UNIVERSIDAD DE INVESTIGACIÓN DE TECNOLOGÍA EXPERIMENTAL YACHAY

Escuela de Ciencias Matemáticas y Computacionales

TÍTULO: Artificial Bee Colony Algorithm for Nurse Scheduling Problem using High-performance computing

Trabajo de integración curricular presentado como requisito para la obtención del título de Ingeniero en Tecnologías de la Información

Autor:

Aristizábal Scacco Daniel Santiago

Tutor:

Ph. D. Fonseca Delgado Rigoberto Salomón

Urcuquí, julio 2023

Autoría

Yo, **DANIEL SANTIAGO ARISTIZÁBAL SCACCO**, con cédula de identidad 1717545204, declaro que las ideas, juicios, valoraciones, interpretaciones, consultas bibliográficas, definiciones y conceptualizaciones expuestas en el presente trabajo; así cómo, los procedimientos y herramientas utilizadas en la investigación, son de absoluta responsabilidad de el autor del trabajo de integración curricular. Así mismo, me acojo a los reglamentos internos de la Universidad de Investigación de Tecnología Experimental Yachay.

Urcuquí, julio 2023.

Daniel Santiago Aristizábal Scacco

CI: 1717545204

Autorización de publicación

Yo, **DANIEL SANTIAGO ARISTIZÁBAL SCACCO**, con cédula de identidad 1717545204, cedo a la Universidad de Investigación de Tecnología Experimental Yachay, los derechos de publicación de la presente obra, sin que deba haber un reconocimiento económico por este concepto. Declaro además que el texto del presente trabajo de titulación no podrá ser cedido a ninguna empresa editorial para su publicación u otros fines, sin contar previamente con la autorización escrita de la Universidad.

Asimismo, autorizo a la Universidad que realice la digitalización y publicación de este trabajo de integración curricular en el repositorio virtual, de conformidad a lo dispuesto en el Art. 144 de la Ley Orgánica de Educación

Urcuquí, julio 2023.

Daniel Santiago Aristizábal Scacco

CI: 1717545204

Dedication

I would like to dedicate this thesis to my family, who helped me finish this work by giving me their unconditional support all the way of my studies. A special dedication to my dad, who I know would be proud watching me from heaven.

Daniel Aristizábal

Acknowledgment

I would like to thank my adviser for his invaluable patience and feedback. I am also grateful to my teachers for their knowledge that helped me mount the bases for this work. Lastly, I thank my family for their unconditional support that kept my motivations high during this process.

Daniel Aristizábal

Resumen

El problema de la lista de enfermeras (NRP) es un problema de optimización combinatoria que se aborda asignando un conjunto de enfermeras con diferentes habilidades y contratos a diferentes tipos de turnos, durante un período de programación predefinido. Un proceso eficiente de programación de enfermeras permite a los hospitales manejar mejor los cambios de última hora y cumplir con las leyes laborales. También garantiza una comunicación fluida y eficaz entre los empleados y entre los programadores y los empleados. El problema de la lista de enfermeras se caracteriza por la presencia de un gran conjunto de restricciones, que generalmente se dividen en dos categorías: restricciones duras y blandas. Las restricciones estrictas son aquellas que se aplican rígidamente y deben cumplirse en todo momento. Las violaciones de las restricciones blandas deben evitarse, si es posible. Se propone el algoritmo de optimización de colonias de abejas (BCO) para abordar el NRP probado en un conjunto de datos de 30 enfermeras presentado en la Primera Competencia Internacional de Listas de Enfermeras (INRC2010). El algoritmo busca iterativamente la solución pasando por un proceso de validación que implica filtrar la solución a través de todas las restricciones. Los resultados producidos por el algoritmo propuesto son muy prometedores en comparación con los producidos por los métodos de los cinco ganadores de INRC2010. Todavía es necesaria una mayor investigación para mejorar aún más el algoritmo propuesto.

Palabras Clave:
Problema de lista de enfermeras, Algoritmo de optimización de colonias de abejas, Método de inteligencia de enjambre, Algoritmo inspirado en la naturaleza.

Abstract

The Nurse Rostering Problem (NRP) is a combinatorial optimization problem that is tackled by assigning a set of nurses with different skills and contracts to different types of shifts, over a predefined scheduling period. An efficient nurse scheduling process allows hospitals to better handle last-minute changes and stay in compliance with labor laws. It also ensures smooth and effective communication among employees, and between schedulers and employees. The nurse rostering problem is characterized by the presence of a large set of constraints, which are usually divided into two categories: hard and soft constraints. Hard constraints are those that are rigidly enforced and should be met at all times. Violations of the soft constraints are to be avoided, if possible. Bee Colony Optimization (BCO) algorithm is proposed to tackle the NRP tested on a dataset of 30 nurses introduced in the First International Nurse Rostering Competition (INRC2010). The algorithm iteratively searches for the solution by going through a validation process which involves filtering the solution through all the constraints. The results produced by the proposed algorithm are very promising when compared with those produced by the five INRC2010 winners' methods. Further investigation is still necessary for further improvement of the proposed algorithm.

Keywords:
Nurse Rostering Problem, Bee Colony Optimization Algorithm, Swarm Intelligence Method, Nature-inspired algorithm.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

Nurse rostering has become a very attractive research area within the field of management science/operational research and artificial intelligence, especially since the 1990's. Personnel rostering is defined as the problem of placing human resources into slots in a pattern, where the pattern denotes a set of legal shifts defined in terms of work to be done subject to given constraints [1]. Healthcare institutions recognize that good rosters add to the quality of care and to the (mental) health and social well-being of large numbers of health workers. Hospitals face different objectives that relate to medical and organizational aspects, but also to the perspective of individual nurses. In constructing a roster, healthcare institutions are constrained by various legal, management, and staff requirements that are often incompatible in their nature [2]. For example, requirements for the cover, i.e. the required mix of staff qualifications for a particular shift, from a medical care point of view, are often in conflict with the maximum working hours that are allowed for the available nurses, and may also be in conflict with individual staff preferences for that shift. In addition, hospital wards often have to deal with a lack of personnel, which makes the nurse rostering problem even more difficult. Being complex and highly constrained, nurse rostering problems have been the subject of interest within both the artificial intelligence and operational research communities.

The topic of nurse rostering (NRP) is one of the timetabling issues commonly studied

by researchers in the area of organizational analysis and artificial intelligence. The NRP is characterized as an NP-hard problem, subject to a set of hard and soft constraints, as an assignment of a set of eligible nurses to a different set of shifts over a predetermined scheduling period [3]. The hard constraints are the form that must be met in order for the roster to be feasible, whereas breaches of soft constraints in the NRP are tolerated, but as far as possible should be minimized. It is notable that the consistency of the roster is calculated in a feasible roster by the fulfillment of the soft constraints. The fundamental aim of NRP is to produce a high quality roster that is feasible. Studies in the NRP domain, however, have shown that a roster that meets all constraints is almost impossible to find because the NRP is known as a problem of combinatorial optimization [4]. It is a very difficult and demanding job to have a good quality roster due to the combinatorial and highly restricted existence of the NRP. Naturally, over the past five decades, investigations of various strategies for tackling NRP in the timetabling area have increased. Some of the earliest NRP approaches used include integer programming, target programming, case-based reasoning, and programming of constraints.

By combining other methods, metaheuristic techniques, especially the Bee Colony Optimization Algorithm (BCO), can be easily adapted to tackle a larger number of NP-hard combinatorial optimization problems. It is possible to break the metaheuristic approach into local search methods and global search methods. To leverage the search space of the problem, local search methods such as tabu search, simulated annealing, and the Nelder-Mead Methods are used while global search methods such as scatter search, genetic algorithms, and Bee Colony Optimization concentrate on exploring the search space. The natural behavior of organisms in swarm intelligence will obey a simple basic rule to structure their environment. There would be no centralized mechanism for the agents to monitor other people; it uses local interactions between the agents to evaluate the agents' complex global behavior [5]. Bird flocking, ant colony, fish training, and animal herding strategies provide some of the influenced natural behavior of swarm intelligence. The different algorithms include the optimization algorithm for the ant colony, the genetic algorithm and the algorithm for particle swarm optimization [6]. Honey bees' normal foraging behavior has inspired the bee algorithm. All honey bees will start collecting nectar from different

locations around their new hive, and the community decision of honey bees is the method of finding the best nectar spot. The honey bees' mode of communication is carried out by the waggle dance method to tell hive mates about the location of rich food sources.

This thesis discusses the NRP dataset proposed by the First International Nurse Rostering Competition (INRC2010), which is organized by the study group CODeS at Katholieke Universiteit Leuven in Belgium, the Norwegian SINTEF Group and the Italian University of Udine. The INRC2010 dataset is categorized into three tracks: sprint, medium, and long, varying in size and complexity. In accordance with their publication period in the competition, each track is grouped into four categories: early, late, secret, and hint. Specifically, the dataset used in this study, which is further discussed in chapter 4, is composed of thirty nurses with their respective schedule requirements. During and after the competition, few approaches suggested to overcome the INRC2010 dataset are checked in Chapter 3 as well as other approaches used to tackle the NRP. It is worth noting that research in the domain of NRP is still active, since exact solution has as yet been found for the INRC2010 dataset which required further investigations using other algorithmic techniques. The main purpose of this study is to investigate whether the use of the Bee Colony Optimization (BCO) algorithm could improve the state-of-the-art results for the INRC2010 dataset.

1.2 Problem statement

The Nurse Rostering Problem (NRP) is a complex combination problem because the search space of possible solutions increase exponentially depending of different factors. Also, there are NRP scenarios where does not exist a solution that satisfies all the constraints. Explore all the possible solutions, to find the best one, will demand a huge computational power. Heuristic optimization algorithms do not explore all the search space to find viable solutions, that is why the Bee Colony Optimization (BCO) algorithm presents a useful strategy to tackle combination problems. The properly application of the BCO algorithm requires to study the problem, then define the variables that will be involved in the optimization process.

This work aims to use of the BCO algorithm to increase the efficacy of very critical healthcare problems such as the NRP. In this work, the Artificial Bee Colony Algorithm seeks to optimize the schedules of nurses by efficiently organizing their shifts throughout the week. In order to do this, the algorithm must take in account the conditions of the problem and the requirements of the nurses.

1.3 Objectives

1.3.1 General Objective

Elaborate an optimization system based on BCO algorithm applied in multiobjective NRP to satisfy hard and soft constraints in an efficient time compared with the state of the art.

We expect that this approach will help with the organization of nurse staff members by developing a more effective schedule for the nurses and for the hospital, consequently, improving the client attention.

1.3.2 Specific Objectives

- Minimize the total cost of the rostering problem using BCO, satifying hard and soft contraints proposed by the NRP.
- Analyse the efficiency of the proposed optimization system based on BCO algorithm. By comparing the feasibility and solution cost with state of the art methods used to solve the NRP.

1.4 Contribution

The BCO optimization system was applied successfully in multiobjective NRP, satisfying hard and soft contraints. The proposed system was tested on a dataset of 30 nurses introduced in the INRC2010. The results produced by the proposed system showed competitive performance metrics when compared with those produced by the five INRC2010 winners' methods.

Chapter 2

Theoretical Framework and State of the Art

In the course of all the research carried out to complete this work, the knowledge of several topics was essential. These topics helped me to reach my goal, which was to implement the algorithm of the bee colony for the optimization of nurses' schedules. All this theoretical framework is explained below to provide the foundations that are needed to better understand all the scientific and practical concepts discussed in this work.

2.1 NP-Hard Problem

As described in chapter 1, the nurse rostering problem is characterized as an NP-hard problem. In computational complexity theory, NP-hardness (non-deterministic polynomial-time hardness) is the defining property of a class of problems that are informally "at least as hard as the hardest problems in NP" [7]. A simple example of an NP-hard problem is the subset sum problem. A more precise specification is: a problem H is NP-hard when every problem L in NP can be reduced in polynomial time to H; that is, assuming a solution for H takes 1 unit time, H's solution can be used to solve L in polynomial time. As a consequence, finding a polynomial time algorithm to solve any NP-hard problem would give polynomial time algorithms for all the problems in NP. As it is suspected that P $\neq$ NP, it is unlikely that such an algorithm exists. A common misconception is that the NP in "NP-hard" stands for "non-polynomial" when in fact it stands for "non-deterministic polynomial acceptable problems". It is suspected that there are no polynomial-time algo-

rithms for NP-hard problems, but that has not been proven. Moreover, the class P, in which all problems can be solved in polynomial time, is contained in the NP class.

A decision problem H is NP-hard when for every problem L in NP, there is a polynomial-time many-one reduction from L to H. An equivalent definition is to require that every problem L in NP can be solved in polynomial time by an oracle machine with an oracle for H. Informally, an algorithm can be thought of that calls such an oracle machine as a subroutine for solving H and solves L in polynomial time if the subroutine call takes only one step to compute. Another definition is to require that there be a polynomial-time reduction from an NP-complete problem G to H [8]. As any problem L in NP reduces in polynomial time to G, L reduces in turn to H in polynomial time so this new definition implies the previous one. Awkwardly, it does not restrict the class NP-hard to decision problems, and it also includes search problems or optimization problems. An example of an NP-hard problem is the decision subset sum problem: given a set of integers, does any non-empty subset of them add up to zero? That is a decision problem and happens to be NP-complete. Another example of an NP-hard problem is the optimization problem of finding the least-cost cyclic route through all nodes of a weighted graph. This is commonly known as the traveling salesman problem.

There are decision problems that are NP-hard but not NP-complete such as the halting problem. That is the problem which asks ”given a program and its input, will it run forever?” That is a yes/no question and so is a decision problem. It is easy to prove that the halting problem is NP-hard but not NP-complete. For example, the Boolean satisfiability problem can be reduced to the halting problem by transforming it to the description of a Turing machine that tries all truth value assignments and when it finds one that satisfies the formula it halts and otherwise it goes into an infinite loop [1]. It is also easy to see that the halting problem is not in NP since all problems in NP are decidable in a finite number of operations, but the halting problem, in general, is undecidable. There are also NP-hard problems that are neither NP-complete nor Undecidable. For instance, the language of true quantified Boolean formulas is decidable in polynomial space, but not in non-deterministic polynomial time (unless NP = PSPACE).

Every NP-complete problem can be solved by exhaustive search [9]. Unfortunately, when the size of the instances grows the running time for exhaustive search soon becomes forbiddingly large, even for instances of fairly small size. For some problems it is possible to design algorithms that are significantly faster than exhaustive search, though still not polynomial time. This survey deals with such fast, super-polynomial time algorithms that solve NP-complete problems to optimality. In recent years there has been growing interest in the design and analysis of such super-polynomial time algorithms. This interest has many causes.

