the definition of α–reliability, they tried to minimize the travel-time budget (amount of time) required to ensure a certain level of on-time arrival probability. They classified travelers risk attitude towards travel time uncertainty as "risk-averse" with on-time arrival probability, $\alpha > 0.5$, "risk-neutral" with $\alpha = 0.5$ and "risk-seeking" with $\alpha < 0.5$. They examined the performance of their algorithms on large-scale networks, showing that their algorithms performed significantly better than others under risk-averse scenarios. However, as they stated, their study was restricted to normally distributed link times, and hence its performance on links following other distributions remains unknown.

In [30], an improved Real-Time Dynamic Programming (RTDP) algorithm is proposed which is guaranteed to terminate in finite time and return an optimal path, an improvement on that proposed in [19] which converged asymptotically, providing no guarantee to convergence in finite time. They compared the performance of the improved RTDP algorithm with the Value Iteration (VI) algorithm based on the Bellman equation [23], showing that it outperformed the VI algorithm by orders of magnitude on some benchmark problems with large state spaces. Vladimirsky [226] defined and discussed a class of multimode SSPs, defining conditions under which label-setting methods can be applied to them.

In the field of Swarm Intelligence, Sudholt and Thyssen [214] proposed a simple ACO for SSPs, considering various noise models from general, arbitrary noise to independent gamma-distributed noise, showing instances where ACO can discover optimal paths efficiently. They showed however, that for independent gamma-distributed noise, ACO took exponential time to find a good approximate solution. Momtazi, et al. [157] proposed a PSO method to solve the SSP, comparing their results with those reported for Distributed Learning Approaches

proposed in [154] and [22] on 10 and 15-instance SSPs. They reported that their algorithm outperformed those with which it was compared, achieving optimal paths with much fewer iterations.

2.2.1.4 Constrained Shortest Path Problems (CSP)

The addition of constraints on the resource(s) consumed by including a given edge in the path, as well as the restriction of the graph type to digraphs leads to various variants of the SPP. Though the elementary form of SPP is solvable in $O(n^2)$ time or less [266], the addition of one or more constraints makes it NP-complete [152]. As edges are added to the path under construction, the amount of consumed resources accumulates, each resource being associated with an upper limit/constraint which must not be exceeded. The Constrained Shortest Path Problem (CSP), which has also been referred to in literature as the Resource-Constrained Shortest Path Problem (RCSPP) or the Shortest Path Problem with Resource Constraint (SPP-RC), is a generalization of the SPP with constraints, where *number of resources* ≥ 1. Two well-known specializations of the CSP are Weight-Constrained SPP (WCSPP) where *number of resources* = 1, and Shortest Path Problem with Time Windows (SPPTW) where *number of resources* = 2 [112].

As is highlighted in [112], CSPs can be classified by the manner of resource accumulation, fitness definition, under-lying network, and the existence of path-structural constraints excluding specific paths. Examples of such variants include Maximum Capacity Path Problem (Bottleneck SPP) [184], Balanced SPP [148], Weight-Constrained SPP with Replenishment [28], Variance-Constrained SPP [206], Equity-Constrained SPP[91], and the Hop-Constrained SPP [54]. A variant worthy of note is the Multi-Objective SPP (MOSPP) [236] where all pareto-optimal solutions which minimize two or more objective functions are sought. The variant with only two objectives is known as the Bi-criteria SPP [43]. Multi-objective optimization is discussed in section 2.4, while CSPs are treated in much greater detail in [112] and [152].

For CSPs, Label-setting [64, 68, 198] as well as Label-correcting [28, 125] algorithms have also been developed. Lagrangean Relaxation methods have also been used [33, 160] for the

Resource-Constrained Shortest Path Problem. These algorithms are explained in the following section.

2.2.2 Solution Approaches to the SPP: Exact Methods

As has been earlier stated, the elementary form of the SPP assumes a single source and destination node. The aim here is to find the path of minimal cost from the identified source node to the destination node [65]. A slightly varied but similar case is when the aim is to find the shortest path from a single source destination to all other nodes in the graph. Both of these problems are referred to as the Single-Source Shortest Path Problem (SSSP) [170, 218].

Due to the tractable nature of the SPP, most of the research effort on it has been towards achieving greater computational efficiency. Exact algorithms which have been proposed to solve this problem usually fall into two categories: *label-setting* and *label-correcting* algorithms [83]. Note that many of these solution methods for the SSSP have been extended or applied to solve other variants of the SPP derived by introducing constraints, seeking for multiple objectives or re-defining the minimization function.

2.2.2.1 Label-setting

Label-setting algorithms associate a *label* $L_{(i)}$ to each node which represents an upper bound on the cost of getting from the source node, *s* to that node through a preceding node $P_{(i)}$. Each node is also associated with a status $S_{(i)}$ which could either be *unreached, labeled* or *scanned.* The algorithm begins by setting $L_{(i)} = \infty$, $P_{(i)} = null$, and $S_{(i)} = unreached$ for all nodes, and is initialized by setting $L_{(s)} = 0$ and $S_{(s)} = labeled$. At each iteration, a scan is made from the current labeled node across all its emanating nodes, and they are each processed thus: If $L_{(i)} + c_{ij} < L_{(j)}$, set $S_{(i)} = labeled$, $P_{(j)} = i$ and $L_{(j)} = L_{(i)} + c_{ij}$, and let *j* become the current node. If $L_{(i)} + c_{ij} \geq L_{(j)}$, set $S_{(j)} = scanned$, where (i,j) is an edge from *i* to *j*. Out of the *unreached* nodes, the one with the minimum $L_{(i)}$ is selected, its $S_{(i)}$ value is set to *labeled* and it becomes the current node. Note that once a node has become *labeled,* it is no longer processed. This state is also referred to in literature as *permanently labeled*.

The algorithm terminates either when all nodes have been processed (in the single source, multiple destination problem) or when the destination node, *t* is reached (in the single source,

single destination problem) [10, 83, 256]. It can be inferred, therefore, that algorithms which solve the single-source, multiple destinations case inherently solve the single source, single destination case.

A formal general statement of label-correcting algorithms is presented in **Algorithm 2-II**:

Algorithm 2-II: Label-Setting Algorithm

1. Initialize *scanned* list
2. For i in node-set A:
3. Set $L_{(i)} = \infty$, $P_{(i)} = null$, and $S_{(i)} = unreached$
4. Set $L_{(s)} = 0$ and $S_{(s)} = labeled$
5. Add s to *scanned*
6. Set s as current node, i
7. For all nodes, j adjacent to i:
8. If($S_{(j)} != labeled$)
9. If($L_{(i)} + c_{ij} < L_{(j)}$)
10. Set $L_{(j)} = L_{(i)} + c_{ij}$
11. Set $S_{(i)} = labeled$
12. Set $P_{(j)} = i$
13. Remove i from *scanned*
14. Set j as current node, i
15. Else
16. Set $S_{(j)} = scanned$
17. Add j to *scanned*
18. Select node n with minimum $L_{(m)}$ from *scanned*
19. Set n as current node, i
20. If ($i == t$)
21. Stop
22. Else
23. Go to Step 7

Label-setting algorithms have the advantage of having better worst-case complexity bounds than label-correcting algorithms, as well as being useful for solving single-source single-destination SPPs. However, they are less general. That is, they do not apply to all classes of problems such as problems with negative edge weights [10].

2.2.2.2 Label-correcting

In label-correcting algorithms, the same variables and initialization procedures are maintained as with label-setting. The key difference with label-correcting algorithms rises in the labelling procedure. In label-correcting algorithms, no label is denoted as *permanent* until the final iteration. In addition, the procedure of selecting the next node to be processed differs. While label-setting algorithms select the adjacent node with the minimum $L_{(i)}$, label-correcting algorithms mostly use a First-In-First-Out (queuing) procedure, regardless of the $L_{(i)}$ value. It processes each node in the same way, updating $L_{(i)}$ values when a lower alternative is found [10]. Label-correcting algorithms are suitable for problems of a more general class such as the Shortest Path Problem with negative edge weights, though with that generality some level of performance optimality in terms of computational cost is sacrificed [24].

2.2.3 Solution Approaches to the SPP: Non-Nature-Inspired Heuristics

Improvements on the performance of these exact algorithms have been attempted as a result of their excessive computational intensity when applied to real-time one-to-one traffic networks. These are in the form of heuristics which take advantage of available information about the location of nodes in the graph, network structure, and other application-specific information to reduce the search effort involved in the algorithm. This in turn reduces the practical computational time. Some of the notable heuristics in this regard are considered: *A* algorithms*, algorithms involving *bi-directional search*, and *branch pruning* methods [83].

*2.2.3.1 A**

The A* algorithm was first introduced by Hart, et al. [100] in 1986 as an admissible (guaranteed to converge) method for solving the SPP. It can be seen as a generalization of the label-setting algorithm [166] since it involves node scanning, expansion and selection. However, it makes use of a heuristic evaluation function $F_{(i)} = L_{(i)} + e_{(i,d)}$ as node labels, with the algorithm proceeding in a best first search manner. That is, it maintains two lists: an *open* and a *closed* list. The *open* list is ordered based on nodes' F values and contains all nodes

which have not been expanded yet. The *closed* list contains nodes which have already been expanded. See **Algorithm 2-III** for details of the A* algorithm [195].

Algorithm 2-III: A* Algorithm

1. Set *s_pred* = *null*, $F_s = 0$, and place *s* in *open*
2. For all nodes in *open*
3. Select node *i* with minimum F_i value
4. If (*i* == *t*)
5. Stop
6. For all *j* adjacent to *i*
7. If (*j* is not in *closed*)
8. If (*j* is not in *open*)
9. Add *j* to *open*
10. Else
11. If ($new\_F_j < old\_F_j$)
12. $F_j = new\_F_j$
13. *j_pred* = *i*
14. Add *i* to *closed*
15. Stop

where *s_pred* and *j_pred* refer to the previous node to nodes *s* and *j* respectively in a path, *s* represents the source node, *t*, the destination node, and $e_{(i,d)}$ is an optimistic estimate of the sum cost from node *i* to node *d*..

The estimation of the $e_{(i,d)}$ function is a key factor in ensuring the admissibility of the algorithm. As long as $e_{(i,d)}$ is selected in such a way that it is a lower bound (and not an overestimation) of the actual distance from node *i* to the destination node *d*, the algorithm is guaranteed to find the optimal solution. Such an estimation is usually done based on information from the problem domain. A classic example given in [100] is the problem of finding the shortest path between cities on a road network. Estimating the value of $e_{(i,d)}$ based on the cost of travelling by air from city *i* to city *d*, which is the lowest possible travelling cost by road, would guarantee the optimal path being found. The A* algorithm has the advantage of being able to solve the SPP without prior knowledge of the network, constructing the network concurrently with its solution. Another advantage is that the search

space is reduced to less than 11% if the edge costs are generated by Euclidean distances [89], and hence, computational time is greatly reduced [18, 83].

2.2.3.2 Bi-directional search

Bi-directional search algorithms attempt to improve the search for the shortest path by splitting the graph into two and searching in a label-setting fashion from both ends: one forwards from the source node, and the other backwards from the end node. The first publication introducing bi-directional algorithm was made by Dantzig [57] but it was described as being "ambiguous and vague" [181]. It was in Nicholson's work [165] that a clear and concise definition of a bidirectional algorithm was made. His contributions are based on the following definitions [181]: $d(x)$ denotes an upper bound on the cost of getting from s to node x and $d_t(x)$ denotes an upper bound on the cost of getting from node x to t. A contains nodes which have been labelled as *permanent*, B contains nodes which have been labelled *scanned* and C contains nodes which are still *unreached*. Sets D, E and F are symmetrical sets corresponding to A, B and C respectively in the backward-directional search. Two key aspects of Nicholson's algorithm were the manner in which searches in both directions were made, and the termination criteria. Regarding the search order, he proposed an improvement to the alternating search order of Dantzig [57]. While Dantzig's search order alternated between the forward and backward searches, Nicholson [165] proposed that instead of alternating, the search should be based on which node is closest to the source and destination nodes. That is, if the node is closer to s, the search is conducted in a forward manner, and if it is closer to t, the search is conducted in a backward manner. Where there is a tie, the nodes were to be expanded simultaneously (within the same iteration). Secondly, he defined an improved termination criteria thus [182]:

$$\min_{x_i \in A \cap D} d(x_i) + d_t(x_i) \leq \min_{x_i \in B} d(x_i) + \min_{x_i \in E} d_t(x_i) \tag{2.15}$$

However, Dreyfus [67] proposed an improvement to this termination criteria. His improvement involves the algorithm terminating immediately some node n is selected which is contained both in A and D, and the shortest path is achieved by examining $n \cup Y$, $where\ Y = \{x_i : x_i \in A \cap E\}$:

$$min\ (d(x_i) + d_t(x_i), d(n) + d_t(n))\ \forall x_i \in Y \tag{2.16}$$

Pohl [181] proposed an improvement to Nicholson's search procedure by stating that treating all ties simultaneously is unnecessary. He proposed that the ties be handled individually, which would lead to time efficiency, especially when the tie happens at a point where a node occurs in the intersection of *A* and *D*.

It is important to note that thus far, no heuristic component had been introduced to the bi-directional search. However, Pohl in [181, 182] introduced A* heuristic search to both directions of the bi-directional search with an aim of improving its performance. He called the algorithm the Bi-directional Heuristic Path Algorithm (BHPA) and noted that an improvement in performance would only be achieved by the heuristic search if both searches met near the middle of their separation. This did not seem to happen in his computations. The forward and backward search trees would usually grow almost to completion before there would be an intersection, thereby making the algorithm twice computationally intensive than the uni-directional version. He concluded that the heuristic search trees were highly unlikely to intersect while growing in opposite directions, likening the situation to a scenario of two missiles independently aimed at each other's base 'missing' each other. This conclusion was taken for granted in the works of other researchers like de Champeaux [60], Sint and de Champeaux [205] and Fu, et al. [83], and led to the development of 'front-to-front' algorithms which directed a search tree at the front of the opposite search tree, and not at the root [60, 183, 205]. The front-to-front algorithms were however excessively computationally demanding or had no admissibility [58, 118].

The notion of search trees 'missing' each other like missiles was refuted by Kaindl and Kainz [118], who stated that rather than the search trees missing each other, they 'pass through' each other. They also identified the cause of the poor performance of the previous bi-directional heuristic algorithms as the major computational effort spent on searching for a better solution than that which is found when the search trees initially meet. Even when the optimal solution is found, additional effort is spent on ensuring that it is indeed the best solution possible, since the algorithm does not 'know' that the optimal solution is indeed optimal until it has generated all the remaining nodes. They then devised a new generic

approach to bi-directional heuristic search which utilizes dynamic heuristic front-to-end evaluations which can be embedded efficiently. They also showed instances of problems where bi-directional search is more suitable and efficient than unidirectional search. In addition, Ikeda, et al. [111] proposed a way to combine a bidirectional version of Dijkstra's algorithm with A* search in an admissible way, and based on his work, some improvements on bidirectional shortest path algorithms were achieved in [126].

Major recent improvements on bidirectional heuristics as regards search space reduction, and hence improvement in computational efficiency, include the scalar projections method proposed by Whangbo [241] and the new lower bound proposed by Pijls and Post [179].

2.2.3.3 Branch pruning

Branch pruning involves a continual evaluation of the edges in the graph to find weak ones which are unlikely to form part of the solution and directing the search away from them. According to [83], the edges are evaluated based on the inequality

$$L_{(i)} + e_{(i,d)} \leq E_{(o,d)} \tag{2.17}$$

(where $e_{(i,d)}$ is the estimated cost from node i to the destination node, d, and $E_{(o,d)}$ is the estimated cost from the source node to the destination node), and edges which violate this inequality are pruned from the search.

Just like in A*, the performance of branch pruning algorithms depends on the accuracy of the estimated costs. Hence, a weakness of branch pruning algorithms arises when the algorithm is not able to find the optimal path as a result of the estimation functions evaluating to relatively low values. This could lead to the optimal path being pruned from the search. In fact, there exists the chance that it succeeds at finding no solution at all because all branches from the source node are pruned before the destination node is reached. A way of remedying this fault is presented in [82].

Of the three heuristic approaches discussed, A* heuristic search has been found to reduce computational intensity by up to 50%. The viability of bidirectional search algorithms has

also been discussed and proven, as seen in [118], [241] and [195], against the conclusion made in [83].

2.3 The Submission Decision Problem

Paper publication rate, citation rate, citation networks and other matters related to scientific research publication have been widely studied over the years and even more intensely in recent times, as they relate to issues which lie close to the heart of every researcher, easily summarized as the desire for recognition and relevance. An easy (and widespread) way of judging a researcher's level of achievement as regards these two goals is the total number of citations which they have accumulated over their paper publication history. Measures such as the Impact Factor and h-index all in one way or the other factor citation count into their assessment of a journal's relevance or an author's level of contribution to the scientific community.

Despite the recent introduction of the h-index [107] as a response to the wide criticisms leveled at the use of the Impact Factor [103], the Impact Factor still continues to be widely used by scientists to judge the quality of a journal, and by universities and research institutes, in conjunction with other measures like the h-index and citation count, to judge the quality of an author's contribution (number of articles published in high Impact Factor journals) to knowledge. These judgements are then used to make administrative decisions like promotion, appointment/election to prestigious positions, and awarding of prizes [200]. As a result, deciding which journal to send a manuscript has been and still remains a crucial decision problem faced by scientists all over the world. A naïve solution would be to simply sort all target journals by their Impact Factor and then work down the list, but various other important factors come into play such as whether the manuscript gets accepted, the time it takes till it gets published as well as how long it takes before the authors get feedback (review outcome) on the submitted manuscript. These factors have to be taken into consideration in order to strike an acceptable balance between getting a reasonably high number of citations (which generally determines and/or indicates the relevance of the work), and the time it takes to get the desired citations.

Various measures of impact which differ with respect to the target of the assessment being carried out have been proposed in literature over the years. Three main assessment targets found in literature are author, paper and venue (journal) [200], and various impact measures have been proposed which assess one or a combination of these targets. An old but still popular impact measure is the citation count [200], which is a summation of all citations obtained by a paper, venue or author over some time period. Citation count is more widely applied in the assessment of a paper's worth than venue or author, but when used to measure an author's impact, it has a few shortcomings. One of such shortcomings is its being easily skewed by a few 'popular' papers which have very high number of citations but were co-written with other authors, and hence may not represent the individual's capabilities or contributions. It also gives undue weight to review articles which are generally highly cited but make no significant original contribution to scientific knowledge [107].

A more recent and widely used measure of an author's impact is the *h*-index, which was proposed by Hirsch in 2005 [107], and is computed as the maximum *h* for which an author has *h* papers with more than *h* citations. To put it simply, an author has an *h-index* of *x* if he has published *x* papers which each have at least *x* number of citations. Though the validity of the *h-index* has been attested to by various researchers [108, 146], a major weakness was pointed out by Shah and Song [200] which is its failure to offer a fair comparison between authors whose publications' citation distribution differ significantly. Pan and Fortunato [173] pointed out another key weakness of the h-index which is the fact that its assessment covers the entire career of an author, whereas authors' performances vary significantly with time. A good case in point is that of Nobel Laureates, whose publication trajectory tends to significantly fall after they have been awarded the Nobel Prize, mostly as a result of other non-publishing engagements that come as a result of the award. Also, some authors only make a single significant contribution to science, while others continue to make great contributions over their career horizon. Thus, a performance measure which collapses important publication/performance dynamics into a single value may not be a suitable tool for comparing between different authors' performance at a particular point in time, and one

of the key reasons why assessment measures are created is the performance of such comparisons.

Pan and Fortunato [173] therefore proposed another impact measure which extends the ubiquitous Impact Factor (IF) to authors. The usual (and widespread) use of IF in the context of authorship is basically to assess performance based on the number of publications that have been made in relatively high IF journals within the respective fields. Pan and Fortunato however made a formal extension of the IF to author assessment. They reported that their new measure is dynamic in nature and overcomes the shortcomings found in the usual definition of Impact Factor as regards its failure to capture current 'hot' researchers, papers, journals and so on. They termed the extended IF measure Author Impact Factor (AIF), and conducted varied empirical studies to prove its validity. Thus, it can be concluded that IF and IF-based measures are still a widely-used and relatively viable measure of assessing an author's research impact.

Assessing the quality of a research paper is a critical process which authors undergo, as they have to make choices about which publications to cite out of a large pool of related articles, the quality of publications cited within a paper tending to contribute to its general quality as perceived by the research community. To the best of our knowledge, the major and most widely-used measure of a paper is the citation count [200, 211], a fact which is demonstrated by its widespread use in author citation network analyses [250] and models for predicting future/long term impacts of scientific publications [90, 193]. It is therefore a reasonable conclusion that a schedule which maximizes the citation count of a paper will generally increase its recognition and in turn lead to higher relevance in the scientific world.

Deciding which journal to send a paper to, which is the focus of this work, obviously has a direct correlation with the way a journal's quality is measured. It remains an established fact that the Journal Impact Factor (JIF) is the most widely-used measure for assessing journal quality, despite the various faults which have been highlighted in the literature [16, 76, 200]. The idea of an Impact Factor was first introduced by Garfield in 1955 [72], but formalized by Garfield and Sher in the late 1960s [84] as a means of deciding which journals would be selected for the *Science Citation Index (SCI).* The most critical factors taken into

consideration at its creation were recognizing large reputable journals with high citation count while not neglecting ones which were not as large, but were also important in their fields. To put it succinctly, they wanted to compare journals fairly irrespective of their size. The JIF was defined by them as [200]:

$$J = \frac{\sum_{p \in P} C(p)}{|P_2|} \tag{2.18}$$

where P_2 is the set of papers published within the last two years, and *C(p)* is the citation count of paper *p,* and $|P_2|$ refers to the number of 'citable publications' in the last two years [32].

In simple terms, the JIF is the total sum of citations which have been accumulated in the current year by all articles which were published in a journal within the last two years divided by the number of citable publications made in the same journal within the last two years. The JIF is reported yearly by Thomson Reuters in their Journal Citation Reports (JCR), which has assumed the position of a legitimate authority in ranking journals [103, 135]. The strong points of the JIF, as pointed out in literature are its comprehensibility, stability, simplicity and popularity [87]. On the other hand, quite a few weaknesses of the JIF have been pointed out in literature, many of which relate to its misuse in comparing journals across unrelated fields which have varying sizes and publication rates [32, 98, 103], the ambiguity of the term 'citable publications' and its lack of justification and tacit encouragement of 'negotiations' with Thomson Reuters [16, 209], the shortness of the 2-year horizon which puts fields where citation rates rise relatively slowly at a disadvantage [32, 108], its susceptibility to self-citation and coercive citation measures (journals directly or indirectly inferring that citing self-published papers will influence acceptance) [16, 242] and so on. As a result of this widespread criticism, other measures have been proposed, notable among which are the PageRank system used by Google [76], EigenFactor [29], an extension of h-index to journal assessment [32], and the s-index [200].

Considering the fact that the JIF is a citation-based measure, most flaws associated with it which are a result of its use of citation count would also apply to measures which incorporate citation count in their computations, such as the h-index. As has been stated earlier, though

the JIF has been so widely criticized, it still remains the most widely-used journal assessment measure. This implies that it is still very much factors into an author's assessment of a journal and administrators' assessment of an author [135], [212], [87], [70], and this justifies the inclusion of the JIF in the SDP model developed by Salinas and Munch [196].

2.3.1 Models for the SDP

As was earlier stated, the SDP problem was formally introduced in 2105 by Salinas and Munch [196]. It involves an author deciding which submission order to follow in submitting their manuscripts to journals, based on a goal of maximizing citation count while minimizing the number of resubmissions on one hand, or the amount of time it takes from submission to publication on the other hand. While modeling this problem, they considered various important factors like the *acceptance rate* of the journal, the average amount of *time spent in review*, the probability of getting scooped (*scoop ratio*). Being scooped refers to a similar article being published by a competing author before an author has a chance to publish his work. Following Markov Decision Process theory, they proposed the following model which gives the expected number of citations, *C*, over *N* submissions in a given order [196]:

$$C = q^{-1}\sum_{j=1}^{N}\alpha_j\lambda_j\,[T - \sum_{k=1}^{j}\tau_j - (j-1)t_R]^{+}\prod_{k=1}^{j-1}(1-\alpha_k)(1-s)^{\tau_k+t_R} \tag{2.19}$$

where α is the acceptance rate, λ is the expected number of citations (taken as the JIF), T is the time horizon over which the author intends to maximize number of citations, τ is the time from submission to publication, t_R is the amount of time spent in revision, s is the scoop probability per day, and q is a normalization constant. The + superscript indicates that negative values within the inner summation term will be replaced with 0. All time units are in days.

The model representing the expected number of submissions, *R*, is given as:

$$R = q^{-1}\sum_{j=1}^{N} j\alpha_j\prod_{i=1}^{j-1}(1-\alpha_i)\,(1-s)^{t_R+\tau_i}\mathrm{H}\left(T - \sum_{k=1}^{j}\tau_j - (j-1)t_R\right) \tag{2.20}$$

H is the Heaviside function which ensures that the expected number of citations is calculated within the specified time horizon, *T*. It takes a value 1 when its argument it positive, and 0 otherwise.

To estimate the expected number of days from first submission to acceptance, *P*, they proposed the following model:

$$P = q^{-1}\sum_{j=1}^{N}[\tau_j(j-1)t_R]\alpha_j\prod_{i=1}^{j-1}(1-\alpha_i)(1-s)^{t_R+\tau_i}\,\mathrm{H}\left(T-\sum_{k=1}^{j}\tau_j-(j-1)t_R\right) \quad (2.21)$$

They also modeled a means through which journal's comparative desirability can be evaluated, factoring the mentioned variables like acceptance rate and so on. The index value *V* which depends on *T* being large relative to publication and revision times, of a given journal *j*, is given as:

$$V_j = \frac{\alpha_j\lambda_j\left(1-\frac{\tau_j}{T}\right)}{1-\left(1-\frac{\tau_j}{T}-\frac{t_R}{T}\right)(1-\alpha_j)(1-s)^{t_R+\tau_j}} \quad (2.22)$$

A few assumptions were made for model simplification purposes, and as a result of the unavailability of data. Some key assumptions among them were that all manuscripts are of equal value to all sampled journals, that time to publication is the averagely same for all papers submitted to a journal, that acceptance and scoop probability are the same for all manuscripts, and obviously the use of a static JIF measure to represent the expected number of citations for a journal. Though quite a few of these assumptions are not exactly realistic, it is believed that they do not affect the quality of the models to a great extent.

2.3.2 Solution Approach to the SDP: Metropolis Algorithm

As a result of the very recent introduction of the SDP, the only existing method which has been applied to it is the Metropolis algorithm [196]. The Metropolis algorithm, introduced by Metropolis, et al. [153], is a Markov Chain Monte Carlo [86] algorithm generalized as the Metropolis-Hastings algorithm [102]. It has been very widely applied to various problems in the field of physics and computer science, and is also applied in Simulated Annealing, an algorithm which has been effectively used for a wide variety of COPs [21]. It is initialized

by randomly generating an initial solution. The initial solution is progressively improved by generating other random solutions and testing to see if they are better than the current solution. The new solution becomes the current one only if it is better by a specified probability. It is presented in **Algorithm 2-IV** [47].

Algorithm 2-IV: Metropolis Algorithm

1. Initialize $x_0 \in \mathcal{X}$
2. Repeat for i = 1 to n-1
    a. Generate x_i from $Q(x_i, x)$
    b. Generate u from $\mathcal{U}(0,1)$
    c. If $u < \frac{\rho(x)}{\rho(x_i)}$, then
        i. Set $x_{i+1} = x$
    d. else,
        i. Set $x_{i+1} = x_i$
3. Return $x_0, x_1 \ldots x_n$

where Q is a stochastic matrix ($Q = (Q_{ab}: a, b \in \mathcal{X})$) which represents the stochastically distributed state space of the Markov Chain, and $\rho(x_i)$ in proportional to the probability of x_i.

In [196], the Metropolis algorithm is applied to SDP by representing states (x) in the Markov chain as submission schedules, and transitions between states (moving form x_i to x_{i+1}) as a random swap of the positions of two journals. Also, $\rho(x)$ is defined as the *C, R, or P* evaluation of the submission schedule, *x*. Their adapted algorithm for the *R* and *C* bi-objective optimization case (without loss of generality) is presented in **Algorithm 2-V**.

Algorithm 2-V: Metropolis Algorithm for SDP

4. Initialize $x_0 \in \mathcal{X}$
5. Repeat for i = 1 to n-1
    a. Generate x from $Q(x_i, x)$
    b. Generate u from $\mathcal{U}(0,1)$

c. If $u < \frac{C(x_i)}{C(x)}\ AND\ u < \frac{R(x)}{C(x_i)}$, then

i. Set $x_{i+1} = x$

d. else,

i. Set $x_{i+1} = x_i$

6. Return $x_0, x_1 \ldots x_n$

In the following section, we discuss NI-based heuristics for both the TSP and SPP, bearing in mind that for the SDP, no existing literature which applies NI-based heuristics has been found.

2.4 Solution Approaches to the TSP and SPP: Nature-Inspired Heuristics

In the field of meta-heuristics, methods have also been developed to solve the TSP and SPP. Specifically, Nature-Inspired algorithms have been widely applied to these and many other COPs in literature [248]. Nature-Inspired algorithms are basically algorithms which derive their inspiration from natural phenomena, either biological, physical, chemical or otherwise. A very large percentage of these algorithms are based on some characteristics of biological systems. These algorithms are called bio-inspired algorithms, and within them is a very popular field known as Swarm Intelligence [79].

Swarm Intelligence (SI) is a branch of computational intelligence that mimics the collective intelligence of autonomous self-organizing agents as inspired by social animals (swarms). This leads to the derivation of some form of emergent behavior that could lead to the solution of complex problems [3]. The fascinating ability of unsophisticated individuals in these real swarms or colonies to solve complex problems and exhibit complex organization and coordination without central control of any kind (known as *emergence*) has attracted both biologists and computer scientists to both study them, and apply their methods in solving life's problems and challenges.

Fister Jr, et al. [79] give a broad grouping of NI heuristics as SI-based, Bio-inspired but not SI-based, Chemistry and Physics-based, and Other algorithms. The discussion of NI methods to solve the TSP and SPP in this work is restricted to two popular SI-based techniques, ACO

and PSO, and two popular Bio-inspired but not SI-based techniques, GA and Physarum algorithms.

2.4.1 Genetic Algorithms (GA)

Genetic Algorithms (GAs) can be traced as far back as the 1940s when the use of evolution to solve problems was suggested by Turing [222]. Though the introduction of GAs has been attributed to Holland [109], other researchers had been using similar methods before him [53]. GAs are inspired by the biological processes of natural selection and genetics, and how better individuals evolve and emerge from these processes over many generations. In GAs, candidate solutions are represented as *chromosomes which* are made of *genes*. Genes can take 2 or more values, which are known as *alleles* [109].

GAs usually start with the generation of an initial population of chromosomes which encode feasible solutions to the problem. Two major and widely-used methods exist for the encoding of solutions as chromosomes: binary encoding and real encoding [53]. In binary encoding, the solution is broken down or encoded into a binary string which will be decoded at the fitness evaluation stage. In real encoding on the other hand, chromosomes are made of actual parameters that make up feasible solutions, invalidating the need for a decoding procedure.

Once the initial population is generated, three distinct reproduction operators act on the population to produce successive generations [53]:

- *Selection*: This operator uses some means (tournament, roulette wheel) to select the parents which will be used to reproduce children which will form the next generation.
- *Crossover*: This operator is said to be the soul of GAs. It combines parent chromosomes to produce a child. Various crossover schemes exist, the major of which are the single point, multi-point and uniform schemes.
- *Mutation*: The mutation operator serves as a means of maintaining diversity in the population, or exploring the search space. It randomly selects and alters the value of one or more genes in the chromosome. Mutation rate can be used to bias the algorithm either towards exploration or exploitation.

Various GA methods have been used to solve the SPP with varying levels of success. The earliest work applying a GA to solve SPP which we were able to find in literature appeared in 1997 [85]. In their work, a GA for the SPP is proposed in which a priority-based indirect method of encoding is used, with various eligible node sets being used as a means of preventing infeasible paths from being constructed during the decoding process. Infeasible paths include those which contain loops and those that terminate in non-destination nodes. They were able to solve SPP instances of up to 70 nodes and 211 edges accurately. In another work, Ahn and Ramakrishna [9] used a direct encoding method to represent chromosomes, with a repair function being designed for chromosomes representing invalid routes. They also proposed a population-sizing equation which can be used to choose the right population for the GA. They compared their algorithm's performance with other GAs in literature, and found theirs to have a better accuracy rate on problem instances up to 50 nodes in size.

In [243], a GA is proposed for SPP in which chromosomes are made of nodes which form feasible paths, genes being constrained so as to ensure that chromosomes do not represent infeasible paths. They tested their algorithm on an undirected SPP instance of 100 nodes using a population size of 10 with maximum generation of 100, and found it to be at the same level of accuracy as Floyd-Warshal's algorithm. Hasan, et al. [101] proposed a Heuristic GA (HGA) for SPP which combines exact as well as heuristic techniques into the GA for optimal results. Their algorithm prove to be useful in selecting a promising initial population and optimizing accuracy and execution time for the GA when tested on SPP instances ranging from size 10 to 100.

GAs have also been severally applied to the TSP. In [234], two local optimization strategies are merged into the traditional GA to form a HGA. Both local optimization strategies have time complexities of $O(n)$ and $O(n^3)$ respectively. The results were compared with the traditional GA and showed better performance. However, they noted that with larger instances the computational complexity of the proposed algorithm grows very fast, causing it to require more time to obtain accurate results. Nagata and Soler [161] also used a GA method to solve the ATSP, however, it was combined with a local search procedure based on 3-opt for initial population generation. Their algorithm was successfully tested on 153

ATSP instances from the TSPLIB [192] in addition to 123 self-generated instances, and reported competitive results.

2.4.2 Physarum Algorithm

Physarum polycephalum is a unicellular amoeboid organism (also known as *true slime mold*) whose body shape resembles an intricate network of tubular components forming a means of intracellular transportation of protoplasmic material [162]. The shape and physiological features of the organism metamorphose through the disassembling and reassembling of tubes with respect to changes in the environment (such as presentation of new food source(s)). This adaptation ensures the maintenance of an efficient tubular network design which will lead to efficient streaming of protoplasm between each pair of local parts of the plasmodium. Intracellular exchange of protoplasm plays a significant role in chemical communication within the cell, determining the survival and maintenance of the cell as a unified singular individual [162]. The first work which investigated and channeled this unique behavior of the slime mold was done by Nakagaki [162] in the year 2000. He carried out series of experiments with the slime mold, placing it in a maze with two food sources at its entry and exit points. After a period of time, the mold had metamorphosed into a three-part shape consisting of two sheet-like parts covering the two food sources (absorbing nutrients from the food sources), and a thick tube lying in the shortest path between these two parts,

maintaining an efficient means of intracellular communication between them (See Figure 2.3).

