REPUBLIQUE ALGERIÉNNE DÉMOCRATIQUE ET POPULAIRE
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Mohamed Khider - BISKRA
Faculté des Sciences Exactes et des Sciences de la Nature et de la Vie
Département d'informatique

THÈSE

En vue d'obtenir le titre de

Docteur en Informatique 3éme cycle LMD
Option : Image et Vie Artificielle

Contribution au développement de concepts et outils d'aide à la décision pour l'optimisation via le plongement dans un réseau d'interconnexion parallèle.

Présentée par

Aymen Takie Eddine SELMI

Devant le jury composé de :

- Pr. Leila DJEROU	Université de Biskra	Présidente
- Dr. Mohamed Faouzi ZERARKA	Université de Biskra	Directeur de thèse
- Dr. Abdelhakim CHERIET	ENSIA, Alger	Co-Directeur de thèse
- Pr. Hammadi BENNOUI	Université de Biskra	Examinateur
- Dr. Okba TIBERMACINE	ENSIA, Alger	Examinateur

PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA Ministry of Higher Education and Scientific Research MOHAMED KHIDER UNIVERSITY - BISKRA
Faculty of Exacts Sciences, Nature and Life Sciences
Department of Computer Science

THESIS

To obtain the academic degree of

Doctor in Computer Sciences LMD 3rd Cycle
Option: Image and Artificial Life

Contribution to the Development of Concepts and Decision Support Tools for Optimization Through Embedding via a Parallel Interconnection Network

---

Presented by

Aymen Takie Eddine SELMI

Members of the jury:

- Pr. Leila DJEROU	University of Biskra	President
- Dr. Mohamed Faouzi ZERARKA	University of Biskra	Director of thesis
- Dr. Abdelhakim CHERIET	ENSIA, Algiers	Co-Director of thesis
- Pr. Hammadi BENNOUI	University of Biskra	Examinator
- Dr. Okba TIBERMACINE	ENSIA, Algiers	Examinator

Not all words can express the gratitude, love, respect, and recognition; it is simply that I dedicate this modest work to:

To my dear mother, You represent for me the symbol of kindness par excellence, the source of tenderness, and the example of dedication that has never ceased to encourage me and pray for me. Your prayer and blessing have been of great help to me in successfully completing my studies.

To my dear father, No dedication can express the love, esteem, dedication, and respect that I have always had for you. This work is the result of your sacrifices that you have made for my education and training.

To my very dear brother RAMI who is present in all my exam moments with his moral support and sweet surprises. I wish you a future full of joy, happiness, success, and serenity. Through this work, I express to you my feelings of fraternity and love.

To all my colleagues at the University of Biskra.

Acknowledgments

I would like to extend my deepest appreciation to my thesis advisor, Dr. Mohamed Faouzi ZERARKA, for his invaluable guidance throughout this research. His willingness to provide timely feedback, thoughtful discussions, and words of encouragement and love given to me as a father were instrumental in helping me develop as a researcher. I am especially grateful for his patience and support during the many challenges of this work. This achievement would not have been possible without his expertise, insights, and belief in my abilities. I feel extremely fortunate to have worked with such a knowledgeable and caring mentor. His dedication to nurturing my intellectual growth is something I will carry with me in my future endeavors.

I wish to express my sincere gratitude to Dr. Abdelhakim CHERIET for his invaluable assistance during my doctoral research. His generosity in providing access to computing resources was instrumental in enabling me to complete my thesis work. His contributions have been truly motivating and will not be forgotten.

I extend my sincere appreciation to my esteemed colleagues and the entire team at LAHE especially Dr. Abdallah LABADI for their warm welcome and the pleasant working atmosphere they have cultivated. I am truly grateful for the camaraderie and support extended to me, which has made my integration into the team seamless and enjoyable.