- It is now commonly believed that P=NP, and that super-polynomial time algorithms are the best we can hope for when we are dealing with an NPcomplete problem. There is a handful of isolated results scattered across the literature, but we are far from developing a general theory. In fact, we have not even started a systematic investigation of the worst case behavior of such super-polynomial time algorithms.

- Some NP-complete problems have better and faster exact algorithms than others. There is a wide variation in the worst case complexities of known exact (super-polynomial time) algorithms. Classical complexity theory can not explain these differences. Do there exist any relationships among the worst case behaviors of various problems? Is progress on the different problems connected? Can we somehow classify NP-complete problems to see how close we are to the best possible algorithms?

- With the increased speed of modern computers, large instances of NPcomplete problems can be solved effectively. For example it is nowadays routine to solve travelling salesman (TSP) instances with up to 2000 cities.

- And if the data is nicely structured, then instances with up to 13000 cities can be handled in practice. There is a huge gap between the empirical results from testing implementations and the known theoretical results on exact algorithms.

- Fast algorithms with exponential running times may actually lead to practical algorithms, at least for moderate instance sizes. For small instances, an algorithm with

an exponential time complexity of O(1.01n) should usually run much faster than an algorithm with a polynomial time complexity of O(n4).

2.2 Swarm Intelligence

The Bee Colony Optimization Algorithm is a type of swarm intelligence algorithm. Swarm Intelligence is a relatively new interdisciplinary field of research, which has gained huge popularity in these days. Algorithms belonging to the domain, draw inspiration from the collective intelligence emerging from the behavior of a group of social insects (like bees, termites and wasps). When acting as a community, these insects even with very limited individual capability can jointly (cooperatively) perform many complex tasks necessary for their survival [10]. Problems like finding and storing foods, selecting and picking up materials for future usage require a detailed planning, and are solved by insect colonies without any kind of supervisor or controller. An example of particularly successful research direction in swarm intelligence is Ant Colony Optimization (ACO), which focuses on discrete optimization problems, and has been applied successfully to a large number of NP hard discrete optimization problems including the traveling salesman, the quadratic assignment, scheduling, vehicle routing, etc., as well as to routing in telecommunication networks [11]. Particle Swarm Optimization (PSO) is another very popular SI algorithm for global optimization over continuous search spaces. Since its advent in 1995, PSO has attracted the attention of several researchers all over the world resulting into a huge number of variants of the basic algorithm as well as many parameter automation strategies.

Swarm intelligence refers to a subset of artificial intelligence (AI). It has been identified as an emerging field which was coined for the first time by Gerardo Beni and Jing Wang in 1989 in the context of developing cellular robotic systems. There are multiple reasons responsible for the growing popularity of such SI-based algorithms, most importantly being the flexibility and versatility offered by these algorithms [12]. The self-learning capability and adaptability to external variations are the key features exhibited by the algorithms which has attracted immense interest and identified several application areas. In recent times, swarm intelligence has grown in popularity with the increasing prominence of

NP-hard problems where finding a global optima becomes almost impossible in real-time scenario. The number of potential solutions which may exist in such problems often tends to be infinite. In such situations, finding a workable solution within time limitations becomes important. SI finds its utility in solving nonlinear design problems with real-world applications considering almost all areas of sciences, engineering and industries, from data mining to optimization, computational intelligence, business planning, in bioinformatics and in industrial applications [13]. Some high-end application areas include navigation control, interferometry, planetary motion sensing, micro-robot control, malignant tumour detection and control and image processing technologies. Being an emerging topic of research, not many publications are available which relate to swarm intelligence, except for few of the dominant approaches, which again has been over applied. Hence, the authors aim to present a review which discusses certain handpicked swarm intelligence algorithms and their future scope.

2.3 Object-oriented Programming

For the implementation of the code it was necessary to use object-oriented programming due to the complexity of the algorithm. Object-oriented programming (OOP) is a computer programming model that organizes software design around data, or objects, rather than functions and logic. An object can be defined as a data field that has unique attributes and behavior.

OOP focuses on the objects that developers want to manipulate rather than the logic required to manipulate them. This approach to programming is well-suited for programs that are large, complex and actively updated or maintained. This includes programs for manufacturing and design, as well as mobile applications; for example, OOP can be used for manufacturing system simulation software [14]. The organization of an object-oriented program also makes the method beneficial to collaborative development, where projects are divided into groups. Additional benefits of OOP include code reusability, scalability and efficiency.

The first step in OOP is to collect all of the objects a programmer wants to manipulate and identify how they relate to each other – an exercise known as data modeling. Examples of an object can range from physical entities, such as a human being who is described by properties like name and address, to small computer programs, such as widgets [15]. Once an object is known, it is labeled with a class of objects that defines the kind of data it contains and any logic sequences that can manipulate it. Each distinct logic sequence is known as a method. Objects can communicate with well-defined interfaces called messages.

The structure, or building blocks, of object-oriented programming include the following:

- Classes are user-defined data types that act as the blueprint for individual objects, attributes and methods.
- Objects are instances of a class created with specifically defined data. Objects can correspond to real-world objects or an abstract entity. When class is defined initially, the description is the only object that is defined.
- Methods are functions that are defined inside a class that describe the behaviors of an object. Each method contained in class definitions starts with a reference to an instance object. Additionally, the subroutines contained in an object are called instance methods. Programmers use methods for reusability or keeping functionality encapsulated inside one object at a time.
- Attributes are defined in the class template and represent the state of an object. Objects will have data stored in the attributes field. Class attributes belong to the class itself.

Object-oriented programming is based on the following principles:

- Encapsulation. This principle states that all important information is contained inside an object and only select information is exposed. The implementation and state of each object are privately held inside a defined class. Other objects do not have access to this class or the authority to make changes. They are only able to call a list of public functions or methods. This characteristic of data hiding provides greater

program security and avoids unintended data corruption. Abstraction. Objects only reveal internal mechanisms that are relevant for the use of other objects, hiding any unnecessary implementation code. The derived class can have its functionality extended. This concept can help developers more easily make additional changes or additions over time.

- Inheritance. Classes can reuse code from other classes. Relationships and subclasses between objects can be assigned, enabling developers to reuse common logic while still maintaining a unique hierarchy. This property of OOP forces a more thorough data analysis, reduces development time and ensures a higher level of accuracy.

- Polymorphism. Objects are designed to share behaviors and they can take on more than one form. The program will determine which meaning or usage is necessary for each execution of that object from a parent class, reducing the need to duplicate code. A child class is then created, which extends the functionality of the parent class. Polymorphism allows different types of objects to pass through the same interface.

2.4 Applied Algorithms for NRP

Valouxis et al. applied Integer Programming (IP) in [16] to solve the NRP using the INRC2010 dataset in which their solution process consists of two phases: the first step consists of assigning various nurses to working days, while the second phase includes arranging those shifts for nurses assigned to working days. In the first step, the authors used three additional neighborhood structures: (i) rescheduling one day for another time in the roster, (ii) rescheduling two days for another time in the roster, and (iii) reshuffling the shifts among nurses for the medium and long track of the data set. In all three tacks in the dataset, the approach ranked first. The presentation of two methods for solving the dataset of INRC2010 is discussed in [17]. The authors used the ejection chain-based approach for the sprint track dataset in their work, while the branch and price method is used for the INRC2010 dataset's medium and long tracks. For the medium and long tracks, the branch and price method achieved second rank, while the ejection method was fourth in the dataset's sprint track. INCR2010 dataset modeling as a Constraint Optimization

Problem (COP) is given in [18]. In order to further boost the results and the methodology ranked second, third, fourth in the sprint, medium and long tracks of INRC2010, the author used the "COP solver" based on tabu scan.

The application of a taboo-based adaptive local search to solve the INRC2010 dataset is presented in [19] in which the solution approach is also split into two phases. The first stage includes the use of a random heuristic technique to produce a feasible roster, while in the second stage the use of two neighborhood structures (i.e. transfer and swap) was used to maximize the solution. It is worth noting that the methodology kept the previous rosters in an elite pool. If, with the help of local search method, the consistency of the roster could not be improved within a given number of iterations, then one of the elite rosters is randomly selected to restart the second level. In the sprint and medium tracks, the system achieved third and fourth position, respectively.

Bilgin et al. introduced the hybridization of a hyper-heuristic with a greedy shuffle step to solve an INRC2010 dataset [20]. Simulated hyper-heuristic annealing was used at the initial stage to produce a feasible roster where soft constraints are met to the greatest extent possible. The selfish shuffle was used for the roster's further development. The hybrid hyper-heuristic technique was third in the long track, and fifth in the INRC2010 dataset's sprint and medium tracks. A heuristic method for solving the INRC2010 dataset is considered in the introduction of [21]. In constructing a feasible roster, the heuristic approach was used as well as trying to achieve the fulfillment of five predefined soft constraints. To boost the roster further, the authors used three local search procedures. The technique reached its fifth long-term position. The harmony search algorithm (HSA) adaptation was suggested for NRP using the INRC2010 dataset in [22] where the results achieved on small data set instances indicate that the method is very promising. In another development, the HSA was later changed to include unique local search procedures in the pitch adjustment operator to mitigate violations of soft constraints in the pitch adjustment operator [22]. With the hybridization of greedy shuffle local search technique, which was used to boost the new solution locally at each iteration, the efficiency of updated HSA is further improved [23]. Other works relating to HSA that have been used to resolve NRP can be found in

[24],[25], [26]. It is worth noting that NRP research is still active, since the INRC2010 dataset, which needed further investigations using other algorithmic techniques, has yet to find an exact solution.

Berrada et al. [27] considered multiple objectives to tackle the nurse scheduling problem by considering various ordered soft constraints. The soft constraints are ordered based on priority level, and this determines the quality of the solution. Burke et al. [28] proposed a multiobjective Pareto-based search technique and used simulated annealing based on a weighted-sum evaluation function towards preferences and a dominated-based evaluation function towards the Pareto set. Many mathematical models are proposed to reduce the cost and increase the performance ofthe task. The performance of the problem greatly depends on the type of constraints used [4]. Dowsland [29] proposed a technique of chain moves using a multistate tabu search algorithm. This algorithm exchanges the feasible and infeasible search space to increase the transmission rate when the system gets disconnected. But this algorithm fails to solve other problems in different search space instances.

Burke et al. [30] proposed a hybrid tabu search algorithm to solve the NRP in Belgian hospitals. In their constraints, the authors have added the previous roster along with hard and soft constraints. To consider this, they included heuristic search strategies in the general tabu search algorithm. This model provides flexibility and more user control. A hyper heuristic algorithm with tabu search is proposed for the NRP by Burke et al. [31]. They developed a rule based reinforcement learning, which is domain specific, but it chooses a little low-level heuristic to solve the NRP. The indirect genetic algorithm is problem dependent which uses encoding and decoding schemes with genetic operator to solve NRP. Burke et al. [32] developed a memetic algorithm to solve the nurse scheduling problem, and the authors have compared memetic and tabu search algorithm. The experimental result shows a memetic algorithm outperforms with better quality than the genetic algorithm and tabu search algorithm.

Simulated annealing has been proposed to solve the NRP. Hadwan and Ayob [33] introduced a shift pattern approach with simulated annealing.The authors have proposed a

greedy constructive heuristic algorithm to generate the required shift patterns to solve the NRP at UKMMC (University Kebangsaan Malaysia Medical Centre). This methodology will reduce the complexity of the search space solution to generate a roster by building two- or three-day shift patterns. The efficiency of this algorithm was shown by experimental results with respect to execution time, performance considerations, fairness, and the quality of the solution. This approach was capable of handling all hard and soft constraints and produces a quality roster pattern. Sharif et al. [34] proposed a hybridized heuristic approach with changes in the neighborhood descent search algorithm to solve the NRP at UKMMC. This heuristic is the hybridization of cyclic schedule with non cyclic schedule. They applied repairing mechanism, which swaps the shifts between nurses to tackle the random shift arrangement in the solution. A variable neighborhood descent search algorithm (VNDS) is used to change the neighborhood structure using a local search and generate a quality duty roster. In VNDS, the first neighborhood structure will reroster nurses to different shifts and the second neighborhood structure will do repairing mechanism.

Aickelin and Dowsland [35] proposed a technique for shift patterns; they considered shift patterns with penalty, preferences, and number of successive working days. The indirect genetic algorithm will generate various heuristic decoders for shift patterns to reconstruct the shift roster for the nurse. A qualified roster is generated using decoders with the help of the best permutations of nurses. To generate best search space solutions for the permutation of nurses, the authors used an adaptive iterative method to adjust the order of nurses as scheduled one by one. Asta et al. [36] and Anwar et al. [37] proposed a tensor-based hyper heuristic to solve the NRP. The authors tuned a specific group of datasets and embedded a tensor-based machine learning algorithm. A tensor-based hyper heuristic with memory management is used to generate the best solution. This approach is considered in life-long applications to extract knowledge and desired behavior throughout the run time.

Todorovic and Petrovic [38] proposed the Bee Colony Optimization approach to solve the NRP; all the unscheduled shifts are allocated to the available nurses in the constructive phase. This algorithm combines the constructive move with local search to improve the

quality of the solution. For each forward pass,the predefined numbers of unscheduled shifts are allocated to the nurses and discarded the solution with less improvement in the objective function. The process of intelligent reduction in neighborhood search had improved the current solution. In construction phase, unassigned shifts are allotted to nurses and lead to violation of constraints to higher penalties.

Several methods have been proposed using the INRC2010 dataset to solve the NRP; the authors have considered five latest competitors to measure the effectiveness of the proposed algorithm. Asaju et al. [39] proposed Artificial Bee Colony (ABC) algorithm to solve NRP. This process is done in two phases; at first heuristic based ordering of shift pattern is used to generate the feasible solution. In the second phase, to obtain the solution, ABC algorithm is used. In this method, premature convergence takes place, and the solution gets trapped in local optima. The lack of a local search algorithm of this process leads to yielding higher penalty. Awadallah et al. [40] developed a meta heuristic technique hybrid artificial bee colony (HABC) to solve the NRP. In ABC algorithm, the employee bee phase was replaced by a hill climbing approach to increase exploitation process. Use of hill climbing in ABC generates a higher value which leads to high computational time.