Thus, it was proven that *Physarum polycephalum* is capable of solving shortest path problems, even in complex structures like mazes. A key part of modelling this process of shortest path resolution is understanding the mechanism through which the protoplasmic material (also referred to as 'sol') is pumped within the cell. At the food sources where a sponge-like part of the organism is formed, *actin-myosin* fibers exhibit rhythmic contractions which exert pressure on the sol, making it flow into the tube (s) and out at the other end. This flow is bi-directional, involving periodic changes in direction known as protoplasmic *shuttle streaming*.

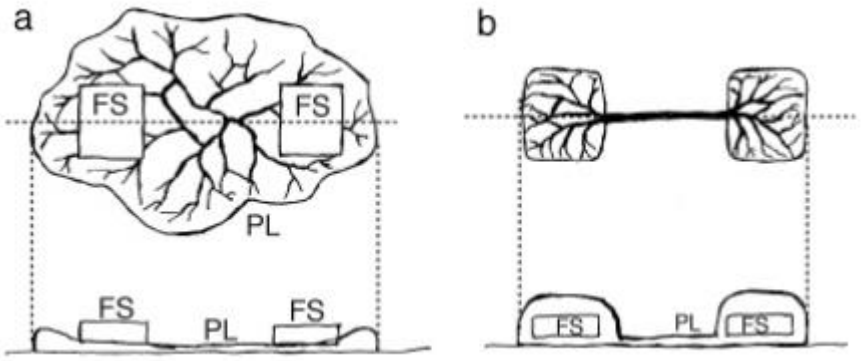


Figure 2.3: An illustration of change in shape and structure of a true slime mold in response to new food sources (FS). (a) shows top and side views of the mold with food sources just introduced. (b) shows the new shape of the mold, with the interconnecting tube between the two sheet-like parts

Hence, the sponge-like parts of the organism act as sources of mechanical force/pressure at the food sources. The formation and thickness of tubes is directly proportional to the length of time as well as pressure with which shuttle streaming is maintained in a particular direction [217].

In the formation of a mathematical model for the flow of sol through tubes [217], the shape of the *Physarum* is represented as a graph, where a tube is an edge, and a junction between tubes is a node. Two special nodes corresponding to the two food sources are denoted as N_1 and N_2, with N_1 as the source node and N_2 as the sink node. Other nodes are designated as N_3, N_4, N_5, and so on. M_{ij} represents the edge between N_i and N_j, Q_{ij} represents the flux between

N_i and N_j through $M_{ij,}$, p_i and p_j represent the power at nodes i and j respectively, and L_{ij} and a_{ij} are the respective length and radius of the tube corresponding to M_{ij}. Assuming that the flow along the tube is approximately a Poiseuille flow, Q_{ij} is given as [217, 258]:

$$Q_{ij} = \frac{D_{ij}}{L_{ij}}(p_i - p_j) \tag{2.23}$$

with D_{ij} usually set at

$$D_{ij} = \frac{\pi a_{ij}^4}{8\kappa} \tag{2.24}$$

By taking into cognizance the conservation of sol,

$$\sum_i Q_{ij} + I_0 = 0\,, (j \neq 1, 2). \tag{2.25}$$

And, for the source and sink nodes, N_1 and N_2 respectively, the following equations hold:

$$\sum_i Q_{i1} + I_0 = 0, \quad \sum_i Q_{i2} - I_0 = 0 \tag{2.26}$$

where I_0 is the flux from the source node or into the sink node, and is constant.

Physarum algorithms have been quite successfully applied to the SPP in literature. In [260], a Rapid Physarum Algorithm was proposed to solve the SPP, in which a heuristics rule determined from experimentation and statistics is integrated to avoid redundancy and improve efficiency. They tested their algorithm on graphs of size ranging from 15 nodes and 23 edges, to 2000 nodes and 4044 edges. They reported that their algorithm was able to solve the SPP for a fully connected network with 1000 nodes in under 50 iterations, and a randomly distributed network with 2000 nodes and 4044 edges in under 100 iterations. They compared the performance reported for their improved algorithm with the basic Physarum model, and found theirs to perform better. Similarly, Zhang, et al. [258] improved the basic Physarum model for SPP by combining it with a new 'energy' parameter, accelerating the search speed and reducing the number of iterations required in the basic version. They compared their results with three other algorithms: the basic Physarum, ACO and Dijkstra's algorithm, showing that their new algorithm outperforms both the basic Physarum and ACO in both running time and execution speed, while performing comparably to Dijkstra as well as

proposing more than one shortest path at the same time. Zhang, et al. [259] applied the Physarum algorithm to the Constrained SPP (CSP) problem in a directed graph for the first time, combining it with Lagrangean relaxation. They perform some experimental analysis to prove the ability of Physarum algorithms to also deal with CSPs.

2.4.3 Ant Colony Optimization (ACO)

Ant Colony Optimization (ACO) was introduced by Dorigo [52] in 1991 as a means of solving difficult optimization problems like the SPP, Travelling Salesman Problem (TSP), Job Shop Scheduling Problems (JSSP) and others, and has been widely used for solving the SPP specifically because it is most naturally related to ants' food-finding activities. ACO is based on the behavior of colonies or nests of ants and how they solve their day-to-day problems, the most common of which is finding the shortest path to a food source. A worker ant leaves its nest in search of food, and as it moves along, it deposits *pheromone,* a chemical substance which evaporates with time. Other ants also searching for food are biased to follow the path with a stronger pheromone scent when they are faced with a branch. As more ants choose a particular path, they also deposit pheromone, making the pheromone scent even stronger. Conversely, a path which is less used will have an increasingly weaker pheromone scent because of the evaporation of the pheromone. This situation is more likely to occur because the shorter the path, the smaller the time it will take for ants to get to and from the food source, leading to the path being more used, and therefore, have an increasingly denser pheromone deposit. Note that once an ant finds a food source, it tends to walk back to the nest on its own pheromone trail. Eventually, all ants will choose the shortest path since it will have a much higher pheromone level than other paths [110].

Various ACO systems are modelled after the behavior of the ants to solve the SPP. These systems have the following basic structure [245, 253]

1. Initialization:
    a. Generate and position *m* ants on the *m* nodes of the graph
    b. Set $\tau_{ij}(0) = c, c \neq 0$, where $\tau_{ij}(t)$ is the pheromone trail on the edge between nodes *i* and *j* at iteration *t*.

2. For each ant, *k* at node *i*, select next adjacent node *j*, thus*:*

$$P_{in}^{k}(t) = \begin{cases} \arg \max\limits_{s \in allowed_k^i} \tau_{iz}(t)\eta_{iz}^{\beta}(t), & q \le q_0 \\ s, & q > q_0 \end{cases} \quad (2.27)$$

where $allowed_k^i$ contains nodes adjacent to *i* which have not been visited by ant *k*. $\eta_{iz}(t)$ denotes a the inverse of the total cost between *i* and *z*. η_{iz} is a heuristic function, and is used to estimate the a-priori 'desirability' of the decision to move to node *z*. q and q_0 are random values within the interval [0,1]. *s* is a randomly selected node based on the probability distribution:

$$P_{in}^{k}(t) = \begin{cases} \dfrac{\tau_{iz}(t)\eta_{iz}^{\beta}(t)}{\sum_{x \in allowed_k^i} \tau_{iz}(t)\eta_{iz}^{\beta}(t)}, & s \in allowed_k^i \\ 0, & s \notin allowed_k^i \end{cases} \quad (2.28)$$

3. When an ant has reached the destination node, update its trail thus:

$$\tau_{ij}(t+1) = (1 - \rho)\tau_{ij}(t) + \rho\Delta\tau_{ij}(t) \quad (2.29)$$

where

$$\Delta\tau_{ij}(t) = \sum_{k=1}^{m} \Delta\tau_{ij}^{k}(t) \quad (2.30)$$

$\Delta\tau_{ij}(t)$ represents the pheromone gain that will be introduced after the current tour and ρ is the pheromone decay parameter. $\Delta\tau_{ij}(t) = 0$ if ant *k* has not visited edge *(i,j).* This is called the *local updating rule.*

4. At the end of an iteration, when all ants have reached the destination node, *global update* is performed to update the pheromone of the shortest path which is denoted as $L_{localmin}$:

$$\tau_{ij}(t+1) = \tau_{ij} + \mu\Delta\tau_{ij} \quad (2.31)$$

where

$$\Delta\tau_{ij}(t) = \begin{cases} \frac{1}{L_{localmin}}, & (i,j) \in localmin \\ 0, & otherwise \end{cases} \quad (2.32)$$

ACOs have been applied successfully to the TSP. Recently, Mavrovouniotis and Yang [151] integrated ACO with immigrant schemes to address the Dynamic TSP with traffic factors (changing edge costs and so on), considering two types of dynamic environments: those where the changes in traffic factors are entirely random, and those where the changes are cyclic and have a chance of recurring. They proposed three variants of the algorithm, with Random, Elitism-based and Memory-based Immigrant schemes, which they reported to perform well in quickly and significantly, slowly and slightly, and cyclic changing environments respectively. Furthermore, Yang, et al. [247] developed and ACO for the GTSP which included a mutation process to avoid getting stuck on local optima. They also used a 2-opt heuristic for local search, testing their results on 20 instances from the TSPLIB [192]. The method was able to successfully handle instances of size less than 200 cities.

In [17], an ACO algorithm called n-ANT is presented for the Single Destination SPP on Directed Acyclic Graphs (DAGs), based on Neumann and Witt [164]'s 1-ANT ACO algorithm. More importantly, they conducted a rigorous analysis of the running time of ACO algorithms in general, and proved polynomial-time bounds for their algorithm specifically, which was a better performance than that recorded in [164]. Improving on their work, Sudholt and Thyssen [213] presented another ACO algorithm called $MMAS_{SDSP}$, with results extended from DAGs to arbitrary directed graphs. They also showed that their algorithm transfers to the All Pairs Shortest Path Problem (APSP). They then compared the performance of the proposed ACO algorithm with evolutionary and genetic algorithms using number of fitness evaluations as a performance measure, and showed that ACO is a state-of-the-art algorithm for the APSP. Zakzouk, et al. [253] proposed an ACO algorithm to solve SPPs with fuzzy constraints. They demonstrated that raising the number of ants while making other parameters like pheromone evaporation coefficient a heuristic selection can lead to the best result in the best time. Yabo, et al. [245] provided three improvements to the classical ACO for the SPP to prevent it from falling into local optima: direction guiding via setting of heuristic initial pheromone concentration level, pheromone redistribution to prevent the

pheromone concentration of the optimal path from being overdamped by the pheromone decay process, and introducing a dynamic factor to the global renewal process. Their results were tested on a 30-node graph, and the results showed that the proposed modifications in the classical algorithm can greatly increase the probability of finding the optimal path. ACO has also been applied to solve the Stochastic Shortest Path Problem with favorable results [66].

2.4.4 Particle Swarm Optimization

Particle Swarm Optimization was introduced in 1995 by Kennedy and Eberhart [119] as a method to solve continuous optimization problems (COPs). It originated from their study of the swarming behavior of flocks of birds and schools of fish and attempt to simulate that behavior computationally. However, its introduction saw a great favorable response by the scientific community, chiefly because of its speed of convergence, simplicity, ease of implementation and wide applicability [175]. The classical PSO and its many variants have been applied to solve numerous COPs with a high level of success [175]. PSO algorithms generally have the following components:

1. *Particles:* These are simple agents which represent an individual fish or bird and are used to search out solutions in the problem's search space.
2. *Position:* A position represents a candidate solution, and at any given time during the execution of a PSO algorithm, each particle is at a particular position in the search space.
3. *Swarm:* A set of particles act together as a swarm in their search endeavor. They communicate through simple interactions to share knowledge on good solutions, guiding their search around them. Each particle in the swarm keeps track of its best position gained so far (*pbest)*, as well as the best position gained by the entire swarm (*gbest)*.
4. *Velocity*: Each particle flies through the search space at a particular velocity which dictates both its direction and speed of motion. Its velocity is majorly influenced by its *pbest* and the swarm's *gbest*, together with some element of randomness for search diversity.

Two equations for velocity and position update were proposed by Kennedy and Eberhart [119] in the original PSO:

$$V_i^{t+1} = V_i^t + c_1 r_1^t (pbest_i^t - X_i^t) + c_2 r_2^t (gbest^t - X_i^t) \quad (2.33)$$

$$X_i^{t+1} = X_i^t + V_i^{t+1} \quad (2.34)$$

where *t* is the iteration counter, *i* is the particle index, *pbest* is the personal best, *gbest* is the global best, r_1 and r_2 are random numbers in the interval [0, 1], c_1 is the referred to as the cognitive parameter which dictates how biased the particle's search will be biased towards its personal best position, and c_2 is the social parameter which dictates how much its search will be biased towards the global best. The c_1 and c_2 parameters can be tweaked to favor either exploitation or exploration respectively.

However, with the high level of research that went into the PSO [3, 15], various improvements which focused on a key aspect of the PSO algorithm, velocity update, were proposed. The current state-of-the-art PSO variant is the constriction coefficient PSO [175] proposed in [51]:

$$V_i^{t+1} = \chi[V_i^t + c_1 r_1^t (pbest_i^t - X_i^t) + c_2 r_2^t (gbest^t - X_i^t)] \quad (2.35)$$

$$X_i^{t+1} = X_i^t + V_i^{t+1} \quad (2.36)$$

Clerc & Kennedy defined χ thus:

$$\chi = \frac{2}{\left|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}\right|} \quad (2.37)$$

where $\varphi = c_1 + c_2$, and $\varphi > 4$

The default settings for χ is considered to be 0.729, and that for c_1 and c_2 is 2.05 [175]. The general PSO format of the PSO algorithm is shown in **Algorithm 2-VI**.

Algorithm 2-VI: Particle Swarm Optimization

1. Initialize *swarm, gbest*
2. For particle *i* in *swarm*

3. Initialize X_i^0
4. Set $pbest_i = X_i^0$
5. For particle *i* in *swarm*
6. If($pbest_i$ < gbest) // minimization problem
7. Set gbest = $pbest_i$
8. While (termination criteria is not met)
9. For particle *i* in *swarm*
10. Compute V_i^{t+1} using eq. (2.35) and X_i^{t+1} using eq. (2.36)
11. If($f(X_i^{t+1}) < pbest$) // minimization problem
12. Set $pbest_i = X_i^{t+1}$
13. For particle *i* in *swarm*
14. If($pbest_i$ < gbest)
15. Set gbest = $pbest_i$
16. End While
17. Return *gbest*

2.4.5 Discrete Particle Swarm Optimization

Though the PSO was originally designed for Continuous Optimization Problems, various efforts have been made to discretize it for real world problem which are discrete in nature, such as TSP, JSSP, SPP and others. Because of the prevalence of COPs as applications of real-world problems, a lot of intensive research has gone into discretizing PSO to solve them, with various approaches and schemes being proffered. These schemes can be classified into six categories from literature [44, 131]: Binary, Swap-operator-based, Space transformation-based, Fuzzy matrix-based, Set-based, and Hybrid DPSOs.

2.4.5.1 Binary PSO

Kennedy and Eberhart proposed a discretized version of the PSO algorithm known as Binary PSO (BPSO) [121] in 1997. In BPSO, positions are indirectly represented as strings of bits, and velocities are changes in the probability that a particular bit will either be zero or one. A candidate solution could for example be represented as [100101001110101001], with each

bit representing some problem-specific information. Also, some works define operations like addition, subtraction and multiplication in terms of bit-wise OR, XOR, and other bit-wise operations. In addition to the initial work done by Kennedy and Eberhert [121], various other variants of binary PSO have been developed. These include Modified Binary PSO (MBPSO) by Shen et al [201] in which the fitness function is modified as a static function, *C(p)*, and is applied to variable selection for both Multiple Linear Regression (MLR) and Partial Least-Squares (PLS). The modified BPSO showed satisfactory performance in variable selection and convergence rate. A Probability Binary PSO (PBPSO) was introduced by Wang, et al. [232] which included a novel probability-based strategy for the update of a particle's position. They claimed that their algorithm is relatively simple and improves optimization ability, testing their results on the Multi-dimensional Knapsack Problem (MKP), and finding it better than other existing algorithms in speed of convergence and global search ability.

2.4.5.2 Swap-operator-based DPSO

This scheme was introduced by Wang, et al. [230], but a clear outline and definition was given by Clerc [50]. It defines the position of a particle as a permutation of numbers, and velocity as a set of swaps or 'exchanges' of these numbers to achieve a different permutation. For example, if a position, p is given as (3, 6, 2, 1, 8), a velocity, v = [(2, 3), (3, 1), (4, 2)], then when p is added to v, we get, successively for each velocity element,

(3, 2, 6, 1, 8) (swap position 2 with position 3)

(2, 3, 6, 1, 8) (swap position 3 with position 1)

(2, 3, 1, 6, 8) (swap position 4 with position 2)

Definitions for other operators are given in [50], some of which are left to the DPSO designer to be tuned with respect to specific problems. Shi, et al. [203] proposed a swap operator-based DPSO with an uncertain searching strategy and crossover eliminated technique. This algorithm was extended to the General TSP (GTSP), being the first DPSO to be used for GTSP, and shown to be able to solve graph instances of large sizes in both cases.

2.4.5.3 Space transformation-based DPSO

In space transformation schemes, a transformation method is found which maps the search space to the problem space. That is, the search space is 'constructed' in way as to suit the discrete nature of the problem, with the search space being made of discrete points which correspond to feasible solutions which are in turn potential positions for a particle. For example, in [197] where a DTSP is proposed for the Task Assignment Problem (TAP), the search space is constructed as an *M*-dimensional space corresponding to an *M* task assignment problem, such that a position is a vector of *M* tasks. Comparisons with a GA approach for the TAP showed that their method is viable. In [174], the search space is defined as a continuous Cartesian *N*-dimensional space, where *N* is the number of cities for the TSP, and a mapping was made from the continuous space to discrete which applies to TSP. They included local search techniques as well as chaotic operations to prevent premature convergence, and demonstrated the proposed algorithm's ability to find good solutions in short time by testing it on four benchmark TSP instances.

2.4.5.4 Fuzzy DPSO

In fuzzy discretization for PSO [238], position and velocity are represented as a fuzzy *n-dimensional* square matrix whose elements denote degree of membership in the target position. A defuzzification process is performed on the position matrix to generate a valid TSP tour. This is done by the use of a 'Max Number Method' to go through the position matrix row by row, selecting unique column indices with the highest membership score. Upon completion of the selection process, the column indices form a set which represents a TSP tour. A Similar fuzzy matrix scheme is implemented by Liao, et al. [140] in developing a DPSO algorithm for the flow shop scheduling problem. They combined the DPSO with a local search algorithm, and were able to solve both single- and multi-objective variants of the flow shop scheduling problem with results which compared favorably with those existing in literature. In the same vein, Hajforoosh, et al. [99] applied a Fuzzy DPSO to the Plug-in Electric Vehicle (PEV) charging co-ordination problem which involves minimizing costs associated with energy generation and grid losses when electric vehicles plug into an electric grid for charging, while maximizing the power delivered to them. The algorithm was tested

on a 449-node electric network populated with PEVs having different driving patterns, battery sizes and charging rates, and prove to be viable.

2.4.5.5 Set-based DPSO

In set-based schemes, a particle's position is represented as a mathematical set of unique elements which represents a solution in the search space, the search space being characterized as a universal set. All operators in the two update equations are defined as set operations, though the specific implementation of these operations vary from problem to problem and from algorithm to algorithm. For the TSP, this set can either be made of edges that form a tour, or ordered nodes which also form a tour. Therefore, a typical set for a 7-node graph could be either (2, 1, 4, 6, 3, 5, 7, 2) or [(2,1), (1,4), (4,6), (6,3), (3,5), (5,7), (7,2)].

In [263], a novel algorithm is propounded for discretizing PSO for the TSP, which they called C3DPSO, introducing a 'mutation factor', c3, to prevent premature convergence of the classical discretized PSO. They discretized particles' position as a set of edges with associated probability. They also used the same representation for velocity. Operators were redefined with respect to these representations, and they are discussed in the paper. Six classic TSP benchmarks from TSPLIB [192] were used in comparing the results of their improved algorithm with those obtained from ACO. They found out that the C3DPSO was much better on all benchmarks than all other compared algorithms both in precision and in computational cost. Fang, et al. [74] proposed "an advanced PSO algorithm with Simulated Annealing", aiming at using simulated annealing to reduce the tendency of the classical PSO to get trapped in local minimum, "slow down the degeneration of the PSO swarm, and increase swarm diversity". They also used a set-based approach in discretizing PSO, where a position is a set represented as $\{s[1], s[2], \ldots, s[n]\}$, where $s[i] = j$ indicates that city j is visited from city i. In analyzing their algorithm's efficiency, they compared it with basic Simulated Annealing, basic Genetic Algorithm and basic Ant Colony Algorithm on the benchmark problems Oliver30 and Att48 from the TSPLIB [192]. They found out that their algorithm is significantly better than the other compared algorithms, and a more effective means of solving the TSP problem.

Chen, et al. [44] developed a novel set-based PSO method for Discrete Optimization problems, which they called Set-based PSO (S-PSO) aimed at using a set-based representation scheme to characterize the search space of continual optimization problems, and which could be used to extend the various PSO variants to their discrete versions. They used their algorithm to develop a discrete version of the Comprehensive Learning PSO(CLPSO) proposed in [138]. They evaluated the algorithm by using it to solve some benchmark TSP problems and comparing the results with those of both existing PSO-based approaches and some meta-heuristic algorithms. At the end, they found their algorithm to be promising. Also, Ma, et al. [147] proposed an "efficient discrete PSO algorithm" in order to achieve optimization in discrete space. This was a modification of the work of Qu, et al. [186] called Locally Informed Particle Swarm (LIPS) based on set-theory. They called the algorithm Set-based LIPS (S-LIPS). They then compared the performance of S-LIPS on some benchmark TSP instances with that of S-CLPSO, and found the results to be promising, especially in the case of large scale problems. A Discrete Particle Swarm Optimization (DPSO) technique is also presented in [6] to solve graph-based TSP. Its convergence was tested, and the result found favorable. Wu, et al. [244] proposed a discrete PSO (DPSO) algorithm with scout particles for optimizing library material acquisition subject to certain constraints including budget and number of materials. The efficiency of the proposed DPSO was established through empirical study and simulation.

The majority of the DPSO algorithms which have been proposed specifically for the (elementary) SPP have been found to also use the set-based approach [252], [155], and [156]. These algorithms all use an indirect method for encoding paths as positions, mainly because of the fact that this allows for less invalid paths than would be produced by a direct representation made of a mere random permutation of nodes. Another major reason is that the updating of particle positions is based on set arithmetic operations which would lead to invalid nodes if they are applied on direct node representations. The disadvantage of this indirect method is the need for a decoding process to translate the encoded representation into path for fitness evaluation [155]. Two major approaches exist for this indirect encoding: *priority-based* and *weight-bias-based*. In *priority-based* encoding (used in [155]), a position

is a set whose indices represent nodes in the graph, and whose values are numbers representing allocated priorities. The path is constructed through a path-growth procedure which starts from the source node, iteratively selecting the adjacent node with the highest priority, until the destination node is reached. In the *weight-biased* encoding method, each node has an allocated weight, and the edge costs are biased based on the weights of their two adjacent nodes. The formula used for edge cost bias in [156] was inspired by that presented in [187], which is given in equation (2.38).

$$c'_{ij} = c_{ij} + w_i + w_j \tag{2.38}$$

where c_{ij} is the cost of edge (i,j) and w_i is the weight of node i.

For the weight-biased encoding scheme which was used in [156], the path-growth procedure is used, but in this case, adjacent edges are evaluated and selected based on

$$j = \min \left\{ c_{ij}\beta_j \middle| (i,j) \in E \right\}, \beta_j \in [-1.0, 1.0] \tag{2.39}$$

where c_{ij} represents the cost of the edge (i,j), β_j is the associated weight, and E is the set of edges in the graph.

According to Mohemmed, et al. [156], the weight-biased method is superior to the priority-based one when implemented for PSO. They presented a hybrid DPSO for the SPP which uses weight-biased method and incorporates a noising metaheuristic and periodic velocity re-initialization to enhance the search capability of the algorithm. They used their algorithm to solve randomly generated SPP instances of sizes ranging between 50 nodes and 160 edges and 100 nodes and 200 edges, achieving average accuracy levels of about 90% in all cases. However, they do not discuss the discretization scheme used, nor do they mention the use of position-specific information to enhance and consolidate their algorithm's search process.

The same authors, in [155], presented a modified path encoding scheme in which they used a priority-based approach. They introduced a heuristic operator to reduce the probability of 'no-exit nodes' which cause the formation loops or invalid paths, and compared their results on instances ranging from 15 to 70 nodes with those produced by GA methods existing in literature. Their algorithm was shown to perform better than those with which it was

compared, both in speed and accuracy. Their algorithm also produced near-optimal paths with costs of up to 95% of the optimal cost.

In [252], an improvement is made on the algorithm in [155]. They maintained a priority-based encoding approach, though they introduce a new idea of randomly selecting priority values and incorporating search decomposition to improve the performance of the algorithm. Their algorithm's performance was different instances of real-world data with size ranging from 13 to 38 nodes, and the performance of the improved algorithm was better than the previous one. However, they similarly do not describe their PSO discretization procedure, and therefore nothing can be said about how they went about operator redefinitions and so on. Of all three DPSO methods for SPP which were found in literature, the one which showed the most promise was [156], where 100-node instances of the SPP were solved with accuracy of up to 90% in as few as 107 iterations.

2.4.5.6 Hybrid DPSO

Hybrid methods incorporate other meta-heuristics into DPSO. In [235], the ideas of a swarm, *gbest* and *pbest* from PSO were combined with a novel quantum bit expression mechanism based on Quantum-inspired Evolutionary algorithm to develop a Quantum Swarm Evolutionary (QSE) algorithm. Experiments on the 0-1 Knapsack problem and TSP showed that the algorithm outperformed Tabu Search and Simulated Annealing, though the performance was not optimal when the number of cities is large for TSP. Afshinmanesh, et al. [7] introduced a novel DPSO which combines Binary PSO with Artificial Immune Systems to create a novel Binary PSO which performed better than other Binary PSO in literature, as well as a GA method. Others hybrid approaches incorporate problem-dependent local search methods. In [37], a Hybrid DPSO (HDPSO) algorithm is proposed which combines a novel discretization technique with a local search algorithm based on Simulated Annealing to improve quality. Their algorithm was specifically designed for the no-wait flow shop scheduling problem, and upon comparing it with another Hybrid DPSO designed for the same problem in literature, they found that their algorithm executes at much higher speed. Finally, in [145] a hybrid model between PSO and a GA-based local search (called Fast Local Search) is proposed for the blind TSP. Instances ranging between 76 and 2103 cities in size

were solved with error rates varying between 1 and 2.538%. This served to demonstrate the robustness of the proposed HDPSO. A summary of the discretization schemes discussed is presented in Table 2.1.

Table 2.1: Summary of PSO discretization schemes used in literature

Binary	Afshinmanesh, et al. [7], Kennedy and Eberhart [121], Khanesar, et al. [122], Khanesar, et al. [123], Taşgetiren and Liang [216], Wang, et al. [232]
Swap operator-based	Clerc [50], Wang, et al. [230]
Space transformation	Pang, et al. [174], Salman, et al. [197]
Fuzzy	Abraham, et al. [1], Hajforoosh, et al. [99], Liu, et al. [143], Wei, et al. [238]
Set-based	Chen, et al. [44], Langeveld and Engelbrecht [131], Liu and Maeda [144], Ma, et al. [147], Mohemmed, et al. [155], Mohemmed, et al. [156], Momtazi, et al. [157], Wei-Neng, et al. [237], Yusoff, et al. [252]
Hybrid	Afshinmanesh, et al. [7], Chandrasekaran, et al. [37], Lope and Coelho [145], Wang, et al. [235]

2.4.6 Memetic DPSO

Memetic algorithms have been successfully used to solve complex optimization problems because they have two components: the global and local search components, which lead to a balance in both exploration and exploitation of the search space. The term ‘memetic’ comes from ‘meme’, a word which was coined by Dawkins in 1976 in his book, the *Selfish Gene* [59]. The meme is a unit of intellectual or cultural information, as opposed to the gene, which is a unit of biological information. The key advantage of a meme is that an individual can improve its meme’s information through learning and experience during its lifetime, a feat which is impossible with genes. A memetic algorithm usually consists of memes with which global search is done akin to other Evolutionary Algorithms. However, it then uses its local search component to attempt to improve each meme iteratively. As a result, a balance can be

achieved between exploration and exploitation, where such a balance has been found difficult to achieve with the general class of Evolutionary Algorithms, as well as some Particle Swarm Optimization (PSO) variants [73, 178, 228]. In this work, the Random Walk with Direct Exploitation (RWDE) algorithm presented in [178] is implemented with the application of *Schematic 1* described in the same paper, where local search is performed on the global best at each iteration of the PSO. The local search algorithm is presented in **Algorithm 2-VII** [178].

Algorithm 2-VII: Random Walk with Direct Exploitation

1. Let $X^{(t)}$ be a vector representing current position at the tth iteration, $X^{(t+1)}$ be the new value at the (t+1)th iteration, and $z^{(t)}$ be a vector. Set $\lambda = \lambda_{init}$ and find $F(X^{(t)})$, the fitness value of position $X^{(t)}$
2. Set t = t + 1. If t is greater than t_{max}, end. Otherwise randomly generate a z with elements within the range (0,3].
3. Compute F' = F($X^{(t)} + \lambda z$).
4. If F' < F
    a. Set $X^{(t+1)} = X^{(t)} + \lambda z$, $t = t + 1$, $\lambda = \lambda_{init}$, $F^{(t)} = F'$.
    b. Check if $t > t_{max}$. If yes, terminate. If no, go to (3).
5. If F' >= F, set $X^{(t+1)} = X^{(t)}$ and go to (2).

2.5 Multi-Objective Optimization (MOO)

In multi-objective optimization (MOO) problems, two or more competing and/or conflicting functions are to be minimized/maximized at the same time. As a result of the conflicting nature of the objectives to be optimized, it is often near-impossible to find solutions which optimize all objectives. As a result, trade-offs in optimizing the various objectives are usually provided [194]. These tradeoffs are presented in form of a set of solutions which are not strictly better than each other in all objectives to be minimized (one is better than the other in at least one objective), referred to as *pareto-optimal* solutions [231]. Given $\vec{x} =$

$[x_1, x_2, \dots, x_d]$ where d is the dimension of the decision variable space, multi-objective optimization problems are usually of the form [221]:

$$minimize\ \vec{f}(\vec{x}) = [f_1(\vec{x}), f_2(\vec{x}), \dots, f_m(\vec{x})] \tag{2.40}$$

Subject to

$$g_j(\vec{x}) \leq 0,\ \ j = 1,2, \dots, J \tag{2.41}$$

$$h_k(\vec{x}) = 0\ \ \ k = 1,2, \dots, K \tag{2.42}$$

where $f_i : \mathbb{R}^n \rightarrow \mathbb{R},\ i = 1, \dots, m$ are the objective functions, $g_j(\vec{x})$, $h_k(\vec{x}) : \mathbb{R}^n \rightarrow \mathbb{R},\ j = 1, \dots, J,\ \ k = 1,2, \dots, K$ are the inequality and equality constraint functions, respectively, of the multi-objective problem.

The concept of *pareto-optimality* was first introduced in 1906 by Vilfredo Pareto, an economist [36]. In order to formally define this concept, a few important terms are defined below [158, 194]:

Dominance: A vector, $\vec{x}$ is said to *dominate* another vector, $\vec{y}$ (denoted by $\vec{x} \prec \vec{y}$) if $\vec{f_i}(\vec{x}) \leq \vec{f_i}(\vec{y})$ for $i = 1, \dots, k$ and $\vec{f_i}(\vec{x}) < \vec{f_i}(\vec{y})$ for at least one $i = 1, \dots, k$. That is, $\vec{x}$ not worse than $\vec{y}$ in all objectives, and $\vec{x}$ better than $\vec{y}$ in at least one objective. A graphical illustration is given in Figure. 2.4.

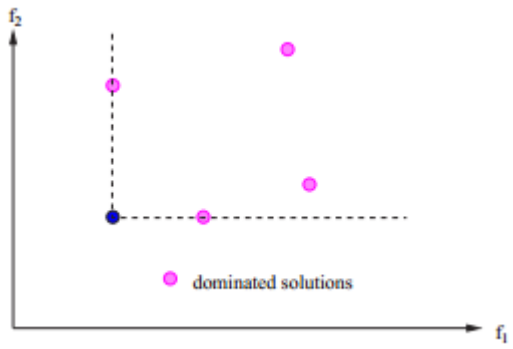


Figure. 2.4: Illustrating dominance in a 2-objective space ([194])

Non-dominance: A vector, $\vec{x} \in \mathcal{X} \subset \mathbb{R}^n$ is said to be *non-dominated* with respect to $\mathcal{X}$ if there exists no other $\vec{y} \in \mathcal{X}$ such that $\vec{f}(\vec{x}) \prec \vec{f}(\vec{x})$. In simple words, there exists no $\vec{y}$ which is better than $\vec{x}$ in all objectives.

Pareto-optimality: Pareto-optimality of a vector $\vec{x}$ within a feasible region, $\mathcal{F}$, therefore, implies that the vector $\vec{x}$ is non-dominated with respect to $\mathcal{F}$.

Pareto-optimal set: Denotes the set $\mathcal{P}^*$, of all pareto-optimal solutions in $\mathcal{F}$. Formally, $\mathcal{P}^*$ is defined as:

$$\mathcal{P}^* = \{\vec{x} \in \mathcal{F} \mid \vec{x} \text{ is pareto-optimal}\}$$

Pareto Front: The Pareto Front, $\mathcal{PF}^*$is the set of all objective values corresponding to all variables in the pareto-optimal set. A Pareto-front is shown in Figure. 2.5. Formally,

$$\mathcal{PF}^* = \{\vec{f}(\vec{x}) \in \mathbb{R}^k | \vec{x} \in \mathcal{P}^*\}$$

The goal in solving multi-objective optimization problems (MOOPs), therefore, is to detect the "highest possible number of Pareto-optimal solutions that correspond to an adequately spread Pareto front, with the smallest possible deviation from the true Pareto front" [177].