I extend my heartfelt gratitude to the esteemed members of my jury: Professor Leila DJEROU, Professor Hammadi BENNOUI, and Dr. Okba TIBERMACINE. Their dedication in investing the time required to thoroughly review and comprehend the manuscript is deeply appreciated.

Finally, I would like to express my deep gratitude to my family who supported and encouraged me throughout my studies, as well as all the teachers who contributed to my training.

Abstract

In a world characterized by complex and interconnected challenges, effective decision-making is paramount for addressing issues spanning environmental sustainability, transportation infrastructure improvement, and medical innovation. However, the growing complexity of these problems often exceeds traditional reasoning capabilities. Decision support systems, leveraging artificial intelligence techniques, offer promising avenues for navigating these challenges. This thesis focuses on addressing one such complex problem, the Traveling Salesman Problem (TSP), which finds applications in logistics, network planning, and bioinformatics. Despite advancements in TSP-solving methods, scalability and adaptability to dynamic scenarios remain persistent challenges. This research proposes a parallel simulation via an interconnection network topology-based optimization tool integrating advanced artificial intelligence techniques to tackle these issues. The methodology includes hierarchical clustering representations, graph embeddings, and hybrid parallel-solving strategies. Key contributions include novel clustering algorithms tailored for TSP optimization, integration with parallel computing architectures, and experimental validation showcasing superior performance compared to traditional methods. The thesis outlines theoretical foundations, explores parallel computing architectures and graph embedding techniques with the best quality, and presents a comprehensive evaluation of the proposed methodology. The findings contribute to enhancing decision-making processes and offer a robust framework for addressing complex optimization challenges in dynamic real-world settings.

Keywords: complex problems, modeling, embedding, parallel interconnection networks, discrete event systems, clustering, optimization.

Resumé

Dans un monde caractérisé par des défis complexes et interconnectés, la prise de décision efficace est primordiale pour aborder des problèmes touchant à la durabilité environnementale, à l'amélioration des infrastructures de transport et à l'innovation médicale. Cependant, la complexité croissante de ces problèmes dépasse souvent les capacités de raisonnement traditionnelles. Les systèmes d'aide à la décision, exploitant les techniques d'intelligence artificielle, offrent des perspectives prometteuses pour naviguer dans ces défis. Cette thèse se concentre sur l'adresse d'un tel problème complexe, le Problème du Voyageur de Commerce (TSP), qui trouve des applications dans la logistique, la planification de réseau et la bio-informatique. Malgré les avancées dans les méthodes de résolution du TSP, la scalabilité et l'adaptabilité aux scénarios dynamiques demeurent des défis persistants. Cette recherche propose une simulation parallèle via un outil d'optimisation basé sur une topologie de réseau d'interconnexion intégrant des techniques avancées d'intelligence artificielle pour aborder ces problèmes. La méthodologie inclut des représentations de regroupement hiérarchique, des plongements de graphes et des stratégies hybrides de résolution parallèle. Les contributions clés comprennent de nouveaux algorithmes de regroupement adaptés à l'optimisation du TSP, une intégration avec les architectures informatiques parallèles et une validation expérimentale démontrant des performances supérieures par rapport aux méthodes traditionnelles. La thèse décrit les fondements théoriques, explore les architectures informatiques parallèles et les techniques de plongement de graphes de la meilleure qualité, et présente une évaluation complète de la méthodologie proposée. Les résultats contribuent à améliorer les processus de prise de décision et offrent un cadre robuste pour aborder les défis d'optimisation complexes dans des environnements réels dynamiques.

Mots clés : Problèmes complexes, modélisation, plongement, réseau d’interconnexion parallèle, systèmes d’évènements discrets, clustering, optimisation.