The global best harmony search with pitch adjustment design is used to tackle the NRP in [24].The author adapted the harmony search algorithm (HAS) in exploitation process and particle swarm optimization (PSO) in exploration process. In HAS, the solutions are generated based on three operator, namely, memory consideration, random consideration, and pitch adjustment for the improvisation process. They did two improvisations to solve the NRP, multipitch adjustment to improve exploitation process and replaced random selection with global best to increase convergence speed. The hybrid harmony search algorithm with hill climbing is used to solve the NRP in [41]. For local search, meta heuristic harmony and hill climbing approach are used. The memory consideration parameter in harmony is replaced by PSO algorithm. The derivative criteria will reduce the number of iterations towards local minima. This process considers many parameters to construct the roster since improvisation process is to be at each iteration.

Santos et al. [42] used integer programming (IP) to solve the NRP and proposed monolith compact IP with polynomial constraints and variables. The authors have used both upper and lower bounds for obtaining optimal cost. They estimated and improved lower bound values towards optimum, and this method requires additional processing time.

2.5 Performance Metrics

The performance of the proposed system is assessed by comparing with five different competitor methods. Here six performance metrics are considered to investigate the significance and evaluate the experimental results. The metrics are listed in this section.

2.5.1 Least Error Rate

Least Error Rate (LER) is the percentage of the difference between known optimal value and the best value obtained. The LER can be calculated using 2.1

$$\text{LER}(\%) = \sum_{i=1}^{r} \frac{\text{Optimal}_{\text{NRP-Instance}} - \text{fitness}_i}{\text{Optimal}_{\text{NRP-Instance}}}. \tag{2.1}$$

2.5.2 Average Convergence

The Average Convergence is the measure to evaluate the quality of the generated population on average. The Average Convergence (AC) is the percentage of the average of the convergence rate of solutions. The performance of the convergence time is increased by the Average Convergence to explore more solutions in the population. The Average Convergence is calculated using 2.2

$$\text{AC} = \sum_{i=1}^{r} 1 \frac{\text{Avg\_fitness}_i - \text{Optimal}_{\text{NRP-Instance}}}{\text{Optimal}_{\text{NRP-Instance}}} * 100, \tag{2.2}$$

where (r) is the number of instances in the given dataset.

2.5.3 Standard Deviation

Standard deviation (SD) is the measure of dispersion of a set of values from its mean value. Average Standard Deviation is the average of the standard deviation of all instances taken

from the dataset. The Average Standard Deviation (ASD) can be calculated using 2.3

$$\text{ASD} = \sqrt{\sum_{i=1}^{r}(\text{value obtained in each instance}_i - \text{Mean value of the instance})^2}, \quad (2.3)$$

where (r) is the number of instances in the given dataset.

2.5.4 Convergence Diversity

The Convergence Diversity (CD) is the difference between best convergence rate and worst convergence rate generated in the population. The Convergence Diversity can be calculated using 2.4

$$\text{CD} = \text{Convergence}_{\text{best}} - \text{Convergence}_{\text{worst}}, \quad (2.4)$$

where $\text{Convergence}_{\text{best}}$ is the convergence rate of best fitness individual and $\text{Convergence}_{\text{worst}}$ is the convergence rate of worst fitness individual in the population.

2.5.5 Cost Diversion

Cost reduction is the difference between known cost in the NRP Instances and the cost obtained from our approach. Average Cost Diversion (ACD) is the average of cost diversion to the total number of instances taken from the dataset. The value of ACD can be calculated from 2.5

$$\text{ACD} = \sum_{i=1}^{r} \frac{\text{Cost}_i - \text{Cost}_{\text{NRP-Instance}}}{\text{Total number of instances}}, \quad (2.5)$$

where (r) is the number of instances in the given dataset.

Chapter 3

Methodology

3.1 Phases of Problem Solving

3.1.1 Description of the Problem

The NRP is a real-world problem at hospitals; the problem is to assign a predefined set of shifts (like S1-day shift, S2-noon shift, S3-night shift, and S4-Free-shift) of a scheduled period for a set of nurses of different preferences and skills in each ward. Figure 3.1 shows the illustrative example of the feasible nurse roster, which consists of four shifts, namely, day shift, noon shift, night shift, and free shift (holiday), allocating five nurses over 11 days of scheduled period [1]. Each column in the scheduled table represents the day and the cell content represents the shift type allocated to a nurse. Each nurse is allocated one shift per day and the number of shifts is assigned based on the hospital contracts. This problem will have some variants on a number of shift types, nurses, nurse skills, contracts, and scheduling period. In general, both hard and soft constraints are considered for generating and assessing solutions.

Hard constraints are the regulations which must be satisfied to achieve the feasible solution. They cannot be violated since hard constraints are demanded by hospital regulations. The hard constraints HC1 to HC5 must be filled to schedule the roster. The soft constraints SC1 to SC14 are desirable, and the selection of soft constraints determines the quality of the roster. Tables 3.1 and 3.2 list the set of hard and soft constraints considered to solve the NRP.

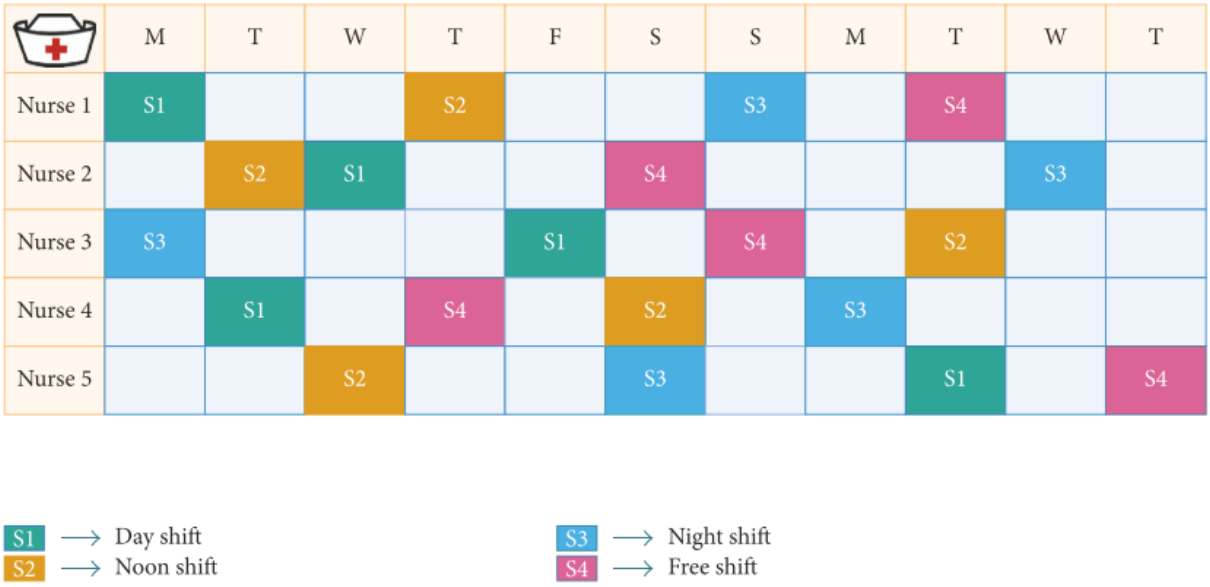

Figure 3.1: Illustrative example of Nurse Rostering Problem.

Hard constraints	
HC1	All demanded shifts assigned to a nurse.
HC2	A nurse can work with only a single shift per day.
HC3	The minimum number of nurses required for the shift.
HC4	The total number of working days for the nurse should be between the maximum and minimum range.
HC5	A day shift followed by night shift is not allowed.

Table 3.1: Hard constraints of Nurse Rostering Problem.

Soft constraints	
SC1	The maximum number of shifts assigned to each nurse.
SC2	The minimum number of shifts assigned to each nurse.
SC3	The maximum number of consecutive working days assigned to each nurse.
SC4	The minimum number of consecutive working days assigned to each nurse.
SC5	The maximum number of consecutive working days assigned to each nurse on which no shift is allotted.
SC6	The minimum number of consecutive working days assigned to each nurse on which no shift is allotted.
SC7	The maximum number of consecutive working weekends with at least one shift assigned to each nurse.
SC8	The minimum number of consecutive working weekends with at least one shift assigned to each nurse.
SC9	The maximum number of weekends with at least one shift assigned to each nurse.
SC10	Specific working day.
SC11	Requested day off.
SC12	Specific shift on.
SC13	Specific shift off.
SC14	Nurse not working on the unwanted pattern.

Table 3.2: Soft constraints of Nurse Rostering Problem.

3.1.2 Analysis of the Problem

The NRP consists of a set of nurses n = 1,2,...,N, where each row is specific to particular set of shifts s = 1,2,...,S, for the given set day d = 1,2,...,D. The solution roster ς for the 0/1 matrix dimension N*SD is as in equation 3.1.

$$\varsigma = \begin{cases} 1 & \text{if nurse n works s shift for day d} \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

HC1. In this constraint, all demanded shifts are assigned to a nurse as shown in equation 3.2.

$$\sum_{n=1}^{N} \varsigma_{d,s}^{n} = E_{ds}, \forall d \epsilon D, s \epsilon S, \tag{3.2}$$

where E_{ds} is the number of nurses required for a day (d) at shift (s) and $\varsigma_{d,s}$ is the allocation of nurses in the feasible solution roster.

HC2. In this constraint, each nurse can work not more than one shift per day as shown in equation 3.3.

$$\sum_{s=1}^{S} \varsigma_{n,d}^{n} \leq 1, \forall n \epsilon N, d \epsilon D, \tag{3.3}$$

where $\varsigma_{n,d}$ is the allocation of nurses (n) in solution at shift (s) for a day (d).

HC3. This constraint deals with a minimum number of nurses required for each shift as shown in equation 3.4.

$$\sum_{n=1}^{N} \varsigma_{d,s}^{n} \geq \min_{d,s}^{n}, \forall d \epsilon D, s \epsilon S, \tag{3.4}$$

where $\min_{d,s}^{n}$ is the minimum number of nurses required for a shift (s) on the day (d).

HC4. In this constraint, the total number of working days for each nurse should range between minimum and maximum range for the given scheduled period as shown in equation 3.5.

$$W_{\min} \leq \sum_{d=1}^{D} \sum_{s=1}^{S} \leq W_{\max}, \forall n \epsilon N. \tag{3.5}$$

The average working shift for nurse can be determined by using equation 3.6

$$W_{\text{avg}} = \frac{1}{N} \left(\sum_{d=1}^{D} \sum_{s=1}^{S} \varsigma_n^{d,s}, \forall n \epsilon N \right), \tag{3.6}$$

where $W_{\min}$ and $W_{\max}$ are the minimum and maximum number of days in scheduled period and W_{avg} is the average working shift of the nurse.

HC5. In this constraint, shift 1 followed by shift 3 is not allowed; that is, a day shift followed by a night shift is not allowed as shown in equation 3.7.

$$\sum_{n=1}^{N} \sum_{d=1}^{D} \varsigma_{s3}^{n,d} + \varsigma_{s1}^{n,d+1} \leq 1, \forall s \epsilon S. \tag{3.7}$$

SC1. The maximum number of shifts assigned to each nurse for the given scheduled period is as equation 3.8:

$$\max \left(\left(\sum_{d=1}^{D} \sum_{s=1}^{S} \varsigma_n^{d,s} - \Phi_n^{ub} \right), 0 \right), \forall n \epsilon N, \tag{3.8}$$

where Φ_n^{ub} is the maximum number of shifts assigned to nurse (n).

SC2. The minimum number of shifts assigned to each nurse for the given scheduled period is as equation 3.9:

$$\max \left(\left(\Phi_n^{lb} - \sum_{d=1}^{D} \sum_{s=1}^{S} \varsigma_n^{d,s} \right), 0 \right), \forall n \epsilon N, \tag{3.9}$$

where Φ_n^{lb} is the minimum number of shifts assigned to nurse (n).

SC3. The maximum number of consecutive working days assigned to each nurse on which a shift is allotted for the scheduled period is as equation 3.10:

$$\sum_{k=1}^{\Psi_n} \max\left(\left(\mathbb{C}_n^k - \Theta_n^{ub}\right), 0\right), \forall n \epsilon N, \tag{3.10}$$

where Θ_n^{ub} is the maximum number of consecutive working days of nurse (n), Ψ_n is the total number of consecutive working spans of nurse (n) in the roster, and $\mathbb{C}_n^k$ is the count of the kth working spans of nurse (n).

SC4. The minimum number of consecutive working days assigned to each nurse on which a shift is allotted for the scheduled period is as equation 3.11:

$$\sum_{k=1}^{\Psi_n} \max\left(\left(\Theta_n^{lb} - \mathbb{C}_n^k\right), 0\right), \forall n \epsilon N, \tag{3.11}$$

where Θ_n^{lb} is the minimum number of consecutive working days of nurse (n), Ψ_n is the total number of consecutive working spans of nurse (n) in the roster, and $\mathbb{C}_n^k$ is the count of the the working span of the nurse (n).

SC5. The maximum number of consecutive working days assigned to each nurse on which no shift is allotted for the given scheduled period is as equation 3.12:

$$\sum_{k=1}^{\Gamma_n} \max\left(\left(\delta_n^k - \varphi_n^{ub}\right), 0\right), \forall n \epsilon N, \tag{3.12}$$

where φ_n^{ub} is the maximum number of consecutive free days of nurse (n), Γ_n is the total number of consecutive free working spans of nurse (n) in the roster, and δ_n^k is the count of the kth working span of the nurse (n).

SC6. The minimum number of consecutive working days assigned to each nurse on which no shift is allotted for the given scheduled period is as equation 3.13:

$$\sum_{k=1}^{\Gamma_n} \max\left(\left(\varphi_n^{lb} - \delta_n^k\right), 0\right), \forall n \epsilon N, \tag{3.13}$$

where φ_n^{lb} is the minimum number of consecutive free days of nurse (n), Γ_n is the total number of consecutive free working spans of nurse (n) in the roster, and δ_n^k is the count of the kth working span of the nurse (n).

SC7. The maximum number of consecutive working weekends with at least one shift assigned to nurse for the given scheduled period is as equation 3.14:

$$\sum_{k=1}^{\Upsilon_n} \max\left(\left(\zeta_n^k - \Omega_n^{ub}\right), 0\right), \forall n \epsilon N, \tag{3.14}$$

where Ω_n^{ub} is the maximum number of consecutive working weekends of nurse (n), Υ_n is the total number of consecutive working weekend spans of nurse (n) in the roster, and ζ_n^k is the count of the kth working weekend span of the nurse (n).