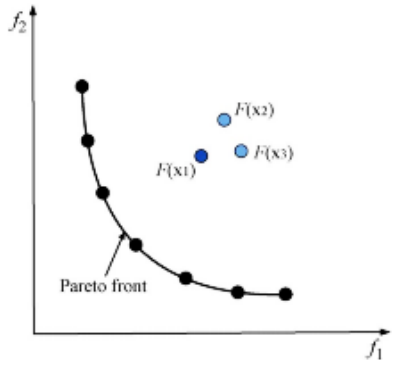


Figure. 2.5: Pareto front for a two-objective minimization problem

2.5.1 Multi-Objective Particle Swarm Optimization (MOPSO)

It is clear from the above definitions that the usual single-objective PSO would be inefficient or even outright fail when applied to MOO. Due to the nature of MOO, the following issues arise when extending PSO to solve MOOPs [194]:

1. **Choice of leader:** Choice of the right particle to act as a leader for the particle flight process since the concept of a single global best no longer applies. The choice of leader should be such that only non-dominated solutions are selected.
2. **Set of non-dominated solutions:** An optimal way of maintaining an updated set of non-dominated solutions found over iterations, which are well-spread along the Pareto front.
3. **Diversity promotion:** Maintaining the swarm in such a way that diversity is promoted, and prevent convergence to a single solution is prevented.

Various MOPSO algorithms have been presented in literature, proposing various ways of handling these issues. Some of the key proposals based on these three issues are highlighted subsequently.

2.5.1.1 Choice of leader

In MOPSOs, rather than having a single leader (global/local best) directing the flight of particles, a set of non-dominated solutions referred to as the *external archive* is usually maintained, and the leader for each particle in each iteration selected from this set. The choice of which leader to select has been approached from different angles. While a few researchers decide to randomly select leaders from the external archive [124], the general approach taken by researchers in literature is to define some *quality* measure by which the closeness of particles within a swarm can be measured. Some of the important quality measures in literature include the Nearest neighbor density estimator [62] and kernel density estimator [61]. A very widely-used quality measure is the *sigma method* proposed by Mostaghim and Teich [159] which assigns a value σ_i to each solution. Sigma (σ) is defined for the bi-objective case as

$$\sigma = \frac{f_1^2 - f_2^2}{f_1^2 + f_2^2} \tag{2.43}$$

This value is extended to a 3- or more objective case thus:

$$\vec{\sigma} = \frac{\begin{pmatrix} f_1^2 - f_2^2 \\ f_2^2 - f_3^2 \\ f_3^2 - f_1^2 \end{pmatrix}}{(f_1^2 + f_2^2 + f_3^2)} \tag{2.44}$$

Thus, a particle in the course of performing its position update will select the particle in the external archive with the closest sigma-value to its own as its leader.

Another measure is the *crowding distance* proposed by Raquel and Naval Jr [190], which provides an estimate of the density of solutions surrounding a given solution in the objective space. To calculate crowding distance, all solutions are sorted in an ascending order based on each objective function value. Solutions with highest and lowest objective values are given infinite crowding distances, and the crowding distance for each remaining solution is calculated by first obtaining the average distance of its two neighboring solutions in the sorted list, for each objective. Its final crowding distance is then the sum of the crowding distances obtained for each objective function. A group of non-dominated solutions with

highest crowding distances are used as the leaders of the swarm, each particle randomly selecting its leader from this group. A few more recent techniques for selecting leaders include the use of hyper-heuristics [34], and parallel cell coordinate system [229].

2.5.1.2 Set of non-dominated solutions

As a result of the complexity of maintaining an external archive as well as the chance of its size exploding, some researchers have tried to avoid its use by proposing objective function aggregation methods which perform a weighted combination of all objective functions into a single one thus:

$$F(x) = \sum_{i=1}^{k} w_i f_i(x) \tag{2.45}$$

where the sum of all the weights is usually normalized to 1:

$$F(x) = \sum_{i=1}^{k} w_i = 1 \tag{2.46}$$

Some notable such approaches include the Conventional Weight Approach (CWA), Bang-Bang Weighted Approach (BWA) and the Dynamic Weighted Approach (DWA), all adapted from Jin et al [117] for MOPSO by Parsopoulos and Vrahatis [176]. However, these methods have a strong limitation in that the algorithms have to be run repeatedly, hopefully finding a different non-dominated solution at each run, for the pareto front to be achieved.

For pareto-based approaches, the size of the external archive poses a challenge. Whenever a solution which is non-dominated by a particle's *pbest* is found, it is to be added to the external archive. There is therefore a tendency for the size of the external archive to increase significantly after a few iterations, which results in dominance checks which have to be carried out during every iteration becoming computationally expensive. This has led to the necessity of restricting the size of the external archive to some reasonable and computationally manageable value. Some researchers do this by determining a fixed size for the external archive, and as soon as the predetermined size is reached, applying truncation and clustering techniques to delete old archive candidates while maintaining a good spread

of solutions [127, 265]. However, the problem of deciding on a condition (better than mere dominance) based on which a solution will be added to or removed from the external archive still needs to be solved. The major scheme adopted by researchers in literature to address this problem is the simple, yet effective ϵ-dominance scheme proposed by Mostaghim and Teich [158]. To explain the concept of ϵ-dominance, they gave the following definitions:

ϵ-domination: For some ε > 0, a decision vector, $\vec{x}_1 \in \mathcal{X}$ is said to dominate a decision vector $\vec{x}_2 \in \mathcal{X}$ (denoted by $\vec{x}_1 \prec_\epsilon \vec{x}_2$) iff:

- $f_i(\vec{x}_1)/(1+\epsilon) \; < \; f_i(\vec{x}_2) \forall \; i = 1, \ldots, m.$
- $f_i(\vec{x}_1)/(1+\epsilon) \leq f_i(\vec{x}_2)$ for at least one $i = 1, \ldots, m.$

ϵ-approximate Pareto Front: Let $F \subseteq \Re^m$ be a set of vectors and $\epsilon > 0$. The ϵ-approximate Pareto Front $F_\epsilon \subseteq F$ contains all vectors $\vec{x} \in F$, which are not ϵ-dominated by any vector $\vec{x}_2 \in F$:

$$\forall \vec{x}_2 \in F : \exists \; \vec{x}_1 \text{ such that } \; \vec{x}_1 \prec_\epsilon \vec{x}_2 \tag{2.47}$$

Their scheme provides an effective way of filtering the contents of the external archive and keeping its size at a reasonable level [177], and on comparison with other well-known clustering techniques, it was shown to find solutions faster and with even better diversity. An illustration of how ϵ-dominance enlarges the area dominated by a solution is given in Figure 2.6.

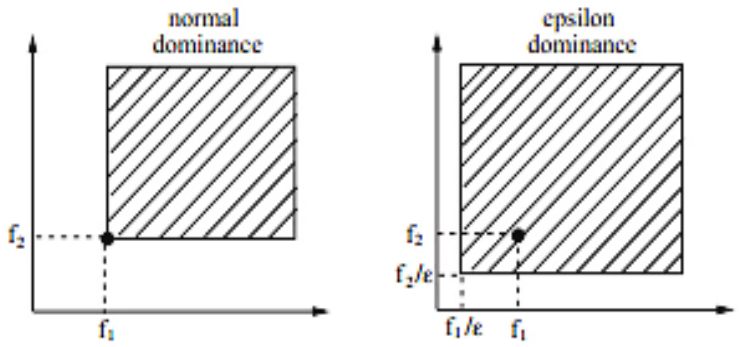


Figure 2.6: Dominance and ϵ-dominance [158]

Recently, a mechanism known as *stripes* which improves on ϵ-dominance was proposed in [225]. Though the proposed technique was shown to perform better than ϵ-dominance in selecting leaders, it exhibits much greater complexity compared to the simple nature of ϵ-dominance.

2.5.1.3 Diversity promotion

One of the strengths of PSO is its fast convergence. However, with fast convergence comes the danger of getting trapped on local optima, as has been pointed out by various PSO researchers [3, 4]. Therefore, there is the need for a means through which diversity can be maintained within the swarm so as to counter-balance fast convergence. One of the major factors which affects swarm diversity is inertia weight, and so tuning or optimizing the inertia weight scheme can significantly improve the diversity of the swarm. In [14], two chaotic inertia weight strategies are proposed which are proven to enhance swarm diversity while maintaining fast convergence, with the methods tested successfully on benchmark global optimization problems.

Another strategy which has been widely used in MOPSOs is mutation, a concept borrowed from Genetic Algorithm (GA). Mutation is usually carried out based on a user-defined value called *mutation rate*. Upon choosing a solution to be mutated, a decision is made on changing each component of the solution or not based on the mutation rate (probability). Thus, mutation can be a very useful tool in helping swarms to escape being trapped on local optima [194]. Recent proposed algorithms for maintaining swarm diversity in MOPSOs include Partitioning Around Medoid (PAM) clustering algorithm as well as uniform design proposed by [264]. Here, the pareto front is partitioned into K clusters and then uniform crossover employed on the minimal cluster, with PAM being used to decide which solutions are to be removed or inserted in the external archive. Daneshyari and Yen [55] also introduced a cultural-based MOPSO in which a cultural framework is used to adapt the personalized flight parameters of mutated particles. Another approach to maintaining diversity is the use of a dynamic multi-swarm technique, as was recently adapted to MOPSO by Liang, et al. [139] showing remarkable performance on various benchmark test functions.

A summary of the general form of MOPSO algorithms is presented in **Algorithm 2-VIII** [194].

Algorithm 2-VIII: General MOPSO Procedure

1. Initialize swarm
2. Initialize external archive
3. Compute quality measure of leaders in external archive
4. While (termination criteria not met)
5. For each particle in swarm
6. Select leader (based on quality measure)
7. Perform particle flight (based on equations (6) & (7)
8. Perform mutation
9. Evaluate fitness
10. Update pbest and add discovered non-dominated solution to external archive
11. Update external archive
12. Compute quality measure of leaders in external archive
13. Return external archive as pareto front

2.6 Inspirations for New Algorithms

Having reviewed the existing literature with special emphasis on DPSOs for TSPs and SPPs, the following inspirations gave rise to the proposed DPSOs for both problems.

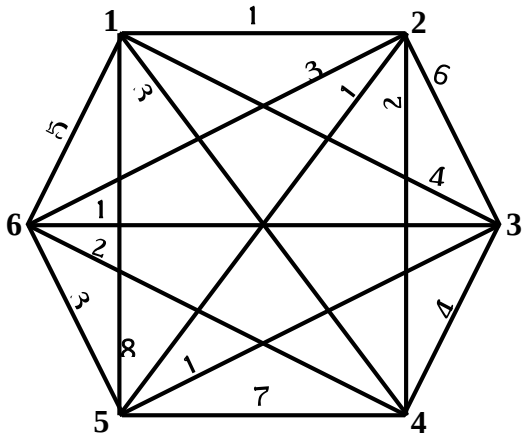


Figure 2.7: Sample 6-node-sized graph

As regards the TSP, it was discovered in reviewed literature that none of the DPSOs actively factored in the distribution of edge costs within the input graph in directing the particles' search towards more promising solution regions. To illustrate this concept, consider the graph in Figure 2.7. Given the distribution of edge costs within the graph, edges (5,1), (5,4), and (3,2) have relatively high costs. A means was therefore sought through which these high-cost edges can be identified, and the search directed away from them towards much cheaper edges such as (1,2), (2,8), (5,3). This inspired the development of the Enhanced Communication DPSO, as is fully presented and discussed in Chapter three.

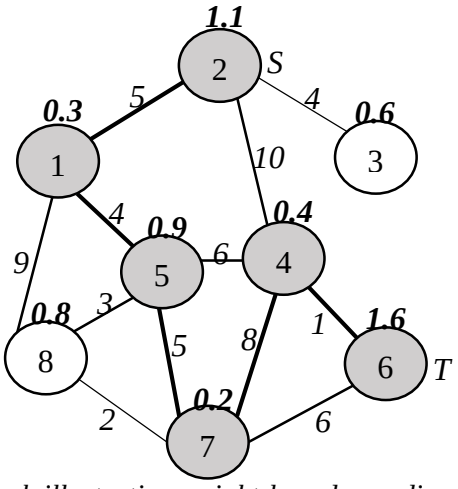


Figure 2.8: Graph illustrating weight-based encoding and resulting path

In addition, for the SPP, it was noted that the majority of DPSOs used in the literature are set-based. Consider the graph in Figure 2.8 which shows a path from node 2 to node 6. In the *weight-biased* encoding which has been successfully employed for SPP in literature, this path, (2, 1, 5, 7, 4, 6), will be represented as (0.3, 1.1, 0.6, 0.4, 0.9, 1.6, 0.2, 0.8). The general *weight-biased* path decoding scheme chooses the path with the minimum cumulative biased edge cost (equation (2.38)). This implies that varying permutations of the same weights within a set represent different paths.

As a simple example, if the set earlier given is rearranged as (0.4, 0.2, 1.1, 0.6, 0.8, 0.3, 1.9, 0.9), the resulting path will be changed to (2, 1, 5, 4, 6), as highlighted in Figure 2.9, which happens to be a cheaper path. Therefore, the node allocations of these weights uniquely identify a path, and need to be taken into consideration during the search process, especially as the *gbest* and *pbest* need to be uniquely identified for the particles to be able to effectively converge towards them. This prompted the development of the second algorithm which is proposed for the SPP, presented in Chapter four.

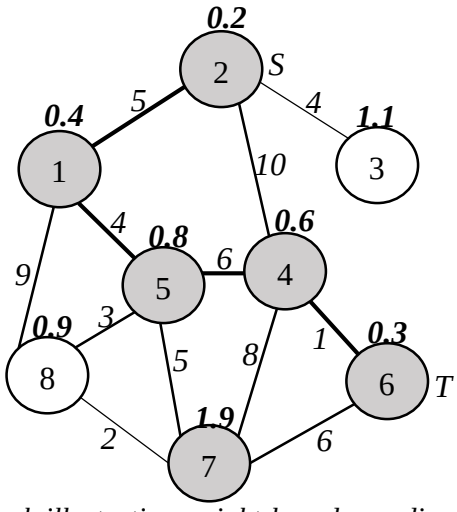


Figure 2.9: Graph illustrating weight-based encoding and resulting path

2.7 Summary

A review has been conducted of the Shortest Path Problem and its variants, with a highlight on Stochastic Shortest Path Problems. Also, the methods used for solving SPPs in literature have been discussed, highlighting exact, non-NI heuristic methods and NI heuristics

including Particle Swarm Optimization. The various discretization techniques used for PSO which exist in literature have also been discussed, and the existing DPSO methods specific to the SPP and TSP highlighted. Three existing DPSO algorithms for SPP were found in literature, but of all three, none explicitly described the PSO discretization process which they used as is expected, and as was the case with other DPSOs presented in literature. It was also observed that of all the discretization techniques used for TSP which were found in literature, none took advantage of edge costs as a means of driving the particles' search towards more promising regions. Furthermore, it was observed that the existing method used for solving the SDP in literature is a Metropolis algorithm which entails the generation of about 3,200,000 different submission schedules. This involves unnecessary computational burden, both time and memory-wise.

Based on the aforementioned observations, this research focusses on developing two new DPSO algorithms. The first DPSO algorithm, which is tailored towards the TSP, introduces the idea of driving particles' search by their experience of how 'good' an edge is, leading them to concentrate on more promising regions within the search space, and hence speeding up convergence. A DPSO algorithm for the SPP is also proposed which clearly describes the discretization process and incorporates node position-specific information. This research also deals with the application of the DPSO algorithm proposed for SPP to solve the SPP, MOSSP, and SDP.

Chapter Three

An Enhanced Communication Discrete Particle Swarm Optimizer for the Travelling Salesman Problem

3.0 Introduction

The discretization of the Particle Swarm Optimization (PSO) technique for the Travelling Salesman Problem (TSP) in the proposed case is such that a position is a certain permutation of edges that form a Hamiltonian cycle. Hence, if for example, six cities are being considered, all interconnected in a mesh, then a position could be the set of edges: {(1, 2), (2, 3), (3, 4), (4, 5), (5, 6), (6, 1)}. The concept of the Enhanced Communication DPSO (EC-DPSO) is embedded in the attempt to directly portray velocity as a measure of the amount of change that will be effected on a particular position to produce a new position, and drive particle's search by their communal concept of how 'good' or 'bad' an edge is.

3.1 PSO Discretization

In the process of adapting equations (2.33) & (2.34) to the discrete form of PSO, and more specifically EC-DPSO, the following definitions were made:

Definition 1: (x, y) – represents an undirected edge between two cities, x and y, where the cost of travelling from city x to city *y* is equal to the cost of travelling from city *y* to city *x*.

***Definition 2*:** Velocity, *V* represents the number of edges in the current position to be substituted for new edges. Where the value of *V* exceeds the number of edges required to form a Hamiltonian cycle, it will be clamped down to the number of edges required to form a Hamiltonian cycle. For example, if the number of cities is 6, then the number of edges required to form a Hamiltonian cycle is also 6. Now, if the value of *V* is, say 9, its value will be brought back down to 6. The rationale behind this is that the highest number of edges that can be changed in a particular position is obviously the number of edges in that position. Therefore, if *V* exceeds this value, the value that most closely represents the value of *V* is that number. Also, since velocity is discretized as a scalar value, the question arises about the

case when this value is not an integer. Since the value of *V* has to be an integer, it was decided that any floating point value gotten will be approximated to the nearest integer.

Definition 3: Subtraction –Between two position vectors, A and B, a subtraction, represented as A – B, is defined as the index values of edges in *B*, which are not in *A*. The motivation behind defining subtraction this way arises from the two instances where this operation occurs in equation (2.33), that is *pbest – X* and *gbest – X*. This operation, in the classical PSO, is aimed at measuring the deviation of the position of the current particle from its personal best and the global best respectively, so as to make the subsequent positions to oscillate around those values. Now, since the velocity, V represents the number of edges to be changed in the process of finding a new position, and considering the fact that the result of *pbest – X* and *gbest – X* will invariably contribute to this value of *V*, subtraction should be carried out in such a way that the result constitutes those edges in the current position which are not in the *pbest* and *gbest*, and hence show which edges are to be changed in the current position to obtain the new one. Recall that at this point, these values are the best known values, and the aim is to use the information they contain to aid the search for new positions. An example will suffice to illustrate the redefined subtraction operation:

If $A = \{(2,3),(5,6),(3,4),(6,2),(1,5),(1,4),$ and

$B = \{(1,4),(1,3),(4,2),(6,5),(5,2),(6,3)\},$

Then $A - B = \{2,3,5,6\}$

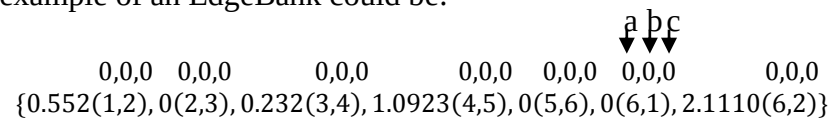
The numbers, 2, 3, 5 and 6 are the indices of the edges in B which differ from those in A.

Definition 4: Multiplication – is defined thus: $x * A = \{x_1, x_2, \ldots, x_n\} = x * |A|$. For example, 3 * {1, 3, 7} = 9.

Definition 5: Addition maintains its meaning as a sum of two scalar quantities, A and B. For example, 3 + 4 = 7.

Definition 6: EdgeBank – EdgeBank is a list of all edges in the input graph from which each particle will select the edges to form its new position. Each edge in EdgeBank has an

associated *goodnessIndex* which influences its chance of being selected as part of a new position. An example of an EdgeBank could be:



All *goodnessIndex* values of the edges will be initialized to zero, and will only be updated at the end of the current iteration, that is after all particles have been processed.

As can be seen from the example above, the EdgeBank will also be used to track the 'opinions' of the different particles about a particular edge. To do this, the EdgeBank will contain, for each edge, three integer values: the first, *a*, will track how many particles included edge *x*, the second, *b*, will track how many particles consider edge *x* to be a 'good' edge, as will be defined below, and the third, *c*, will track how many particles consider edge *x* to be a 'bad' edge.

- The concept of 'goodness' of an edge is defined thus:
    - If during the processing of a particle, the fitness function of the new position evaluates to a better result than the one obtained at the previous iteration, the edge is considered to be 'good'. That is, the value of *b* will increase by 1.
    - If on the other hand, the fitness function of the new position evaluates to a worse result than the one obtained at the previous iteration, the edge is considered to be 'bad'. That is, the value of *c* will increase by 1.
    - If it is neither better nor worse, nothing will be done to the edge, or it is considered 'neutral'. That is, *a* and *b* will remain unchanged.
- After the processing of each particle, the value of *a* for all edges constituting the new position will be increased by 1.
- After all particles have been processed, the *goodnessIndex* values of the all the edges in EdgeBank will be updated thus:

- If the value of *b* is greater than or equal to half the value of *a*, the *goodnessIndex* of that edge will be increased by a constant, *Q*. The value of *Q* is discussed in section 3.1.1.

- If the value of *c* is greater than or equal to half the value of *a*, the *goodnessIndex* of that edge will be reduced by *Q*.

- In the case where neither of the above happens, or *b* and *c* are the same, the *goodnessIndex* of that edge will not be changed.

Finally, there will be a fixed value, *K*, above which if any edge's *goodnessIndex* gets, it will be always selected by particles in their generation of a new position. The parameter *K* is also discussed in section 3.1.2.

Definition 7: The update operation in equation 2.34 has also been re-defined. As has been earlier stated, the velocity is the number of edges to be replaced in *X*. In the update process, V_i^{t+1} random edges will be selected from EdgeBank and used to replace the ones in X_i^t. It involves both the selection of the specific edges in X_i^t which are going to be replaced, and the selection of the edges in EdgeBank which will replace them.

The selection of edges to be replaced in X_i^t will be in three stages:

1. Compute the approximate value of $c_1 r_1^t (pbest_i^t - X_i^t)$, x, and then randomly choose, from the set of indices resulting from $pbest_i^t - X_i^t$, *x* indices whose edges in X_i^t will be replaced.
2. Compute the approximate value of $c_2 r_2^t (gbest^t - X_i^t)$, *y*, and then randomly choose, from the set of indices resulting from $gbest^t - X_i^t$, *y* indices whose edges in X_i^t will be replaced.
3. Where the value of x + y is less than V_i^{t+1}, randomly choose the remaining V_i^{t+1} – (x + y) edges which will be replaced in X_i^t.

The selection of the edges from EdgeBank which will be used in the replacement will also be thus:

1) Randomly select, if possible, V_i^{t+1} edges whose *goodnessIndex* values are greater than or equal to *K*. If the number of edges selected is not up to V_i^{t+1}, then proceed to 2, with z as the number of edges already selected. Else, terminate the selection process.

2) Randomly generate a number, *R*, within the interval [0, *T*], where *T* is a value which will continue to increase as the number of iterations increase. The value of *T* is elaborated upon in section 3.1.3.

3) From EdgeBank, randomly select $V_i^{t+1} - z$ edges whose *goodnessIndex* values are higher than *R*. In the case where the number of edges whose *goodnessIndex* values are higher than *R* is not up to $V_i^{t+1} - z$, then randomly select the remaining number of edges needed to form a Hamiltonian cycle, ensuring that no edge(s) are selected which are already part of the new position. It is worthy of note that in selecting the edges, the selection of the last edge to form the new position does not really have any element of stochasticity in it, as only a particular edge in EdgeBank can be selected that will make up a Hamiltonian cycle (the nearest neighbour). Also, in the selection of edges, one must be careful not to select an edge that will lead to an invalid subtour by adhering to the following rules:

    a) In the final list of edges to constitute the new position, each node must appear exactly twice.
    b) Do not select any edge which consists of a node which already appears up to two times in the new position being generated.
    c) Do not select any edge that will cause a cycle to be formed except it is the last edge, i.e. prevent the forming of sub-tours.

Definition 8: Fitness Function – is defined as the sum of the costs of all the edges that form the new position.

Definition 9: *N* is the number of particles in the entire search space.

Therefore, the new update equations for the EC-DPSO are thus:

$$V_i^{t+1} = V_i^t \oplus c_1 r_1^t (pbest_i^t - X_i^t) \oplus c_2 r_2^t (gbest^t - X_i^t) \quad (3.1)$$

$$X_i^{t+1} = X_i^t \odot V_i^{t+1} \quad (3.2)$$

3.1.1 Analysis of the Constant, Q

It will be observed that the value of *Q* will be based, to a large extent, on the degree or exploitation or exploration that is desired. However, it was decided that a value that will more or less maintain a balance between exploration and exploitation and prevent particles from becoming stuck on local optimum should be selected. If there are *v* nodes in the graph, and there are *m* edges in EdgeBank, then the singular probability of one edge being selected, i.e. not considering that *x* edges are to be selected at once to form a position, is $\frac{1}{m}$ Therefore, *Q* is set at

$$\mathrm{Q} = \frac{1}{m} \quad (3.3)$$

Thus, an edge grows in 'goodness' by a factor of its probability of being selected among *n* other edges.

3.1.2 Analysis of the constant, K

In the course of simulating the iteration process, it became evident that among the edges making up the graph, some were very 'good', i.e. they had very low cost, and would almost always influence the overall fitness of any position in which they were included positively. Conversely, some edges were also noticed which were very 'bad', i.e., they had such high cost that whenever a particle selected them, the fitness value was almost always poor. A way was therefore sought, through which those edges could be tracked, and a difference could be made between the really good and the really bad ones. Therefore, the pegging of a certain value, *K*, above which if an edge's *goodnessIndex* can climb, it will always be selected is motivated by the attempt to single out exceptionally 'good' edges and hasten convergence towards the best position by making particles always select them, instead of wasting time selecting edges which are not so 'good'. Obviously, this is a measure which can easily lead

to the particles getting trapped on local optimum. This is the reason why it was decided that the decision of an edge being 'good' will be based on the opinions of all of the particles which have used it. More so the value of *K* will be selected such that only if over a series of iterations, the *goodnessIndex (*a *cumulative* value) of that edge is up to *K* will it be permanently selected as a good or desirable edge. It should also be noted that when a particle marks an edge as 'good', it is based on the value of the fitness function, which takes into cognizance all other edges that make up that position. Therefore, other edges also contribute in the decision, and the possibility is still reasonably high of an edge's *goodnessIndex* being reduced because it was selected together with other edges whose costs are high, even though its own cost is low. Also, where the number of edges which have *goodnessIndex* values higher than *K* is greater than the velocity, the required number of edges will be randomly selected from amongst all those who qualify. It is obvious then, that the selection of the value of *K* has to take into consideration the probability of an edge being selected among the *n* edges in the edge bank, as well as the probability of that edge being among the group of *m* edges that will form a Hamiltonian cycle which will be selected from among the *n* edges in EdgeBank.

The probability of an edge being selected from *n* edges is $\frac{1}{n}$. Therefore, the probability of selecting an edge among the *m* edges to be selected from the *n* edges in EdgeBank is $\frac{m}{n}$. The value of *K* is hence expressed as

$$\mathrm{K} = \frac{m}{n} \tag{3.4}$$

3.1.3 Analysis of the Value, T

The value *T* is introduced so as to give the edges whose *goodnessIndex* values are higher than those of others a minutely better chance of being selected. It is important that the value of *T* start at a small value and proceed higher as the maximum *goodnessIndex* value of the edges increases, so as to continually give better edges a higher prospect than poorer ones. Since the value of *T* is dynamic, the initial value of *T* is defined as

$$T_0 = \frac{1}{n} \tag{3.5}$$

The subsequent values of T will be defined as

$$T_k = \max(\{p_1, p_2, \ldots, p_n\}) + Q \tag{3.6}$$

Where:

p_i = *the goodnessIndex of edge i in EdgeBank*

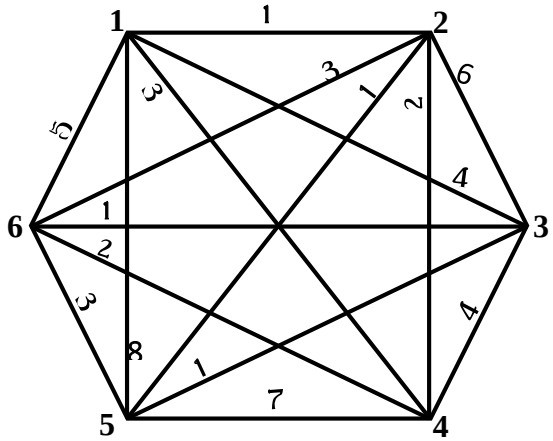
n = number of edges in EdgeBank

k = the number of iteration

Thus, *T* will always maintain a lead ahead of the highest possible *goodnessIndex* value an edge can have.

3.2 Simulation

A short simulation of this algorithm will be done so as to show the nature of the algorithm and give a clearer picture of how it operates. In this simulation of the EC-DPSO algorithm, the search space will be made of 2 particles going through three iterations. The selected parameters are for illustration purpose only. A 6-city TSP will be considered, with the graph representation earlier given in Figure 2.7, which is reproduced here:



$c_1 = 0.8$, $c_2 = 0.8$, $w = 1$

Initialization

Particle 1:

X_0^0 $= \{(1,6), (6,2), (2,3), (3,4), (4,5), (5,1)\}$ $f(X_0^0) = 33$

$pbest_0^0$ $= \{(1,6), (6,5), (5,3), (3,2), (2,4), (4,1)\}$ $f(pbest_0^0) = 20$

$V_0^0 = 4$

Particle 2:

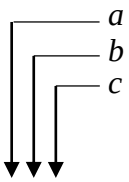
X_1^0 $= \{(2,3), (3,4), (4,5), (5,6), (6,1), (1,2)\}$ $f(X_1^0) = 26$

$pbest_1^0$ $= \{(2,4), (4,1), (1,5), (5,6), (6,3), (3,2)\}$ $f(pbest_1^0) = 23$

$V_0^0 = 2$

Communication

$gbest^0$ = {(1,6), (6,5), (5,3), (3,2), (2,4), (4,1)} $f(gbest^0) = 20$



$$EdgeBank = \{\overset{0,0,0}{0}(1,2), \overset{0,0,0}{0}(2,3), \overset{0,0,0}{0}(3,4), \overset{0,0,0}{0}(4,5), \overset{0,0,0}{0}(5,6), \overset{0,0,0}{0}(6,1), \overset{0,0,0}{0}(6,2), \overset{0,0,0}{0}(6,3), \overset{0,0,0}{0}(6,4), \overset{0,0,0}{0}(4,1), \overset{0,0,0}{0}(4,2), \overset{0,0,0}{0}(2,5), \overset{0,0,0}{0}(5,1), \overset{0,0,0}{0}(5,3), \overset{0,0,0}{0}(3,1)\}$$

A graph illustrating $gbest^0$ is shown in

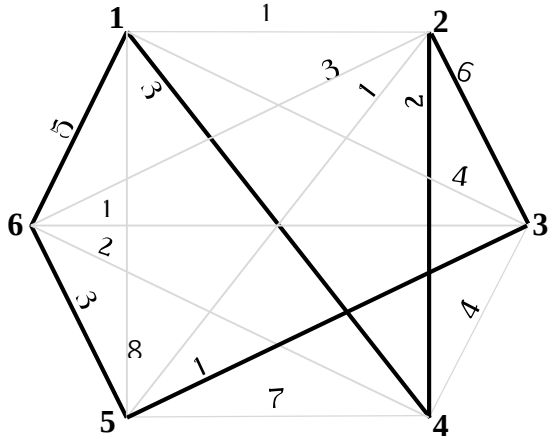


Figure 3.1: Graph showing $gbest^0$

From (3.4), (3.5) and (3.6), with *n* = 15 and *m* = 6, we will respectively have:

$Q = \frac{1}{15} = 0.067$, $T_0 = \frac{1}{15} = 0.067$, $K = \frac{6}{15} = 0.4$

1st Iteration

Particle 1:

Let $r_1 = 0.2, r_2 = 0.8$

$pbest_0^0 - X_0^0 = \{2, 4, 5, 6\}$ $\quad c_1 r_1 (pbest_0^0 - X_0^0) = 1$

$gbest^0 - X_0^0 = \{2, 4, 5, 6\}$ $\quad c_2 r_2 (gbest^0 - X_0^0) = 3$

$V_0^1 = V_0^0 + c_1 r_1 (pbest_0^0 - X_0^0) + c_2 r_2 (gbest^0 - X_0^0)$

$= 4 + 1 + 3$

$= 8$

Clamping this to the maximum velocity,

$V_0^1 = 6$

Let R = 0.017

Edges to be replaced: 1, 2, 3, 4, 5, 6

Edges to be used for replacement (randomly selected according to the rules stated in definition 7): 1, 11, 9, 8, 14, 13

$X_0^1 = \{(1,2), (2,4), (4,6), (6,3), (3,5), (5,1)\}$ $\quad f(X_0^1) = 15$

$pbest_0^1 = \{(1,2), (2,4), (4,6), (6,3), (3,5), (5,1)\}$ $\quad f(pbest_0^1) = 15$

$$EdgeBank = \{0^{1,1,0}(1,2), 0^{0,0,0}(2,3), 0^{0,0,0}(3,4), 0^{0,0,0}(4,5), 0^{0,0,0}(5,6), 0^{0,0,0}(6,1), 0^{0,0,0}(6,2), 0^{1,1,0}(6,3), 0^{1,1,0}(6,4), 0^{0,0,0}(4,1), 0^{1,1,0}(4,2),$$
$$0^{0,0,0}(2,5), 0^{1,1,0}(5,1), 0^{1,1,0}(5,3), 0^{0,0,0}(3,1)\}$$

Note that the values of *a* and *b* have been updated based on definition 6.