ملخص:

في عالم مميز بالتحديات المعقدة والمترابطة، تكون عمليات اتخاذ القرار الفعّالة أمرًا حيويًا للتعامل مع مسائل تتعلق بالاستدامة البيئية وتحسين البنية التحتية للنقل، والابتكار الطبي. ومع ذلك، فإن تزايد تعقيد هذه المشاكل في كثير من الأحيان يتجاوز قدرات التفكير التقليدية. تقدم أنظمة دعم القرار، باستخدام تقنيات الذكاء الاصطناعي، مسارات متحمسة للتنقل في هذه التحديات. تركز هذه الأطروحة على التعامل مع مشكلة معقدة من هذا القبيل، وهي مشكلة بائع الجوال، التي تجد تطبيقات في اللوجستيات وتخطيط الشبكات وعلم الأحياء الحاسوبي. على الرغم من التقدمات في طرق حل مشكلة بائع الجوال، إلا أن قابلية التوسع والتكيف مع السيناريوهات الديناميكية تظل تحديات مستمرة. تقترح هذه البحث محاكاة متوازية عبر أداة تحسين استنادية لتوبولوجيا الشبكة المترابطة تكامل تقنيات الذكاء الاصطناعي المتقدمة للتعامل مع هذه المشاكل .تشمل منهجية البحث تمثيلات التجميع الهرمية، وتضمين الرسوم البيانية، واستراتيجيات الحل المتوازية المختلطة. من بين المساهمات الرئيسية هي خوارزميات التجميع الجديدة المصممة خصيصًا لتحسين هذه المشكلة ، والتكامل مع هندسة الحواسيب المتوازية، والتحقق التجريبي الذي يظهر أداءً متفوقًا بالمقارنة مع الطرق التقليدية. توضح الأطروحة الأسس النظرية، وتستكشف هندسة الحواسيب المتوازية وتقنيات تضمين الرسوم البيانية بأعلى جودة، وتقدم تقييمًا شاملًا للمنهجية المقترحة. تساهم النتائج في تعزيز عمليات اتخاذ القرار وتقدم إطارًا قويًا للتعامل مع التحديات الشديدة للتحسين في البيئات الحقيقية الديناميكية

الكلمات الافتتاحية: المشكلات المعقدة، النمذجة، تضمين، شبكة الاتصال المتوازية، أنظمة الأحداث المتقطعة، تجميع (كلوسترنج)، الأمثلة.

Contents

List of Figures

List of Tables

Chapter 1

General Introduction

1.1 Context and Motivation

In a rapidly changing world, humanity faces increasingly complex and interconnected challenges. Sustainable management of environmental resources, improving transportation infrastructure, and seeking innovative medical solutions are just a few areas where decisions made today will have a significant impact on the future. However, the growing complexity of these problems often surpasses our traditional reasoning capabilities.

All these changes are based on *decision-making*. A decision relies on various elements. The first is identifying what problem needs to be addressed. In other words, having a good understanding and visibility of the problem to make the best decisions that need to be made. The second is developing a valid reasoning process. Finally, the third is the method of elaborating and structuring this reasoning. The decisions that current leaders are required to make within tight deadlines involve numerous interdependent entities that are difficult to analyze. However, the *efficiency* of the choices made must be maximized and must simultaneously meet many interdependent criteria. Throughout history, the solution humanity has found to compensate for its weaknesses is the development of tools to extend its capabilities. In the context of complex decisions, one of the most commonly used tools is called *decision support systems*. These often rely on various techniques and tools for developing artificial intelligence. Currently, there are several decision support systems on the market, but constraints typically limit them to a single application domain and they are often inaccessible in terms of costs and training.

The Traveling Salesman Problem (TSP) stands out as a notoriously challenging optimization puzzle, particularly in its Euclidean form, where the objective is to find the shortest route visiting each city exactly once, represented as points in space [47]. This conundrum finds widespread application in critical domains such as logistics [3, 7], network planning [92, 154], and bioinformatics [111, 120], where efficient resource management and cost minimization are paramount concerns.

Over the years, researchers have tackled the Euclidean TSP using a variety of approaches, ranging from classical mathematical techniques to approximation algorithms like the Christofides

algorithm [29]. Additionally, metaheuristic methods such as genetic algorithms [73] and simulated annealing [88] have been employed to address this problem. Despite significant progress, certain research gaps persist, notably in scalability and adaptability to dynamic scenarios.