SC8. The minimum number of consecutive working weekends with at least one shift assigned to nurse for the given scheduled period is as equation 3.15:

$$\sum_{k=1}^{\Upsilon_n} \max\left(\left(\Omega_n^{lb} - \zeta_n^k\right), 0\right), \forall n \epsilon N, \tag{3.15}$$

where Ω_n^{lb} is the minimum number of consecutive working weekends of nurse (n), Υ_n is the total number of consecutive working weekend spans of nurse (n) in the roster, and ζ_n^k is the count of the kth working weekend span of the nurse (n).

SC9. The maximum number of weekends with at least one shift assigned to nurse in four weeks is as equation 3.16:

$$\sum_{k=1}^{\Xi_n} \max\left(\left(\upsilon_n^k - \omega_n^{ub}\right), 0\right), \forall n \epsilon N, \tag{3.16}$$

where υ_n^k is the number of working days at the kth weekend of nurse (n), ω_n^{ub} is the maximum number of working days for nurse (n), and Ξ_n is the total count of the weekend in the scheduling period of nurse (n).

SC10. The nurse can request working on a particular day for the given scheduled period as in equation 3.17.

$$\sum_{d=1}^{D} \lambda_n^d = 1, \forall n \epsilon N, \tag{3.17}$$

where λ_n^d is the day request from the nurse (n) to work on any shift on a particular day (d).

SC11. The nurse can request that they do not work on a particular day for the given scheduled period as in equation 3.18.

$$\sum_{d=1}^{D} \lambda_n^d = 0, \forall n \epsilon N, \tag{3.18}$$

where λ_n^d is the request from the nurse not to work on any shift on a particular day .

SC12. The nurse can request working on a particular shift on a particular day for the given scheduled period as in equation 3.19.

$$\sum_{d=1}^{D} \sum_{s=1}^{S} \Upsilon_n^{d,s} = 1, \forall n \epsilon N, \tag{3.19}$$

where $\Upsilon_n^{d,s}$ is the shift request from the nurse (n) to work on a particular shift on particular day (d).

SC13. The nurse can request that they do not work on a particular shift on a particular day for the given scheduled period as in equation 3.20.

$$\sum_{d=1}^{D} \sum_{s=1}^{S} \Upsilon_n^{d,s} = 0, \forall n \epsilon N, \tag{3.20}$$

where $\Upsilon_n^{d,s}$ is the shift request from the nurse (n) not to work on a particular shift on particular day (d).

SC14. The nurse should not work on unwanted pattern suggested for the scheduled period as in equation 3.21.

$$\sum_{u=1}^{\varrho_n} \mu_n^u, \forall n \epsilon N, \tag{3.21}$$

where μ_n^u is the total count of occurring patterns for nurse (n) of type u; ϱ_n is the set of unwanted patterns suggested for the nurse (n).

The objective function of the NRP is to maximize the nurse preferences and minimize the penalty cost from violations of soft constraints in equation 3.22.

$$\min f(\varsigma_{n,d,s}) = \sum_{SC=1}^{14} P_{SC}\left(\sum_{n=1}^{N}\sum_{s=1}^{S}\sum_{d=1}^{D}\varsigma_{n,d,s}\right) * T_{SC}\left(\sum_{n=1}^{N}\sum_{s=1}^{S}\sum_{d=1}^{D}\varsigma_{n,d,s}\right). \tag{3.22}$$

Here SC refers to the set of soft constraints indexed in 3.2, $P_{SC}(x)$ refers to the penalty weight violation of the soft constraint, and $T_{SC}(x)$ refers to the total violations of the soft constraints in roster solution. It has to be noted that the usage of penalty function in the NRP is to improve the performance and provide the fair comparison with another optimization algorithm.

3.1.3 Algorithm Design

Swarm intelligence is an emerging discipline for the study of problems which requires an optimal approach rather than the traditional approach. The use of swarm intelligence is the part of artificial intelligence based on the study of the behavior of social insects [43]. The swarm intelligence is composed of many individual actions using decentralized and self-organized system. Swarm behavior is characterized by natural behavior of many species such as fish schools, herds of animals, and flocks of birds formed for the biological requirements to stay together. Swarm implies the aggregation of animals such as birds, fishes, ants, and bees based on the collective behavior [11]. The individual agents in the swarm will have a stochastic behavior which depends on the local perception of the neighborhood. The communication between any insects can be formed with the help of the colonies, and it promotes collective intelligence among the colonies.

The important features of swarms are proximity, quality, response variability, stability, and adaptability. The proximity of the swarm must be capable of providing simple space and time computations, and it should respond to the quality factors. The swarm should allow diverse activities and should not be restricted among narrow channels. The swarm should maintain the stability nature and should not fluctuate based on the behavior [12]. The adaptability of the swarm must be able to change the behavior mode when required. Several hundreds of bees from the swarm work together to find nesting sites and select the

best nest site. Bee Colony Optimization is inspired by the natural behavior of bees. The bee optimization algorithm is inspired by group decision-making processes of honey bees. A honey bee searches the best nest site by considering speed and accuracy.

In a bee colony there are three different types of bees, a single queen bee, thousands of male drone bees, and thousands of worker bees.

1. The queen bee is responsible for creating new colonies by laying eggs.
2. The male drone bees mated with the queen and were discarded from the colonies.
3. The remaining female bees in the hive are called worker bees, and they are called the building block of the hive. The responsibilities of the worker bees are to feed, guard, and maintain the honey bee comb.

Based on the responsibility, worker bees are classified as scout bees and forager bees. A scout bee flies in search of food sources randomly and returns when the energy gets exhausted. After reaching a hive scout bees share the information and start to explore rich food source locations with forager bees. The scout bee's information includes direction, quality, quantity, and distance of the food source they found. The way of communicating information about a food source to foragers is done using dance. There are two types of dance, round dance and waggle dance [44]. The round dance will provide direction of the food source when the distance is small. The waggle dance indicates the position and the direction of the food source; the distance can be measured by the speed of the dance. A greater speed indicates a smaller distance; and the quantity of the food depends on the wriggling of the bee. The exchange of information among hive mates is to acquire collective knowledge. Forager bees will silently observe the behavior of scout bee to acquire knowledge about the directions and information of the food source.

The group decision process of honey bees is for searching best food source and nest site. The decision-making process is based on the swarming process of the honey bee. Swarming is the process in which the queen bee and half of the worker bees will leave their hive to explore a new colony [45]. The remaining worker bees and daughter bee will remain in the

old hive to monitor the waggle dance. After leaving their parental hive, swarm bees will form a cluster in search of the new nest site. The waggle dance is used to communicate with quiescent bees, which are inactive in the colony. This provides precise information about the direction of the flower patch based on its quality and energy level. The number of follower bees increases based on the quality of the food source and allows the colony to gather food quickly and efficiently [46]. The decision-making process can be done in two methods by swarm bees to find the best nest site. They are consensus and quorum; consensus is the group agreement taken into account and quorum is the decision process taken when the bee vote reaches a threshold value.

Bee Colony Optimization (BCO) algorithm is a population-based algorithm. The bees in the population are artificial bees, and each bee finds its neighboring solution from the current path [47]. This algorithm has a forward and backward process. In forwarding pass, every bee starts to explore the neighborhood of its current solution and enables constructive and improving moves. In forward pass, entire bees in the hive will start the constructive move and then local search will start. In backward pass, bees share the objective value obtained in the forward pass. The bees with higher priority are used to discard all non improving moves. The bees will continue to explore in next forward pass or continue the same process with neighborhood [48]. The flowchart for BCO is shown in figure 3.2 . The BCO is proficient in solving combinatorial optimization problems by creating colonies of the multiagent system. The pseudocode for BCO is described in figure 3.3. The bee colony system provides a standard well-organized and well-coordinated teamwork, multitasking performance.

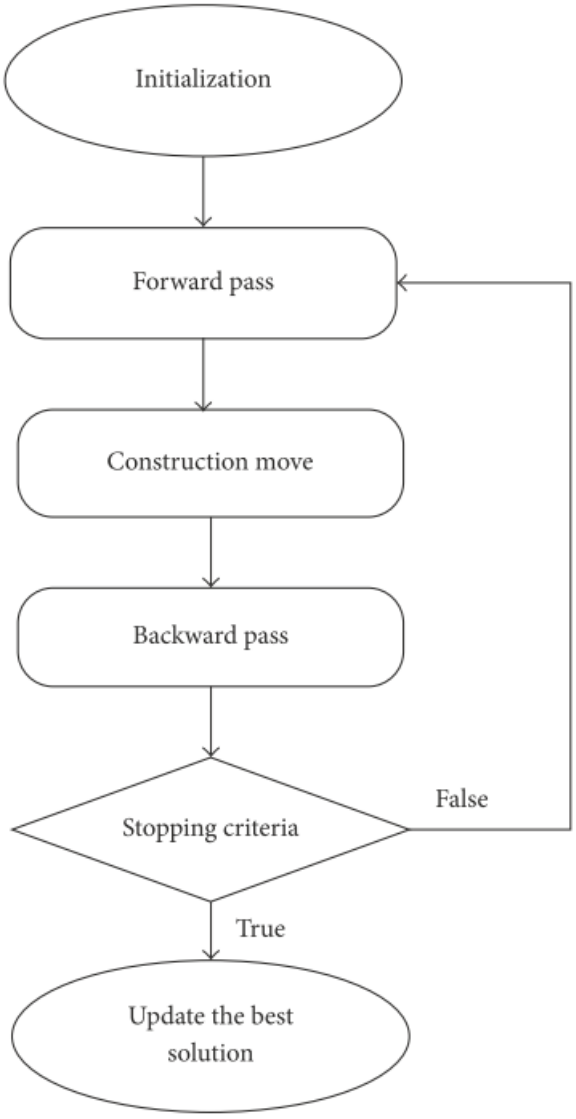

Figure 3.2: Flowchart of BCO algorithm.

Bee Colony Optimization
(1) Initialization: Assign every bee to an empty solution.
(2) Forward Pass
For every bee
(2.1) set $i = 1$;
(2.2) Evaluate all possible construction moves.
(2.3) Based on the evaluation, choose one move using Roulette Wheel.
(2.4) $i = i + 1$ if $(i \leq N)$ Go to step (2.2)
where i is the counter for construction move and N is the number of construction moves during one forward pass.
(3) Return to Hive.
(4) Backward Pass starts.
(5) Compute the objective function for each bee and sort accordingly.
(6) Calculate probability or logical reasoning to continue with the computed solution and become recruiter bee.
(7) For every follower, choose the new solution from recruiters.
(8) If stopping criteria is not met Go to step (2).
(9) Evaluate and find the best solution.
(10) Output the best solution.

Figure 3.3: Pseudocode of BCO.

3.2 Model Proposal

In this section, the concepts of Bee Colony Optimization (BCO) algorithm as adapted for the NRP is discussed. The adapted BCO algorithm involves changing its continuous nature with integration of different neighbourbood structures in order to cope with the solution search space of NRP.

The nurse roster (i.e. solution) is represented as a vector of allocations $x = (x_1, x_2, ..., x_N)$ where each allocation contains three values (nurse, day, shift). For instance, let x = (1, 1, 1); (1, 2, 3),..., (5, 5, 4) be a feasible nurse roster. The roster is interpreted by ABC algorithm as follows: the allocation $x_1 = (1, 1, 1)$ means nurse n1 is assigned to shift s1 at day d1. The second allocation $x_2 = (0, 1, 2)$ means to nurse n_0 assigned to shift s_2 at day d_1, and so on. Note that representation of this roster is adopted in [25]. The description of six main procedural steps of ABC as algorithm adapted for tackling NRP are given as follows:

1) Initialization of BCO and INRC2010 parameters: This step involves initialization of the three control parameters of adapted ABC that are needed for tackling the NRP: solution number (SN) which is the number of food sources in the population and similar to

the population size in GA; maximum cycle number (MCN) which represents the maximum number of iterations; and limit that is responsible for the abandonment of solution, if there is no improvement for certain number of iterations and basically use in diversifying the search. Similarly, the NRP parameters that are drawn from the INRC2010 dataset are also initialized. They are the set of nurses, the set of skill categories, the set of shift types, the scheduling period, the set of work contracts, matrix of weekly nurse demand, matrices of nurses preferences, and eventually the set of unwanted patterns. The job specification which includes: total number of shifts, minimum number of shifts, maximum number of consecutive working days, minimum number of consecutive working days, maximum number of consecutive free days, minimum number of consecutive free days, and maximum working weekend in four weeks.

2) Initialization of the Food Source Memory: The food source memory (FSM) is a memory allocation that consists of sets of feasible food source (i.e. rosters) which is determined by SN as shown in 3.23. In this step, the feasible rosters are generated using the heuristic ordering approach and stored in ascending order in FSM according to the objective cost values that is $f(x_1), f(x_2), ..., f(x_{SN})$. The function of heuristic ordering is to sorts the daily shifts in ascending order based on the level of difficulty. It noteworthy that the lowest weekly nurses demand is the most difficulty and thus, the required nurses of the ordered shifts will be scheduled starting with the most difficult and ending with less difficult one.

$$\mathbf{FSM} = \begin{bmatrix} X_1(1) & X_1(2) & \cdots & X_1(N) \\ X_2(1) & X_2(2) & \cdots & X_2(N) \\ \vdots & \vdots & \ddots & \vdots \\ X_{SN}(1) & X_{SN}(2) & \cdots & X_{SN}(N) \end{bmatrix} \begin{bmatrix} f(X_1) \\ f(X_2) \\ \vdots \\ f(X_{SN}) \end{bmatrix} \tag{3.23}$$

3) Send the Employed Bee to the Food Sources: in this step, the employed bee operator selects feasible nurse rosters sequentially from the FSM and exploits each roster using the neighbourhood structures to produce a new set of neighbouring solutions. The neighbourhood structures utilized by employed bee are:

- Move Neighbourhood Structure (MNS): The nurse of chosen allocation x_j is replaced with another nurse selected randomly to solve the violations of the soft constraint.
- Swap Neighbourhood Structure (SNS): The shift of selected allocation x_j is swapped with another shift on the same day for another selected allocation x_k.
- Swap Unwanted Pattern (SUP): This exchange a group of shifts among two nurses in which the chosen allocation x_j is replaced with another group of shifts on the same day for another chosen allocation x_k.
- Token Ring Move (TRM). The nurse of chosen allocation x_j is replaced by another nurse selected randomly, if the soft constraint S7 is violated. Furthermore, the shift of a selected allocation x_j will be exchanged with another shift on which another nurse is working on the same day, for another selected allocation x_k to solve the violation of the soft constraint S8.

The fitness of each new roster is calculated. If it is better than that of candidate roster (i.e. food source), then it replaces the parent roster in FSM. This process is implemented for all solutions. The detailed of this process is given in Figure 3.4.