Particle 2

Let $r_1 = 0.5, r_2 = 0.6$

$pbest_1^0 - X_1^0 = \{2,3,5,6\}$ $c_1 r_1(pbest_1^0 - X_1^0) = 2$

$gbest^0 - X_0^0 = \{2,3,6\}$ $c_2 r_2(gbest^0 - X_1^0) = 1$

$V_1^1 = V_1^0 + c_1 r_1(pbest_1^0 - X_1^0) + c_2 r_2(\text{gbest}^0 - \text{X}_1^0)$

= 2 + 2 + 1

= 5

Let R = 0.04

Edges to be replaced: 2, 5, 3, 1, 6

Edges to be used for replacement: 13, 9, 15, 2, 11

X_1^1 = {(5,6), (5,1), (6,4), (1,3), (3,2), (2,4)} $f(X_1^1) = 25$

$pbest_1^1$ = {(2,4),(4,1),(1,5),(5,6),(6,3),(3,2)} $f(pbest_1^1) = 23$

$$EdgeBank = \{0^{1,1,0}(1,2), 0^{1,1,0}(2,3), 0^{0,0,0}(3,4), 0^{0,0,0}(4,5), 0^{1,1,0}(5,6), 0^{0,0,0}(6,1), 0^{0,0,0}(6,2), 0^{2,2,0}(6,3), 0^{2,2,0}(6,4), 0^{0,0,0}(4,1), 0^{2,2,0}(4,2),$$
$$0^{0,0,0}(2,5), 0^{2,2,0}(5,1), 0^{1,1,0}(5,3), 0^{1,1,0}(3,1)\}$$

Communication

$gbest^1$ = {(1,2), (2,4), (4,6), (6,3), (3,5), (5,1)} $f(gbest^1) = 15$

$$EdgeBank = \{0.067^{0,0,0}(1,2), 0.067^{0,0,0}(2,3), 0^{0,0,0}(3,4), 0^{0,0,0}(4,5), 0.067^{0,0,0}(5,6), 0^{0,0,0}(6,1), 0^{0,0,0}(6,2), 0.067^{0,0,0}(6,3),$$
$$0.067^{0,0,0}(6,4), 0^{0,0,0}(4,1), 0^{0,0,0}(4,2), 0^{0,0,0}(2,5), 0.067^{0,0,0}(5,1), 0.067^{0,0,0}(5,3), 0.067^{0,0,0}(3,1)\}$$

Graph illustrating $gbest^1$:

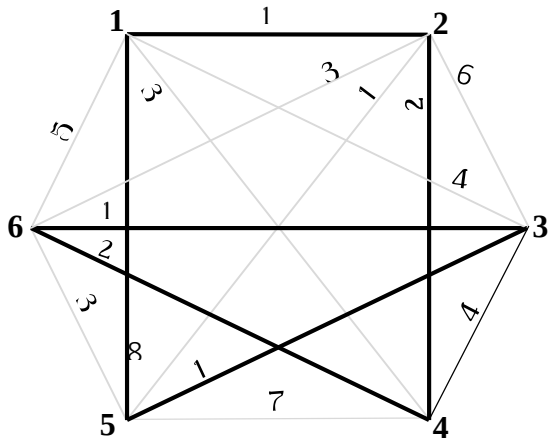


Figure 3.2: Graph showing $gbest^1$

From (9), T = 0.134

2nd Iteration

Particle 1:

Let $r_1 = 0.3, r_2 = 0.4$

$pbest_0^1 - X_0^1 = \{\}$ $c_1 r_1 (pbest_0^1 - X_0^1) = 0$

$gbest^1 - X_0^1 = \{\}$ $c_2 r_2 (gbest^1 - X_0^1) = 0$

$V_0^2 = V_0^1 + c_1 r_1 (pbest_0^1 - X_0^1) + c_2 r_2 (gbest^1 - X_0^1)$

= 6 + 0 + 0

= 6

$V_0^2 = 6$

Let R = 0.025

Edges to be replaced: 1, 2, 3, 4, 5, 6

Edges to be used for replacement: 5, 9, 11, 2, 15, 13

X_0^2 = {(5,6), (6,4), (4,2), (2,3), (3,1), (1,5)} $f(X_0^2) = 25$

$pbest_0^2$ = {(1,2), (2,4), (4,6), (6,3), (3,5), (5,1)} $f(pbest_0^2) = 15$

$$EdgeBank = \{0.067^{0,0,0}(1,2), 0.067^{1,0,1}(2,3), 0^{0,0,0}(3,4), 0^{0,0,0}(4,5), 0.067^{1,0,1}(5,6), 0^{0,0,0}(6,1), 0^{0,0,0}(6,2), 0.067^{0,0,0}(6,3),$$
$$0.067^{1,0,1}(6,4), 0^{0,0,0}(4,1), 0^{1,0,1}(4,2), 0^{0,0,0}(2,5), 0.067^{1,0,1}(5,1), 0.067^{0,0,0}(5,3), 0.067^{1,0,1}(3,1)\}$$

Particle 2

Let $r_1 = 0.9, r_2 = 0.2$

$pbest_1^1 - X_1^1$ = {2, 5} $c_1 r_1 (pbest_1^1 - X_1^1) = 2$

$gbest^1 - X_0^1$ = *{1, 4, 5}* $c_2 r_2 (gbest^1 - X_1^1) = 0$

$V_1^1 = V_1^1 + c_1 r_1 (pbest_1^1 - X_1^1) + c_2 r_2 (gbest^1 - X_1^1)$

= 5 + 2 + 0

= 7

$V_1^2 = 6$ (velocity clamping)

Let R = 0.105

Edges to be replaced: 1, 2, 3, 4, 5, 6

Edges to be used for replacement: 1, 12, 5, 8, 3, 14

X_1^2 = {(1,2), (2,5), (5,6), (6,3), (3,4), (4,1)} $f(X_1^2) = 13$

$pbest_1^2$ = {(1,2), (2,5), (5,6), (6,3), (3,4), (4,1)} $f(pbest_1^2) = 13$

$$EdgeBank = \{0.067^{1,1,0}(1,2), 0.067^{1,0,1}(2,3), 0^{1,1,0}(3,4), 0^{0,0,0}(4,5), 0.067^{1,1,1}(5,6), 0^{0,0,0}(6,1), 0^{0,0,0}(6,2), 0.067^{0,0,0}(6,3),$$
$$0.067^{1,0,1}(6,4), 0^{1,1,0}(4,1), 0^{1,0,1}(4,2), 0^{1,1,0}(2,5), 0.067^{1,0,1}(5,1), 0.067^{0,0,0}(5,3), 0.067^{1,0,1}(3,1)\}$$

Communication

$gbest^2$ = {(1,2), (2,5), (5,6), (6,3), (3,4), (4,1)} $f(gbest^2) = 13$

$$EdgeBank = \{0.134^{0,0,0}(1,2), 0^{0,0,0}(2,3), 0.067^{0,0,0}(3,4), 0^{0,0,0}(4,5), 0.067^{0,0,0}(5,6), 0^{0,0,0}(6,1), 0^{0,0,0}(6,2), 0.067^{0,0,0}(6,3),$$

$$0^{0,0,0}(6,4), 0.067^{0,0,0}(4,1), 0^{0,0,0}(4,2), 0.067^{0,0,0}(2,5), 0^{0,0,0}(5,1), 0.067^{0,0,0}(5,3), 0^{0,0,0}(3,1)\}$$

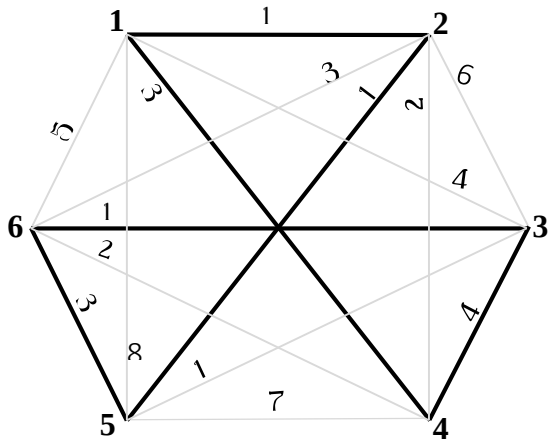


Figure 3.3: Graph showing $gbest^2$

T = 0.201, from (9).

At the end of the second iteration, the fitness value obtained can be seen to be approaching the exact solution, and will eventually converge to it. It is also interesting to observe how some of the low-cost edges already have higher *goodnessIndex* values than others. The exact solution is {(1,2), (2,5), (5,3), (3,6), (6,4), (4,1)}

.

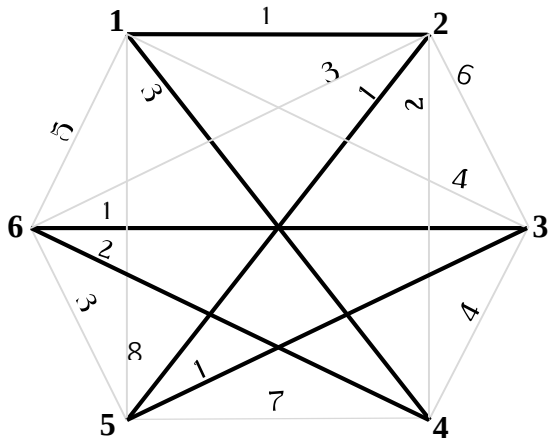


Figure 3.4: Graph showing exact solution for simulation TSP

3.3 EC-DPSO Formal Algorithm

Below is a formal presentation of the EC-DPSO algorithm:

Algorithm 3-I: EC-DPSO Algorithm

```
1:    Randomly initialize the whole swarm
2:    Compute Q and K, and T0
3:    Populate EdgeBank with all edges in the graph and set a, b and c of each edge to 0
      and all goodnessIndex values to 0.0
4:    while(termination criteria not met) {
5:          for(t = 1; t ≤ swarmsize; t++) {
6:                calculate(vt)
7:                update(xt)
8:                evaluate f(xt)
9:                if (f(xt) < f(pbestt-1)) pbestt = xt
10:               update EdgeBank              // using definition 6(a)
11:         }
12:         for(t = 1; t ≤ swarmsize; t++) {
13:               if (f(pbestt) < f(gbest)) gbest = pbestt
14          }
15:         update EdgeBank                    // using definition 6(b)
16:         compute Tk
17:   }
```

3.4 Results

3.4.1 Comparative Analysis of EC-DPSO$_B$, EC-DPSO$_R$, and EC-DPSO

The algorithm was implemented in the Java programming language and run on a Toshiba C660 dual-core computer running @2.00GHz with 4.0GB RAM. The values of *c1* and *c2* were both set at 1.49. The performance of the algorithm was investigated by testing three variants of the proposed algorithm: basic EC-DPSO (without implementing *R* or *K* - EC-DPSO$_B$), EC-DPSO with *R* alone (without *K*: EC-DPSO$_R$), and the full EC-DPSO algorithm, (with both *R* and *K:* EC-DPSO). Each algorithm was run fifty (50) times on 10, 15 and 20-city TSPs (complete graphs), and the average best result, average time taken in seconds and the standard deviation recorded. The termination criterion was set at convergence, i.e. when 130 consecutive iterations give the same *gbest*. The result is presented in Tables 3.1, 3.2 and 3.3.

Table 3.1: Comparison of ECDPSO variant performances on 10-city TSP

Algorithm Variants	Average Time (s)	Average gbest	Standard Deviation for Time (SD)	Average No. of iterations
EC-DPSO$_B$	0.3893	131	14.8007	264
EC-DPSO$_R$	0.2345	108.28	7.8896	182
EC-DPSO	0.2476	108.14	5.7464	181

Table 3.2: Comparison of ECDPSO variant performances on 15-city TSP

Algorithm Variants	Average Time (s)	Average gbest	Standard Deviation for Time (SD)	Average No. Of Iterations
EC-DPSO$_B$	1.7275	225.18	18.7199	237
EC-DPSO$_R$	2.0674	132.04	5.0748	330
EC-DPSO	2.1131	129.82	2.1351	320

Table 3.3: Comparison of ECDPSO variant performances on 20-city TSP

Algorithm Variants	Average Time (s)	Average gbest	Standard Deviation for Time (SD)	Average No. Of Iterations
EC-DPSO$_B$	5.1041	311.44	19.8228	237
EC-DPSO$_R$	8.5543	140.1	8.2764	450
EC-DPSO	10.1541	137.18	8.4169	519

From the results presented above, it can be seen that the basic algorithm, EC-DPSO$_B$ performed worst in the 10-city, 15-city and 20-city TSP instances. In the 10-city TSP instance, the EC-DPSO performed best in both computational cost and speed of convergence, closely followed by EC-DPSO$_R$. It is also interesting to observe that in this smallest-sized instance of the three tested instances, the number of iterations for EC-DPSO$_B$ was much higher than those of the remaining two, whose values are very close. Looking across the standard deviations obtained for the three variants, it can be seen that the disparity of *gbest* values over the 50 runs decreases from EC-DPSO$_B$ through EC-DPSO. In the 15-city TSP instance, a similar trend is observed, the differences being that EC-DPSO took longer time than all the rest, though with better results. It is interesting to note that the number of iterations of the EC-DPSO$_B$ variant is significantly lower than those of the rest. When its result is compared with those of the other two, it becomes evident that this must have been as a result of pre-mature convergence/getting stuck on local optima, since the standard deviation is high, indicating very low agreement in results over the 50 runs. In the case of the 20-city TSP, the tendency of EC-DPSO to take more time to converge gets more pronounced, as does its ability to provide better results. The relative weakness of EC-DPSO$_B$ and its tendency towards premature convergence and getting stuck on local optima is much more pronounced here. In summary, it is evident that the implementation of the concept of enhanced communication through the introduction of the constants *R* and *K* helps to not only lead to much smaller convergence times, but also keep the algorithm from getting stuck in local optima. Also, the constant *K* in particular leads to more accurate results but slower convergence.

3.4.2 Comparative Analysis of EC-DPSO and S-CLPSO

To further test the performance of the new EC-DPSO algorithm, three TSP instances from the TSPLIB[192], eil51, berlin52, st70, eil76, and pr76 were used. On each instance, the algorithm, which was implemented in Java, was run 50 times on a computer with Intel® Core™i7 CPU at 3.10GHz with 8.00GB RAM running a Windows 8.1 Operating System. The results were compared with those reported on the same problem instances in [44], and are presented below:

Table 3.4: Comparison of ECDPSO performance with S-CLPSO with respect to minimal cost route found

S/N	TSP Instance	**EC-DPSO**			**S-CLPSO**		
		Best	Mean	Worst	Best	Mean	Worst
1	eil51	478.7	511	566.8	426	433	427.9
2	berlin52	8446.9	9190	9919.6	7542	7662	7548.4
3	st70	844.4	911.1	1058.6	675	680.1	690
4	eil76	679.5	718.4	785.8	538	549	541.7
5	pr76	133438	147348.5	166226.8	108159	108690	110255

On the eil51 TSP instance, it can be seen that the best result produced by EC-DPSO is comparable to that produced by S-CLPSO, since, though the best produced by S-CLPSO is better, the results are close. As for the other instances, the performance of the S-CLPSO is evidently much better than that of the EC-DPSO. However, the results shown in the five instances indicate that the EC-DPSO algorithm has prospects.

To buttress this point, regression analysis was conducted to determine how well the means of both algorithms predict the best values gotten, with the mean values as independent variable, and the best values as dependent variable. The results for both EC-DPSO and S-CLPSO are shown:

Table 3.5: Regression Analysis for EC-DPSO

Model		Unstandardized Coefficients		Standardized Coefficients	t	Sig.
		B	Std. Error	Beta		
1	(Constant)	46.797	27.093		1.727	.183
	EC-DPSO_Mean	.905	.000	1.000	2206.274	.000

a. Dependent Variable: EC-DPSO_Best

Table 3.6: Regression Analysis for S-CLPSO

Model		Unstandardized Coefficients		Standardized Coefficients	t	Sig.
		B	Std. Error	Beta		
1	(Constant)	-23.826	20.247		-1.177	.324
	S-CLPSO_Mean	.995	.000	1.000	2395.447	.000

a. Dependent Variable: S-CLPSO

The mean values of the EC-DPSO predict the best results with a correlation coefficient of 0.905. This shows that the prediction of the best result by the means is highly significant, and hence, indicates that the performance of the EC-DPSO is reliable and promising, though the S-CLPSO shows a slightly better result.

It is believed that, as is recommended in section 3.5, when the novel idea of enhanced communication between particles based on the ‘goodness’ of individual edges is introduced into other algorithms, their performance can be greatly improved. Further research will also be made on improving the performance of the EC-DPSO algorithm itself via optimized parameter selection, initialization techniques and velocity discretization.

3.5 Swarm Size Investigations for EC-DPSO

The impact of variations in swarm size on the algorithm was also investigated. According to [220], the population (swarm) size can be varied to encourage either exploration or exploitation of the search space. El-Gallad, et al. [71] state that very small settings of the swarm size will greatly increase the chance of the algorithm being trapped in local minima. However, if the value of swarm size is set too high, computational time requirements will be increased. Therefore, to see impact of these variations more clearly, bays29 from [191] was used, and the size of the swarm was varied from 10 to 100 percent of the TSP instance size, *n*. The results in terms of time and deviation from optimal value are displayed below.

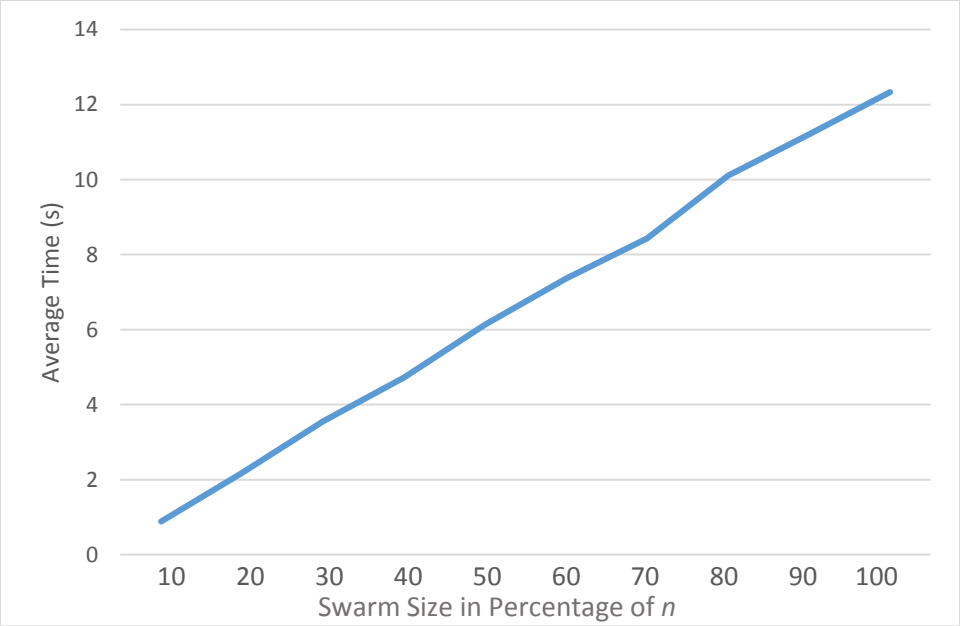


Figure 3.5: Chart showing speed of convergence of EC-DPSO on bays29 with swarm size ranging from 10 – 100% of n

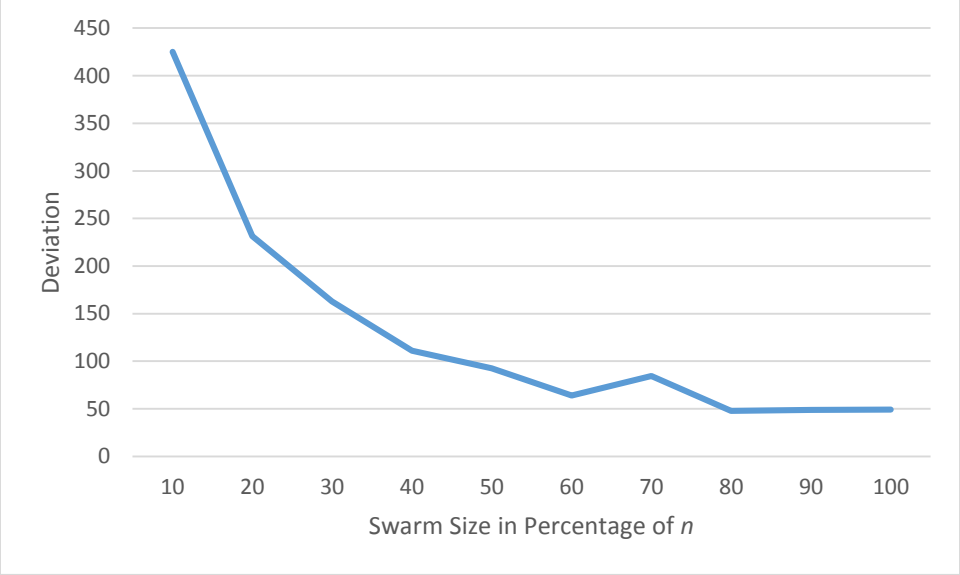


Figure 3.6: Chart showing divergence from optimal solution of EC-DPSO on bays29 with swarm size ranging from 10 – 100% of n

From the charts above, it can be seen that, as expected, speed increased almost perfectly uniformly as swarm size increased. However, the accuracy in terms of divergence from the optimal solution was not that uniform. There was a steady fall in divergence from the optimal solution between 10 and 60 percent, after which it can be argued that the performance was generally within the same range. This indicates that if the swarm size is set at 60 percent of *n*, the performance will not be significantly different from when it is set at *n*.

3.6 Summary and Conclusion

After successfully discretizing the Particle Swarm Optimization technique and successfully optimizing it through the achievement of enhanced communication between particles, it can be concluded that the EC-DPSO algorithm is a very promising one. The great improvement in results achieved by directing the particles towards positions formed by 'good' edges clearly proves this. By taking into account the impact which the inclusion of an edge makes on the fitness value achieved in the new position, the chances of edges with relatively high cost getting included was greatly reduced, and that of those with relatively very low cost increased. In addition, swarm size experimentations indicated that setting the swarm size to

around 50-60 percent of the size of the TSP instance *n*, as against the previous setting of *n* as the swarm size, significantly improved speed, while maintaining accurate results.

Chapter Four

Proposed Discrete Particle Swarm Optimization Algorithm for the Shortest Path Problem

4.0 Introduction

As has been highlighted in the literature review, various methods have been proffered for discretizing PSO, including binary methods [120, 123, 216], set-based methods [116, 144, 237], and fuzzy methods [1, 143, 238]. Most of the algorithms which are used to address SPP adopt the set-based approach. In the set-based method, the set represents a feasible path from source to destination node, where the order or arrangement of elements within the set uniquely identifies that path. This implies that the position of each element within the set is critical in path representation, and any computation upon the set which disrupts or disregards this order/position ultimately disregards the unique path which it represents. The proposed PSO discretization is therefore performed in such a way that operations such as addition, subtraction and so on which are intrinsic to the PSO algorithm are performed such that the position information of each element is maintained. The new DPSO algorithm for the Shortest Path Problem is presented in this chapter.

4.1 Position Encoding for SPP

A highly important factor when discretizing PSO is how a position is represented in discrete space. There are two general methods of representing a position, the *direct* and *indirect* encoding methods [156]. In the *direct* method which was used in [157], position is represented as a set of vertices (or edges) which directly represent a feasible *s*-*t* path in the graph. When position is represented as a set of vertices, numbers (IDs) are allocated to represent each vertex. For example, in the graph in Figure 4.1, the highlighted path would be represented as (2, 1, 8, 5, 4, 6).

In this encoding method, the size of the set representing a position varies with the length of the path. Hence, (2, 3, 6) and (2, 4, 5, 7, 6) are both valid positions. In some cases the source and destination nodes are left out of the set, since they are always the same, and are added when fitness evaluation is to be done. The *direct* method suffers a major disadvantage of having a much higher chance of representing invalid paths, since a permutation of nodes can represent any path containing edges which are not contained in the graph under consideration [156]. For example, a path represented by (2, 1, 4, 6) could be generated, but it is infeasible, as it does not lead to the destination node (Figure 4.1). The chance of generating invalid paths can be reduced if an *indirect* approach is used. Another distinguishing feature between direct and indirect encoding is that directly encoded positions can have varying lengths. That is, (1, 4, 2, 7) and (1, 2, 7) could be valid positions for a particle, but in indirect encoding, the cardinality of the position set is always fixed. For a graph with *n* nodes, a particle's position will always also be of size *n* The *indirect* encoding method has been successfully applied by various researchers, as can be seen in [85], [155] and [156]. In the indirect method, the information represented by the position *leads to* the generation of a path, as opposed to representing the path directly. This will however necessitate a means of decoding the information to produce the path represented by such a position. There are two ways of encoding a position indirectly: *priority-based* and *weight-based*. Gen, et al. [85] and

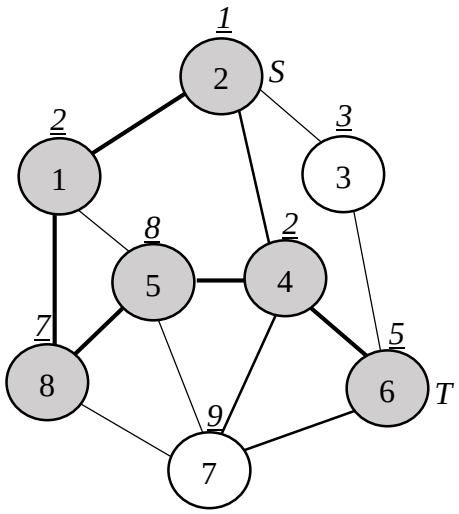


Figure 4.1: Graph showing nodes, their sample priority allocations and the corresponding path from nodes 2 to 6

Mohemmed, et al. [155] use the priority-based method for encoding an *s*-*t* path for SPP. In this method, each node in the graph is allocated a priority, and a position is a set of such priorities with each index in the set representing the priority of the corresponding node. Consequently, the size of the set is always fixed in a priority-based encoding scheme. Path generation (decoding) proceeds in a step-wise manner, starting from the source node, selecting the adjacent node with least priority. The process is repeated with the selected node until the destination node is reached.

For example, Figure 4.1 shows a position represented as (2, 1, 3, 2, 8, 5, 9, 7), which will be decoded as the path (2, 1, 8, 5, 4, 6) for an SPP with source and destination nodes as 2 and 6 respectively.

Although the priority-based method has been found to perform acceptably for SPP, its main drawback is that edge costs are not taken into consideration in the priority allocation. This is where the weight-based encoding approach proves to be a better option. Raidl and Julstrom [187] as well as Mohemmed, et al. [155] use the weight-based encoding method for SPP. In this method which was first described in [172], each vertex is associated with a weight. The cost of each edge is then biased by adding the weight of both its adjacent vertices to its cost:

$$c'_{ij} = w_i + c_{ij} + w_j \tag{4.1}$$

Therefore, each position is a set of weights whose indices represent the IDs of the respective vertices. However, Raidl and Julstrom [187] found that multiplying the edge costs by weights generated from a log-normal distribution led to a better performance than adding them. Therefore they performed cost biasing thus:

$$c'_{ij} = w_i \cdot c_{ij} \cdot w_j \tag{4.2}$$

They chose their weights from the distribution $(1+\gamma)^{\mathcal{N}(0,1)}$, where $\mathcal{N}(0,1)$ is the normal distribution. γ represents a parameter called *biasing strength*, which they set at 1.5, and their approach is adopted in this work. An example of a position represented with the weight-based encoding scheme is (0.3, 1.1, 0.6, 0.4, 0.9, 1.6, 0.2, 0.8) as depicted in Figure 4.2, which would generate the path [2, 1, 5, 7, 4, 6].

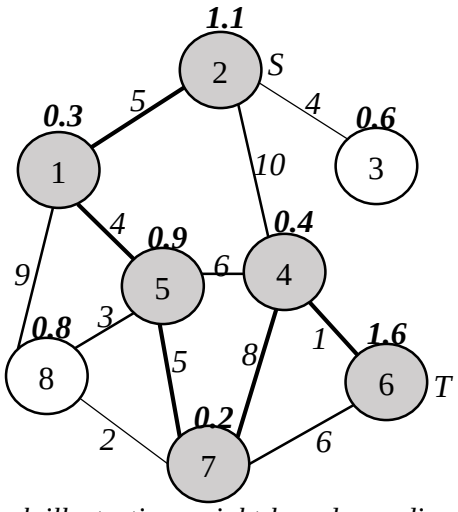


Figure 4.2: Graph illustrating weight-based encoding and resulting path

4.2 Path Decoding

The path decoding scheme which is used to transform particle positions to their corresponding paths for fitness evaluation is discussed in this section. A general decoding procedure is presented in Algorithm 4-I:

Algorithm 4-I – General Path Decoding Procedure

1. Initialize empty path, P
2. Add source vertex s to P
3. while (destination node not reached)
    a. Add node v with lowest biased weight to P
    b. Set added node as current node, v
4. Return P
5. End

Though Algorithm 4-I selects nodes based on lowest biased weight, it can be trivially adapted to select nodes based on highest priority.

An important issue is how to handle cases when a partial path leads to a dead end, that is, the path ends with a non-destination node. In [155] and [262], such nodes are referred to as "no exit nodes". The strategy adopted in the two aforementioned works is to terminate the path decoding process once such a node is detected, resulting in an invalid path which will be discarded. Other methods of handling such invalid paths include the use of a penalty value/function [249] or a repair function [9]. A better way of handling such a case without any chance of creating an invalid path is proposed. At the point in the path decoding process when a 'no-exit' node is detected, that node is discarded, and a simple backtracking is done to select the node with the next lowest biased cost. Also, in order to prevent loop formation, a simple check is conducted, and any node which is already part of the partial path discarded. Algorithm 4-II gives a complete description of the proposed decoding procedure:

Algorithm 4-II: Proposed Path Decoding Procedure

```
1. Initialize Stack P, source node, s and destination node, t, List invalid_nodes
2. P.push(s)
3. Set curr = s, next = null
4. Set min_weight = -∞
5. While (curr ≠ t){
6.     For i in adj[curr] {
7.         if (!(P.contains(i) or invalid_nodes.contains(i)){ // Loop prevention check
8.         biased_weight = weight[i] * weight[curr]
9.             if(biased_weight  < min _weight){
10.                     Set next = i
11.                     Set min_weight = biased_weight
12.                 }
13.             }
14.         }
```

```
15.       if(min_weighty == -∞) { // Detection of "no exit node" and backtracking
16.          invalid = P.pop()
17.          curr = P.peek()
18.          invalid_nodes.add(invalid)
19.          continue
20.       }
21.       curr = next
22.       P.push(curr)
23.    }
24.    return P
```

Though the proposed algorithm selects nodes based on lowest biased weight, it can be trivially adapted to select nodes based on highest priority. The proposed DPSO can therefore also be used both for priority-based and weight-biased encoding schemes with some minor adaptations to suit each case.

4.3 PSO Equation Discretization

As a reminder, the constriction-coefficient PSO which is adopted in this work has the following equations [51]:

$$V_i^{t+1} = \chi[V_i^t + c_1 r_1^t (pbest_i^t - X_i^t) + c_2 r_2^t (gbest^t - X_i^t)] \quad (4.3)$$

$$X_i^{t+1} = X_i^t + V_i^{t+1} \quad (4.4)$$

where t is the iteration counter, i is the particle index, *pbest* is the personal best, *gbest* is the global best, r_1 and r_2 are random numbers in the interval [0, 1], c_1 is the referred to as the cognitive parameter which dictates how biased the particle's search will be biased towards its personal best position, and c_2 is the social parameter which dictates how much its search will be biased towards the global best. The c_1 and c_2 parameters can be tweaked to favor either exploitation or exploration respectively.

χ is defined as

$$\chi = \frac{2}{\left|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}\right|} \tag{4.5}$$

where $\varphi = c_1 + c_2$, and $\varphi > 4$

It must be noted here that the manner in which the various elements of the PSO equations are discretized will have a very significant impact on the performance of the discrete PSO algorithm. A new set-based discretization method which is believed to give the proposed algorithm an edge over existing PSO discretization methods used for SPP is presented below.

1. **Velocity:** In a set-based discretization scheme, operations such as addition and subtraction are usually redefined in terms of set union and difference operators [44]. It was discovered that when these operations are carried out in discretization schemes that map priorities or weights to node indices, vital data related to the index mappings of either node priorities or weights are lost. Obviously, the major point in such schemes is the priority/weight-to-node mapping. For example, consider two positions, *A = (0.2, 0.6, 1.1, 0.5)* and *B = (0.9, 0.6, 1.1, 2.1)*. The result of the operation, *A-B* would be *(0.2, 0.5)*. Priority/weight 0.5, which was originally mapped to node 4 in *A*, now points to node 2 in the resulting difference set. This does not represent the original sets which were involved in the set difference operation. Furthermore, in the course of the PSO algorithm, the result of this difference operation will be combined in some way with other sets to form a new position. The new position then becomes completely different from the previous one in terms of node-priority/weight mappings, severely limiting the search capability of particles. For this reason, a different scheme of representation for velocity is proposed which retains this index mapping information throughout the velocity and position update operations. Velocity is defined in terms of *direction* and *magnitude*, direction indicating the specific node indices within a position whose priorities/weights will be altered and their associated priorities/weights which will be used in replacement, and magnitude, the number of indices whose priority/weight values will be replaced. That is, velocity is defined as

$$V = \{(u_1, v_1), (u_2, v_2), \dots, (u_n, v_n)\}_k \tag{4.6}$$

where u_i is the ith index, v_i is its associated priority/weight, and k is the magnitude of V.

For example, a velocity could be $\{(7, 0.2), (5, 0.4), (1, 0.3), (4, 0.1), (2, 0.3)\}_3$.

2. **Subtraction ($\ominus$):** In line with the above definition of velocity, subtraction is redefined thus: Given two sets, *A* and *B*, *A*-*B* is gotten by comparing the elements in *A* to those in *B* index-wise. That is, compare the element at index 1 in *A* with the element at index 1 in *B* and so on, forming the resulting set with those elements in *A* which are different from their counterparts in *B.* Thus, the resulting set maintains information about node-weight mappings. Therefore, for the example given above where *A = (0.2, 0.6, 1.1, 0.5)* and *B = (0.9, 0.6, 1.1, 2.1), A – B = {(1, 0.2), (4, 0.5)}*$_2$, where 2 is the size of the resulting set, which will be useful subsequently.
3. **Multiplication ($\odot$):** A multiplication operation between a real number and a velocity set results in that set with its magnitude component changed to the closest approximate integer resulting from a multiplication operation between that number and the magnitude component of the velocity set. Continuing with the above example, if $C = A - B$, then

$$0.4 \times C = \{(1, 0.2), (4, 0.5)\}_{0.4\times2}$$
$$\Rightarrow 0.4 \times C = \{(1, 0.2), (0.5)\}_1$$

4. **Addition ($\oplus$):** In adding two velocity sets *A* with magnitude k_1 and *B* with magnitude k_2 together, randomly select k_1 elements from *B* and then k_2 elements from *A* to form the resulting sum set with a resultant magnitude of $k_1 + k_2$. An example is given below:
   Let $A = \{(2, 0.6), (4, 0.5)\}_1$ and $B = \{(3, 0.3), (2, 1.3), (1, 0.1), (7, 2.3)\}_2$.
   Then $A + B = \{(1, 0.1), (3, 0.3), (2, 0.6)\}_3$
   In selecting elements, it should be ensured that no elements with the same index values are present in the resulting sum. Also, it was decided to pick first from *B* so as to favor *gbest* over *pbest* in equation (4.3), and hence speed up convergence.
5. **Position Update ($\otimes$):** Position update is conducted as per equation (4.8) by replacing *k* randomly selected priorities/weights in the current position with those in the updated velocity according to their associate indices. An example will suffice to make this concept clearer.
   Let X_i^t *= (1.3, 0.5, 0.8, 0.3, 0.2, 1.0)* and V_i^{t+1} *= {(1, 0.1), (3, 0.3), (2, 0.6)}*$_2$.
   To obtain X_i^{t+1}, replace the value at index 2 of X_i^t with 0.6 and the value at index 3 with 0.3. Thus, the new position will be:
   X_i^{t+1}*= (1.3, 0.6, 0.3, 0.2, 1.0)*

Thus, the updated equations will be:

$$V_i^{t+1} = \chi[V_i^t \oplus c_1 r_1^t \odot (pbest_i^t \ominus X_i^t) \oplus c_2 r_2^t \odot (gbest^t \ominus X_i^t)] \quad (4.7)$$

$$X_i^{t+1} = X_i^t \otimes V_i^{t+1} \quad (4.8)$$

4.4 Simulation

A brief simulation is presented to demonstrate the discretization scheme as a whole.