Scalability remains a significant hurdle, particularly concerning large datasets, which limits the practical utility of existing solutions. Furthermore, dynamic TSP scenarios, characterized by changing environments or variable point-to-point distances, present additional challenges requiring systematic exploration. Incorporating real-time data, such as traffic conditions [137] or delivery time windows [161, 20], into TSP models represents another avenue for further investigation.

Moreover, the potential of hybrid approaches, which blend various algorithms and integrate machine learning techniques, remains largely untapped in Euclidean TSP research. Bridging these gaps is crucial for enhancing the scalability, adaptability, and real-time applicability of TSP solutions, ensuring their efficacy in tackling intricate challenges in dynamic, practical settings.

1.2 Objectives

The primary objective of this research is to propose a parallel simulation via an interconnection network topology tool that integrates advanced artificial intelligence techniques. This system aims to be open and scalable, capable of adapting to a variety of domains and problems. The first part of this work will explore the theoretical foundations of modeling and parallel simulating discrete event systems supported by interconnection network topology, as well as existing developments in this field. The second part will focus on applying these concepts to optimization, examining different methods and tools available for solving complex problems. Finally, the third part will explore the possibilities of parallelizing simulation processes, examining parallel simulation interconnection network technologies, to speed up process time and obtain optimal solutions more quickly.

By integrating these different elements, this research aims to provide a robust framework for addressing contemporary challenges holistically and effectively, harnessing the potential of artificial intelligence to guide decisions toward outcomes beneficial for humanity and the environment.

1.3 Thesis Contributions

Our work in this thesis consists of creating a platform that integrates: a set of optimization tools that are open and extensible through simulation using parallel representation techniques such as quadtree and octree topologies (interconnection networks). This framework facilitates various decision-making stages, including data acquisition, selection, cleaning, and integration, while enhancing our understanding of the data to provide deeper insights into decision-support problems.

Furthermore, we aim to bolster the efficiency of our tool and concept development by employing

embedding techniques into parallel machines, particularly those utilizing interconnection networks with desirable properties such as crossed cubes. The effectiveness and validity of our proposed components are rigorously assessed through theoretical validation, employing a training-based approach, with a focus on addressing challenges akin to the TSP. Through this integrated approach, we strive to offer a robust platform that not only enhances decision-making processes but also fosters innovation and efficiency in tool development within complex problem domains.

In another term, Our paper introduces a novel approach to solving the Euclidean Traveling Salesman Problem (TSP) by implementing a hierarchical framework that utilizes compressed quadtrees/octrees topology alongside recursive hybrid clustering amalgamating the strengths of classic clustering algorithms including K-Means, Affinity Propagation, and Density Peaks Clustering. Recognizing the need for adaptability to diverse problem domains, we aim to enhance these methods tailored to our specific requirements. Hence, we introduce Enhanced K-Means, K-Affinity Propagation, and K-Density Peaks Clustering. This innovative method optimizes the clustering of cities into a compressed quadtree/octree, prioritizing intra-cluster cohesion and inter-cluster separation while evaluating these aspects through robust metrics such as the Davies-Bouldin index and Gini coefficient. By hierarchically representing the TSP, either in 2D or 3D, we significantly enhance parallelizability, facilitating concurrent optimization.

To further improve scalability, fault tolerance, and resource utilization, we integrate the compressed quadtree/octree into crossed cubes, minimizing distance distortions through a meticulous one-by-one dilation 2 embedding process. Our multi-tiered decomposition strategy strategically abstracts complex optimization landscapes into localized clusters, which are efficiently solved in parallel using heuristics like nearest neighbor and ant colony optimization.

Furthermore, we employ a genetic networking heuristic to interconnect independent intra-cluster solutions, thereby constructing unified inter-cluster routes. This clustering-guided initialization ensures a diverse population of initialized tours, effectively balancing global exploration against localized exploitation.