4) Send the Onlooker Bees to the Food Sources: Subsequent to the completion of employed bees exploitation process, the employed bees share the information of exploited food source (i.e. roster) with onlooker bees. The onlooker bees decide to follow certain employed bees and exploit their corresponding food sources randomly using the set of neighbourhood structures discussed above based on proportional selection probability as shown in 3.24.

$$P_j = \frac{f(X_j)}{\sum_{k=1}^{SN} f(X_k)} \tag{3.24}$$

```
for i = 1 ··· SN do
  i = RND() {RND generates a random integer number
  in range 1 - 4}
  if i = 1 then
    x^{i(new)} = MNS(x^i)
  else
    if i = 2 then
      x^{i(new)} = SNS(x^i)
    else
      if i = 3 then
        x^{i(new)} = SUP(x^i)
      else
        if i = 4 then
          x^{i(new)} = TRM(x^i)
        end if
      end if
    end if
  end if
  if x^{i(new)} is better than x^i then
    x^i = x^{i(new)}
  end if
  next i
end for
```

Figure 3.4: Employed Bee Phase.

Thus, the roster with higher selection probability may be selected and adjusted to its neighbourhood using the same strategy as the employed bee. The fitness of the new roster is calculated and if it is better, then it replaces the current one.

5) Send the Scout to Search for Possible New Food Sources: Owing to continuous exploitation, some food sources may finally be exhausted in which they might be abandoned by its corresponding employed bee. Thus, the associated employed bee turns to a scout bee, and explores the solution search space randomly for a possible new food source to replace the abandoned one. Memorize the fitness of the best food source found so far in FSM.

6) Stopping condition: Repeat steps 3-5 until a stop condition is achieved, which is originally determined by MCN.

3.3 Experimental Setup

The performance of the proposed Bee Colony Optimization (BCO) algorithm for NRP is evaluated using INRC2010 dataset. The Nurse Rostering Problem datasets are taken from the First International Rostering Competition (INRC2010) by PATAT-2010, a leading conference in Automated Timetabling. The INRC2010 dataset is divided based on its complexity and size into three tracks, namely, sprint, medium, and long datasets [19]. Each track is divided into four types as early, late, hidden, and hint with reference to the competition INRC2010. The first track sprint is the easiest and consists of 10 nurses, 33 datasets which are sorted as 10 early types, 10 late types, 10 hidden types, and 3 hint type datasets. The scheduling period is for 28 days with 3 to 4 contract types, 3 to 4 daily shifts, and one skill specification. The second track is a medium which is more complex than sprint track, and it consists of 30 nurses, 18 datasets which are sorted as 5 early types, 5 long types, 5 hidden types, and 3 hint types. The scheduling period is for 28 days with 3 to 4 contract types, 4 to 5 daily shifts, and 1 to 2 skill specifications. The most complicated track is long with 49 to 40 nurses and consists of 18 datasets which are sorted as 5 early types, 5 long types, 5 hidden types, and 3 hint types. The scheduling period for this track is 28 days with 3 to 4 contract types, 5 daily shifts, and 2 skill specifications.

For this work, the medium track instances with a dataset of thirty nurses is used to compute the algorithm. According to Awadallah et al. [23] the medium track is complex but not to complicated to compute because the number of nurses is neither too large for the algorithm to take too long to provide a solution nor too small for inconclusive results to be obtained. Table 3.3 describes the detailed description of the datasets; columns one to three are used to index the dataset to track, type, and instance. Columns four to seven will explain the number of available nurses, skill specifications, daily shift types, and contracts. Column eight explains the number of unwanted shift patterns in the roster. The nurse preferences are managed by shift off and day off in columns nine and ten. The number of weekend days is shown in column eleven. The last column indicates the scheduling period. The symbol “x” shows there is no shift off and day off with the corresponding datasets. Table 3.4 shows the list of medium datasets used in the experiment, and it is classified

based on its type.

Track	Type	Instance	Nurses	Skills	Shifts	Contracts	Unwanted pattern	Shift off	Day off	Weekend	Time period
Medium	Early	01-05	30	1	4	4	0	✓	✓	2	1-01-2010 to 28-01-2010
	Hidden	01-04	30	2	5	4	9	x	x	2	1-06-2010 to 28-06-2010
		05	30	2	5	4	9	x	x	2	1-06-2010 to 28-06-2010
		01	30	1	4	4	7	✓	✓	2	1-01-2010 to 28-01-2010
	Late	02, 04	30	1	4	3	7	✓	✓	2	1-01-2010 to 28-01-2010
		03	30	1	4	4	0	✓	✓	2	1-01-2010 to 28-01-2010
		05	30	2	5	4	7	✓	✓	2	1-01-2010 to 28-01-2010
	Hint	01, 03	30	1	4	4	7	✓	✓	2	1-01-2010 to 28-01-2010
		02	30	1	4	4	7	✓	✓	2	1-01-2010 to 28-01-2010

Table 3.3: The features of the INRC2010 medium track datasets.

Case	Track	Type
Case 1	Medium	Early
Case 2		Hidden
Case 3		Late
Case 4		Hint

Table 3.4: Classification of INRC2010 medium datasets based on the type.

The algorithm is coded with Python on Windows 10 platform on 11th Gen Intel Core i9-11900H @ 2.50GHz processor with 16 GB of RAM. The list of competitor methods chosen to evaluate the performance of the proposed algorithm is shown in Table 3.5.

ID	Method	Reference
M1	Hyper-heuristic combined with a greedy shuffle approach.	[20]
M2	Variable Depth Search Algorithm and Branch and Price Algorithm.	[17]
M3	Tabu search with restart mechanism.	[19]
M4	Constraint Optimization Problem solver.	[18]
M5	Integer programming with set of neighborhood structures.	[16]

Table 3.5: INRC2010 winners' methods.

Given the multi-stage nature of the overall process, the input data of the problem comes from three different sources, called scenario, week data, and history, as explained in the following subsections using a dataset of five nurses as example.

3.3.1 Scenario

The scenario represents the general data common to all stages of the overall process. It contains the following information:

The first line of the scenario file contains the name of the dataset in the format nXXXwY, where XXX is the number of nurses and Y the number of weeks of the planning horizon. This is the identifier of the scenario that is subsequently used in the relating history and week data.

SCENARIO = n005w4

Then it is reported the length of the planning horizon, expressed in number of weeks, and the number and names of skills for nurses.

WEEKS = 4

SKILLS = 2
HeadNurse
Nurse

The shift types section indicates the number of shift types available, and for each one, the identifier (its name), and the minimum and the maximum number of consecutive assignments allowed. For each shift type, it is also detailed the forbidden shift types sequences as <preceding_shift_ type> <number_forbidden_successions> <succeeding_shift_type_list>. In the following example, the successions Late → Early, Night → Early and Night → Late are forbidden.

SHIFT_TYPES = 3
Early (2,5)
Late (2,3)
Night (4,5)

FORBIDDEN_SHIFT_TYPES_SUCCESSIONS
Early 0

Late 1 Early
Night 2 Early Late

In the contract section it is listed the name of the contract type, and the lower and upper limits on working and rest days. In detail, it establishes the minimum and the maximum number of total assignments in the planning horizon, the minimum and the maximum number of consecutive working days, the minimum and the maximum number of consecutive days off, the maximum number of working weekends, and the presence (1) or absence (0) of the complete weekend constraint.

CONTRACTS = 2
FullTime (15,22) (3,5) (2,3) 2 1
PartTime (7,11) (3,5) (3,5) 2 1

Finally, the nurse section reports the total number of nurses available, and for each nurse his/her identifier (the name), the contract type, the number of skills owned and their names.

NURSES = 5
Patrick FullTime 2 HeadNurse Nurse
Andrea FullTime 2 HeadNurse Nurse
Stefaan PartTime 2 HeadNurse Nurse
Sara PartTime 1 Nurse
Nguyen FullTime 1 Nurse

3.3.2 Week data

The week data contains the specific data for the single week. It consists of the following information:

In the week data file, first of all there is the identifier of the corresponding scenario; then all the data about coverage requirements and nurse preferences is listed.

WEEK_DATA
n005w4

A coverage requirement is specified by the shift type, the skill, and for each day of the week (from Monday to Sunday), the minimum coverage and the optimal coverage.

REQUIREMENTS
Early HeadNurse (1,1) (0,0) (0,0) (0,0) (0,0) (1,1) (0,0)
Early Nurse (1,2) (1,1) (1,1) (0,1) (1,1) (1,1) (0,1)
Late HeadNurse (1,1) (0,1) (1,1) (0,0) (0,0) (0,0) (0,0)
Late Nurse (1,1) (1,1) (0,1) (0,1) (1,1) (1,1) (1,1)
Night HeadNurse (0,0) (1,1) (0,0) (0,0) (1,1) (1,1) (0,0)
Night Nurse (0,1) (1,1) (1,1) (1,1) (1,1) (0,1) (1,1)

Finally, the number of shift off requests is reported with the following grammar: <nurse> <shift type> <day>. The special shift type Any means that the nurse would like to have a day off.

SHIFT_OFF_REQUESTS = 3
Sara Any Thu
Sara Night Sat
Stefaan Late Sat

3.3.3 History

The history contains the information that must be carried over from one week to the following one so as to evaluate the constraints correctly. In detail, it reports for each nurse the following information:

The first line of the history file describes the week to which the history refers to (i.e. 0 for the initial history file, 1 after the first week, . . .) and the relating scenario file.

HISTORY
0 n005w4

In addition, the file contains the nurse history, in terms of total number of assignments, total number of worked weekends, last assigned shift type, number of consecutive assignments of the last shift type, number of consecutive worked days and number of consecutive days off.

NURSE_HISTORY
Patrick 0 0 Night 1 4 0
Andrea 0 0 Early 3 3 0
Stefaan 0 0 None 0 0 3
Sara 0 0 Late 1 4 0
Nguyen 0 0 None 0 0 1

3.4 Code

In this section the implemented code is explained in detail. First we install the required libraries and packages for the algorithm to work properly as shown in figure 3.5.

```
from ortools.sat.python import cp_model
import pandas as pd
import numpy as np
```

Figure 3.5: Libraries and packages.

Then, we choose the input files for the algorithm as shown in figure 3.6. There are one scenario file, three history files and ten week data files to choose from. The scenario is the same for all the instances, so the history and week data files will determine the initial conditions for the algorithm.

```
Scenario = input("Nombre del archivo de escenario: ")
historia = input("Nombre del archivo de historia: ")
semana = input("Nombre del archivo de escenario: ")
```

Figure 3.6: Input files.

Then, we create the variable contador_nombres to save the amount of nurses, the dictionary dic_nomb_enferm to save the names of the nurses, the variable contador_turnos to save amount of shifts and the dictionary dic_turnos to save the shifts that are going to be used as shown in figure 3.7.

Later, we open the scenario file to read the nurses, the shift types and the amount of weeks for the problem as shown in figure 3.7.

```
contador_nombres=0
dic_nomb_enferm={}
contador_turnos=0
dic_turnos={}

with open("./n005w4/" + Scenario) as archivo:
    for linea in archivo:
      if "SCENARIO =" in linea: escenario = linea [linea.find('=')+1:-1]
      if "NURSES =" in linea:
        num_nurses=int(linea[linea.find('=')+1:linea.find('n')])
        contador_nombres+=1
      if contador_nombres>=1 and (linea.split())[0]!='NURSES':
        nombre_enfermera=(linea.split())[0]
        dic_nomb_enferm[contador_nombres]=nombre_enfermera
        contador_nombres+=1
      if "SHIFT_TYPES =" in linea:
        num_shifts=int(linea[linea.find('=')+1:linea.find('n')])
        contador_turnos+=1
      if contador_turnos>=1 and contador_turnos<=num_shifts :
        if len (linea.split())==2:
          tipo_turno=(linea.split())[0]
          min_max_turnos=(linea.split())[1]
          dic_turnos[contador_turnos]=[tipo_turno,min_max_turnos]
          contador_turnos+=1
      if "WEEKS =" in linea: num_days=7*int(linea[linea.find('=')+1:linea.find('n')])
```

Figure 3.7: Create variables and open the scenario file.

And then, with the subtracted information from the scenario file we show the name of

the scenario file, the number of nurses, the number of shifts, the number of days, the names of the nurses and the shift types with their respective minimum and maximum ranges as shown in figure 3.8.

```
print('***  DATOS GENERALES DEL PROBLEMA ***')
print(' ')
print('El Escenario es:', escenario)
print('El numero de Enfermeras es:', num_nurses)
print('El numero de Turnos es:', num_shifts)
print('El numero de dias es:', num_days)
print('Los nombres de las enfermeras son:',dic_nomb_enferm)
print('Los tipos de turnos y sus rangos maximos y minimos son:',dic_turnos)
```

Figure 3.8: Show the scenario file data.

Then, we open the history file to know the number of previous consecutive days worked and the number of previous consecutive days not workedas shown in figure 3.9.

```
contador_historico=0
with open("./n005w4/" + historia) as archivo:
    for linea in archivo:
      if 'NURSE_HISTORY' in linea:
        contador_historico+=1
      if contador_historico>=1 and (linea.split())[0]!='NURSE_HISTORY':
        num_dias_trabajados= (linea.split())[-2]
        num_dias_NO_trabajados= (linea.split())[-1]
        dic_nomb_enferm[contador_historico]=[dic_nomb_enferm[contador_historico],num_dias_trabajados,num_dias_NO_trabajados]
        contador_historico+=1

for enf in dic_nomb_enferm.values():
  print('La cantidad de dias consecutivos trabajados previos de ',enf[0],' fueron:', enf[1])
  print('La cantidad de dias consecutivos NO trabajados previos de ',enf[0],' fueron:', enf[2])
```

Figure 3.9: Open the history file.

Then, we again open the scenario file to read the nurses contracts information. At the end we show the maximum values allowed for consecutive days not worked and the types of contracts according to each nurse as shown in figure 3.10.

```
contador_nombres=0
contador_contrato=0
dic_contrato={}
dic_tipo_contrato={}

with open("./n005w4/" + Scenario) as archivo:
    for linea in archivo:
      if "NURSES =" in linea:
        num_nurses=int(linea[linea.find('=')+1:linea.find('n')])
        contador_nombres+=1
      if contador_nombres>=1 and (linea.split())[0]!='NURSES':
        contrato=(linea.split())[1]
        dic_tipo_contrato[contador_nombres]=contrato
        contador_nombres+=1
      if "CONTRACTS =" in linea:
        contador_contrato+=1
      try:
        if contador_contrato>=1 and (linea.split())[0]!='CONTRACTS':
          limites_contrato=(linea.split())[2]
          maximo_limite_NO=limites_contrato[3]
          dic_contrato[(linea.split())[0]]=[maximo_limite_NO]
          contador_contrato+=1
      except:
        pass
print('Los valores maximos permitidos para dias consecutivos no trabajados son:',dic_contrato)
print('Los tipos de contrato segun cada enfermera son:',dic_tipo_contrato)
```

Figure 3.10: Open the scenario file and show the maximum values.