Let $pbest_i^t = (0.4, 2.1, 0.3, 1.1, 1.2, 0{,}9, 2.3)$, $gbest_i = (1.0, 0.2, 0.7, 0.3, 1.7, 2.8, 0.4)$, $X_i^t = (0.6, 2.1, 0.7, 1.3, 1.9, 0.5, 2.3)$, $V_i^t = \{(2, 0.2), (5, 0.8), (1, 1.3), (6, 01)\}_3$.

Then from definition 2,

$pbest_i^t - X_i^t = \{(1, 0.4), (3, 0.3), (4, 1.1), (5, 1.2), (6, 0.9)\}_5$

$gbest - X_i^t = \{(1, 1.0), (2, 0.2), (4, 0.3), (5, 1.7), (6, 2.8), (7, 0.4)\}_6$

Let $c1 = 2.05, r1 = 0.5, c2 = 0.5, r2 = 0.5$

From definition 3,

$c1r1(pbest_i^t - X_i^t) = 1.025\{(1, 0.4), (3, 0.3), (4, 1.1), (5, 1.2), (6, 0.9)\}_5$

$= \{(1, 0.4), (3, 0.3), (4, 1.1), (5, 1.2), (6, 0.9)\}_{5 * 1.025}$

$= \{(1, 0.4), (3, 0.3), (4, 1.1), (5, 1.2), (6, 0.9)\}_5$

$c2r2(gbest - X_i^t) = 0.25\{(1, 1.0), (2, 0.2), (4, 0.3), (5, 1.7), (6, 2.8), (7, 0.4)\}_6$

$= \{(1, 1.0), (2, 0.2), (4, 0.3), (5, 1.7), (6, 2.8), (7, 0.4)\}_{6 * 0.25}$

$= \{(1, 1.0), (2, 0.2), (4, 0.3), (5, 1.7), (6, 2.8), (7, 0.4)\}_2$

From definition 4,

$c1r1(pbest_i^t - X_i^t) + c2r2(gbest - X_i^t) = \{(1, 0.4), (3, 0.3), (4, 1.1), (5, 1.2), (6, 0.9)\}_5$

$+ \{(1, 1.0), (2, 0.2), (4, 0.3), (5, 1.7), (6, 2.8), (7, 0.4)\}_2$

$= \{(2, 0.2), (6, 2.8), (1, 0.4), (3, 0.3), (4, 1.1), (5, 1.2)\}_7$

$V_i^t + c1r1(pbest_i^t - X_i^t) + c2r2(gbest - X_i^t) = \{(2, 0.2), (5, 0.8), (1, 1.3), (6, 0.1)\}_3$

$+ \{(2, 0.2), (6, 2.8), (1, 0.4), (3, 0.3), (4, 1.1), (5, 1.2)\}_7$

$= \{(2, 0.2), (6, 2.8), (1, 0.4), (3, 0.3), (4, 1.1), (5, 1.2)\}_7$

Note that whenever the magnitude sum exceeds *n*, which is 7 in this case, it is clamped it to 7. Also, as was earlier stated, once an index value is present in the resulting sum, no new element with the same index value is added.

Now, applying equation (4.7),

$V_i^{t+1} = \chi[V_i^t \oplus c_1 r_1^t (pbest_i^t \ominus X_i^t) \oplus c_2 r_2^t (gbest^t \ominus X_i^t)]$

$= 0.729 * \{(2, 0.2), (6, 2.8), (1, 0.4), (3, 0.3), (4, 1.1), (5, 1.2)\}_7$

$= \{(2, 0.2), (6, 2.8), (1, 0.4), (3, 0.3), (4, 1.1), (5, 1.2)\}_{7 * 0.729}$

$= \{(2, 0.2), (6, 2.8), (1, 0.4), (3, 0.3), (4, 1.1), (5, 1.2)\}_5$

Note that the χ value recommended in [175] as 0.729 is used here.

From equation (4.8) and definition 5,

$X_i^{t+1} = X_i^t \otimes V_i^{t+1}$

$= (0.6, 2.1, 0.7, 1.3, 1.9, 0.5, 2.3) \otimes \{(2, 0.2), (6, 2.8), (1, 0.4), (3, 0.3), (4, 1.1), (5, 1.2)\}_5$

$= (0.4, 2.1, 0.3, 1.1, 1.2, 2.8, 2.3)$

4.5 Summary and Conclusion

In this chapter, the proposed DPSO algorithm was presented with an explanation of the various modifications and redefinitions that had to be made to the original continuous PSO, and the presentation of a new path decoding algorithm. A brief simulation to illustrate the PSO operation redefinitions in clearer terms was also presented. In the subsequent chapters, this algorithm is applied with a few adaptations where necessary, to the SPP, MOSSP, and SDP.

Chapter Five

A Memetic Discrete Particle Swarm Optimization Method for the Shortest Path Problem

5.0 Introduction

Given an undirected, connected graph *G* with a set *V* of vertices and a set *E* of edges defined as

$$E = \{(i,j)|i,j \epsilon V\} \tag{5.1}$$

each of which have associated costs c_{ij}, the Shortest Path Problem (SPP) seeks to find a path between a source vertex $s \epsilon V$, and a destination or sink vertex $t \epsilon V$, the total cost of whose constituent edges is minimized. Mathematically,

$$\underset{p \epsilon \mathrm{P}}{min} \sum_{(i,j) \epsilon V} c_{ij} \tag{5.2}$$

Where P is the set of all *s*-*t* paths defined as sequences of non-repeating *(i,j)* edges starting at *s* and terminating at *t*. In the problem case under consideration, the shortest path is sought between a specified single source node and single destination node.

Apart from its ubiquitous applications, the SPP is also important because it forms a base for other NP-complete problems such as weight-constrained shortest path problem, multi-objective shortest path problem, bottleneck shortest path problem and so on [68, 223, 236]. Though exact methods have been proffered to solve the classical SPP such as the Dijkstra and Bellman-Ford algorithms [207, 255], these exact methods have a major drawback of not being extendable its variants. They also cannot be used to determine other similar or dissimilar short routes, in addition to the fact that they perform poorly in real-time rapidly changing environments [202, 246].

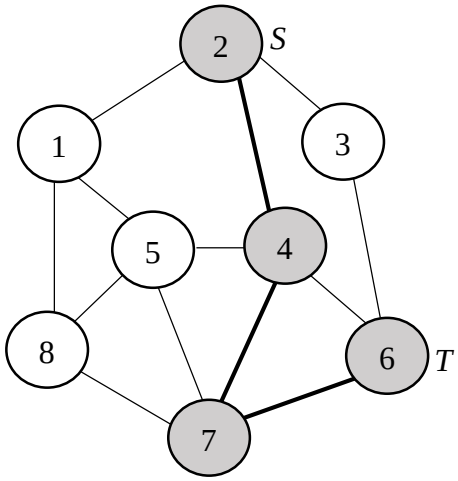


Figure 5.1: Graph displaying a path from node 2 to node 6

Therefore, various heuristic methods such as A* and branch pruning [83, 100] have been proposed to solve the SPP and avoid these drawbacks. Methods in the field of Swarm Intelligence have also been developed to solve the problem. These algorithms have been discussed in sections 2.3 and 2.4..

5.1 Application of Proposed DPSO for SPP

The proposed DPSO algorithm as presented in Chapter four is now applied to the SPP, incorporating the memetic algorithm discussed in section 2.3.6 to improve its local search capability. The results obtained are compared with those reported in [156], [155], and [85].

A randomly negative or positive λ value of 0.5 in the memetic component was found to perform better than other values after experimentation. This takes care of cases where the search needs to be towards regions where weight is lower than the current position as well as regions where weight is higher.

5.2 Values of *c1* and *c2*

The values of *c1* and *c2* are known as the cognitive and social parameters respectively, and they are used to control the level of exploitation or exploration of the algorithm. Where *c1* is higher than *c2*, exploration is favored, and where *c2* is higher than c1, exploitation is being favored [175]. Therefore, upon close examination of the algorithm's performance, it was

Table 5.1: Comparison of Proposed DPSO with two other DPSOs and a GA

S/N	Number of nodes	Number of edges	DPSO in [155]		GA in [85]		DPSO in [156]		Proposed DPSO	
			Success Rate	No of Iterations	Success Rate	No. of Iterations	Success Rate	No of Iterations	Success Rate	No of Iterations
1	6	10	100	50	100	50	-	-	100	**40**
2	32	66	100	100	98	100	-	-	100	**60**
3	50	166	-	-	-	-	95	44	**99**	50
4	60	233	-	-	-	-	89	60	**94**	60
5	70	211, 216, 212	80	200	64	200	91	60	**98**	60
6	70	326	-	-	-	-	86	78	**98**	**60**
7	80	188	-	-	-	-	92	74	**95**	**60**
8	80	232	-	-	-	-	93	77	**96**	**60**
9	90	230	-	-	-	-	88	90	**96**	**60**
10	90	251	-	-	-	-	94	86	**97**	**60**
11	100	255	-	-	-	-	90	95	**94**	**60**
12	100	280	-	-	-	-	90	107	**95**	**60**

discovered that the default settings of the values of both *c1* and *c2* at 2.05 [175] were not favorable in this case. This is because with these values, the *magnitude* value of the operations $c_1 r_1^t(pbest_i^t - X_i^t)$ or $c_2 r_2^t(gbest^t - X_i^t)$ will be at such a high level, that during position update, the particle has a very high probability of flying directly to the position of the *gbest* or *pbest*, as the case may be. This leads to premature convergence and makes the local search capability of the algorithm very poor. Experiments were therefore carried out to find the most suitable setting of c1 and c2 for the algorithm, and it was discovered that setting their values at 1.48 and 0.5 respectively leads to an improved performance. As was explained earlier, this setting encourages exploitative search around the particle's personal best as against a premature direct flight to the swarm's global best position.

5.3 Results and Discussion

A comparison of the three variants of the proposed DPSO algorithm is presented in Table 5.2. The variant with default *c1* and *c2* settings will be called $DPSO_a$, the memetic variant with default *c1* and *c2* settings, $DPSO_b$, and the proposed memetic algorithm with the new *c1* and *c2* values, $DPSO_c$. Each variant was implemented in Java and executed on a computer with Intel® Core™i7 CPU at 3.10GHz with 8.00GB RAM running a Windows 8.1 Operating System. Each graph instance (with a given number of nodes and edges) was randomly generated 100 times, with edge costs taken randomly from the range [10, 1000]. As a result of the stochastic nature of the algorithm, performance slightly varied over each of the 100 runs. Therefore, for each variant and in each instance, the algorithm was run multiple times and the average performance taken, so as to depict a clear picture of its performance both in terms of accuracy and consistency.

Table 5.2: Comparing performance of three DPSO variants for SPP

			$DPSO_a$		**$DPSO_b$**		**$DPSO_c$**	
S/N	**Nodes**	**Edges**	**Accuracy**	**Iterations**	**Accuracy**	**Iterations**	**Accuracy**	**Iterations**
1	200	450	0.83	90	0.84	90	0.91	90
2	150	400	0.83	90	0.87	90	0.90	90
3	100	280	0.83	60	0.86	60	0.93	60
4	100	255	0.82	60	0.88	60	0.91	60
5	90	251	0.81	60	0.91	60	0.92	60
6	90	230	0.84	60	0.88	60	0.92	60
7	80	232	0.85	50	0.88	50	0.93	60
8	80	188	0.82	50	0.86	50	0.93	60
9	70	326	0.85	40	0.87	40	0.96	60
10	70	212	0.85	40	0.87	40	0.93	60
11	60	233	0.85	40	0.9	40	0.93	60
12	50	160	0.89	40	0.92	40	0.95	60

The chart in Figure 5.1 shows the improvement in performance that has been achieved both by selecting the right values for *c1* and *c2*, and by implementing a memetic version with both global and local search components. The nature of the discrete algorithm inherently encourages global search as has been stated in the definition of the addition operation for the velocity update equation. Hence, selecting *c1* and *c2* parameters which favor exploitation has proven to improve the algorithm's local search capability. Further improvement has been achieved by making the algorithm memetic with the addition of a local search component, enabling it to search for better positions around the current global best. Therefore, it can be said that a balance in both exploitation and exploration has been achieved by the addition of these two features to the algorithm, with the resultant significant improvement in its performance.

Table 5.1 shows the viability of the proposed DPSO algorithm in the light of the performances of others reported in [155], [156] and [85].

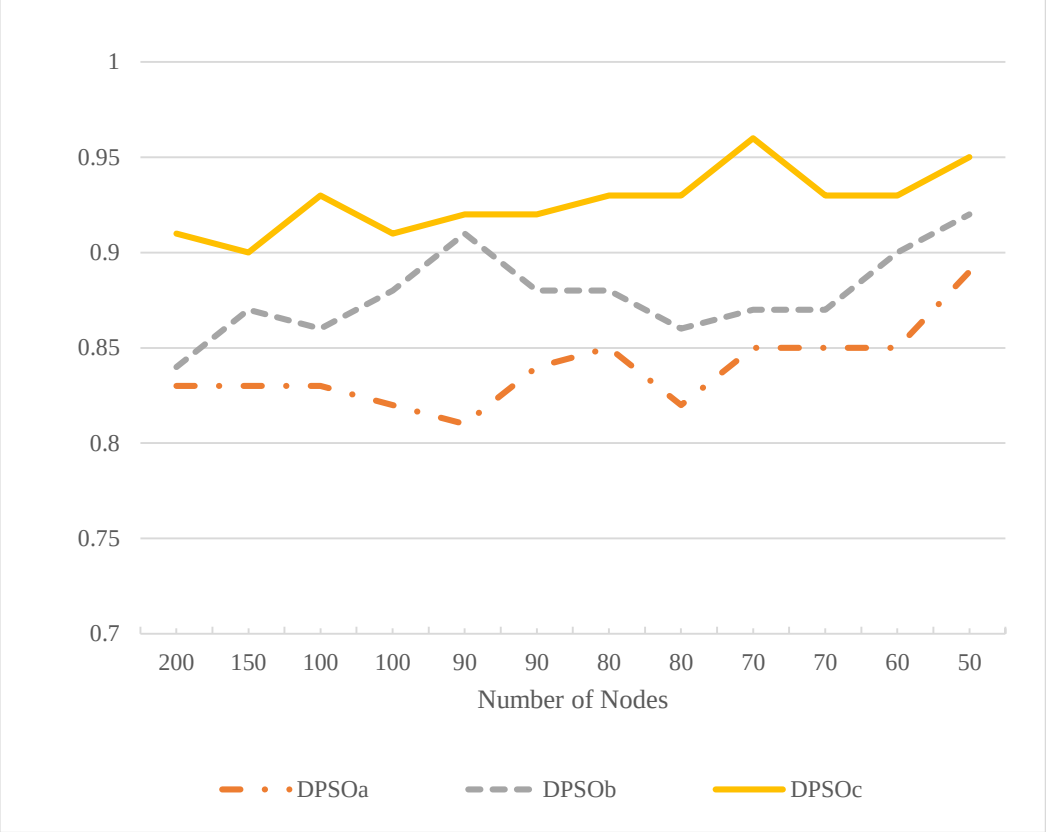


Figure 5.3: Graphical comparison of the performance of three DPSO variants

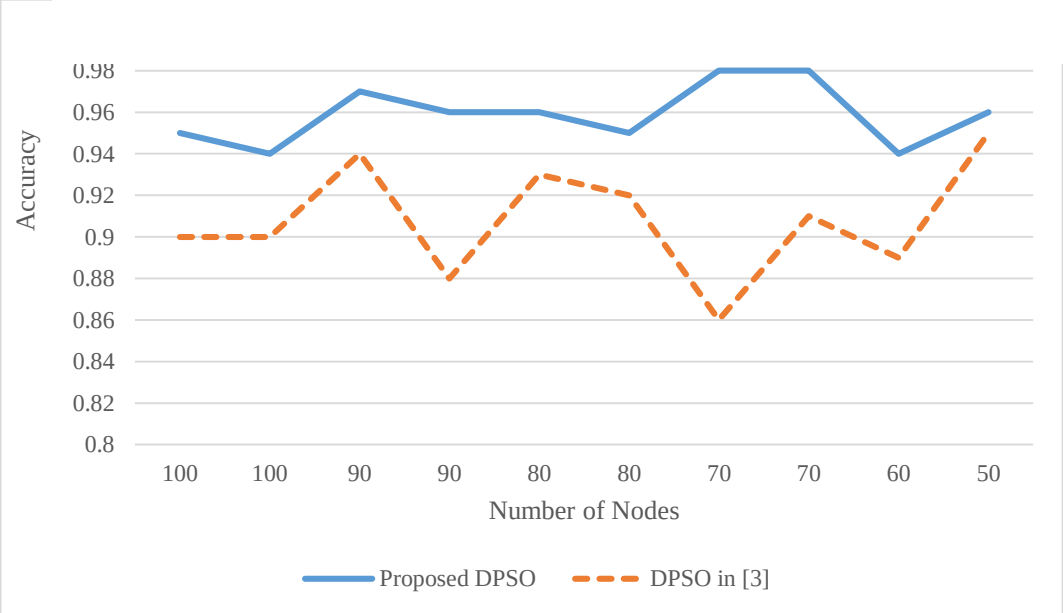


Figure 5.2*: Comparing proposed DPSO with DPSO in [156]*

The proposed algorithm performs comparatively with all compared algorithms in terms of accuracy, and in most cases, with a lower number of iterations. This proves that the novel discretization method introduced with the algorithm is a strong and viable one. As is shown in Table 5.2, even with the basic variant of the algorithm, performance in all cases never goes below 80%. In Table 5.2, it can also be seen that the algorithm was able to solve SPP instances of size of up to 200 nodes at ≥ 90% accuracy, whereas the largest number of nodes solved from sampled literature is 100. This proves that the method used to discretize PSO for a specific combinatorial optimization problem greatly determines the resulting algorithm's efficiency. Tailoring the discretization of the PSO equations to best fit the search space, in this case, the weight-biased representation, has proven to be a very definitive factor in determining performance. The chart in Figure 5.2 highlights the better performance of the proposed algorithm in all shown instances compared to the DPSO proposed in [156], which was used to solve problems of highest size among all three algorithms in literature with which the proposed algorithm is compared.

5.4 Summary and Conclusion

The proposed memetic DPSO algorithm discussed in chapter four was applied to SPP. Optimal values were also selected for *c1* and *c2* with improved results. The local search capability of the memetic component of the algorithm further led to more promising results which, in comparison with those of three other algorithms proposed for the SPP.

Chapter Six

A Discrete Particle Swarm Optimizer for the Multi-Objective Stochastic Shortest Path Problem

6.0 Introduction

The Shortest Path Problem (SPP) which was first defined and solved by Dijkstra in 1958 [65] has been extensively studied by scientists over the years, with various variants being proposed and numerous solution techniques applied to them. However, it has been observed that the SPP in its deterministic form does not adequately represent real-world situations with their inherent uncertainties (such as the road/link congestion, accident occurrence and so on), leading to the solutions proffered for deterministic SPPs ending up failing [249, 262]. The Stochastic Shortest Path problem (SSP) assigns probability distributions associated with costs to each edge in the graph, and is usually solved by seeking the path with the least expected length [157]. A good deal of research has been made into solving various SSPs in literature, from exact methods like value and policy iteration [26] and quasi-convex maximization [167], to heuristics and meta-heuristics like A* [42], Genetic Algorithms (GA) [188] and Particle Swarm Optimization (PSO) [262].

Moreover, most users do not only desire to know the path of least-expected cost, but desire some level of certainty in the chosen path as well. This has led to multi-objective SSPs with objectives like least expected path cost, variance, probability of achieving cost at or below a given value, expected cost to guarantee a given probability, and so on [188]. As was outlined by Rajabi-Bahaabadi, et al. [188], various approaches have also been applied to solve these multi-objective SSPs. In the area of nature-inspired/evolutionary/swarm intelligence algorithms for multi-objective SSPs, GAs have been applied with good results [115, 188].

However, except for the work done in [262], the application of MOPSO to SSP seems not to have received much attention in literature, despite its good qualities of fast convergence, lower computational cost, fewer lines of code and relatively minimal required computational book keeping [119]. In [262], a hybrid algorithm made of Artificial Immune System and a

chaotic PSO is applied to a multi-objective SSP where the mean and variance of path congestion are minimized (stochastic variables) as well as the distance (deterministic variable). The performance of their algorithm was compared with an existing bi-objective Genetic Algorithm reported in [114] and a single-objective PSO method proposed for SSP in [156]. They reported that their algorithm outperformed both algorithms with which it was compared, though it is unclear on what basis they were able to compare the performance of their 3-objective algorithm against 2- and 1- objective algorithms. Theirs was the only work found in literature which applies PSO to MOSSP. In this chapter, the new DPSO for SPP is applied to solve the least expected cost SSP (known as SSP-E [189]) as a means of validating it. Two requirements are then combined (minimizing variance and expected cost) in a multi-objective discrete Particle Swarm Optimization algorithm, and experiments carried out to investigate and validate results.

6.1 Adaptation of Proposed DPSO for SSP and MOSSP

All PSO equation redefinitions, as well as the algorithm presented in **Algorithm 4-II** remain the same for the SSP, except that cost is redefined as expected (mean) cost. However, an important modification had to be made to the proposed weight biasing technique originally proposed in [187] as:

$$w_{i,j} = w_i \cdot w_j \cdot c_{ij} \tag{6.1}$$

In applying equation (6.1) to a multi-objective setting, it was noticed that as expected, the search procedure was being hampered by the bias towards only edge costs at the expense of other objectives such as variance. An attempt was then made to implement objective aggregation methods as proposed in [176] , but the results were not favorable. However, upon further investigation, the following equation which removes bias towards any of the objective functions was used with very favorable results:

$$w_{i,j} = w_i * w_j \tag{6.2}$$

In single-objective cases however, the use of equation (6.1) is maintained.

A possible position could be (0.3, 1.4, 2.1, 0.8) for a 4-node sized graph as illustrated in Figure 6.1.

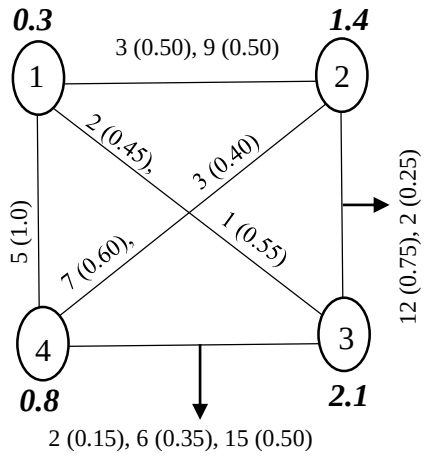


Figure 6.1: Sample graph showing edge cost probability distributions and node weights (emphasized)

6.2 Results of Proposed DPSO on SSP

In this section, the results of applying the proposed DPSO algorithm to the SSP-E where the aim is to minimize total expected cost in a stochastic graph is presented. The objective function for the SSP-E is [39, 113]:

$$Min\ E\left[\sum_{(i,j)\in\mathcal{A}} \mu_{ij}x_{ij}\right] \tag{6.3}$$

Subject to

$$\sum_{(1,j)\in\mathcal{A}} x_{1j} - \sum_{(j,1)\in\mathcal{A}} x_{j1} = 1, \tag{6.4}$$

$$\sum_{(i,j)\in\mathcal{A}} x_{ij} - \sum_{(j,i)\in\mathcal{A}} x_{ji} = 0, \quad 2 \le i \le n-1 \tag{6.5}$$

$$\sum_{(n,j)\in\mathcal{A}} x_{nj} - \sum_{(j,n)\in\mathcal{A}} x_{jn} = -1, \tag{6.6}$$

$$x_{ij} \in \{0,1\}, \forall (i,j) \in \mathcal{A} \tag{6.7}$$

where μ_{ij} is the expected length of edge (i,j), $\mathcal{A}$ is the set of all edges in the graph, constraints (14) – (16) ensure that the edges in the path are feasible, connecting from origin 1 to destination *n*, and constraint (17) ensures that the decision variables are binary.

In [157], a discrete PSO method was proposed and applied to the SSP-E problem, and the result compared with Distributed Learning Automata methods in [154] and [22]. Their results proved the applicability of DPSO to SSP-E by reducing the number of iterations by over 9000. They also compared their results with the results gotten from solving the problem with Dijkstra's algorithm, reporting the same optimal paths as that gotten from Dijkstra's but with lower cost. It is however unclear how they were able to achieve different costs on the same path using the same objective function for SSP-E stated above. The algorithm was tested with *c1* and *c2* settings as 1.48 and 0.5 respectively (after experimentation) on the three test cases (Graphs 1 – 3) presented in their paper. The results obtained are compared with theirs with varying swarm sizes for Graph 3 (15 nodes, 43 edges) over 12 runs in Table 6.1.

Table 6.1: Comparing Performance of Proposed Algorithm with DPSO in [157]

Swarm Size	Best Converged Path		Cost of Converged Path		Number of Convergences		Percentage of Convergence	
	DPSO in [157]	Proposed DPSO	DPSO in [157]	Proposed DPSO	DPSO in [157]	Proposed DPSO	DPSO in [157]	Proposed DPSO
25	1 2 5 15	1 2 5 15	63.292	64.5	4	12	33.33%	100%
50	1 2 5 15	1 2 5 15	61.1428	64.5	8	12	66.6%	100%
100	1 2 5 15	1 2 5 15	62.7442	64.5	6	12	50.0%	100%
150	1 2 5 15	1 2 5 15	63.0834	64.5	9	12	75.0%	100%

Results obtained are also compared with those obtained by Dijkstra's algorithm and those in [157] over all three test cases in Table 6.2:

Table 6.2: Comparison of Proposed DPSO with Dijkstra and DPSO in [157] over three instances

Graphs	Shortest Path for DPSO in [157]	Path Cost	Shortest Path from Proposed DPSO	Path Cost	Shortest Path from Dijkstra	Path Cost
Graph 1	1 3 7 10	14.84535	1 3 7 10	15.22	1 3 7 10	15.22
Graph 2	1 4 9 10	15.49471	1 4 9 10	16.1	1 4 9 10	16.1
Graph 3	1 2 5 15	62.46404	1 2 5 15	64.5	1 2 5 15	64.5

The performance of the proposed DPSO is clearly better than that proposed in [157] since it was able to consistently find the path with least expected cost over all tested swarm sizes with a 100% accuracy. As a further proof of the viability of the algorithm, three more graphs of dimension (50, 150), (100, 250), and (200, 400) were randomly generated with probability distributions generated with the scheme detailed in [208]. Associated integer costs were also randomly generated within the interval [1, 100). For each dimension, 50 cases were randomly generated and the algorithm's performance tested over swarm sizes of 25, 50, 100, and 150. Number of iterations was fixed at 50. The algorithm's accuracy was validated against Dijkstra's algorithm and the percentage accuracy reported. The results are presented in Table 6.3.

Table 6.3: Performance of Proposed DPSO on SSP instances of various sizes

Instances	Swarm Size	Accuracy	Average Time (s)
50, 150	25	100%	0.0526
	50	100%	0.0940
	100	100%	0.1937
100, 250	25	100%	0.1792
	50	100%	0.3774

	100	100%	0.7584
200, 400	25	92%	0.9880
	50	100%	2.2301
	100	100%	3.2276

Table 6.3, shows that the proposed DPSO algorithm is effective in solving the SSP-E problem. However, in practical applications, users are usually not just interested in the path of least expected cost. Most are also concerned with the reliability of the recommended path, especially in applications that have to do with road networks and expected travel time [39]. This leads to the need for a multi-objective algorithm which will proffer pareto-optimal solutions, giving users the chance to make a compromise between these conflicting objectives.

6.3 Results of Proposed DPSO on MOSSP

An attempt was made to compare the results obtained upon applying the proposed DPSO with those presented in [262], but the data presented was unusable, as the probability distributions presented had no associated costs. Because, as was stated earlier, Zhang, et al. [262]'s work was the only one found in literature solving MOSSPs using PSO, and the only other recent work which solves MOSSPs with a Swarm Intelligence approach (Ant Colony Optimization) which was presented in [75] deals with stochastic nodes which is beyond the scope of this work, we randomly generated and present our own data. To prove the viability of the proposed algorithm, three SSP instances with 25 nodes and 50 edges (small instances for the sake of space constraints) were randomly generated with the following probability distributions: $N(\mu, \sigma^2)$ denoting a normal distribution, $E(\lambda)$ an exponential distribution, $U(a, b)$ a uniform distribution, and $T(a, b, c)$ a triangular distribution. The three instances are presented in Table 6.4 - Table 6.6.

Table 6.4: Graph Instance 1

Arc	Distribution Function	Arc	Distribution Function	Arc	Distribution Function	Arc	Distribution Function	Arc	Distribution Function
(1, 17)	E(9)	(11, 22)	N(28, 4)	(21, 11)	T(15, 18, 15)	(3, 11)	U(4, 23)	(18, 6)	T(15, 25, 25)
(2, 14)	U(4, 11)	(12, 23)	U(6, 15)	(22, 3)	U(8, 26)	(15, 4)	T(9, 28, 14)	(5, 21)	U(9, 19)
(3, 25)	U(13, 15)	(13, 4)	N(24, 4)	(23, 5)	N(25, 4)	(20, 18)	U(12, 27)	(20, 9)	T(12, 17, 13)
(4, 12)	E(9)	(14, 7)	U(22, 29)	(24, 18)	T(15, 23, 15)	(5, 23)	N(22, 24)	(21, 4)	U(15, 18)

(5, 14)	U(2, 27)	(15, 22)	T(2, 12, 4)	(25, 17)	U(4, 20)	(17, 25)	U(9, 20)	(14, 11)	T(17, 21, 21)
(6, 16)	U(4, 16)	(16, 20)	E(19)	(1, 7)	U(18, 27)	(13, 8)	U(1, 14)	(12, 10)	T(0, 12, 6)
(7, 19)	T(6, 15, 15)	(17, 7)	N(18, 20)	(18, 12)	U(8, 12)	(11, 13)	T(8, 17, 17)	(21, 3)	U(11, 18)
(8, 4)	E(2)	(18, 8)	T(3, 15, 4)	(5, 8)	T(7, 14, 14)	(10, 24)	N(4, 17)	(21, 19)	E(29)
(9, 1)	N(0, 24)	(19, 22)	T(5, 11, 11)	(8, 9)	N(26, 1)	(2, 3)	N(15, 11)	(1, 3)	U(12, 23)
(10, 9)	E(13)	(20, 8)	U(2, 12)	(2, 13)	N(1, 1)	(23, 8)	U(8, 10)	(20, 19)	E(21)

Table 6.5: Graph Instance 2

Arc	Distribution Function	Arc	Distribution Function	Arc	Distribution Function	Arc	Distribution Function	Arc	Distribution Function
(1, 23)	E(12)	(11, 17)	E(1)	(21, 4)	U(6, 17)	(25, 2)	U(17, 18)	(16, 22)	E(19)
(2, 24)	T(18, 21, 21)	(12, 15)	U(11, 23)	(22, 10)	T(11, 20, 19)	(1, 15)	E(7)	(12, 13)	T(9, 25, 21)
(3, 19)	T(6, 19, 19)	(13, 25)	N(10, 15)	(23, 14)	E(23)	(2, 20)	N(9, 20)	(12, 10)	E(10)
(4, 1)	T(1, 18, 4)	(14, 18)	N(8, 15)	(24, 6)	T(2, 15, 6)	(11, 5)	E(10)	(10, 3)	U(17, 29)
(5, 4)	T(7, 16, 16)	(15, 11)	T(1, 8, 4)	(25, 6)	N(5, 25)	(11, 3)	N(2, 18)	(3, 4)	E(15)
(6, 9)	N(4, 28)	(16, 17)	N(16, 11)	(24, 4)	U(5, 21)	(9, 1)	E(6)	(23, 20)	T(2, 4, 3)
(7, 16)	U(0, 9)	(17, 4)	U(15, 22)	(3, 2)	N(10, 26)	(13, 8)	E(8)	(24, 25)	T(15, 27, 21)
(8, 1)	U(16, 25)	(18, 1)	N(2, 9)	(14, 22)	N(8, 19)	(21, 13)	N(9, 14)	(19, 25)	U(7, 20)
(9, 1)	T(18, 19, 18)	(19, 13)	U(13, 25)	(4, 19)	N(18, 5)	(15, 23)	T(15, 24, 15)	(21, 23)	E(2)
(10, 19)	U(7, 27)	(20, 2)	E(13)	(1, 10)	T(13, 16, 15)	(17, 22)	E(21)	(14, 2)	E(29)

Table 6.6: Graph Instance 3

Arc	Distribution Function	Arc	Distribution Function	Arc	Distribution Function	Arc	Distribution Function	Arc	Distribution Function
(1, 7)	N(13, 16)	(11, 4)	E(8)	(21, 23)	T(3, 9, 8)	(13, 3)	T(6, 30, 16)	(17, 15)	N(3, 13)
(2, 22)	E(27)	(12, 15)	U(7, 17)	(22, 12)	U(7, 25)	(14, 24)	T(6, 10, 9)	(14, 2)	E(2)
(3, 8)	T(17, 25, 23)	(13, 8)	E(10)	(23, 3)	T(9, 15, 9)	(20, 17)	N(25, 5)	(3, 11)	E(11)
(4, 22)	T(14, 22, 14)	(14, 11)	U(15, 17)	(24, 12)	U(11, 20)	(8, 15)	T(15, 19, 19)	(25, 12)	T(0, 9, 5)
(5, 11)	T(1, 21, 2)	(15, 25)	T(11, 23, 22)	(25, 20)	U(13, 18)	(15, 24)	U(5, 26)	(4, 10)	U(16, 28)
(6, 1)	U(20, 22)	(16, 20)	T(13, 18, 13)	(22, 5)	E(10)	(17, 13)	E(8)	(11, 4)	U(3, 18)
(7, 17)	T(8, 11, 10)	(17, 20)	T(6, 24, 23)	(12, 19)	T(11, 17, 16)	(6, 18)	N(3, 15)	(1, 24)	U(15, 18)
(8, 14)	N(29, 14)	(18, 6)	E(22)	(13, 22)	U(14, 23)	(18, 19)	U(20, 28)	(21, 3)	T(3, 26, 22)
(9, 25)	U(21, 27)	(19, 8)	E(24)	(22, 19)	N(23, 6)	(18, 12)	U(11, 20)	(17, 10)	U(17, 26)
(10, 23)	N(5, 23)	(20, 6)	U(0, 2)	(15, 25)	E(12)	(13, 10)	T(10, 29, 24)	(4, 22)	N(22, 11)

For the proposed MOPSO, Sigma-value as discussed in section 2.4.1 was used for leader selection, with ϵ-dominance [158] used for the maintenance of the external archive. Mutation

was used as a swarm diversity measure. The algorithm was implemented in Java and executed on a computer with Intel® Core™i7 CPU at 3.10GHz with 8.00GB RAM running a Windows 8.1 Operating System. Dijkstra's algorithm was used to solve each instance for both single-objective cases (minimum expected cost and variance) to verify that the algorithm is able to achieve the two extremes of the Pareto front. Each instance was executed over 10 runs with number of iterations set at 50 for each run to investigate consistency, a mutation rate of 0.3, and an ϵ-value of 0.01. The results of the algorithm on the three instances are presented in Table 6.7. From the results, it is obvious that the aim of multi-objective optimization is achieved, since the user is given the freedom to make a choice on the shortest path to take based on his preference either for the shortest path which has more variability, or the surest path which has higher cost, or, as the results on instances 2 and 3 show, a path which gives a suitable balance between the two desirables. The proposed algorithm is shown to achieve both extremes of the two objective functions on the three tested instances as verified by Dijkstra's algorithm. The algorithm's consistency is also shown by its ability to achieve the Pareto front 100% of the time on all tested instances. Figure 6.2 and Figure 6.3 show the Pareto fronts achieved in instances 2 and 3.