To validate the efficacy of our approach, we conduct extensive experiments, utilizing the generated solutions to seed a simulated annealing metaheuristic. Through this experimental evaluation, we demonstrate our method's superior ability to initialize metaheuristics for TSP instances, bringing them closer to optimality compared to traditional methods.

1.4 Thesis Outline

This thesis explores the development of an advanced parallel simulation framework integrating Artificial Intelligence techniques for optimizing complex problem-solving. Chapter 2 establishes relevant concepts in complex systems, discrete event modeling, optimization problems, and combinatorial algorithms. Topological classifications and quality metrics for graph embeddings are also examined.

Chapter 3 delves into parallel computing architectures and graph embedding techniques. Flynn's model and topological classifications of interconnection networks are investigated.

Embedding problems, types, and quality are defined.

Chapter 4 uses the Traveling Salesman Problem as a case study to introduce classical and hybrid solving methods, focusing on exact, heuristic, and metaheuristic approaches. Parallel formulations are also explored. This chapter finished by providing an overview of the proposed model methodology, emphasizing the feature selection, optimization, and visualization modules.

Chapter 5 details the proposed modeling paradigm for optimizing TSP solutions through hierarchical clustering representations, graph embeddings, and hybrid parallel solving. Feature selection, dimensionality reduction, clustering algorithms, and intra/inter-cluster optimizations are described.

Chapter 6 evaluates the methodology through experimentation, analyzing clustering performances on graphs, evaluating processing algorithms, and assessing runtimes.

Chapter 7 presents conclusions and discusses avenues for future work, such as dynamic clustering adaptations and evaluating on broader problem domains. Overall, this research introduces an innovative optimization toolkit through synergistic simulation, parallelization, and machine learning.

Chapter 2

General concepts

2.1 Introduction

The first chapter introduces fundamental principles in complex systems, discrete events systems, and combinatorial optimization. It is a crucial exploration for effective problem-solving and optimization. The chapter covers:

- **Complex Systems and Discrete Event Systems:** Discussing characteristics, specifications, modeling techniques, and simulation methodologies.
- **Problem Classification and Optimization:** Addressing the concept of the problem, its classification, and basics of combinatorial optimization using graph theory.
- **Combinatorial Optimization and Meta-heuristic Methods:** Focusing on key aspects such as graph theory, landscape definition, and classification of optimization methods.
- **Parallel and Hybrid Meta-heuristic Approaches:** Concluding with advanced methodologies, including parallel meta-heuristics and the integration of machine learning techniques.

2.2 Complex System

A complex system is a set of interconnected elements that interact in a way to form a coherent whole. These systems can be found in various domains, such as physics, biology, computer science, economics, and more [106].

Complex systems are studied across various scientific disciplines, and understanding them is often crucial for solving real-world problems and improving decision-making in diverse contexts. Tools such as mathematical modeling, computer simulation, and network analysis are often employed to study these systems[106].

2.2.1 Characteristics of complex systems

Complex systems, pervasive across a myriad of scientific disciplines, embody a rich tapestry of interconnected elements whose interactions give rise to emergent properties and behaviors. Understanding these systems requires an exploration of their key characteristics, each contributing to the complexity and adaptability observed[74, 58, 105]. There are some key characteristics of complex systems:

1. **Interconnection:** The elements of a complex system are interconnected, meaning they are linked to each other in a way that influences each other's behavior [74].
2. **Adaptability:** Complex systems often have the ability to adapt to internal or external changes. This may be due to feedback mechanisms that adjust the system in response to disturbances [74].
3. **Emergence:** The properties of the system as a whole can emerge from the interactions among its constituent parts. These emergent properties are not always predictable from studying individual parts [74].
4. **Nonlinear Dynamics:** Complex systems can exhibit nonlinear behavior, meaning that small changes can lead to disproportionate effects or abrupt changes in the system [58].
5. **Self-organization:** Some complex systems have the ability to self-organize, meaning they can develop structures and patterns without external intervention [74].
6. **Sensitivity to Initial Conditions:** Some complex systems, particularly in the context of chaos theory, are sensitive to initial conditions, meaning small initial variations can lead to very different outcomes over time [105].