Then, we read the week data file to know how many nurses the problem requires for each day of the week as shown in figure 3.11.

```
contador_req=0
with open("./n005w4/" + semana) as archivo:
    for linea in archivo:
      if 'REQUIREMENTS' in linea:
        contador_req+=1
      if contador_req>=1 and contador_req<=6:
        if (linea.split())[0]!='REQUIREMENTS':
          contador_req+=1
```

Figure 3.11: Read the week data file.

Then, we preprocess the data before being entered into the model. We create the range, that are immutable lists of n consecutive integers that start at 0 and end at the value of the argument - 1. And we print each one of the vectors as shown in figure 3.12.

```
all_nurses = range(num_nurses)
all_shifts = range(num_shifts)
all_days = range(num_days)
print('La secuencia de enfermeras es asi:',list(all_nurses))
print('La secuencia de turnos es asi:',list(all_shifts))
print('La secuencia de dias es asi:',list(all_days))
```

Figure 3.12: Preprocess the data and print the vectors.

Then, we create the model using CpModel from the ortools library. The matrix defines shift assignments to nurses as follows: shifts[(n, d, s)] equals 1 if shift s is assigned to nurse n on day d, and 0 otherwise. Later we create the initial restrictions of the problem, which are: each nurse is assigned to exactly one shift at a time and each nurse works at most one shift per day. Finally, we distribute the shifts evenly, so that each nurse works the same using the variable min_shifts_per_nurse as shown in figure 3.13.

```
model = cp_model.CpModel()

shifts = {}
for n in all_nurses:
    for d in all_days:
        for s in all_shifts:
            shifts[(n, d, s)] = model.NewBoolVar('shift_n%id%is%i' % (n, d, s))
for d in all_days:
    for s in all_shifts:
        model.AddExactlyOne(shifts[(n, d, s)] for n in all_nurses)
for n in all_nurses:
    for d in all_days:
        model.AddAtMostOne(shifts[(n, d, s)] for s in all_shifts)


print('La cantidad de turnos totales enteros es', num_shifts * num_days)
min_shifts_per_nurse = (num_shifts * num_days) // num_nurses
print('El numero minimo de turnos de cada enfermera entero es ' , min_shifts_per_nurse)
```

Figure 3.13: Create the model.

Then, a validation is performed to ensure the balance of shifts when it is not possible for the displayed product to be divisible by the number of nurses, in those cases it is appropriate to assign one more shift. Then, we implement a loop where the number of shifts worked is saved for each nurse and we aggregate this restriction to the model where the number of shifts worked must be less than the maximum number of shifts worked per

nurse. Finally, we apply a solver to the model where we obtain an initial solution with the initial restrictions of the problem as shown in figure 3.14.

```
if num_shifts * num_days % num_nurses == 0:
    max_shifts_per_nurse = min_shifts_per_nurse
else:
    max_shifts_per_nurse = min_shifts_per_nurse + 1

print(' La cantidad maxima de turnos por enfermera es de ',(max_shifts_per_nurse))

for n in all_nurses:
    num_shifts_worked = []
    for d in all_days:
        for s in all_shifts:
            num_shifts_worked.append(shifts[(n, d, s)])

    model.Add(min_shifts_per_nurse <= sum(num_shifts_worked))
    model.Add(sum(num_shifts_worked) <= max_shifts_per_nurse)


solver = cp_model.CpSolver()
solver.parameters.linearization_level = 0
solver.parameters.enumerate_all_solutions = True
```

Figure 3.14: Application of a solver to the model.

Then, we create a function to replace the name of the nurse with the index in the solution as shown in figure 3.15.

```
def reemplazar_nombre(t):
  index_nurse=int(t[t.find('rse')+3:t.find('rse')+5])
  sustituir=dic_nomb_enferm[index_nurse+1][0]
  palabra_reemplazar=' Nurse ' + str(index_nurse)
  t=t.replace(palabra_reemplazar,sustituir)
  return t
```

Figure 3.15: Function that replace the name of the nurse with the index in the solution.

Then, we create a class that inherits the characteristics of the model. First, we create a constructor to save the information obtained before. Then, the function on_solution_callback first goes through the values of the days and for each day it creates a list of shifts, after all

the values of the nurses are gone through and if the nurse did not work then the below is executed and if the value From that number of nurses it is found that they worked, that nurse is saved and the name is replaced inside the object called shifts. Then, if the value of work is false, then it takes the name of the nurses and is told that this nurse did not work. After the replace name function is used, the index is replaced by the name and it is saved within the turns. Afterwards, a validation is carried out as to whether the number of solutions is equal to the number of limit solutions and if that happens then it stops searching and that is the optimal partial solution. Then it saves the results in the full dictionary and finally creates a function where it returns the value of solutions that have been found as shown in figure 3.16.

```python
class NursesPartialSolutionPrinter(cp_model.CpSolverSolutionCallback):
        global dic,dic_total
        dic_total={}

        def __init__(self, shifts, num_nurses, num_days, num_shifts, limit):
            cp_model.CpSolverSolutionCallback.__init__(self)
            self._shifts = shifts
            self._num_nurses = num_nurses
            self._num_days = num_days
            self._num_shifts = num_shifts
            self._solution_count = 0
            self._solution_limit = limit

        def on_solution_callback(self):
            global dic, turnos_, dic_total
            dic={}
            self._solution_count += 1
            for d in range(self._num_days):
                turnos_=[]
                for n in range(self._num_nurses):
                    is_working = False
                    for s in range(self._num_shifts):
                        if self.Value(self._shifts[(n, d, s)]):
                            is_working = True
                            texto_nurse='  Nurse %i works shift %i' % (n, s)
                            t=reemplazar_nombre(texto_nurse)
                            turnos_.append(t)
                    if not is_working:
                        texto_nurse='  Nurse {} does not work'.format(n)
                        t=reemplazar_nombre(texto_nurse)
                        turnos_.append(t)
                dic[d]=turnos_
            if self._solution_count >= self._solution_limit:
                self.StopSearch()
            dic_total[self._solution_count]=dic
        def solution_count(self):
            return self._solution_count
```

Figure 3.16: Class that inherits the characteristics of the model.

Then we choose the solution limit and apply the solver to the model as shown in figure 3.17.

```
solution_limit = 10
solution_printer = NursesPartialSolutionPrinter(shifts, num_nurses,
                                                num_days, num_shifts,
                                                solution_limit)

solver.Solve(model, solution_printer)
```

Figure 3.17: Choose the solution limit and apply the solver to the model.

Then, we create a function to find repeated series as shown in figure 3.18.

```
def count_dups(nums):
    element = []
    freque = []
    if not nums:
        return element
    running_count = 1
    for i in range(len(nums)-1):
        if nums[i] == nums[i+1]:
            running_count += 1
        else:
            freque.append(running_count)
            element.append(nums[i])
            running_count = 1
    freque.append(running_count)
    element.append(nums[i+1])
    return element,freque
```

Figure 3.18: Function that finds repeated series.

Then, we create a function to know the maximum consecutive number of days not worked and then we create a table with the solution by columns to verify the minimum and maximum number of consecutive assignments allowed as shown in figure 3.19.

```
def maximo_NO_trabajos(elementos, frecuencia):
  lista_work=[]
  for i,ele in enumerate(elementos):
    if 'work' in ele:
      lista_work.append(frecuencia[i])
  maximo=max(lista_work)
  return maximo

columnas=[] ; filas=[]
for nomb in dic_nomb_enferm.keys():
  columnas.append(str(dic_nomb_enferm[nomb][0]))
for f in range(num_days):
  filas.append(str(f))
```

Figure 3.19: Function and table.

This section is another validation where all the solutions obtained from the model are traversed and it is saved in a dataframe. Then an iterator is performed that says that for each object in the number of solutions an empty list is created, then the nurses in the nurses dictionary are traversed and the name of that nurse is saved and stored in the list called turnitos. If an error is found, the index of the solution is taken and within the scenario dataframe that value is eliminated and then the value of the column is initialized with the value of 0 and another loop is run. Then, for each column in the scenario, another try is made to capture errors and likewise it starts with a counter of 1 and within the list each of the elements of each of the columns that are obtained in the dataframe is transformed into a list. of the solution. Then the function is decompressed to find repeated series that returns a pair where it will be equal to the elements and frequencies within that list. Then an object called max_nro_trabajados is created, which is equal to the maximum number of days that were not worked and likewise the maximum value allowed is stored in the dictionary of contracts where the name of the contract that matches the number of the worker is taken. Then a conditional is performed where it transforms the maximum allowed value and if that is less than the maximum value of workers then it means that the solution does not meet the main restrictions and throws an error as shown in figure 3.20.

```
for sol in dic_total.items():
  for num_sol in sol:
    df_escenario = pd.DataFrame(columns=columnas, index=filas)
    salir_solucion=0
    try:
      for enfermera in num_sol.keys():
          turnitos=[]
          for nurse in (num_sol[enfermera]):
            nombre=nurse.split()[0] ; turno=nurse.split()[-1]
            turnitos.append(turno)
          df_escenario.loc[enfermera] = turnitos
    except:
      indice_solucion=num_sol
    df_escenario=df_escenario.dropna(how='all')
    indice_columna=0
    for col in df_escenario.columns:
      try:
        indice_columna+=1
        lista=list(df_escenario[col])
        (elementos, frecuencia)=count_dups(list(df_escenario[col]))
        MAX_No_trabajados=maximo_NO_trabajos(elementos, frecuencia)
        VALOR_MAX_permitido=dic_contrato[dic_tipo_contrato[indice_columna]][0]
        if int(VALOR_MAX_permitido)<int(MAX_No_trabajados):
          print('La solucion',indice_solucion,' No cumple la restriccion para la enfermera',col )
          salir_solucion=1
          break
      except:
        pass
    if salir_solucion==1:
      break
df_escenario
```

Figure 3.20: Validation.

Then, we create a function to encode the shifts and replace with the names of the shifts as shown in figure 3.21.

```
def reemplazar_turnos(df):
  for col in df.columns:
    for i in df.index:
      if df[col][i] =='work':
        df[col][i]='libre'
      else:
        for key in dic_turnos.keys():
          try:
            if int(key-1)==int(df[col][i]):
              df[col][i]=dic_turnos[key][0]
          except:
            pass
  return df
data=reemplazar_turnos(df_escenario)
data
```

Figure 3.21: Function that encodes the shifts and replace with the names of the shifts.

To extract the number of non-working days, a function is defined that identifies the maximum number of days allowed and will receive a column as an argument. So for each value in the keys of a dictionary, if the column is inside the dictionary then it saves the object of that dictionary in an object called ”tipo” and saves it in an object called ”maximo” which would be the maximum number of days allowed to work as shown in figure 3.22.

```
def identificar_max_DIAS_permitidos(columna):
  for key in dic_nomb_enferm.keys():
   if columna in dic_nomb_enferm[key][0]:
     tipo=(dic_tipo_contrato[key])
     maximo=(int(dic_contrato[tipo][0]))
  return maximo
```

Figure 3.22: Function that identifies the maximum number of days allowed.

This function is to search for the maximum frequency of days that receives as arguments the name of the column the maximum days allowed and the maximum frequency. A number of total rows is created that is equal to the number of rows in the dataframe and each of the indexes is traversed and the row is assigned the value of that row. Then with a ”for” it iterates through each of the rows and if the name of the column and the row is equal to

free then the counter is added 1 and if it is not for the iterator then that means that it is searching observations that have the day off. Then it is obtained with a comparator if the maximum frequency is the same as the free count, then the value of that row is saved in an object called ”index_libre” and the iterator is stopped. If this is not true, the value of the index returns to 0 and returns the value of the index of free days as shown in figure 3.23.

```
def buscar_fila_frecuencia_max(nomb_col,max_permitido,frecuencia_max):
  filas_totales=len(data)
  for i in data.index:
    fila_i=i
    cont_libre=0
    for fila in range(fila_i,filas_totales):
      if data[nomb_col][fila] =='libre':
        cont_libre+=1
      else:
        break
    if frecuencia_max==(cont_libre):
      index_libre=fila_i
      break
    else:
      index_libre=0
  return index_libre
```

Figure 3.23: function that searches for the maximum frequency of days.

Then, we create a function to exchange shifts and guarantee maximum days off allowed as shown in figure 3.24.

```
def intercambiar_turnos(nom_col,indice_reemplazar, max_permitido):
  for col in data.columns:
    if nom_col!=col:
      if (data[col][indice_reemplazar+max_permitido]).lstrip().rstrip()!='libre' and (data[col][indice_reemplazar+max_permitido-1]).lstrip().rstrip()!='libre':
        turno_reemplazar=data[col][indice_reemplazar+max_permitido]
        data[nom_col][indice_reemplazar+max_permitido]=turno_reemplazar
        data[col][indice_reemplazar+max_permitido]='libre'
        break
```

Figure 3.24: Function that exchanges shifts and guarantee maximum days off allowed.

Then an algorithm is established that serves to replace the shifts that do not meet the restrictions of the maximum non-working days, then for each column in the ”data” variable,

the duplicate count is saved in two elements called element and maximum frequency. from the above list of columns, then the maximum allowed is going to be equal to the function you had before you identified the maximum days allowed per column. Then, if the maximum of that matrix is greater than the maximum allowed, it means that the maximum working days are not being met and therefore the maximum number of shifts is assigned where the buscar_fila_frecuencia_max function is used, which is used to search for the maximum number of shifts and likewise it is stored in an object called maximum number of rows and finally the function of exchanging shifts is used to correctly assign the number of non-working days allowed by each of the nurses as shown in figure 3.25.

```
for col in data.columns:
  elem,frecuencia_max=count_dups(list(data[col]))
  max_permitido= identificar_max_DIAS_permitidos(col)
  if np.max(np.array([frecuencia_max]))>max_permitido:
    print('La enfermera ', col, 'No cumple con los dias maximos No laborables que son ',max_permitido)
    fila_max= buscar_fila_frecuencia_max(col, max_permitido,np.max(np.array([frecuencia_max])) )
    intercambiar_turnos(col,fila_max, max_permitido)
```

Figure 3.25: Algorithm that serves to replace the shifts that do not meet the restrictions of the maximum non-working days.