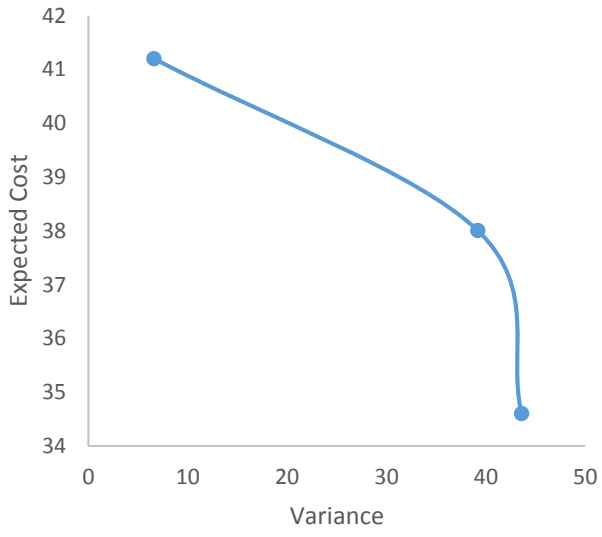


Figure 6.2: Pareto front for graph instance 2

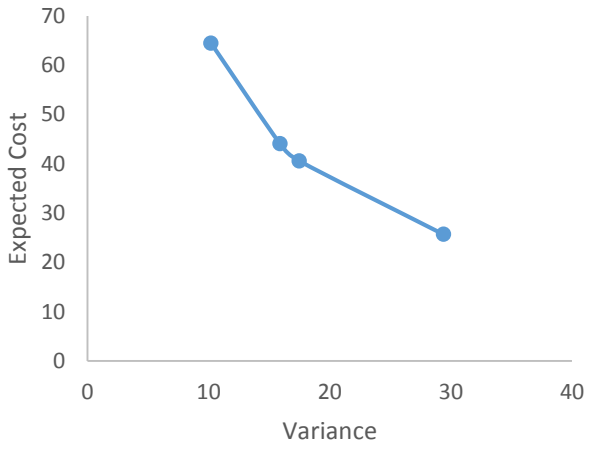


Figure 6.3: Pareto front for graph instance 3

Further experiments were carried out on the impact of ϵ-value used for ϵ-dominance on the algorithm's performance by varying its value from 0.01 to 0.1. The experiments were carried out on a randomly generated graph with 200 nodes and 1000 edges, with the same settings as earlier stated except for the varying ϵ-values. The minimum values found by Dijkstra's

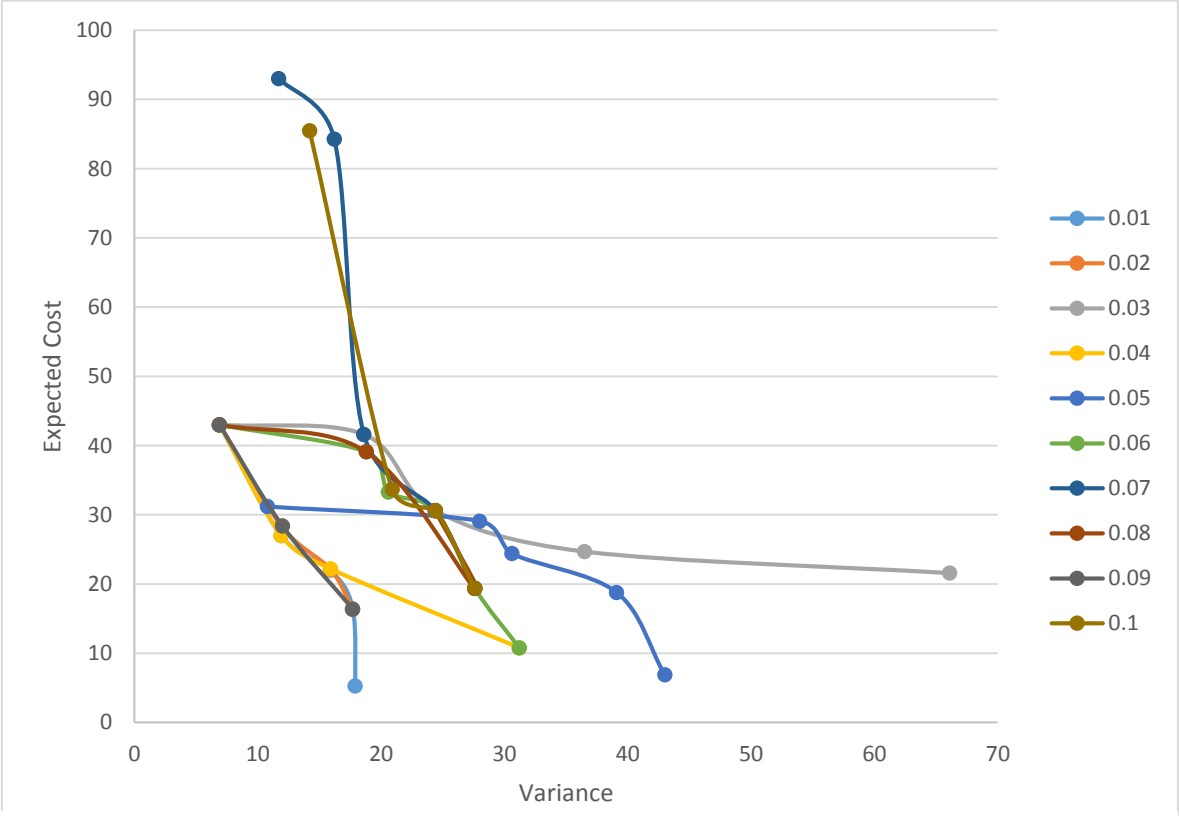


Figure 6.4: Evaluating Proposed DPSO's performance with varying ϵ-values

algorithm for expected length and variance were 5.3 and 4.9 respectively. The algorithm was run 10 times, and results which showed the greatest spread/diversity and accuracy were selected for comparison.

We quickly note that solutions provided for a discrete problem may not be as rich or diverse as those which are achieved for a continuous problem, the obvious reason being the limitation in solution space available for discrete optimization problems. The chart in Figure 6.4 shows the impact that ϵ-value can have on the performance of a MOPSO algorithm, indicating that lower ϵ-values are generally preferable for consistency, accuracy and greater diversity. For most values of ϵ, the algorithm was able to achieve a variance of 6.9, very close to the minimum variance. However, only when ϵ was set to 0.01 was it able to simultaneously find a minimum expected cost of 5.3. For values ranging from between 0.03 and 0.08, performance is generally poorer and farther away from the pareto front which was achieved with $\epsilon = 0.01$. An outlying case occurred with a value of 0.09, where the solution was very close to the pareto front, but the poorer performance trend resumed with a value of 0.1. This trend can be explained given the fact that with larger ϵ-values, the area dominated by a given solution increases, thereby hiding other good solutions which would have been found out. Therefore, for optimal performance, the ϵ-value should be kept reasonably low, preferably between 0.01 and 0.02. This is in agreement with the values used in [158].

6.4 Summary and Conclusion

In this chapter, the proposed DPSO for SPP has been extended to MOSSP, with the introduction of various novel operators and improvements to the generally known decoding process for path construction. Results of the algorithm on expected cost SSP (SSP-E) were compared with those proposed in [157], reporting better performance majorly in terms of consistency on all tested instances. The proposed algorithm was also applied to three randomly generated SSP graph instances with 25 nodes and 50 edges having normal, uniform, exponential and triangular distributions, and the Pareto front accurately approximated, with results by Dijkstra's algorithm solutions to both single-objective versions of the problem. The proposed algorithm was then applied to a problem with 200 nodes and 1000 edges, also successfully approximating the Pareto front. The proposed algorithm's performance with ϵ-

values ranging from 0.01 to 0.1 was evaluated, and a conclusion that ϵ-values between 0.01 and 0.02 tend to give better results than other values reached in line with existing literature.

Table 6.7: Proposed DPSO's performance on 3 graph instances

Problem instance	Path	Expected Cost (DPSO)	Minimum Expected Cost (Dijkstra)	Variance (DPSO)	Minimum Variance (Dijkstra)	Average CPU Time (s)	Convergence Rate (%)
1 (Source: 1, Sink: 20)	1-12-7-3-11-22-4-20	58.6	58.6	34.6	24.2	0.0219	100
	1-12-3-11-22-4-20	74.6		24.2			
2 (Source: 1, Sink: 25)	1-23-14-2-24-25	41.2	34.6	6.6	6.6	0.0109	100
	1-15-11-3-4-19-25	38.0		39.3			
	1-15-11-3-19-25	34.6		43.6			
3 (Source: 1, Sink: 25)	1-24-12-19-8-15-25	64.5	25.7	10.2	10.2	0.0142	100
	1-7-17-15-25	25.7		29.4			
	1-24-12-15-25	44.1		15.9			
	1-7-17-13-8-15-25	40.7		17.5			

Chapter Seven

A Multi-Objective Particle Swarm Optimization Approach to the Submission Decision Process

7.0 Introduction

The SDP was very recently introduced by Salinas and Munch [196] as one of choosing a permutation of journals to which to submit a manuscript (factoring in possibility of rejection and subsequent resubmission) such that the number of citations accumulated over a given time period is maximized, while the number of submissions and time taken from submission to publication are minimized. However, it was observed that the algorithm selected for their computation (Metropolis algorithm) entails an unnecessarily high computational burden and execution time. The proposed DPSO for SPP is therefore extended into a discrete Multi-Objective Particle Swarm Optimization heuristic (MOPSO) so as to address the highlighted issue. Some insight is also provided into the proposed models which can be used to achieve even higher speed increase.

7.1 Application of Proposed DPSO to SDP

As a result of the unique nature of the SDP, a few definition need to be made differently from those made in the proposed DPSO for SPP:

- **Position:** A position must be represented in such a way that it either directly or indirectly represents a feasible solution within the solution space. Position is therefore represented with respect to SDP as a set containing a permutation of journals (represented as numbers/indices) which represents a particular order of manuscript submission. Initially, the set of candidate journals is sorted by name, and indices are assigned to them. These are the indices used to refer to each journal. Thus, a possible position, with number of journals $N = 10$, would be

  $$(2, 4, 7, 5, 1, 9, 3, 8, 0, 6)$$

  Here, 2 could represent a journal named *Ecology*, 6 a journal named *Science*, and so on.

- **Velocity:** The same definition for velocity as was earlier proposed is maintained. An example velocity, specific to the SDP could be $\{(7, 2), (5, 1), (1, 6), (4, 15), (2, 4)\}_3$. That is, for example, the journal at index 7 of a target position will be replaced with another journal represented as 2, and so on.

Furthermore, a redefinition is necessary for the position update operation:

Position Update ($\otimes$): As per equation (4.8), position update is done by first creating a new position and inserting *k* elements in their respective indices in it. Subsequently, any empty indices are filled with values with the original positions, and if any gaps still remain, they are filled with randomly generated values. For example:

If $X_i^t = (2, 5, 1, 3, 4)$ and $V_i^{t+1} = \{(3, 2), (1, 5)\}_2$

To obtain X_i^{t+1}, first build a new position:

$X_i^{t+1} = (5, \ , 2, \ , \ )$

Then, insert all values from the previous position which will not result in duplications within the new position:

$X_i^{t+1} = (5, \ , 2, 3, 4)$

Finally, randomly generate/fill the remaining empty indices:

$X_i^{t+1} = (5, 1, 2, 3, 4)$

The updated equations still remain the same as equations (4.7) and (4.8).

7.2 Results

In the introductory paper by Salinas and Munch [196], a Metropolis algorithm was used in solving the two bi-objective cases of the SDP, obtaining the *C*, *R*, and *P* values for about 3.2 million different submission schedules. They also used multiple starting conditions, going through all $N(N-1)$ possible permutations of the first two journals in a submission schedule. Upon implementing their algorithm, a few areas in which improvements can be made were discovered. The generation of 3.2 million different submission schedules obviously is very time-consuming, also resulting in data which is overly large and cumbersome to process. The proposed discrete MOPSO algorithm was therefore applied to the problem, in which case the reported Pareto front (final contents of the external archive) suffices in helping authors make

their choice. For the experiments, the Sigma value was used for selection of leaders and a mutation rate of 0.5 for diversity maintenance. Also, to be able to properly cover the search space, the first two journals in a sequence were randomly swapped with any other two at every tenth iteration.

All implementations were done in Java and executed on a computer with Intel® Core™i7 CPU at 3.10GHz with 8.00GB RAM running a Windows 8.1 Operating System, and using one core. The same settings applied in [196] (*scoop rate = 0.001*, t_R *= 30*, and *T = 5 years (1826 days)*) were used in implementing both the Metropolis algorithm and the proposed discrete MOPSO algorithm for the bi-objective problem seeking to maximize number of citations, *C*, and minimize number of submissions, *R*. The charts in Figure 7.1 & Figure 7.2 show the results of the Metropolis algorithm and the proposed discrete MOPSO algorithm, respectively.

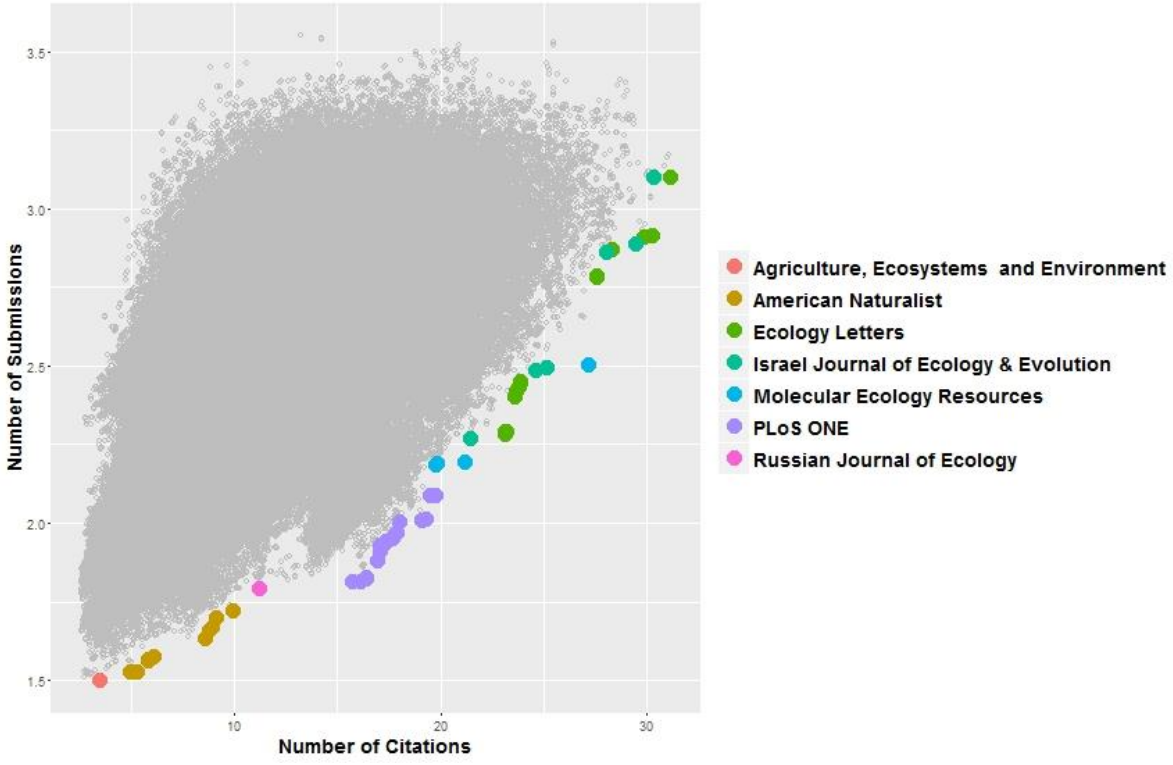


Figure 7.1: A plot of expected Number of Citations against expected Number of Submissions for about 3,150,000 submission schedules using the Metropolis algorithm in [196]

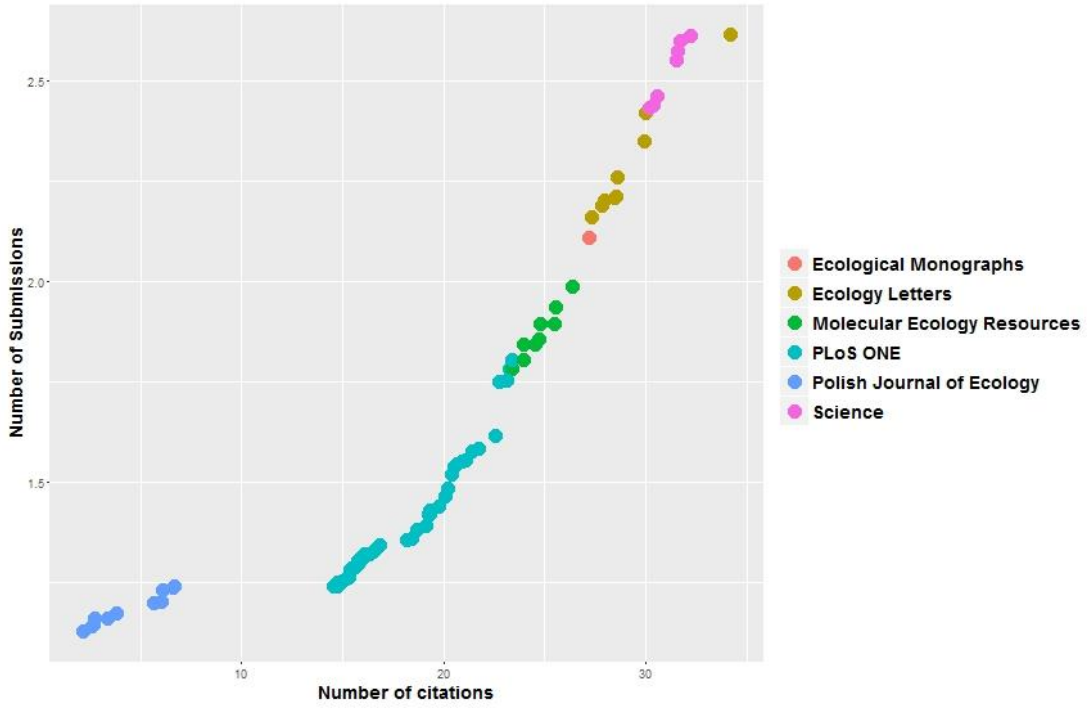


Figure 7.2: A plot of expected Number of Submissions against expected Number of Citations using the proposed MOPSO algorithm

Similarly, the results of optimizing number of citations, *C*, and time spent in review, *P* for both algorithms are shown in Figure 7.3 & Figure 7.4 respectively:

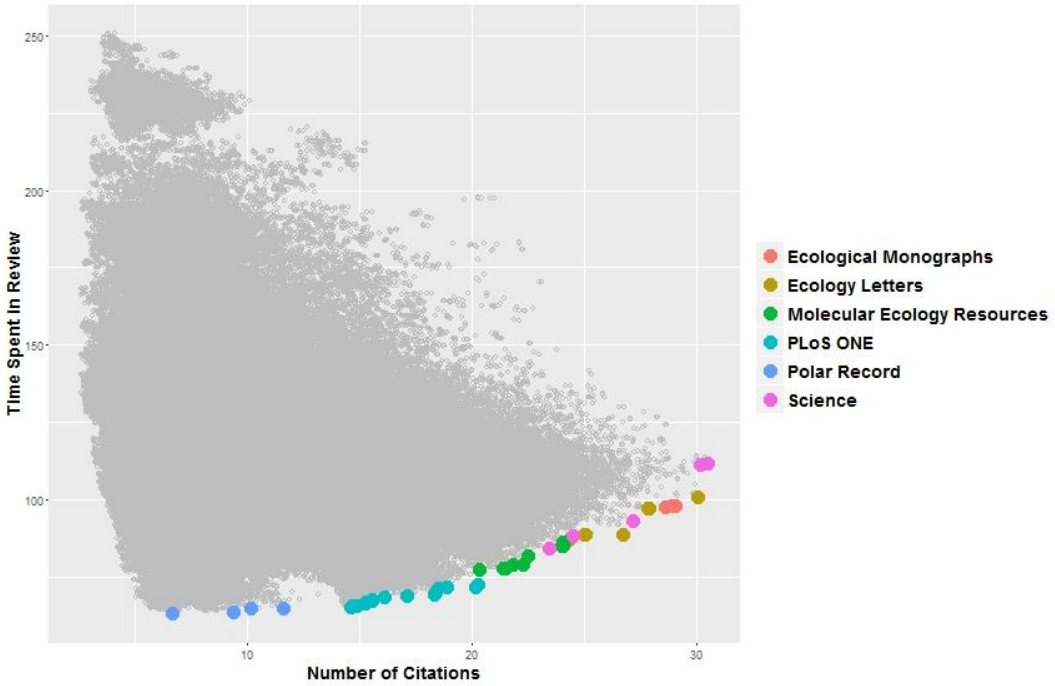


Figure 7.3: A plot of expected Number of Citations against expected Time Spent in Review for about 3,150,000 submission schedules using the Metropolis algorithm in [196]

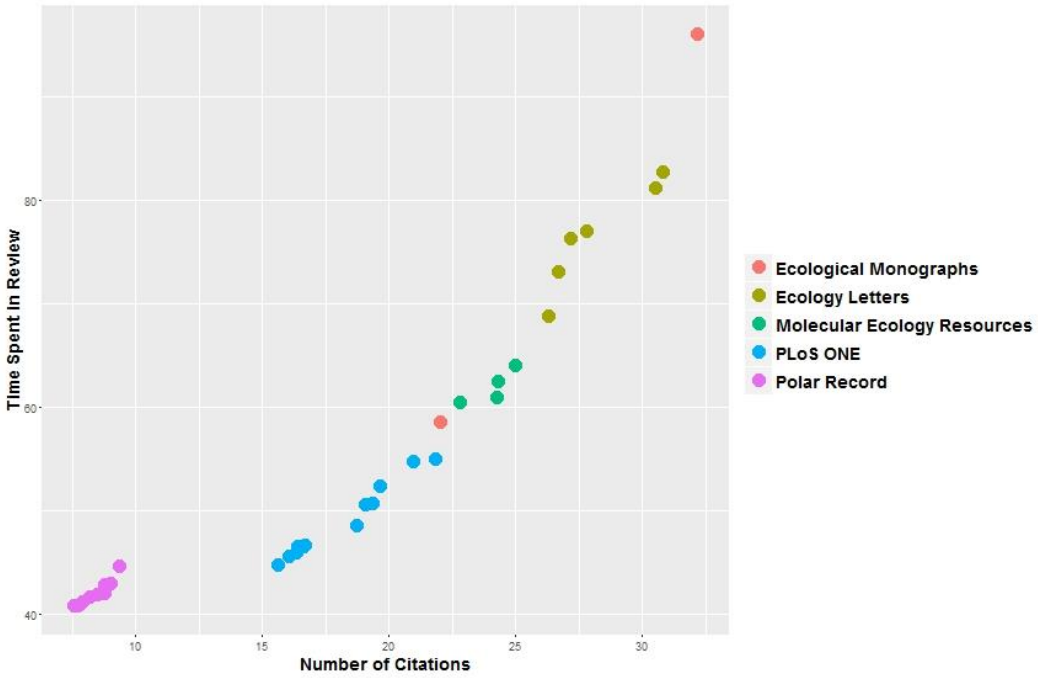


Figure 7.4: A plot of expected Number of Submissions against expected Time Spent in Review using the proposed MOPSO algorithm

Table 7.1 shows the time spent by both algorithms on the two different optimization scenarios.

Table 7.1: Computational time comparison of Metropolis and MOPSO

Objectives	**Algorithm**	**Time (s)**
Maximize *C*, Minimize *R*	Metropolis	1001
	MOPSO	11.957
Maximize *C*, Minimize *P*	Metropolis	966
	MOPSO	12.134

During computations and experiments, it was observed that the number of citations gained increased at a decreasing rate as j increased from 1 to N. Also, as noted in [196], a large N does not seem realistic, as most authors would not have the patience to continue down the submission schedule till the very last journal. Experiments were therefore performed to determine at what point the increase in number of citations is reasonably negligible, and thus to what value N can be reasonably reduced with minimal impact on number of citations gained. A graph of the growth trend for number of citations for 5 randomly generated submission schedules, S1 to S5 is shown in Figure 7.5.

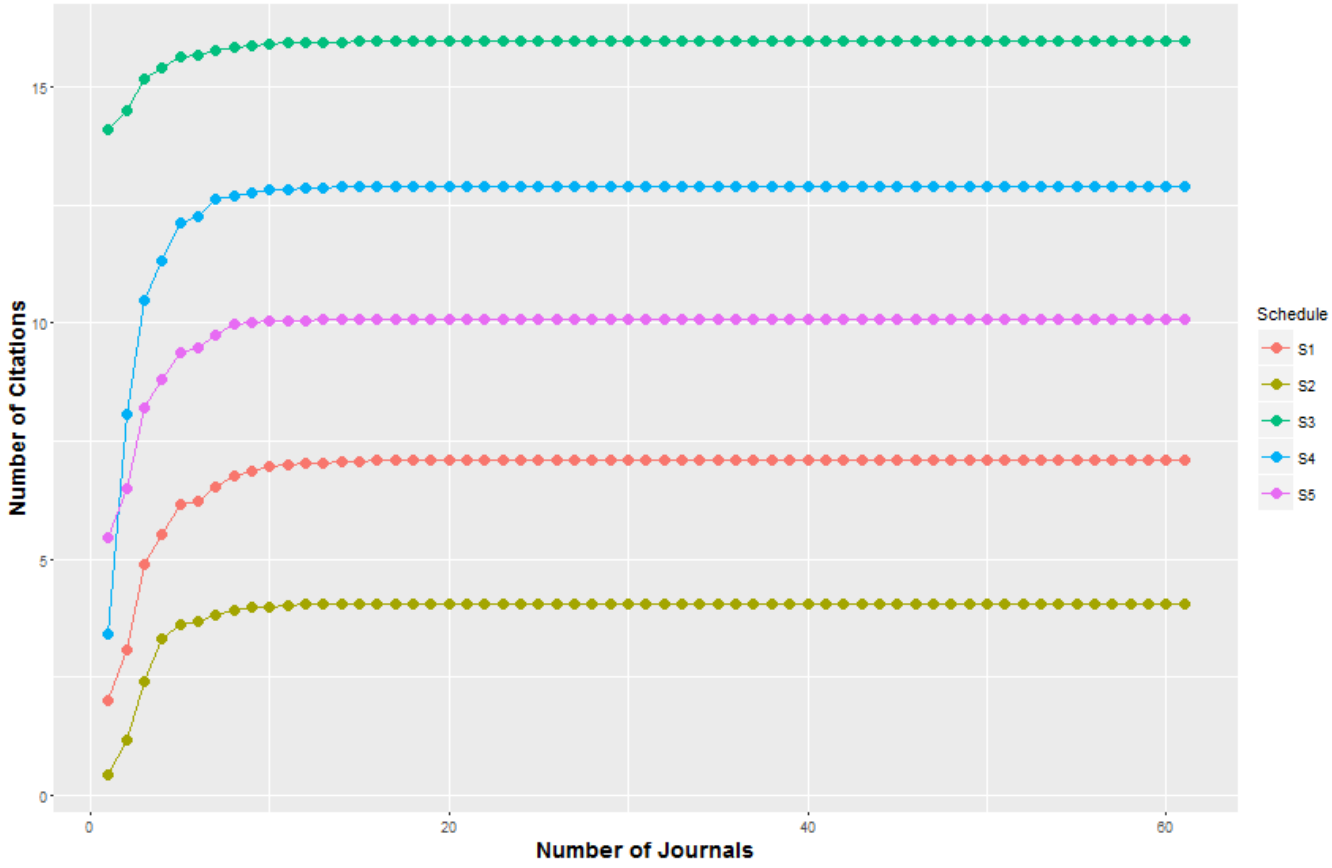


Figure 7.5: Growth trend of citation count over increasing value of N *(number of journals) for 5 randomly generated submission schedules*

As can be observed from Figure 7.5, beyond the point where *N = 10*, the increase in number of citations gained is very little. In line with this observation, the value of *N* was reduced to 10 and the algorithm rerun to see what time savings could be achieved. The result of optimizing *C* and *R* with *N=10* is shown in Figure 7.6, and *C* and *P* in Figure 7.7. A comparison of time spent for all three algorithms: Metropolis, MOPSO, and MOPSO with *N = 10*, denoted as $MOPSO_{10}$, is shown in Table 7.2.

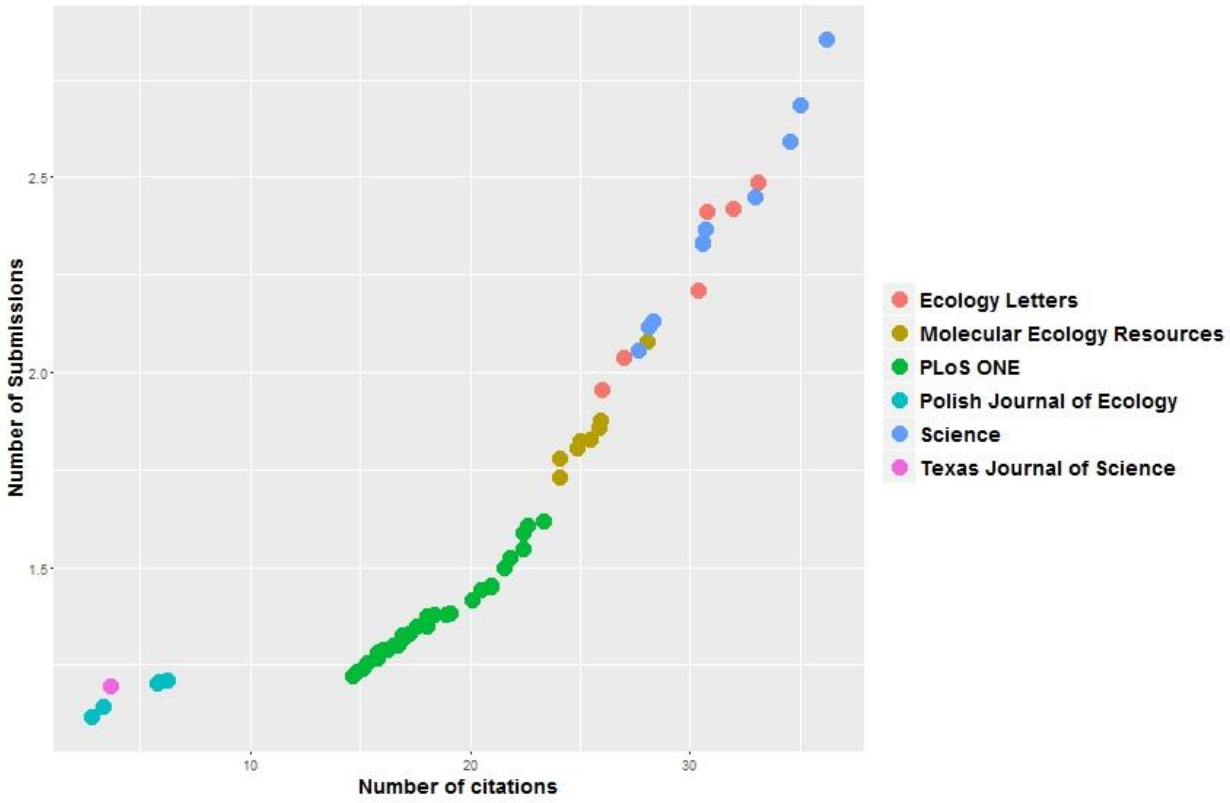


Figure 7.6: A plot of expected Number of Submissions against expected Number of Submissions using the proposed MOPSO algorithm with N = 10

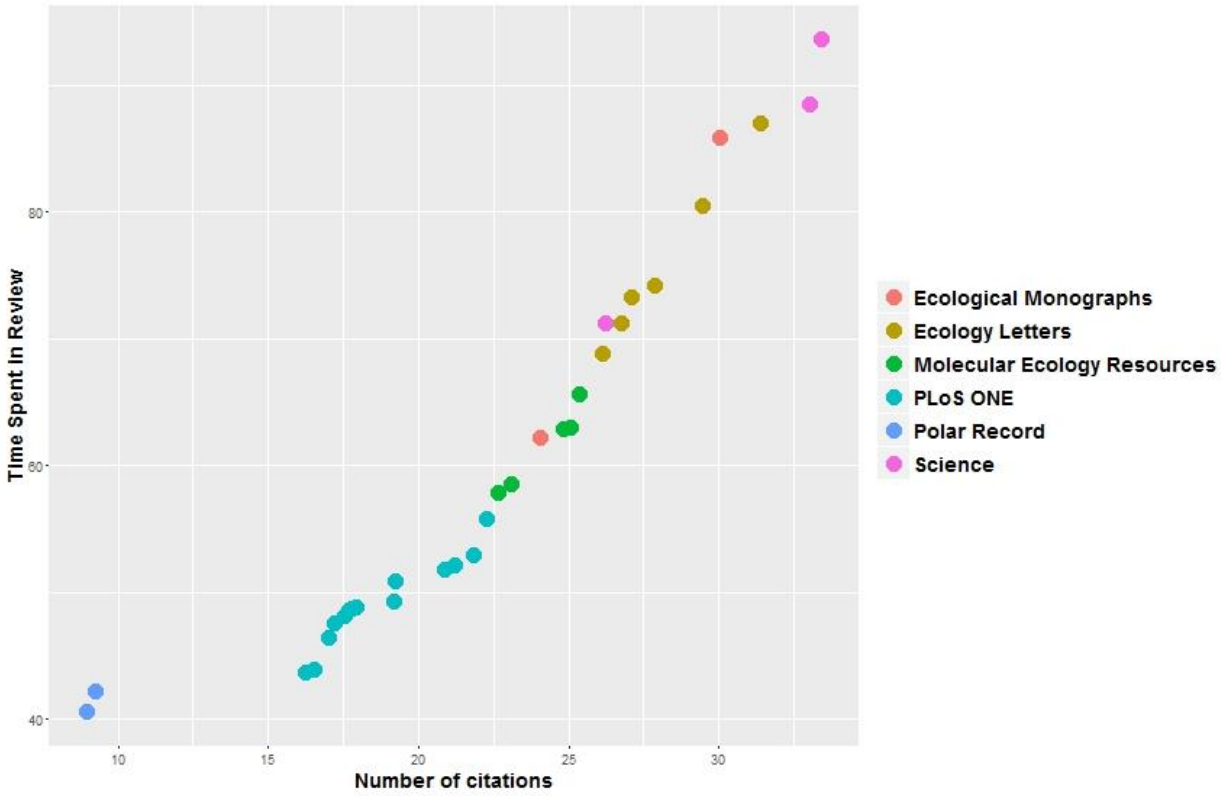


Figure 7.7: A plot of expected Number of Submissions against expected Time Spent in Review using the proposed MOPSO algorithm with N = 10

Table 7.2: Computational time comparison of Metropolis, MOPSO and $MOPSO_{10}$

Objectives	**Algorithm**	**Time (s)**
Maximize *C*, Minimize *R*	Metropolis	1001
	MOPSO	11.957
	$MOPSO_{10}$	0.563
Maximize *C*, Minimize *P*	Metropolis	966
	MOPSO	12.134
	$MOPSO_{10}$	0.539
Maximize *C*, Minimize *P*, Minimize *R*	Metropolis	-
	MOPSO	12.068
	$MOSPO_{10}$	0.63

Finally, the result of combining all objectives into a 3-objective optimization problem and solving with the proposed discrete MOPSO algorithm was investigated. The Pareto front for this 3-objective optimization is presented in Figure 7.8. In Figure 7.9, the same 3-objective optimization is presented, but with *N=10*.