2.2.2 Specification [8]

Careful specification is essential in modeling complex systems computationally, involving decisions on abstraction levels and representation. Explicitly capturing influential components, interactions, and dynamics is key, while less critical elements may be aggregated. Interactions must consider non-linear feedback effects across scales. Environmental factors are parameterized, initial conditions specified, and uncertainties addressed through probabilities or multiple scenarios. Formal modeling frameworks set standards and guide programming, algorithms, and data structure choices. Validation against empirical observations refines specifications, addressing simplifications and uncertainties. The goal is a useful approximation rather than perfect precision. Rigorous yet pragmatic specifications enable computational models to distill insights from complexity for testing hypotheses and generating knowledge about real-world systems.

2.2.3 Modeling [21]

Modeling involves creating a simplified representation of a system, process, or phenomenon to gain a better understanding of its dynamics. Models can take various forms, including mathematical equations, diagrams, physical replicas, or computer simulations. The goal of modeling is to capture the essential features of the system under study while abstracting

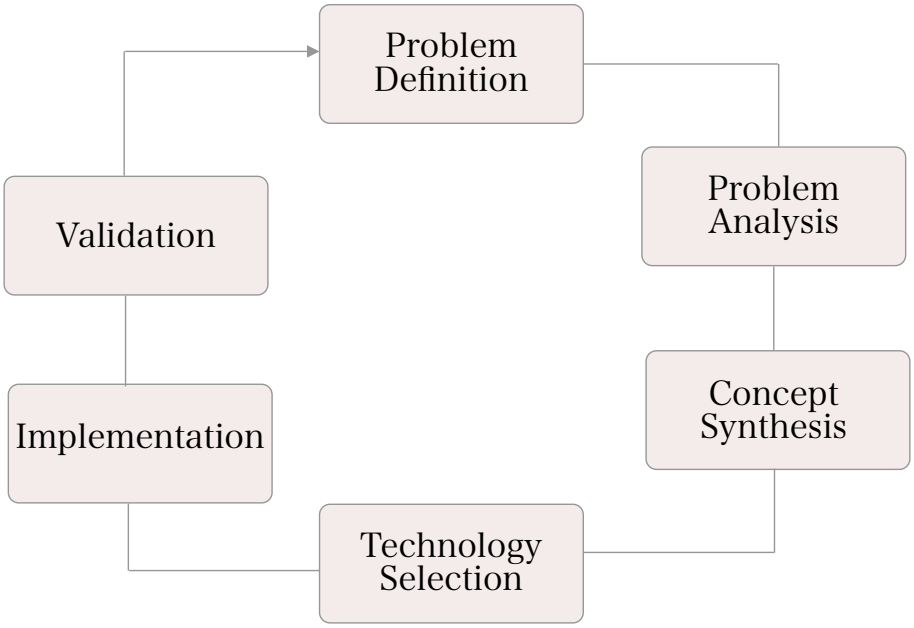


Figure 2.1: Steps of the modeling process.

away unnecessary details. This abstraction enables researchers to analyze and manipulate the model to observe how changes in input parameters impact the system's output.

The modeling process unfolds through a series of distinct stages, as illustrated in Figure 2.1. It commences with the crucial step of precisely defining the problem at hand. Subsequently, a comprehensive analysis of the identified problem ensues, delving into its intricacies and discerning influential factors. The synthesized concepts from this analysis are then organized and integrated, forming the foundational elements of the model. The next stage involves the strategic selection of technology for implementation, necessitating decisions on modeling approaches, software tools, and hardware requirements. Implementation transforms the synthesized concepts into a tangible model, often employing mathematical equations, algorithms, or computer code. Rigorous validation follows, subjecting the model to thorough testing to ensure its accuracy in representing the real-world system. Based on validation results, adjustments may be made, prompting a redefinition of the original problem to align with observed system behavior. This cyclical process of refinement and validation iterates, progressively enhancing the model's accuracy and reliability.