Finally, we show the final solution that meets the restrictions and the respective statistics of the problem. The total cost of the model, the total number of branches, the time it takes to find the solution and the number of solutions found as shown in figure 3.26.

```
print('# La solucion final que cumple las Restricciones es')
print(data)
print()
print('\nStatistics')
print('  - total cost      : %i' % solver.NumConflicts())
print('  - branches        : %i' % solver.NumBranches())
print('  - wall time       : %f s' % solver.WallTime())
print('  - solutions found: %i' % solution_printer.solution_count())
```

Figure 3.26: Final solution.

With regard to the general explanation of the model, what it does is make validations based on the restrictions, then yes or yes, some must be met, which are that each nurse has

to be assigned to a single shift per day and finally that each nurse has to be assigned to a minimum of one shift. From there, if the model does not find an optimal solution where all the restrictions are met, then the model takes the solution that has the least cost. The cost of this model is the minimum number of conflicts that the last solution had, so the less number of conflicts found in the constraints, the better the solution.

Chapter 4

Results

In this section, the effectiveness of the proposed algorithm is compared with other optimization algorithms to solve the NRP using INRC2010 datasets under similar environmental setup, using performance metrics as discussed. The computational analysis on the performance metrics is as follows.

4.1 Best value

The results obtained by BCO with competitive methods are shown in Table 4.1. The performance is compared with previous methods; the number in the table refers to the best solution obtained using the corresponding algorithm. The objective of NRP is the minimization of cost; the lowest values are the best solution attained. In the evaluation of the performance of the algorithm, the authors have considered 18 datasets with medium size. It is apparently shown that the adapted BCO obtained very closed solutions to the optimal values.

Instance	Optimal value	BCO		M1		M2		M3		M4		M5	
		Best	Worst	Best	Worst	Best	Worst	Best	Worst	Best	Worst	Best	Worst
Medium early 01	240	254	270	262	284	247	267	247	262	280	305	250	269
Medium early 02	240	251	267	263	280	247	270	247	263	281	301	250	273
Medium early 03	236	245	261	261	281	244	264	244	262	287	311	245	269
Medium early 04	237	251	273	259	278	242	261	242	258	278	297	247	272
Medium early 05	303	324	339	331	351	310	332	310	329	330	351	313	338
Medium hidden 01	123	155	171	190	210	157	180	157	177	410	429	192	214
Medium hidden 02	243	244	257	286	306	256	277	256	273	412	430	266	284
Medium hidden 03	37	59	75	66	84	56	78	56	69	182	203	61	81
Medium hidden 04	81	98	116	102	119	95	119	95	114	168	191	100	124
Medium hidden 05	130	191	212	202	224	178	202	178	197	520	545	194	214
Medium late 01	157	188	204	207	227	175	198	175	194	234	257	179	194
Medium late 02	18	43	59	53	76	32	52	32	53	49	67	35	55
Medium late 03	29	48	69	71	90	39	60	39	59	59	78	44	66
Medium late 04	35	55	71	66	83	49	57	49	70	71	89	48	70
Medium late 05	107	145	167	179	199	135	156	135	156	272	293	141	164
Medium hint 01	40	50	72	70	90	49	71	49	65	64	82	50	71
Medium hint 02	84	104	116	142	162	95	116	95	115	133	158	96	115
Medium hint 03	129	153	169	188	209	141	161	141	152	187	208	148	166

Table 4.1: Experimental result with respect to best value.

4.2 Error rate

The evaluation based on the error rate shows that our proposed BCO yield lesser error rate compared to other competitor techniques. The computational analysis based on error rate (%) is shown in Table 4.2 and the obtained error rate is 71% for medium sized datasets. A negative value in the column indicates corresponding instances have attained lesser optimum value than specified in the INRC2010. The error rate (%) obtained by using BCO with different algorithms is shown in Figure 4.1.

Case	BCO	M1	M2	M3	M4	M5
Case 1	4.36	9.57	2.73	2.73	16.31	3.93
Case 2	56.21	46.37	27.71	27.71	220.44	40.62
Case 3	37.14	105.40	37.98	37.98	116.36	45.82
Case 4	27.83	63.26	14.97	14.97	54.43	18.00

Table 4.2: Experimental result with respect to error rate.

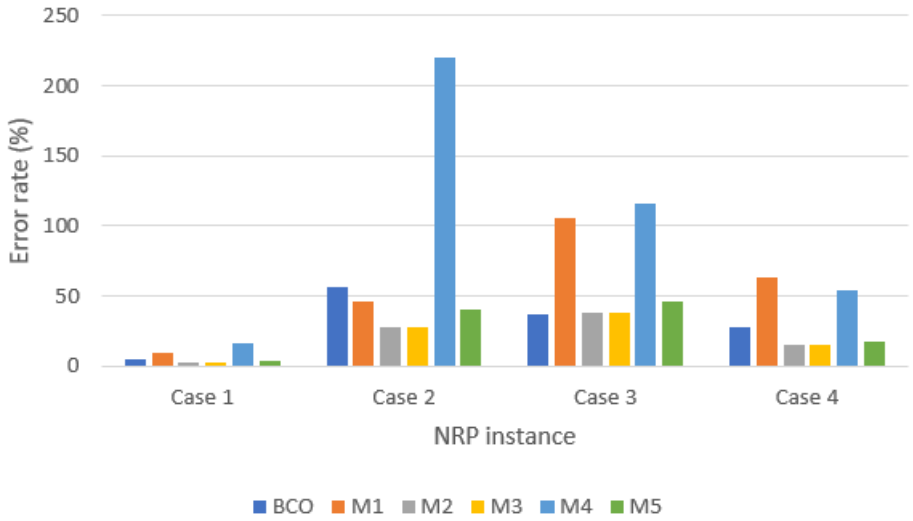

Figure 4.1: Performance analysis with respect to error rate.

4.3 Average convergence

The Average Convergence of the solution is the average fitness of the population to the fitness of the optimal solution. The computational results with respect to Average Convergence are shown in Table 4.3. BCO shows 74% convergence rate in medium size instances. Negative values in the column show the corresponding instances get deviated from optimal solution and trapped in local optima. The Average Convergence rate attained by various optimization algorithms is depicted in Figure 4.2.

Case	BCO	M1	M2	M3	M4	M5
Case 1	104.86	86.49	91.10	92.71	77.29	89.01
Case 2	86.21	45.52	52.87	59.08	-139.09	39.82
Case 3	77.35	-32.73	24.52	26.23	-54.91	9.97
Case 4	98.62	21.72	61.67	68.51	22.95	58.22

Table 4.3: Experimental result with respect to Average Convergence.

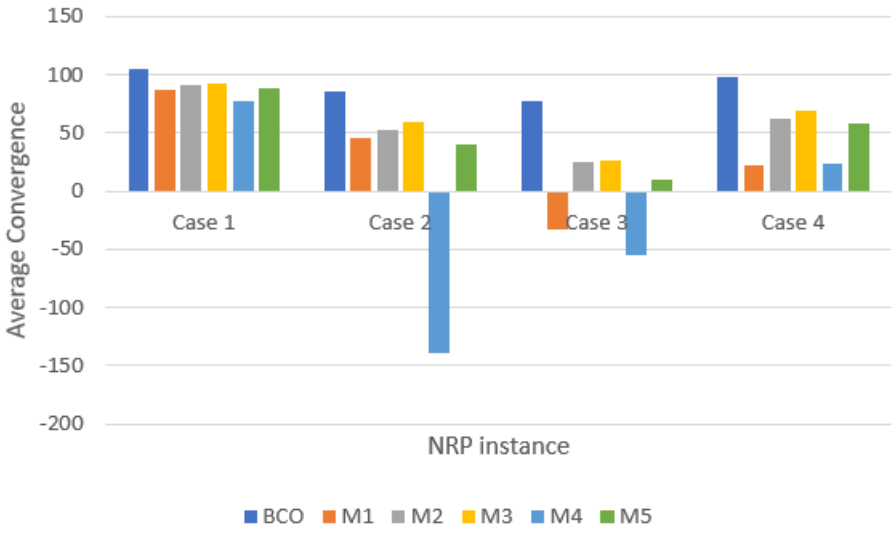

Figure 4.2: Performance analysis with respect to error rate.

4.4 Average Standard Deviation

The Average Standard Deviation is the dispersion of values from its mean value, and it helps to deduce features of the proposed algorithm. The computed result with respect to the Average Standard Deviation is shown in Table 4.4. The Average Standard Deviation attained by various optimization algorithms is depicted in Figure 4.3.

Case	BCO	M1	M2	M3	M4	M5
Case 1	8.042873	9.400025	7.790991	10.28423	13.45243	11.46945
Case 2	9.128496	10.13578	9.109779	9.995431	13.54185	8.020126
Case 3	7.524188	9.069255	9.974609	8.341374	11.50138	10.60612
Case 4	8.366284	9.799198	7.447106	10.05138	11.63378	10.61275

Table 4.4: Experimental result with respect to Average Standard Deviation.

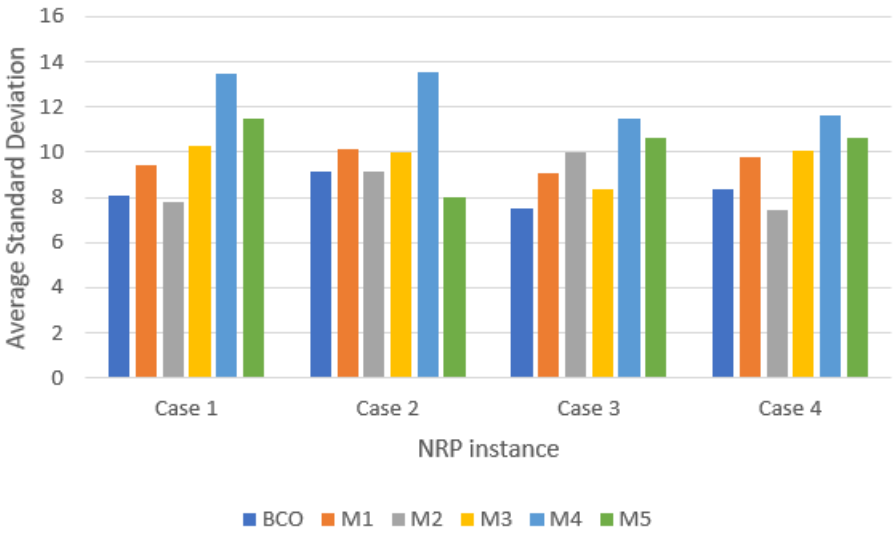

Figure 4.3: Performance analysis with respect to Average Standard Deviation.

4.5 Convergence Diversity

The Convergence Diversity of the solution is to calculate the difference between best convergence and worst convergence generated in the population. The Convergence Diversity and error rate help to infer the performance of the proposed algorithm. The computational analysis based on Convergence Diversity for BCO with another competitor algorithm is shown in Table 4.5. The Convergence Diversity for medium datasets is 39%. Figure 4.4 shows the comparison of various optimization algorithms with respect to Convergence Diversity.

Case	BCO	M1	M2	M3	M4	M5
Case 1	9.13	7.87	8.33	6.71	8.77	9.29
Case 2	23.67	22.21	26.98	19.29	25.45	24.87
Case 3	59.31	54.66	48.13	55.47	50.24	56.18
Case 4	35.95	30.03	31.83	24.11	30.35	29.69

Table 4.5: Experimental result with respect to Convergence Diversity.

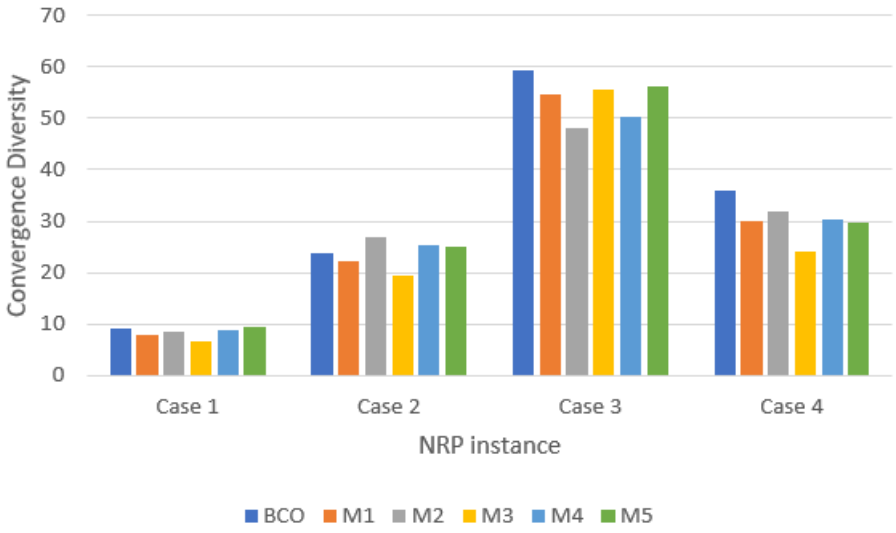

Figure 4.4: Performance analysis with respect to Convergence Diversity.

4.6 Average Cost Diversion

The computational analysis based on cost diversion shows proposed BCO yields less diversion in cost compared to other competitor techniques. The computational analys is with respect to Average Cost Diversion is shown in Table 4.6. For medium dataset 46% of instances got diverged out of which many instances yield optimum value. A negative value in the table indicates corresponding instances have achieved new optimized values. Figure 4.5 depicts the comparison of various optimization algorithms with respect to Average Cost Diversion.

Case	BCO	M1	M2	M3	M4	M5
Case 1	8.4	24.00	6.80	6.80	40.00	9.80
Case 2	21.58	46.40	25.60	25.60	215.60	39.80
Case 3	17.91	46.00	16.80	16.80	67.80	20.20
Case 4	11.72	49.00	10.67	10.67	43.67	13.67

Table 4.6: Experimental result with respect to Average Cost Diversion.