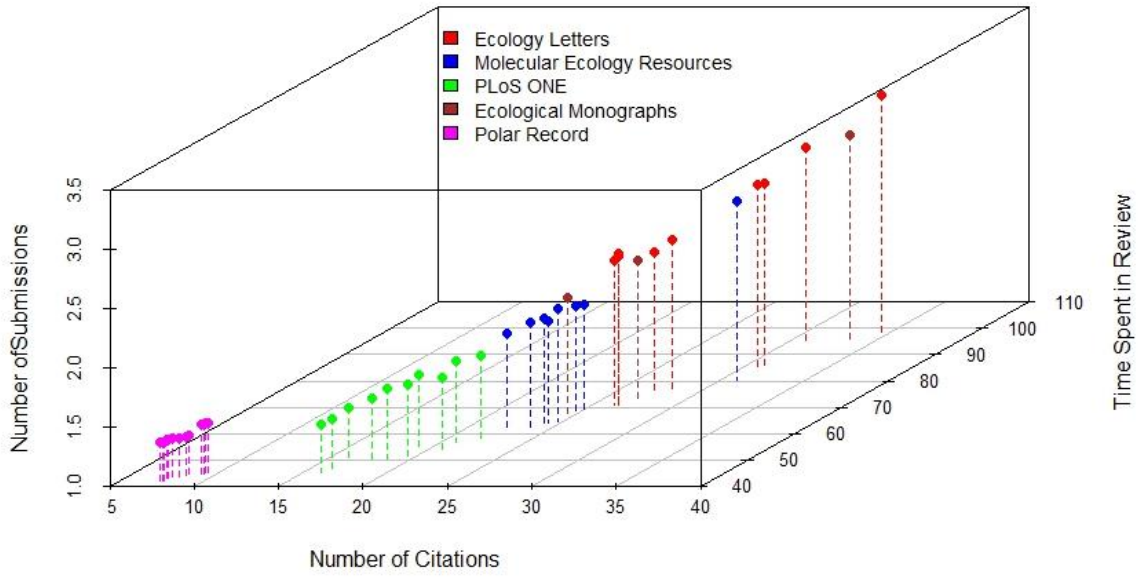


Figure 7.8: 3-Objective optimization plot of expected Number of Submissions, Number of Citations and Time Spent in Review using MOPSO

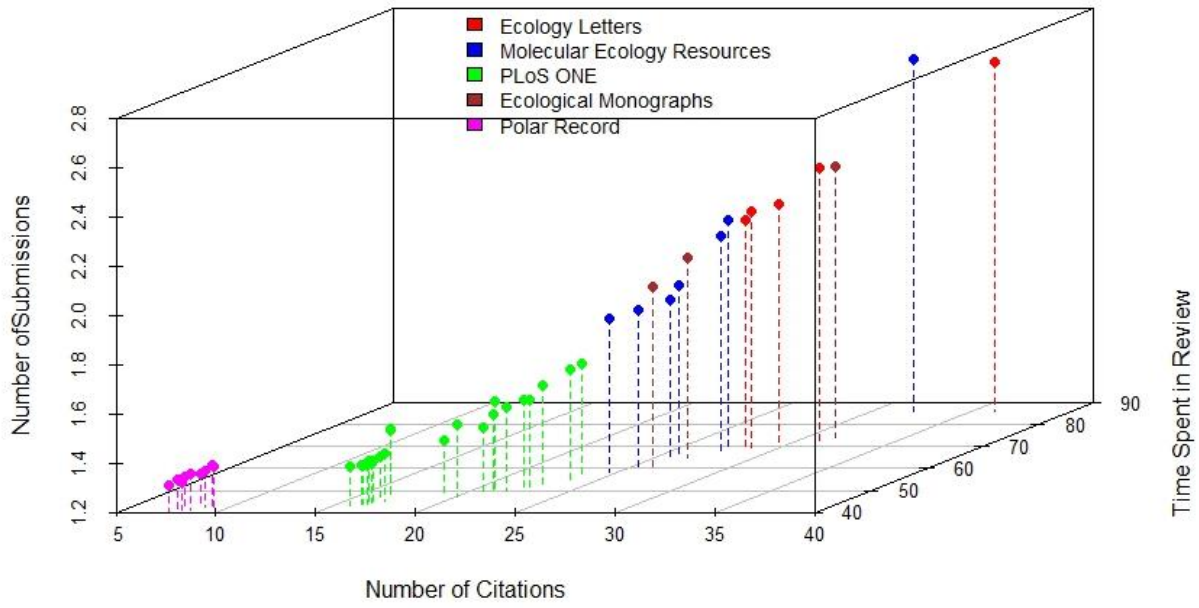


Figure 7.9: 3-Objective optimization plot of expected Number of Submissions, Number of Citations and Time Spent in Review using MOPSO with N = 10

7.3 Discussion

In Figure 7.1, which highlights the Pareto front achieved by the Metropolis algorithm, the top-right portion of the Pareto front is dominated by *Ecology Letters* and *Israel Journal of Ecology and Evolution.* Figure 7.2 which shows the Pareto front achieved by the proposed MOPSO algorithm has the top-right portion dominated by *Ecology Letters* and *Science*. Both algorithms therefore show that a first submission to *Ecology Letters* has a much higher chance of maximizing citation count. Furthermore, the middle-right portions of the Pareto fronts identified by both algorithms are clearly dominated by *PLoS ONE*, indicating that this journal gives the best compromise between both desired objectives. The results similarly reported by both algorithms in Figure 7.3 and Figure 7.4 are also in favor of *Ecology Letters* for maximizing citation count, and *PLoS ONE* for achieving a good compromise between maximizing citation count and minimizing time spent in review. It is also interesting to note that in Figure 7.6 and Figure 7.7 where $N = 10$, the Pareto fronts reported agree with those achieved when $N = 61$.

When all objectives are optimized as a 3-objective optimization problem, *Ecology Letters* stands out clearly as the one which maximizes citation count (though with higher number of submissions and longer time spent in review), while *PLoS ONE* gives a reasonable compromise among all three objectives. This behavior is consistent both when *N=61* and *N=10*.

Therefore, the proposed algorithm, when compared to the Metropolis algorithm in [196], shows similar results: that a schedule with a preference to maximizing number of citations gained during the time horizon over minimizing the number of submissions should begin submission with *Ecology Letters*. Experiments also show that a first submission to *Science* is a good strategy. Other good first journals for maximizing citation count are *Ecological Monographs* and *Molecular Ecology Resources*, as can be seen from the charts. An optimal submission schedule which gives a reasonable compromise between the two objectives should clearly begin submission with *PLoS ONE*. That is, if an author would not mind losing a few citations (5 to 10), his best option would be to first submit his manuscript to *PLoS ONE*. An interesting observation which was also made from experiments is that almost in all cases, for schedules which achieve higher citation counts by first submitting to either *Ecology* or *Science*, the second journal in the submission schedule is *PLoS ONE*. This implies that if upon submission to either of the two journals, one is rejected, the best option would be to submit to *PLoS ONE*. It is also worthy of note that most, if not all of the journals which form the upper to middle (desirable) part of the Pareto front are among the first ten journals when all journals are ranked by their *V* values (equation (2.22)). These journals in descending order of *V* values are: *Ecology Letters, Science, Israel Journal of Ecology and Evolution, Ecological Monographs, Proceedings of National Academy of Sciences USA, Molecular Ecology Resources, Molecular Ecology, ISME Journal, Journal of Evolutionary Biology, and PLoS ONE*. The same results with minute variations apply when optimizing expected number of citations and time spent in review. Most of these observations are similarly made by Salinas and Munch [196].

It has therefore been shown that the SDP problem can be solved in a much smaller amount of time with the proposed MOPSO, and even smaller time when *N* is reduced to 10, without affecting the quality of results obtained.

7.4 Summary and Conclusion

In this chapter, the proposed discrete MOPSO algorithm was applied to the SDP with results showing that it performs much faster than the Metropolis algorithm used in the introducing paper [196] without compromising accuracy. It was also shown that reducing the value of N (Number of journals) to 10 when computing the different objective values C, P, and R leads to a greater reduction in computational time. Furthermore, the proposed algorithm was used to solve a 3-objective optimization problem combining the three objectives C, P and R, as against the existing optimization of only 2 objectives (either C and R or C and P), and the results were reported.

Chapter Eight

Summary, Conclusion and Future Work

8.0 Summary

In this research work, COPs have been discussed with focus on the TSP, SPP and SDP. Exact and heuristic solution approaches to these problems have also been discussed and reviewed. Exact methods discussed include label-setting, label-correcting and branch-and-bound algorithms. The non-NI heuristics which were discussed include A*, heuristic bi-directional search, branch and prune methods for the SPP and nearest-neighbor, *k-opt* and the widely-acclaimed LKH heuristic for the TSP. Nature-Inspired algorithms such as GA, ACO, PSO and the Physarum algorithms were also discussed. With specific regard to PSO, various discretization techniques existing in literature were highlighted such as binary, fuzzy, swap operator-based, set-based, and space transformation-based methods. The SDP was also discussed in the context of pertinent issues related to it in literature. We also briefly presented the Metropolis algorithm which was implemented in literature to solve it.

Two new set-based DPSOs were presented, EC-DPSO and a DPSO for SPP, which involve the use of innovative problem-specific information to speed up convergence and achieve accurate results. The EC-DPSO algorithm was applied to the TSP, while the DPSO proposed for the SPP was applied to the classical SPP and SSP, as well as the MOSSP and SDP. The results achieved are presented in the subsequent sections, and future areas of potential research highlighted. This is done with respect to each of the problems solved.

8.1 Conclusion

A Discrete Particle Swarm Optimization (DPSO) heuristic for the Travelling Salesman Problem (TSP) which was optimized through enhanced communication between particles was presented. A great improvement in results was achieved by attempting to direct the particles towards positions formed by ‘good’ edges. By taking into account the impact which the inclusion of an edge makes on the fitness value achieved in the new position, the chances

of edges with relatively high cost getting included was greatly reduced, and that of those with relatively very low cost increased

A memetic DPSO algorithm for the Shortest Path Problem (SPP) was also presented, introducing new discretization concepts which proved to have a great impact on the global search ability of the DPSO algorithm. Optimal values were also selected for *c1* and *c2* with improved results. The local search ability of the memetic component of the algorithm led to still better results. These results were compared with those of 3 other algorithms proposed in literature for the SPP and found to be better. More so, the size of SPP instances solved by DPSOs found in literature was increased from 100 to 200 cities.

Furthermore, the proposed DPSO algorithm was adapted to the Multi-Objective Stochastic Shortest Path problem (MOSSP), introducing various novel operators and improvements to the generally known decoding process for path construction. Results obtained on expected cost SSP (SSP-E) were compared with those proposed in [157], reporting better performance majorly in terms of consistency on all tested instances.

The proposed algorithm was then applied to MOSSP. Three random SSP graph instances of size 25 nodes and 50 edges were generated, having normal, uniform, exponential and triangular distributions. The Pareto front was accurately approximated, and results validated by Dijkstra's solutions for both single-objective versions of the problem. The algorithm was also applied to a MOSSP instance with 200 nodes and 1000 edges, with the Pareto front also successfully approximated. The performance of the algorithm with ϵ-values ranging from 0.01 to 0.1 were evaluated and reported, and a conclusion that ϵ-values between 0.01 and 0.02 tend to give better results reached in consonance with existing literature.

The Submission Decision Problem was addressed with an adaptation of the proposed discrete MOPSO algorithm. It was shown that the algorithm performs much faster than the Metropolis algorithm used in the paper [196] which introduced the problem. It was also shown that reducing the value of *N* (Number of journals) to 10 in computing the different objective values, expected citation count (*C),* expected time spent in review *(P),* and expected number of submissions (*R)* leads to a greater reduction in computational time without compromising

results. Furthermore, the proposed algorithm was used to solve a 3-objective optimization problem combining the three objectives *C, P* and *R*, as against the existing optimization of only 2 objectives (either *C* and *R* or *C* and *P*).

8.2 Future Work

The DPSO for TSP algorithm which was presented represents a new idea in discretizing PSO for TSP. However, the performance of the algorithm still needs to be improved in terms of speed and accuracy, as we were only able to attempt TSPs of size up to 76 cities. In addition, it was observed in the review of literature that no Particle Swarm Optimization (PSO) algorithm has been applied to the family of constrained Shortest Path problems (CSP). It is therefore recommended that the new DPSO algorithm for SPP be adapted to solve the CSP as a means of testing its robustness and investigating if any improvements can be made on existing results.

The application of discrete MOPSOs to Stochastic Time-Dependent Networks with variants including correlations between link costs, reliability and α-reliability and others will be looked into, as this appears to be an area of active research. Finally, attempts will be made to obtain data specific to the computer science field so as to use them in solving the SDP and reported the results. These results will be of great use to authors in the field of computer science in deciding what journal submission schedule to follow in submitting their manuscripts.

References

[1] A. Abraham, H. Liu, W. Zhang, and T.-G. Chang, "Scheduling Jobs on Computational Grids Using Fuzzy Particle Swarm Algorithm," in *Knowledge-Based Intelligent Information and Engineering Systems*. vol. 4252, B. Gabrys, R. Howlett, and L. Jain, Eds., ed: Springer Berlin Heidelberg, 2006, pp. 500-507.

[2] M. O. Adamu and A. O. Adewumi, "A survey of single machine scheduling to minimize weighted number of tardy jobs," *Journal of Industrial and Management Optimization,* vol. 10, pp. 219-241, 2014.

[3] A. O. Adewumi and A. M. Arasomwan, "An improved particle swarm optimiser based on swarm success rate for global optimisation problems," *Journal of Experimental & Theoretical Artificial Intelligence,* pp. 1-43, 2014.

[4] A. O. Adewumi and A. M. Arasomwan, "Improved particle swarm optimization based on greedy and adaptive features," in *Proceedings of IEEE Symposium on Swarm Intelligence (SIS)*, 2014, pp. 1-6.

[5] A. O. Adewumi, B. A. Sawyerr, and M. Montaz Ali, "A heuristic solution to the university timetabling problem," *Engineering Computations,* vol. 26, pp. 972-984, 2009.

[6] H. Afaq and S. Saini, "A Novel Approach to Solve Graph-based Travelling Salesman Problem Using Particle Swarm Optimization Technique," in *Proceedings of International Conference on Computational Intelligence and Computing Research*, Coimbatore, 2012.

[7] F. Afshinmanesh, A. Marandi, and A. Rahimi-Kian, "A novel binary particle swarm optimization method using artificial immune system," in *Proceedings of the International Conference on Computer as a Tool (EUROCON 2005).* 2005, pp. 217-220.

[8] R. Agarwal, M. Caesar, P. B. Godfrey, and B. Y. Zhao, "Shortest paths in less than a millisecond," in *Proceedings of the 2012 ACM workshop on Workshop on online social networks*, 2012, pp. 37-42.

[9] C. W. Ahn and R. S. Ramakrishna, "A genetic algorithm for shortest path routing problem and the sizing of populations," *IEEE Transactions on Evolutionary Computation,* vol. 6, pp. 566-579, 2002.

[10] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms and Applications*. New Jersey: Prentice-Hall, Inc., 1993.

[11] D. Applegate and W. Cook, "A computational study of the job-shop scheduling problem," *ORSA Journal on computing,* vol. 3, pp. 149-156, 1991.

[12] D. L. Applegate, R. E. Bixby, V. Chvátal, W. Cook, D. G. Espinoza, M. Goycoolea, *et al.*, "Certification of an optimal TSP tour through 85,900 cities," *Operations Research Letters,* vol. 37, pp. 11-15, 2009.

[13] D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook, *The Traveling Salesman Problem: A Computational Study*: Princeton University Press, 2011.

[14] M. A. Arasomwan and A. O. Adewumi, "On adaptive chaotic inertia weights in Particle Swarm Optimization," in *IEEE Symposium on Swarm Intelligence (SIS)*, 2013, pp. 72-79.

[15] M. A. Arasomwan and A. O. Adewumi, "Improved particle swarm optimization with a collective local Unimodal search for continuous optimization problems," *The Scientific World Journal*, vol. 2014, 2014.
[16] É. Archambault and V. Larivière, "History of the journal impact factor: Contingencies and consequences," *Scientometrics*, vol. 79, pp. 635-649, 2009.
[17] N. Attiratanasunthron and J. Fakcharoenphol, "A running time analysis of an ant colony optimization algorithm for shortest paths in directed acyclic graphs," *Information Processing Letters*, vol. 105, pp. 88-92, 2008.
[18] J. L. Bander and C. C. White, III, "A new route optimization algorithm for rapid decision support," in *Vehicle Navigation and Information Systems Conference, 1991*, 1991, pp. 709-728.
[19] A. G. Barto, S. J. Bradtke, and S. P. Singh, "Learning to act using real-time dynamic programming," *Artificial Intelligence*, vol. 72, pp. 81-138, 1995.
[20] J. E. Beasley, "An algorithm for set covering problem," *European Journal of Operational Research*, vol. 31, pp. 85-93, 1987.
[21] I. Beichl and F. Sullivan, "The metropolis algorithm," *Computing in Science & Engineering*, vol. 2, pp. 65-69, 2000.
[22] H. Beigy and M. R. Meybodi, "A New Distributed Learning Automata Based Algorithm For Solving Stochastic Shortest Path Problem," in *JCIS*, 2002, pp. 339-343.
[23] R. Bellman, "Dynamic programming and Lagrange multipliers," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 42, p. 767, 1956.
[24] R. Bellman, "On a routing problem," DTIC Document1956.
[25] H. Bernhard, B. Korte, and J. Vygen, "Combinatorial optimization: Theory and algorithms," ed: Heidelberg: Springer-Verlag, 2008.
[26] D. P. Bertsekas and J. N. Tsitsiklis, "An analysis of stochastic shortest path problems," *Mathematics of Operations Research*, vol. 16, pp. 580-595, 1991.
[27] F. J. Beutler, "Dynamic Programming: Deterministic and Stochastic Models (Dimitri P. Bertsekas)," *SIAM Review*, vol. 31, pp. 132-133, 1989.
[28] M. Bolívar, L. Lozano, and A. Medaglia, "Acceleration strategies for the weight constrained shortest path problem with replenishment," *Optimization Letters*, vol. 8, pp. 2155-2172, 2014.
[29] J. Bollen, H. Van de Sompel, A. Hagberg, and R. Chute, "A principal component analysis of 39 scientific impact measures," *PloS one*, vol. 4, p. e6022, 2009.
[30] B. Bonet and H. Geffner, "Solving stochastic shortest-path problems with RTDP," *University of California, Los Angeles*, 2002.
[31] M. R. Bonyadi, M. R. Azghadi, and H. Shah-Hosseini, "Population-Based Optimization Algorithms for Solving the Travelling Salesman Problem," in *Travelling Salesman Problem*, F. Greco, Ed., ed Croatia: In-Tech, 2008, pp. 1-34.
[32] L. Bornmann, W. Marx, A. Y. Gasparyan, and G. D. Kitas, "Diversity, value and limitations of the journal impact factor and alternative metrics," *Rheumatology International*, vol. 32, pp. 1861-1867, 2012.
[33] W. M. Carlyle, J. O. Royset, and R. Kevin Wood, "Lagrangian relaxation and enumeration for solving constrained shortest-path problems," *Networks*, vol. 52, pp. 256-270, 2008.

[34] O. Castro, Jr. and A. Pozo, "Using Hyper-Heuristic to Select Leader and Archiving Methods for Many-Objective Problems," in *Evolutionary Multi-Criterion Optimization*. vol. 9018, A. Gaspar-Cunha, C. Henggeler Antunes, and C. C. Coello, Eds., ed: Springer International Publishing, 2015, pp. 109-123.

[35] Y. Y. Cha, "Navigation of a free-ranging mobile robot using heuristic local path-planning algorithm," *Robotics and Computer-Integrated Manufacturing*, vol. 13, pp. 145-156, 6// 1997.

[36] S. Chamaani, S. A. Mirtaheri, M. Teshnehlab, and M. A. Shooredeli, "Modified multi-objective particle swarm optimization for electromagnetic absorber design," in *Proceedings of Asia-Pacific Conference on Applied Electromagnetics (APACE)*, 2007, pp. 1-5.

[37] S. Chandrasekaran, S. Ponnambalam, R. K. Suresh, and N. Vijayakumar, "A hybrid discrete particle swarm optimization algorithm to solve flow shop scheduling problems," in *Proceedings of IEEE Conference on Cybernetics and Intelligent Systems*, 2006, pp. 1-6.

[38] W. A. Chang and R. S. Ramakrishna, "A genetic algorithm for shortest path routing problem and the sizing of populations," *IEEE Transactions on Evolutionary Computation*, vol. 6, pp. 566-579, 2002.

[39] A. Chen and Z. Ji, "Path finding under uncertainty," *Journal of advanced transportation*, vol. 39, pp. 19-37, 2005.

[40] B.-Y. Chen, W. H. K. Lam, L. Qingquan, A. Sumalee, and Y. Ke, "Shortest Path Finding Problem in Stochastic Time-Dependent Road Networks With Stochastic First-In-First-Out Property," *IEEE Transactions on Intelligent Transportation Systems*, vol. 14, pp. 1907-1917, 2013.

[41] B. Y. Chen, W. H. Lam, A. Sumalee, Q. Li, H. Shao, and Z. Fang, "Finding reliable shortest paths in road networks under uncertainty," *Networks and spatial economics*, vol. 13, pp. 123-148, 2013.

[42] B. Y. Chen, W. H. K. Lam, A. Sumalee, Q. Li, and M. L. Tam, "Reliable Shortest Path Problems in Stochastic Time-Dependent Networks," *Journal of Intelligent Transportation Systems*, vol. 18, pp. 177-189, 2013.

[43] P. Chen and Y. Nie, "Bicriterion shortest path problem with a general nonadditive cost," *Transportation Research Part B: Methodological*, vol. 57, pp. 419-435, 2013.

[44] W.-N. Chen, J. Zhang, H. S.-H. Chung, W.-L. Zhong, W.-G. Wu, and Y.-H. Shi, "A novel set-based particle swarm optimization method for discrete optimization problems," *IEEE Transactions on Evolutionary Computation*, vol. 14, pp. 278-300, 2010.

[45] Y.-L. Chen and H.-H. Yang, "Shortest paths in traffic-light networks," *Transportation Research Part B: Methodological*, vol. 34, pp. 241-253, 5// 2000.

[46] J. Cheng, S. Kosuch, and A. Lisser, "Stochastic Shortest Path Problem with Uncertain Delays," in *ICORES*, 2012, pp. 256-264.

[47] S. Chib and E. Greenberg, "Understanding the metropolis-hastings algorithm," *The american statistician*, vol. 49, pp. 327-335, 1995.

[48] H. M. Choset, *Principles of robot motion: theory, algorithms, and implementation*: MIT press, 2005.

[49] J. Clausen, "Branch and bound algorithms-principles and examples," *Department of Computer Science, University of Copenhagen*, pp. 1-30, 1999.
[50] M. Clerc, "Discrete Particle Swarm Optimization, illustrated by the Traveling Salesman Problem," in *New Optimization Techniques in Engineering*. vol. 141, ed: Springer Berlin Heidelberg, 2004, pp. 219-239.
[51] M. Clerc and J. Kennedy, "The particle swarm-explosion, stability, and convergence in a multidimensional complex space," *IEEE Transactions on Evolutionary Computation*, vol. 6, pp. 58-73, 2002.
[52] A. Colorni, M. Dorigo, and V. Maniezzo, "Distributed optimization by ant colonies," in *Proceedings of the first European conference on artificial life*, 1991, pp. 134-142.
[53] J. Cooper, "Improving performance of genetic algorithms by using novel fitness functions," Computer Science, Loughborough University, 2006.
[54] G. Dahl and L. Gouveia, "On the directed hop-constrained shortest path problem," *Operations Research Letters*, vol. 32, pp. 15-22, 1// 2004.
[55] M. Daneshyari and G. G. Yen, "Cultural-Based Multiobjective Particle Swarm Optimization," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 41, pp. 553-567, 2011.
[56] G. Dantzig, R. Fulkerson, and S. Johnson, "Solution of a large-scale traveling-salesman problem," *Journal of the operations research society of America*, vol. 2, pp. 393-410, 1954.
[57] G. B. Dantzig, "On the shortest route through a network," *Management Science*, vol. 6, pp. 187-190, 1960.
[58] H. W. Davis, R. B. Pollack, and T. Sudkamp, "Towards a Better Understanding of Bidirectional Search," in *AAAI*, 1984, pp. 68-72.
[59] R. Dawkins, *The Selfish Gene*, 3rd ed. New York: Oxford University Press, 2006.
[60] D. de Champeaux, "Bidirectional heuristic search again," *Journal of the ACM (JACM)*, vol. 30, pp. 22-32, 1983.
[61] K. Deb and D. E. Goldberg, "An investigation of niche and species formation in genetic function optimization," in *Proceedings of the 3rd international conference on genetic algorithms*, 1989, pp. 42-50.
[62] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, pp. 182-197, 2002.
[63] Y. Deng, R. Sadiq, W. Jiang, and S. Tesfamariam, "Risk analysis in a linguistic environment: A fuzzy evidential reasoning-based approach," *Expert Systems with Applications*, vol. 38, pp. 15438-15446, 2011.
[64] M. Desrochers and F. Soumis, "A generalized permanent labeling algorithm for the shortest path problem with time windows," *INFOR Information Systems and Operational Research*, 1988.
[65] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269-271, 1959/12/01 1959.
[66] B. Doerr, A. Hota, and T. Kötzing, "Ants easily solve stochastic shortest path problems," in *Proceedings of the 14th annual conference on Genetic and evolutionary computation*, 2012, pp. 17-24.

[67] S. E. Dreyfus, "An Appraisal of Some Shortest-Path Algorithms," *Operations Research*, vol. 17, pp. 395-412, 1969.
[68] I. Dumitrescu and N. Boland, "Improved preprocessing, labeling and scaling algorithms for the Weight-Constrained Shortest Path Problem," *Networks*, vol. 42, pp. 135-153, 2003.
[69] J. Eaton and L. Zadeh, "Optimal pursuit strategies in discrete-state probabilistic systems," *Journal of Fluids Engineering*, vol. 84, pp. 23-29, 1962.
[70] P. M. Editors, "The impact factor game," *PLoS Med*, vol. 3, p. e291, 2006.
[71] A. El-Gallad, M. El-Hawary, A. Sallam, and A. Kalas, "Enhancing the particle swarm optimizer via proper parameters selection," in *Proceedings of Canadian Conference on Electrical and Computer Engineering (IEEE CCECE 2002)*, 2002, pp. 792-797 vol.2.
[72] G. Eugene, "Citation indexes for science: A new dimension in documentation through association of ideas," *Science*, vol. 122, pp. 108-111, 1955.
[73] M. Eusuff, K. Lansey, and F. Pasha, "Shuffled frog-leaping algorithm: a memetic meta-heuristic for discrete optimization," *Engineering Optimization*, vol. 38, pp. 129-154, 2006/03/01 2006.
[74] L. Fang, P. Chen, and S. Liu, "Particle swarm optimization with simulated annealing for TSP," in *Proceedings of the 6th WSEAS International Conference on Artificial Intelligence, Knowledge Engineering and Data Bases (AIKED'07)*, 2007, pp. 206-210.
[75] M. Farhanchi, R. Hassanzadeh, I. Mahdavi, and N. Mahdavi-Amiri, "A modified ant colony system for finding the expected shortest path in networks with variable arc lengths and probabilistic nodes," *Applied Soft Computing*, vol. 21, pp. 491-500, 8// 2014.
[76] S. Fazel and J. Lamsma, "Beyond the impact factor?," *Evidence Based Mental Health*, pp. 33-35. doi:10.1136/eb-2015-102087, 2015.
[77] P. Festa, "Constrained shortest path problems: State-of-the-art and recent advances," in *Proceedings of the 17th International Conference on Transparent Optical Networks (ICTON)*, 2015, pp. 1-17.
[78] M. Fischetti, J. J. Salazar Gonzalez, and P. Toth, "A branch-and-cut algorithm for the symmetric generalized traveling salesman problem," *Operations Research*, vol. 45, pp. 378-394, 1997.
[79] I. Fister Jr, X.-S. Yang, I. Fister, J. Brest, and D. Fister, "A brief review of nature-inspired algorithms for optimization," *arXiv preprint arXiv:1307.4186*, 2013.
[80] M. M. Flood, "The traveling-salesman problem," *Operations Research*, vol. 4, pp. 61-75, 1956.
[81] H. Frank, "Shortest paths in probabilistic graphs," *Operations Research*, vol. 17, pp. 583-599, 1969.
[82] L. Fu, "Real-time vehicle routing and scheduling in dynamic and stochastic traffic networks," Department of Transportation Engineering, University of Alberta, 1996. Ph.D thesis.
[83] L. Fu, D. Sun, and L. R. Rilett, "Heuristic shortest path algorithms for transportation applications: State of the art," *Computers & Operations Research*, vol. 33, pp. 3324-3343, 2006.

[84] E. Garfield, "Journal impact factor: a brief review," *Canadian Medical Association Journal,* vol. 161, pp. 979-980, 1999.
[85] M. Gen, C. Runwei, and W. Dingwei, "Genetic algorithms for solving shortest path problems," in *Proceedings of IEEE International Conference on Evolutionary Computation,*1997, pp. 401-406.
[86] C. Geyer, "Introduction to Markov Chain Monte Carlo," *Handbook of Markov Chain Monte Carlo,* pp. 3-48, 2011.
[87] W. Glänzel and H. F. Moed, "Journal impact measures in bibliometric research," *Scientometrics,* vol. 53, pp. 171-193, 2002.
[88] R. L. Glass, "A structure-based critique of contemporary computing research," *Journal of Systems and Software,* vol. 28, pp. 3-7, 1995.
[89] B. Golden, "Technical Note—Shortest-Path Algorithms: A Comparison," *Operations Research,* vol. 24, pp. 1164-1168, 1976.
[90] M. Golosovsky and S. Solomon, "Uncovering the dynamics of citations of scientific papers," *arXiv preprint arXiv:1410.0343,* 2014.
[91] R. Gopalan, R. Batta, and M. H. karwan, "The equity constrained shortest path problem," *Computers & Operations Research,* vol. 17, pp. 297-307, 1990.
[92] J. Gottlieb, E. Marchiori, and C. Rossi, "Evolutionary algorithms for the satisfiability problem," *Evolutionary Computation,* vol. 10, pp. 35-50, 2002.
[93] S. Goyal, "A survey on travelling salesman problem," in *Proc. 43rd Midwest Instruction and Computing Symposium (MICS 2010)*, 2010.
[94] F. Greco, *Traveling Salesman Problem*: In-Tech, 2008.
[95] M. Grötschel, L. Lovász, and A. Schrijver, *Geometric algorithms and combinatorial optimization* vol. 2: Springer Science & Business Media, 2012.
[96] J. Gu, P. Purdom, J. Franco, and B. Wah, "Algorithms for the Satisfiability problem: a survey, DIMACS Series on Discrete Mathematics and Theoretical Computer Science 35: 19-151," *American Mathematical Society,* 1997.
[97] M. Guntsch and M. Middendorf, "Pheromone Modification Strategies for Ant Algorithms Applied to Dynamic TSP," in *Proceedings of Applications of Evolutionary Computing: EvoWorkshops 2001: EvoCOP, EvoFlight, EvoIASP, EvoLearn, and EvoSTIM Como*, E. J. W. Boers, Ed., ed Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 213-222.
[98] F. Habibzadeh, "Opinion: Journal Impact Factor: Uses and Misuses," *Archives of Iranian Medicine,* vol. 11, pp. 453-454, 2008.
[99] S. Hajforoosh, M. A. Masoum, and S. M. Islam, "Real-time charging coordination of plug-in electric vehicles based on hybrid fuzzy discrete particle swarm optimization," *Electric Power Systems Research,* vol. 128, pp. 19-29, 2015.
[100] P. E. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics,* vol. 4, pp. 100-107, 1968.
[101] B. S. Hasan, M. Khamees, and A. S. H. Mahmoud, "A heuristic genetic algorithm for the single source shortest path problem," in *Proceedings of IEEE/ACS International Conference on Computer Systems and Applications (AICCSA'07),* 2007, pp. 187-194.
[102] W. K. Hastings, "Monte Carlo sampling methods using Markov chains and their applications," *Biometrika,* vol. 57, pp. 97-109, 1970.