2.2.4 Simulation [21]

Simulation involves running a model over time to observe its behavior and outcomes. Computer simulation, in particular, has become a powerful tool for studying complex systems that may be impractical or impossible to analyze through other means. Through simulation, researchers can explore different scenarios, test hypotheses, and assess the robustness of a system under various conditions.

Simulations can provide insights into the emergent properties of a system and help stakeholders make informed decisions without the need for real-world experimentation, which can be time-

consuming, costly, or ethically challenging. Industries such as aerospace, finance, healthcare, and environmental science heavily rely on simulation techniques to optimize processes, predict outcomes, and mitigate risks.

2.3 Discrete event system

Discrete Event Systems (DES) refer to systems in which events occur at distinct points in time and cause the system to undergo changes in its state. These systems are characterized by the occurrence of events that have a noticeable impact on the system's behavior, and the system's response to these events is often modeled as a series of discrete, distinct steps [25].

Several approaches are employed to model and analyze Discrete Event Systems (DES), each tailored to address specific characteristics and requirements of these systems see Table 2.1.

2.3.1 Petri nets approach

Petri nets are a powerful graphical modeling tool used to represent discrete event systems. They model a system as a collection of places, transitions, and directed arcs connecting places to transitions. Places graphically represent conditions or states and contain tokens to denote the current marking (state) of the system. Transitions represent events that may occur, firing when the appropriate input places contain tokens. When a transition fires, it removes tokens from its input places and adds them to its output places, facilitating the modeling of concurrency, synchronization, and resource allocation dynamics. Analysis techniques like reachability graphs applied to Petri net models aid in system verification and exploring optimization of performance metrics [110].

2.3.2 State-space models

State-space models take an alternative approach by explicitly enumerating all possible system states as vertices in a graph, with transitions between states as edges. While suffering from exponential growth limitations, state-space models effectively optimize properties for small, discrete systems [25, 72].

2.3.3 Agent-based modeling

Agent-based modeling decomposes problems into autonomous software agents that interact locally through message passing within an environment according to behavioral rules. This decentralized approach supports simulation and optimization of complex adaptive systems [17].

2.3.4 Discrete Event System Specification (DEVS)

DEVS provides a formal framework to hierarchically construct modular atomic and coupled component models of large, complex discrete event systems through an interface enforcing separation of model specification and simulation. At its core, DEVS introduces the concept

Approach	**Key Characteristics**	**Application**
Petri Nets	Graph-based model with tokens representing system states. Transitions occur based on rules.	Modeling concurrent processes, workflow, communication protocols. Offers insights into system dynamics.
State-space Models	System states represented as points in a multi-dimensional space. Transitions between states are modeled.	Analyzing system behavior, control theory, studying stability, and identifying reachable states. Common in formal verification.
Agent-based Modeling	Individuals (agents) with autonomous behavior and interactions. Agents adapt and evolve based on local rules.	Simulating complex systems where individual entities influence the overall system behavior. Used in social sciences, ecology, and traffic simulations.
DEVS (Discrete Event System Specification)	Formalism for modeling dynamic systems with discrete events. Hierarchical structure with atomic and composite models. Offers algebraic formalism for modeling.	Modeling and simulation of discrete-event systems. Used in areas like computer systems, communication networks, and manufacturing. Provides a standardized framework for dynamic system representation, incorporating algebraic formalism.

Table 2.1: Discrete Event System Modeling Approaches.