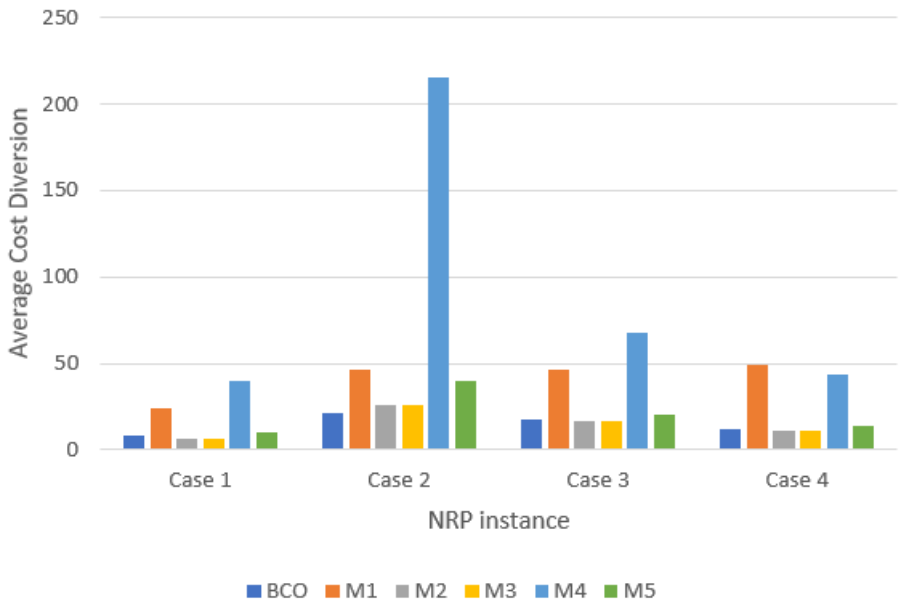

Figure 4.5: Performance analysis with respect to Average Cost Diversion.

4.7 Discussion

The experiments to solve NP-hard combinatorial Nurse Rostering Problem are conducted by our proposed algorithm. The results of BCO algorithm are compared with other competitor methods, and the best values are tabulated in Table 4.1. To evaluate the performance of the proposed algorithm, various performance metrics are considered. Tables 4.1-4.6 show the outcome of our proposed algorithm and other existing methods performance. From Tables 4.1-4.6 and Figures 4.1-4.5, it is evidently shown that the proposed BCO algorithm has very good chances to attain the best value on performance metrics compared to competitor algorithms which use the INRC2010 dataset.

Compared with other existing methods, the mean value of BCO is 24% reduced towards optimum value with other competitor methods, and it attained lesser worst value in addition to the best solution. The used datasets are medium sized with 30 nurses; the standard deviation of BCO is reduced to 5.43%. The error rate of our proposed approach, when compared with other competitor methods with medium sized datasets, reduces to 6.78%. The convergence rate of BCO has achieved 61%. Basically, the proposed adapted BCO produced very competitive results in comparison with those achieved by the five existing

methods in all medium track instances of INRC2010 dataset. This is initial research of adapting BCO algorithm to the INCR2010 dataset. However, it is observed during run of experiment that the proposed methods suffers stagnation in local optima as well as encountering premature convergence. This method balances the exploration and exploitation without any biased nature. Thus BCO converges the population towards an optimal solution at the end of each iteration. Both computational and statistical analyses show the significant performance over other competitor algorithms in solving the NRP.

Chapter 5

Conclusions

This work tackles solving the Nurse Rostering Problem (NRP) using Bee Colony Optimization Algorithm (BCO). To solve the NRP effectively, BCO with an iterative search is chosen for global search. The BCO algorithm as a population-based search method is motivated by intelligent foraging behaviour of honey bee in their hives. BCO has a lot of advantages; it has simplicity, flexibility and robustness, it has the ability to explore local solutions, it has the ability to handle objective cost, it is easy to implement and has broad applicability with complex functions. Object oriented programming played a fundamental role in the development of the code. Due to the complexity of the algorithm it was necessary to create objects that can be easily manipulated in order to have a flexible code. The proposed system is evaluated using the INRC2010 dataset, and the performance of the proposed algorithm is compared with other five existing methods. To assess the performance of our proposed system, 4 different cases of medium sized datasets are chosen, and it is evidently shown that the proposed BCO algorithm has very good chances to attain the best value on performance metrics compared to competitor algorithms which use the INRC2010. Thus, our algorithm contributes with a new deterministic search and effective approach to solve the NRP. Thus, BCO outperforms with classical Bee Colony Optimization for solving NRP by satisfying both hard and soft constraints.

The future work can be projected to:

- adapting proposed BCO for various scheduling and timetabling problems,
- exploring unfeasible solution to imitate optimal solution,

- improve the adapted BCO by introducing more powerful and more structure local search mechanisms to handle specific soft constraints violations while tackling the NRP,
- integrate adapted BCO algorithm with components of other metaheuristic algorithms.

Bibliography

[1] B. Cheang, H. Li, A. Lim, and B. Rodrigues, "Nurse rostering problems - a bibliographic survey," *European Journal of Operational Research*, vol. 151, pp. 447–460, 2003.

[2] U. Aickelin and P. White, "Building better nurse scheduling algorithms," *Annals of Operations Research*, vol. 128, pp. 159–177, 2004.

[3] A. Abuhamdah, W. Boulila, G. M. Jaradat, A. M. Quteishat, M. K. Alsmadi, and I. A. Almarashdeh, "A novel population-based local search for nurse rostering problem." *International Journal of Electrical Computer Engineering (2088-8708)*, vol. 11, 2021.

[4] K. A. Dowsland and J. M. Thompson, "Solving a nurse scheduling problem with knapsacks, networks and tabu search," *Journal of the operational research society*, vol. 51, no. 7, pp. 825–833, 2000.

[5] T. J. Kr¨uger, T. Davidovíc, D. Teodorovíc, and M. Selmíc, "The bee colony optimization algorithm and its convergence," *Int. J. Bio-Inspired Computation*, vol. x, 2014.

[6] X. Li and G. Yang, "Artificial bee colony algorithm with memory," *Applied Soft Computing*, vol. 41, pp. 362–372, 2016.

[7] G. M. Jaradat, A. Al-Badareen, M. Ayob, M. Al-Smadi, I. Al-Marashdeh, M. Ash-Shuqran, and E. Al-Odat, "Hybrid elitist-ant system for nurse-rostering problem," *Journal of King Saud University-Computer and Information Sciences*, vol. 31, pp. 378–384, 2019.

[8] A. M. Turhan and B. Bilgen, "A hybrid fix-and-optimize and simulated annealing approaches for nurse rostering problem," *Computers Industrial Engineering*, vol. 145, p. 106531, 2020.

[9] S. Petrovic and G. V. Berghe, "A comparison of two approaches to nurse rostering problems," *Annals of Operations Research*, vol. 194, pp. 365–384, 2012.

[10] A. Chakraborty and A. K. Kar, "Swarm intelligence: A review of algorithms," *Nature-inspired computing and optimization*, pp. 475–494, 2017.

[11] A. E. Hassanien and E. Emary, *Swarm intelligence: principles, advances, and applications*. CRC press, 2018.

[12] M. Mavrovouniotis, C. Li, and S. Yang, "A survey of swarm intelligence for dynamic optimization: Algorithms and applications," *Swarm and Evolutionary Computation*, vol. 33, pp. 1–17, 2017.

[13] L. Brezočnik, I. F. Jr, and V. Podgorelec, "Swarm intelligence algorithms for feature selection: a review," *Applied Sciences*, vol. 8, p. 1521, 2018.

[14] S. Abbasi, H. Kazi, and K. Khowaja, "A systematic review of learning object oriented programming through serious games and programming approaches," 2017, pp. 1–6.

[15] F. Schuster, T. Tendyck, C. Liebchen, L. Davi, A.-R. Sadeghi, and T. Holz, "Counterfeit object-oriented programming: On the difficulty of preventing code reuse attacks in c++ applications," 2015, pp. 745–762.

[16] C. Valouxis, C. Gogos, G. Goulas, P. Alefragis, and E. Housos, "A systematic two phase approach for the nurse rostering problem," *European Journal of Operational Research*, vol. 219, no. 2, pp. 425–433, 2012.

[17] E. Burke and T. Curtois, "An ejection chain method and a branch and price algorithm applied to the instances of the first international nurse rostering competition," 2010.

[18] K. Nonobe, "An approach using a general constraint optimization solver," 2010.

[19] Z. Lu and J.-K. Hao, "Adaptive local search for the first international nurse rostering competition," 2010.

[20] B. Bilgin, P. Demeester, M. Misir, W. Vancroonenburg, G. Vandenberghe, and T. Wauters, "A hyper-heuristic combined with a greedy shuffle approach to the nurse rostering competition," 2010.

[21] A. A. Constantino, D. Landa-Silva, E. L. de Melo, C. F. X. de Mendonça, D. B. Rizzato, and W. Romão, "A heuristic algorithm based on multi-assignment procedures for nurse scheduling," *Annals of Operations Research*, vol. 218, no. 1, pp. 165–183, 2014.

[22] M. A. Awadallah, A. T. Khader, M. A. Al-Betar, and A. L. Bolaji, "Global best harmony search with a new pitch adjustment designed for nurse rostering," *Journal of King Saud University - Computer and Information Sciences*, vol. 25, pp. 145–162, 2013. [Online]. Available: http://dx.doi.org/10.1016/j.jksuci.2012.10.004

[23] ——, "Nurse rostering using modified harmony search algorithm," in *International Conference on Swarm, Evolutionary, and Memetic Computing*. Springer, 2011, pp. 27–37.

[24] ——, "Global best harmony search with a new pitch adjustment designed for nurse rostering," *Journal of King Saud University-Computer and Information Sciences*, vol. 25, no. 2, pp. 145–162, 2013.

[25] ——, "Hybrid harmony search for nurse rostering problems," in *2013 IEEE Symposium on Computational Intelligence in Scheduling (CISched)*. IEEE, 2013, pp. 60–67.

[26] ——, "Harmony search with novel selection methods in memory consideration for nurse rostering problem," *Asia-Pacific Journal of Operational Research*, vol. 31, no. 03, p. 1450014, 2014.

[27] I. Berrada, J. A. Ferland, and P. Michelon, "A multi-objective approach to nurse scheduling with both hard and soft constraints," *Socio-economic planning sciences*, vol. 30, no. 3, pp. 183–193, 1996.

[28] E. K. Burke, J. Li, and R. Qu, "A pareto-based search methodology for multi-objective nurse scheduling," *Annals of Operations Research*, vol. 196, no. 1, pp. 91–109, 2012.

[29] K. A. Dowsland, "Nurse scheduling with tabu search and strategic oscillation," *European journal of operational research*, vol. 106, no. 2-3, pp. 393–407, 1998.

[30] E. Burke, P. D. Causmaecker, and G. V. Berghe, "A hybrid tabu search algorithm for the nurse rostering problem," in *Asia-Pacific Conference on Simulated Evolution and Learning*. Springer, 1998, pp. 187–194.

[31] E. K. Burke, G. Kendall, and E. Soubeiga, "A tabu-search hyperheuristic for timetabling and rostering," *Journal of heuristics*, vol. 9, no. 6, pp. 451–470, 2003.

[32] E. Burke, P. Cowling, P. De Causmaecker, and G. V. Berghe, "A memetic approach to the nurse rostering problem," *Applied intelligence*, vol. 15, no. 3, pp. 199–214, 2001.

[33] M. Hadwan and M. Ayob, "A constructive shift patterns approach with simulated annealing for nurse rostering problem," in *2010 International Symposium on Information Technology*, vol. 1. IEEE, 2010, pp. 1–6.

[34] E. Sharif, M. Ayob, and M. Hadwan, "Hybridization of heuristic approach with variable neighborhood descent search to solve nurse rostering problem at universiti kebangsaan malaysia medical centre (ukmmc)," in *2011 3rd Conference on Data Mining and Optimization (DMO)*. IEEE, 2011, pp. 178–183.

[35] U. Aickelin and K. A. Dowsland, "An indirect genetic algorithm for a nurse-scheduling problem," *Computers & operations research*, vol. 31, no. 5, pp. 761–778, 2004.

[36] S. Asta, E. Özcan, and T. Curtois, "A tensor based hyper-heuristic for nurse rostering," *Knowledge-based systems*, vol. 98, pp. 185–199, 2016.

[37] K. Anwar, M. A. Awadallah, A. T. Khader, and M. A. Al-Betar, "Hyper-heuristic approach for solving nurse rostering problem," in *2014 IEEE symposium on computational intelligence in ensemble learning (CIEL)*. IEEE, 2014, pp. 1–6.

[38] N. Todorovic and S. Petrovic, "Bee colony optimization algorithm for nurse rostering," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 43, no. 2, pp. 467–473, 2012.

[39] L. B. Asaju, M. A. Awadallah, M. A. Al-Betar, and A. Khader, "Solving nurse rostering problem using artificial bee colony algorithm," in *ICIT 2015 the 7th international conference on information technology*, 2015, pp. 32–38.

[40] M. A. Awadallah, A. L. Bolaji, and M. A. Al-Betar, "A hybrid artificial bee colony for a nurse rostering problem," *Applied Soft Computing*, vol. 35, pp. 726–739, 2015.

[41] M. A. Awadallah, M. A. Al-Betar, A. T. Khader, A. L. Bolaji, and M. Alkoffash, "Hybridization of harmony search with hill climbing for highly constrained nurse rostering problem," *Neural Computing and Applications*, vol. 28, no. 3, pp. 463–482, 2017.

[42] H. G. Santos, T. A. Toffolo, R. A. Gomes, and S. Ribas, "Integer programming techniques for the nurse rostering problem," *Annals of Operations Research*, vol. 239, no. 1, pp. 225–251, 2016.

[43] T.-H. Wu, J.-Y. Yeh, and Y.-M. Lee, "A particle swarm optimization approach with refinement procedure for nurse rostering problem," *Computers Operations Research*, vol. 54, pp. 52–63, 2015.

[44] M. S. Kiran, H. Hakli, M. Gunduz, and H. Uguz, "Artificial bee colony algorithm with variable search strategy for continuous optimization," *Information Sciences*, vol. 300, pp. 140–157, 2015.

[45] G. Sahoo *et al.*, "A two-step artificial bee colony algorithm for clustering," *Neural Computing and Applications*, vol. 28, pp. 537–551, 2017.

[46] D. Teodorovíc, M. Selmíc, and T. Davidovíc, "Bee colony optimization part ii: The application survey," *Yugoslav Journal of Operations Research*, vol. 25, pp. 185–219, 2015.

[47] W.-F. Gao, L.-L. Huang, S.-Y. Liu, and C. Dai, "Artificial bee colony algorithm based on information learning," *IEEE transactions on cybernetics*, vol. 45, pp. 2827–2839, 2015.

[48] K. Z. Gao, P. N. Suganthan, Q. K. Pan, M. F. Tasgetiren, and A. Sadollah, "Artificial bee colony algorithm for scheduling and rescheduling fuzzy flexible job shop problem with new job insertion," *Knowledge-Based Systems*, vol. 109, pp. 1–16, 2016. [Online]. Available: http://dx.doi.org/10.1016/j.knosys.2016.06.014