[103] F. Hecht, B. K. Hecht, and A. A. Sandberg, "The journal “impact factor”: a misnamed, misleading, misused measure," *Cancer genetics and cytogenetics*, vol. 104, pp. 77-81, 1998.
[104] K. Helsgaun, "An effective implementation of the Lin–Kernighan traveling salesman heuristic," *European Journal of Operational Research*, vol. 126, pp. 106-130, 2000.
[105] K. Helsgaun, "An effective implementation of K-opt moves for the Lin-Kernighan TSP heuristic," Roskilde University. Department of Computer Science, 2006.
[106] K. Helsgaun, "General k-opt submoves for the Lin–Kernighan TSP heuristic," *Mathematical Programming Computation*, vol. 1, pp. 119-163, 2009.
[107] J. E. Hirsch, "An index to quantify an individual's scientific research output," *Proceedings of the National academy of Sciences of the United States of America*, vol. 102, pp. 16569-16572, 2005.
[108] D. R. Hodge and J. R. Lacasse, "Evaluating journal quality: Is the H-index a better measure than impact factors?," *Research on Social Work Practice*, 2010.
[109] J. H. Holland, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*: MIT press, 1992.
[110] O. Hussein and T. Saadawi, "Ant routing algorithm for mobile ad-hoc networks (ARAMA)," in *Proceedings of the IEEE International Performance, Computing, and Communications Conference*, 2003, pp. 281-290.
[111] T. Ikeda, M.-Y. Hsu, H. Imai, S. Nishimura, H. Shimoura, T. Hashimoto, *et al.*, "A fast algorithm for finding better routes by AI search techniques," in *Proceedings of Vehicle Navigation and Information Systems Conference*, 1994, pp. 291-296.
[112] S. Irnich and G. Desaulniers, "Shortest Path Problems with Resource Constraints," in *Column Generation*, G. Desaulniers, J. Desrosiers, and M. Solomon, Eds., ed: Springer US, 2005, pp. 33-65.
[113] X. Ji, "Models and algorithm for stochastic shortest path problem," *Applied Mathematics and Computation*, vol. 170, pp. 503-514, 2005.
[114] Z. Ji, Y. S. Kim, and A. Chen, "Multi-objective α-reliable path finding in stochastic networks with correlated link costs: a simulation-based multi-objective genetic algorithm approach (SMOGA)," *Expert Systems with Applications*, vol. 38, pp. 1515-1528, 2011.
[115] Z. Ji, Y. S. Kim, and A. Chen, "Multi-objective α-reliable path finding in stochastic networks with correlated link costs: a simulation-based multi-objective genetic algorithm approach (SMOGA)," *Expert Systems with Applications*, vol. 38, pp. 1515-1528, 2011.
[116] Y.-X. Jin, H.-Z. Cheng, J.-y. Yan, and L. Zhang, "New discrete method for particle swarm optimization and its application in transmission network expansion planning," *Electric Power Systems Research*, vol. 77, pp. 227-233, 2007.
[117] Y. Jin, M. Olhofer, and B. Sendhoff, "Dynamic weighted aggregation for evolutionary multi-objective optimization: Why does it work and how?," in *Proceedings GECCO 2001 Conference, to appear*, 2001.
[118] H. Kaindl and G. Kainz, "Bidirectional heuristic search reconsidered," *Journal of Artificial Intelligence Research*, pp. 283-317, 1997.
[119] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Neural Networks, 1995. Proceedings., IEEE International Conference on*, 1995, pp. 1942-1948 vol.4.

[120] J. Kennedy and R. C. Eberhart, "A discrete binary version of the particle swarm algorithm," in *Systems, Man, and Cybernetics, 1997. Computational Cybernetics and Simulation., 1997 IEEE International Conference on*, 1997, pp. 4104-4108 vol.5.

[122] M. A. Khanesar, H. Tavakoli, M. Teshnehlab, and M. A. Shoorehdeli, "A Discrete Binary Version of the Particle Swarm Algorithm," in *Particle Swamp Optimization*, A. Lazinica, Ed., ed Rijeka, Croatia: In-Tech, 2009, pp. 1-10.

[123] M. A. Khanesar, M. Teshnehlab, and M. A. Shoorehdeli, "A novel binary particle swarm optimization," in *Proceedings of Mediterranean Conference on Control & Automation,(MED '07)*, 2007, pp. 1-6.

[124] L. Ki-Baek and K. Jong-Hwan, "Multiobjective Particle Swarm Optimization With Preference-Based Sort and Its Application to Path Following Footstep Optimization for Humanoid Robots," *IEEE Transactions on Evolutionary Computation*, vol. 17, pp. 755-766, 2013.

[125] D. Kirchler, L. Liberti, and R. Wolfler Calvo, "A Label Correcting Algorithm for the Shortest Path Problem on a Multi-modal Route Network," in *Experimental Algorithms*. vol. 7276, R. Klasing, Ed., ed: Springer Berlin Heidelberg, 2012, pp. 236-247.

[126] G. Klunder and H. Post, "The shortest path problem on large‐scale real‐road networks," *Networks*, vol. 48, pp. 182-194, 2006.

[127] J. D. Knowles and D. W. Corne, "Approximating the nondominated front using the Pareto archived evolution strategy," *Evolutionary computation*, vol. 8, pp. 149-172, 2000.

[128] S. Koenig and M. Likhachev, "Improved fast replanning for robot navigation in unknown terrain," in *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on*, 2002, pp. 968-975 vol.1.

[129] S. Koenig and M. Likhachev, "Fast replanning for navigation in unknown terrain," *IEEE Transactions on Robotics*, vol. 21, pp. 354-363, 2005.

[130] A. B. Kozhokanova, "Bounds for the Maximum-time Stochastic Shortest Path Problem," Department of Computer Science and Engineering, Mississippi State University, Unpublished Masters Thesis, 2014.

[131] J. Langeveld and A. P. Engelbrecht, "Set-based particle swarm optimization applied to the multidimensional knapsack problem," *Swarm Intelligence*, vol. 6, pp. 297-342, 2012.

[132] G. Laporte, "The traveling salesman problem: An overview of exact and approximate algorithms," *European Journal of Operational Research*, vol. 59, pp. 231-247, 1992.

[133] F. Lardeux, F. Saubion, and J.-K. Hao, "GASAT: a genetic local search algorithm for the satisfiability problem," *Evolutionary Computation*, vol. 14, pp. 223-253, 2006.

[134] E. L. Lawler, *Combinatorial optimization: networks and matroids*: Courier Corporation, 1976.

[135] D. Leff, "Making an impact: the rise of the impact factor as a measure of journal quality," *Journal of the American Dietetic Association*, vol. 105, pp. 29-30, 2005.

[136] C. Li, S. G. Anavatti, and T. Ray, "Analytical hierarchy process using fuzzy inference technique for real-time route guidance system," *IEEE Transactions on Intelligent Transportation Systems*, vol. 15, pp. 84-93, 2014.

[137] W. Li, L. Yushu, D. Hongbin, and X. Yuanqing, "Obstacle-avoidance Path Planning for Soccer Robots Using Particle Swarm Optimization," in *Proceedings of IEEE International Conference on Robotics and Biomimetics (ROBIO '06)*, 2006, pp. 1233-1238.

[138] J. J. Liang, A. K. Qin, P. N. Suganthan, and S. Baskar, "Comprehensive learning particle swarm optimizer for global optimization of multimodal functions," *IEEE Transactions on Evolutionary Computation*, vol. 10, pp. 281-295, 2006.

[139] J. J. Liang, B. Y. Qu, P. N. Suganthan, and B. Niu, "Dynamic Multi-Swarm Particle Swarm Optimization for Multi-objective optimization problems," in *Proceedings of 2012 IEEE Congress on Evolutionary Computation (CEC)*, 2012, pp. 1-8.

[140] C.-J. Liao, C.-T. Tseng, and P. Luarn, "A discrete version of particle swarm optimization for flowshop scheduling problems," *Computers & Operations Research*, vol. 34, pp. 3099-3111, 2007.

[141] S. Lin and B. W. Kernighan, "An effective heuristic algorithm for the traveling-salesman problem," *Operations research*, vol. 21, pp. 498-516, 1973.

[142] J. D. Little, K. G. Murty, D. W. Sweeney, and C. Karel, "An algorithm for the traveling salesman problem," *Operations research*, vol. 11, pp. 972-989, 1963.

[143] H. Liu, A. Abraham, and M. Clerc, "An Hybrid Fuzzy Variable Neighborhood Particle Swarm Optimization Algorithm for Solving Quadratic Assignment Problems," *J. UCS*, vol. 13, pp. 1309-1331, 2007.

[144] T. Liu and M. Maeda, "A Modified Hybrid Algorithm of Set-Based Particle Swarm Optimization for Traveling Salesman Problem," *Journal of Signal Processing*, vol. 17, pp. 293-297, 2013.

[145] H. S. Lope and L. S. Coelho, "Particle swarn optimization with fast local search for the blind traveling salesman problem," in *Hybrid Intelligent Systems, 2005. HIS'05. Fifth International Conference on*, 2005, pp. 245-250.

[146] J. Lopez, S. M. Susarla, E. W. Swanson, N. Calotta, and S. D. Lifchez, "The Association of the H-Index and Academic Rank Among Full-Time Academic Hand Surgeons Affiliated With Fellowship Programs," *The Journal of Hand Surgery*, 2015.

[147] Y.-y. Ma, Y.-j. Gong, W.-n. Chen, and J. Zhang, "A set-based locally informed discrete particle swarm optimization," in *Proceedings of the 15th annual conference companion on Genetic and evolutionary computation*, 2013, pp. 71-72.

[148] S. Martello, W. R. Pulleyblank, P. Toth, and D. de Werra, "Balanced optimization problems," *Operations Research Letters*, vol. 3, pp. 275-278, 1984.

[149] S. Martello and P. Toth, *Knapsack problems: algorithms and computer implementations*: John Wiley & Sons, Inc., 1990.

[150] R. Matai, M. L. Mittal, and S. Singh, *Traveling salesman problem: An overview of applications, formulations, and solution approaches*: In Tech Open Access Publisher, 2010.

[151] M. Mavrovouniotis and S. Yang, "Ant colony optimization with immigrants schemes for the dynamic travelling salesman problem with traffic factors," *Applied Soft Computing*, vol. 13, pp. 4023-4037, 2013.

[152] K. Mehlhorn and M. Ziegelmann, "Resource Constrained Shortest Paths," in *Algorithms - ESA 2000*. vol. 1879, M. Paterson, Ed., ed: Springer Berlin Heidelberg, 2000, pp. 326-337.

[153] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *The journal of chemical physics*, vol. 21, pp. 1087-1092, 1953.
[154] M. Meybodi and H. BEYGY, "Solving stochastic shortest path problem using distributed learning automata," in *Proceeding of Annual International CSI Computer Conference*, 2001.
[155] A. W. Mohemmed, N. C. Sahoo, and T. K. Geok, "Solving shortest path problem using particle swarm optimization," *Applied Soft Computing*, vol. 8, pp. 1643-1653, 2008.
[156] A. W. Mohemmed, N. C. Sahoo, and G. Tan Kim, "Particle swarm optimization with noising metaheuristics for solving network shortest path problem," in *IEEE International Conference on Telecommunications and Malaysia International Conference on Communications (ICT-MICC 2007)*, 2007, pp. 354-359.
[157] S. Momtazi, S. Kafi, and H. Beigy, "Solving Stochastic Path Problem: Particle Swarm Optimization Approach," in *New Frontiers in Applied Artificial Intelligence*. vol. 5027, N. Nguyen, L. Borzemski, A. Grzech, and M. Ali, Eds., ed: Springer Berlin Heidelberg, 2008, pp. 590-600.
[158] S. Mostaghim and J. Teich, "The role of ε-dominance in multi objective particle swarm optimization methods," in *Evolutionary Computation, 2003. CEC'03. The 2003 Congress on*, 2003, pp. 1764-1771.
[159] S. Mostaghim and J. Teich, "Strategies for finding good local guides in multi-objective particle swarm optimization (MOPSO)," in *Proceedings of the 2003 IEEE Swarm Intelligence Symposium (SIS'03)*, 2003, pp. 26-33.
[160] R. Muhandiramge and N. Boland, "Simultaneous solution of Lagrangean dual problems interleaved with preprocessing for the weight constrained shortest path problem," *Networks*, vol. 53, pp. 358-381, 2009.
[161] Y. Nagata and D. Soler, "A new genetic algorithm for the asymmetric traveling salesman problem," *Expert Systems with Applications*, vol. 39, pp. 8947-8953, 8// 2012.
[162] T. Nakagaki, "Smart behavior of true slime mold in a labyrinth," *Research in Microbiology*, vol. 152, pp. 767-770, 2001.
[163] T. Nakagaki, H. Yamada, and A. Toth, "Intelligence: Maze-solving by an amoeboid organism," *Nature*, vol. 407, pp. 470-470, 2000.
[164] F. Neumann and C. Witt, *Runtime analysis of a simple ant colony optimization algorithm*: Springer, 2006.
[165] T. A. J. Nicholson, *Finding the shortest route between two points in a network*: United Kingdom Atomic Energy Authority, 1966.
[166] G. Nicosia and G. Oriolo, "An approximate A∗ algorithm and its application to the SCS problem," *Theoretical Computer Science*, vol. 290, pp. 2021-2029, 2003.
[167] E. Nikolova, J. A. Kelner, M. Brand, and M. Mitzenmacher, "Stochastic shortest paths via quasi-convex maximization," in *Algorithms–ESA 2006*, ed: Springer, 2006, pp. 552-563.
[168] C. Nilsson, "Heuristics for the traveling salesman problem," Tech. Report, Linköping University, Sweden2003.

[169] A. Orda and R. Rom, "Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length," *J. ACM*, vol. 37, pp. 607-625, 1990.
[170] J. B. Orlin, K. Madduri, K. Subramani, and M. Williamson, "A faster algorithm for the single source shortest path problem with few distinct positive lengths," *Journal of Discrete Algorithms*, vol. 8, pp. 189-198, 6// 2010.
[171] M. Padberg and G. Rinaldi, "Optimization of a 532-city symmetric traveling salesman problem by branch and cut," *Operations Research Letters*, vol. 6, pp. 1-7, 1987.
[172] C. C. Palmer and A. Kershenbaum, "Representing trees in genetic algorithms," in *Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence*, 1994, pp. 379-384.
[173] R. K. Pan and S. Fortunato, "Author Impact Factor: tracking the dynamics of individual scientific impact," *Scientific reports*, vol. 4, 2014.
[174] W. Pang, K.-P. Wang, C.-G. Zhou, L.-J. Dong, M. Liu, H.-Y. Zhang, *et al.*, "Modified particle swarm optimization based on space transformation for solving traveling salesman problem," in *Proceedings of 2004 International Conference on Machine Learning and Cybernetics*, 2004, pp. 2342-2346.
[175] K. E. Parsopoulos, *Particle Swarm Optimization and Intelligence: Advances and Applications: Advances and Applications*: IGI Global, 2010.
[176] K. E. Parsopoulos and M. N. Vrahatis, "Recent approaches to global optimization problems through particle swarm optimization," *Natural computing*, vol. 1, pp. 235-306, 2002.
[177] K. E. Parsopoulos and M. N. Vrahatis, "Multi-objective particles swarm optimization approaches," *Multi-objective optimization in computational intelligence: Theory and practice*, pp. 20-42, 2008.
[178] Y. G. Petalas, K. E. Parsopoulos, and M. N. Vrahatis, "Memetic particle swarm optimization," *Annals of Operations Research*, vol. 156, pp. 99-127, 2007.
[179] W. Pijls and H. Post, "A new bidirectional search algorithm with shortened postprocessing," *European Journal of Operational Research*, vol. 198, pp. 363-369, 2009.
[180] D. Pisinger, "Algorithms for knapsack problems," Ph.D, Computer Science, University of Copenhagen, Copenhagen, Denmark, 1995.
[181] I. Pohl, "Bi-directional and heuristic search in path problems," Dept. of Computer Science, Stanford University., 1969.
[182] I. Pohl, *Bi-directional search*: IBM TJ Watson Research Center, 1970.
[183] G. Politowski and I. Pohl, "D-Node Retargeting in Bidirectional Heuristic Search," in *AAAI*, 1984, pp. 274-277.
[184] M. Pollack, "The Maximum Capacity through a Network," *Operations Research*, vol. 8, pp. 733-736, 1960.
[185] H. Psaraftis, "Dynamic vehicle routing: Status and prospects," *Annals of Operations Research*, vol. 61, pp. 143-164, 1995.
[186] B.-Y. Qu, P. N. Suganthan, and S. Das, "A distance-based locally informed particle swarm model for multimodal optimization," *IEEE Transactions on Evolutionary Computation*, vol. 17, pp. 387-402, 2013.

[187] G. R. Raidl and B. A. Julstrom, "A weighted coding in a genetic algorithm for the degree-constrained minimum spanning tree problem," in *Proceedings of the 2000 ACM symposium on Applied computing*, vol 1, Italy, 2000.
[188] M. Rajabi-Bahaabadi, A. Shariat-Mohaymany, M. Babaei, and C. W. Ahn, "Multi-objective path finding in stochastic time-dependent road networks using non-dominated sorting genetic algorithm," *Expert Systems with Applications*, vol. 42, pp. 5056-5064, 2015.
[189] M. Randour, J.-F. Raskin, and O. Sankur, "Variations on the stochastic shortest path problem," in *Verification, Model Checking, and Abstract Interpretation*, 2014, pp. 1-18.
[190] C. R. Raquel and P. C. Naval Jr, "An effective use of crowding distance in multiobjective particle swarm optimization," in *Proceedings of the 7th Annual conference on Genetic and Evolutionary Computation*, 2005, pp. 257-264.
[191] G. Reinelt, "TSPLIB—A traveling salesman problem library," *ORSA journal on computing*, vol. 3, pp. 376-384, 1991.
[192] G. Reinelt, "TSPLIB http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/index.html," *TSPLIB95*, 1995.
[193] P. Z. Revesz, "A method for predicting citations to the scientific publications of individual researchers," in *Proceedings of the 18th International Database Engineering & Applications Symposium*, 2014, pp. 9-18.
[194] M. Reyes-Sierra and C. C. Coello, "Multi-objective particle swarm optimizers: A survey of the state-of-the-art," *International Journal of Computational Intelligence Research*, vol. 2, pp. 287-308, 2006.
[195] L. H. O. Rios and L. Chaimowicz, "PNBA*: A Parallel Bidirectional Heuristic Search Algorithm," *Proceedings of the XXXI Congressa da Sociedade Brasileira de Computaçao*, 2011.
[196] S. Salinas and S. B. Munch, "Where should I send it? Optimizing the submission decision process," *PloS one*, vol. 10, 2015.
[197] A. Salman, I. Ahmad, and S. Al-Madani, "Particle swarm optimization for task assignment problem," *Microprocessors and Microsystems*, vol. 26, pp. 363-371, 2002.
[198] P. Sanders and L. Mandow, "Parallel Label-Setting Multi-objective Shortest Path Search," in *Proceedings of the 2013 IEEE 27th International Symposium on Parallel & Distributed Processing (IPDPS)*, 2013, pp. 215-224.
[199] A. Schrijver, "On the history of the shortest path problem," *Documenta Mathematica*, pp. 155-167, 2012.
[200] N. Shah and Y. Song, "S-index: Towards Better Metrics for Quantifying Research Impact," *arXiv preprint arXiv:1507.03650*, 2015.
[201] Q. Shen, J.-H. Jiang, C.-X. Jiao, G.-l. Shen, and R.-Q. Yu, "Modified particle swarm optimization algorithm for variable selection in MLR and PLS modeling: QSAR studies of antagonism of angiotensin II antagonists," *European Journal of Pharmaceutical Sciences*, vol. 22, pp. 145-152, 2004.
[202] Y. Shengxiang, C. Hui, and W. Fang, "Genetic Algorithms With Immigrants and Memory Schemes for Dynamic Shortest Path Routing Problems in Mobile Ad Hoc

Networks," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews,* vol. 40, pp. 52-63, 2010.
[203] X. H. Shi, Y. C. Liang, H. P. Lee, C. Lu, and Q. Wang, "Particle swarm optimization-based algorithms for TSP and generalized TSP," *Information Processing Letters,* vol. 103, pp. 169-176, 2007.
[204] L. Silva Junior, N. Nedjah, and L. de Macedo Mourelle, "Routing for applications in NoC using ACO-based algorithms," *Applied Soft Computing,* vol. 13, pp. 2224-2231, 2013.
[205] L. Sint and D. de Champeaux, "An improved bidirectional heuristic search algorithm," *Journal of the ACM (JACM),* vol. 24, pp. 177-191, 1977.
[206] R. A. Sivakumar and R. Batta, "The Variance-Constrained Shortest Path Problem," *Transportation Science,* vol. 28, pp. 309-316, 1994.
[207] S. Skiena, "Dijkstra's Algorithm," *Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica, Reading, MA: Addison-Wesley,* pp. 225-227, 1990.
[208] N. A. Smith and R. W. Tromble, "Sampling uniformly from the unit simplex," *Johns Hopkins University, Tech. Rep,* pp. 1-6, 2004.
[209] R. Smith, "Commentary: the power of the unrelenting impact factor—is it a force for good or harm?," *International Journal of Epidemiology,* vol. 35, pp. 1129-1130, 2006.
[210] A. Souza, *Combinatorial Algorithms: Lecture Notes*: Huboldt University, Berlin, 2010.
[211] C. Stegehuis, N. Litvak, and L. Waltman, "Predicting the long-term citation impact of recent publications," *arXiv preprint arXiv:1503.09156,* 2015.
[212] D. I. Stern, "High-ranked social science journal articles can be identified from early citation information," *PLoS ONE,* vol9, 2014. doi: 10.1371/journal.pone.0112520.
[213] D. Sudholt and C. Thyssen, "Running time analysis of ant colony optimization for shortest path problems," *Journal of Discrete Algorithms,* vol. 10, pp. 165-180, 2012.
[214] D. Sudholt and C. Thyssen, "A simple ant colony optimizer for stochastic shortest path problems," *Algorithmica,* vol. 64, pp. 643-672, 2012.
[215] A. Tajdin, I. Mahdavi, N. Mahdavi-Amiri, and B. Sadeghpour-Gildeh, "Computing a fuzzy shortest path in a network with mixed fuzzy arc lengths using -cuts," *Computers & Mathematics with Applications,* vol. 60, pp. 989-1002, 2010.
[216] M. F. Taşgetiren and Y.-C. Liang, "A binary particle swarm optimization algorithm for lot sizing problem," *Journal of Economic and Social Research,* vol. 5, pp. 1-20, 2004.
[217] A. Tero, R. Kobayashi, and T. Nakagaki, "A mathematical model for adaptive transport network in path finding by true slime mold," *Journal of theoretical biology,* vol. 244, pp. 553-564, 2007.
[218] M. Thorup, "Undirected single-source shortest paths with positive integer weights in linear time," *J. ACM,* vol. 46, pp. 362-394, 1999.
[219] W. F. Tichy, P. Lukowicz, L. Prechelt, and E. A. Heinz, "Experimental evaluation in computer science: A quantitative study," *Journal of Systems and Software,* vol. 28, pp. 9-18, 1995.
[220] I. C. Trelea, "The particle swarm optimization algorithm: convergence analysis and parameter selection," *Information processing letters,* 2003.

[221] P. K. Tripathi, S. Bandyopadhyay, and S. K. Pal, "Multi-objective particle swarm optimization with time variant inertia and acceleration coefficients," *Information Sciences,* vol. 177, pp. 5033-5049, 2007.
[222] A. M. Turing, "Intelligent Machinery, A Heretical Theory," *The Turing Test: Verbal Behavior as the Hallmark of Intelligence,* p. 105, 1948.
[223] L. Turner, "Variants of Shortest Path Problems," *2012,* vol. 6, p. 14, 2012-01-03 2012.
[224] V. V. Vazirani, *Approximation algorithms*: Springer Science & Business Media, 2013.
[225] M. A. Villalobos-Arias, G. T. Pulido, and C. A. C. Coello, "A new mechanism to maintain diversity in multi-objective metaheuristics," *Optimization,* vol. 61, pp. 823-854, 2012.
[226] A. Vladimirsky, "Label-setting methods for multimode stochastic shortest path problems on graphs," *Mathematics of Operations Research,* vol. 33, pp. 821-838, 2008.
[227] J. Wainer, C. G. Novoa Barsottini, D. Lacerda, and L. R. Magalhães de Marco, "Empirical evaluation in Computer Science research published by ACM," *Information and Software Technology,* vol. 51, pp. 1081-1085, 2009.
[228] H. Wang, I. Moon, S. Yang, and D. Wang, "A memetic particle swarm optimization algorithm for multimodal optimization problems," *Information Sciences,* vol. 197, pp. 38-52, 2012.
[229] H. Wang and G. G. Yen, "Adaptive Multiobjective Particle Swarm Optimization Based on Parallel Cell Coordinate System," *IEEE Transactions on Evolutionary Computation,* vol. 19, pp. 1-18, 2015.
[230] K.-P. Wang, L. Huang, C.-G. Zhou, and W. Pang, "Particle swarm optimization for traveling salesman problem," in *Proceedings of 2003 International Conference on Machine Learning and Cybernetics,* 2003, pp. 1583-1585.
[231] L. Wang and C. Singh, "Environmental/economic power dispatch using a fuzzified multi-objective particle swarm optimization algorithm," *Electric Power Systems Research,* vol. 77, pp. 1654-1664, 2007.
[232] L. Wang, X. Wang, J. Fu, and L. Zhen, "A novel probability binary particle swarm optimization algorithm and its application," *Journal of software,* vol. 3, pp. 28-35, 2008.
[233] Q. Wang, Z. Zhang, Y. Zhang, and Y. Deng, "Fuzzy shortest path problem based on biological method," *Journal of information and computational science,* vol. 95, pp. 1365-1371, 2012.
[234] Y. Wang, "The hybrid genetic algorithm with two local optimization strategies for traveling salesman problem," *Computers & Industrial Engineering,* vol. 70, pp. 124-133, 4// 2014.
[235] Y. Wang, X.-Y. Feng, Y.-X. Huang, D.-B. Pu, W.-G. Zhou, Y.-C. Liang, *et al.*, "A novel quantum swarm evolutionary algorithm and its applications," *Neurocomputing,* vol. 70, pp. 633-640, 2007.
[236] A. Warburton, "Approximation of Pareto Optima in Multiple-Objective, Shortest-Path Problems," *Operations Research,* vol. 35, pp. 70-79, 1987.
[237] C. Wei-Neng, Z. Jun, H. S. H. Chung, Z. Wen-Liang, W. Wei-gang, and S. Yu-hui, "A Novel Set-Based Particle Swarm Optimization Method for Discrete Optimization

Problems," *Evolutionary Computation, IEEE Transactions on*, vol. 14, pp. 278-300, 2010.

[238] P. Wei, W. Kang-Ping, Z. Chun-Guang, and D. Long-Jiang, "Fuzzy discrete particle swarm optimization for solving traveling salesman problem," in *Proceedings of the Fourth International Conference on Computer and Information Technology (CIT '04)*, 2004, pp. 796-800.

[239] C. Weiler, B. Biesinger, B. Hu, and G. R. Raidl, "Heuristic Approaches for the Probabilistic Traveling Salesman Problem," in *Computer Aided Systems Theory–EUROCAST 2015*, ed: Springer, 2015, pp. 342-349.

[240] A. Weld, Kolobov Mausam, D. S, and H. Geffner, "Heuristic search for generalized stochastic shortest path MDPs," 2011.

[241] T.-K. Whangbo, "Efficient modified bidirectional A* algorithm for optimal route-finding," in *New Trends in Applied Artificial Intelligence*, ed: Springer, 2007, pp. 344-353.

[242] A. W. Wilhite and E. A. Fong, "Coercive citation in academic publishing," *Science*, vol. 335, pp. 542-543, 2012.

[243] W. Wu and Q. Ruan, "A gene-constrained genetic algorithm for solving shortest path problem," in *Proceedings of the 7th International Conference on Signal Processing*, 2004, vol.3, pp. 2510-2513.

[244] Y.-L. Wu, T.-F. Ho, S. J. Shyu, and B. M. Lin, "Discrete particle swarm optimization with scout particles for library materials acquisition," *The Scientific World Journal*, vol. 2013, 2013.

[245] Y. Yabo, L. Yi, and W. Bin, "A modified ant colony algorithm to solve the shortest path problem," in *Proceedings of the 2014 International Conference on Cloud Computing and Internet of Things (CCIOT)*, 2014, pp. 148-151.

[246] D. Yanfang, T. Hengqing, and Z. Xiedong, "Dynamic shortest path in stochastic traffic networks based on fluid neural network and Particle Swarm Optimization," in *Proceedings of the 2010 Sixth International Conference on Natural Computation (ICNC)*, 2010, pp. 2325-2329.

[247] J. Yang, X. Shi, M. Marchese, and Y. Liang, "An ant colony optimization method for generalized TSP problem," *Progress in Natural Science*, vol. 18, pp. 1417-1422, 2008.

[248] X.-S. Yang, *Nature-inspired metaheuristic algorithms*: Luniver press, 2010.

[249] J. Yao, K. Li, Y. Kong, and Y. Shen, "Solving the deterministic/stochastic SPR problems by PSO with predatory escaping behavior," *International Journal of Digital Content Technology and its Applications*, vol. 6, p. 42, 2012.

[250] Q. F. Ying, S. Venkatramanan, and D. M. Chiu, "Modeling and Analysis of Scholar Mobility on Scientific Landscape," in *Proceedings of the 24th International Conference on World Wide Web Companion*, 2015, pp. 609-614.

[251] D. Yong, S. Wenkang, D. Feng, and L. Qi, "A new similarity measure of generalized fuzzy numbers and its application to pattern recognition," *Pattern Recognition Letters*, vol. 25, pp. 875-883, 2004.

[252] M. Yusoff, J. Ariffin, and A. Mohamed, "Discrete particle swarm optimization with a search decomposition and random selection for the shortest path problem," *J. Comput. Inf. Syst. Ind. Manag. Appl*, vol. 4, pp. 578-588, 2012.

[253] A. A. Zakzouk, H. M. Zaher, and R. A. Z. El-Deen, "An ant colony optimization approach for solving shortest path problem with fuzzy constraints," in *Proceedings The 7th International Conference on Informatics and Systems (INFOS)*, 2010, pp. 1-8.

[254] M. V. Zelkowitz and D. R. Wallace, "Experimental models for validating technology," *Computer*, vol. 31, pp. 23-31, 1998.

[255] F. B. Zhan, "Three fastest shortest path algorithms on real road networks: Data structures and procedures," *Journal of geographic information and decision analysis*, vol. 1, pp. 69-82, 1997.

[256] F. B. Zhan and C. E. Noon, "A comparison between label-setting and label-correcting algorithms for computing one-to-one shortest paths," *Journal of Geographic Information and Decision Analysis*, vol. 4, pp. 1-11, 2000.

[257] R. Zhang, S. Song, and C. Wu, "A hybrid artificial bee colony algorithm for the job shop scheduling problem," *International Journal of Production Economics*, vol. 141, pp. 167-178, 2013.

[258] X. Zhang, Q. Wang, A. Adamatzky, F. T. Chan, S. Mahadevan, and Y. Deng, "An improved physarum polycephalum algorithm for the shortest path problem," *The Scientific World Journal*, vol. 2014, 2014.

[259] X. Zhang, Y. Zhang, Y. Hu, Y. Deng, and S. Mahadevan, "An adaptive amoeba algorithm for constrained shortest paths," *Expert Systems with Applications*, vol. 40, pp. 7607-7616, 2013.

[260] X. Zhang, Y. Zhang, Z. Zhang, S. Mahadevan, A. Adamatzky, and Y. Deng, "Rapid Physarum Algorithm for shortest path problem," *Applied Soft Computing*, vol. 23, pp. 19-26, 2014.

[261] Y. Zhang, X. Deng, Y. Li, J. Wu, X. Sun, and Y. Deng, "E-commerce security assessment under group decision making," *Journal of Information and Computational Science*, vol. 8, pp. 7-15, 2011.

[262] Y. Zhang, Y. Jun, G. Wei, and L. Wu, "Find multi-objective paths in stochastic networks via chaotic immune PSO," *Expert Systems with Applications*, vol. 37, pp. 1911-1919, 2010.

[263] W.-l. Zhong, J. Zhang, and W.-n. Chen, "A novel discrete particle swarm optimization to solve traveling salesman problem," in *Proceedings of IEEE Congress on Evolutionary Computation, (CEC 2007)*, 2007, pp. 3283-3287.

[264] X. Zhu, J. Zhang, and J. Feng, "Multiobjective Particle Swarm Optimization Based on PAM and Uniform Design," *Mathematical Problems in Engineering*, vol. 501, p. 126404, 2015.

[265] E. Zitzler and L. Thiele, "Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach," *IEEE Transactions on Evolutionary Computation*, vol. 3, pp. 257-271, 1999.

[266] X. Zongyan, L. Haihua, and G. Ye, "A Study on the Shortest Path Problem Based on Improved Genetic Algorithm," in *2012 Fourth International Conference on* 2012 *Computational and Information Sciences (ICCIS)*, pp. 325-328.