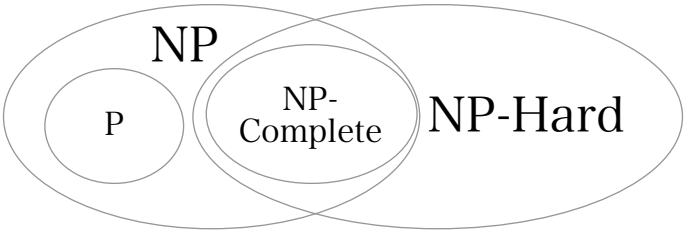


Figure 2.2: Problem complexity classes [31].

of **Atomic Models**, the fundamental building blocks that encapsulate the behavior of a system at a specific level of abstraction. These models respond to input events, triggering state transitions and producing output events. The hierarchical nature of DEVS is manifested in the concept of **Coupled Models**, enabling the composition of atomic models into complex systems, fostering modularity and scalability. Interaction between models occurs through designated **Ports**, where input events drive the state changes within a model, and output events reflect the system's response. The system's behavior is delineated by a set of **States**, defining its possible conditions or configurations. DEVS facilitates simulation through various techniques, allowing for the dynamic analysis of system performance over time. This formalism finds application in diverse domains such as control systems, communication networks, and manufacturing processes, providing a structured approach to understanding and designing complex systems [158].

2.4 The problem concept

In computer science, a problem can be defined as a computational task that a computer program is expected to solve [135]. More formally:

A problem P is a pair (I, Q) where:

- I is a set of instances or inputs to the problem.
- Q is a functional relation that associates each instance $i \in I$ with a set of outputs $Q(i)$.

The aim is to find an algorithm or computer program that implements Q, i.e., for each input $i \in I$, the program should produce a valid output $q \in Q(i)$.

2.5 Classification of problems

Problems can be categorized based on the certainty and predictability of their outcomes. This classification often relies on two fundamental concepts: deterministic Turing machines and non-deterministic Turing machines.

This classification is rooted in the computational complexity of problems and the resources necessary to solve them effectively [95]. See Figure 2.2.

The main complexity classes used to categorize optimization problems based on computational difficulty are P, NP, NP-hard, and NP-complete. P contains problems that can be solved in

Characteristic	Opposite Characteristic
Single-objective	Multi-objective
Linear	Non-linear
Convex	Non-convex
Static	Dynamic
Continuous	Discrete
One-dimensional	Multidimensional
With constraints	Without constraints
Homogeneous types	Heterogeneous types
Deterministic	Stochastic

Table 2.2: Characteristics of optimization problems

polynomial time by a deterministic algorithm. Problems in NP cannot necessarily be solved in polynomial time but can be verified in polynomial time by a non-deterministic algorithm. NP-hard problems are at least as hard as any problem in NP. Even if a polynomial time solution is found for one NP-hard problem, it does not imply that all NP problems can be solved in polynomial time. NP-complete comprises problems in NP that are also NP-hard they are the hardest problems in NP in that every problem in NP can be reduced to an NP-complete problem in polynomial time. Determining whether P = NP, meaning whether every problem for which a solution can be verified quickly can also be solved quickly, remains one of the most significant unsolved problems in computer science [31].

Optimization problems are not confined to a single type or nature. Instead, they often involve a combination of different characteristics, making them heterogeneous in nature [89]. The different natures of a problem are presented in Table 2.2.

2.5.1 Discrete problems

Discrete optimization problems comprise the class of optimization challenges where the variables can only take on discrete, separate values rather than continuous ranges [30]. More formally, discrete optimization involves finding optimal solutions where the decision variables are restricted to be integer or categorical rather than real-valued. The key characteristics of discrete problems include:

- Discrete search space: The set of possible solutions consists of distinct, separated options rather than continuous values. This discrete nature restricts solutions to integer or logical domains [48].
- Discrete variables: The decision variables that define candidate solutions can only take on values from a finite set of discrete alternatives rather than continuous spectra [51].
- Non-differentiability: As solutions lie on a discrete lattice rather than a smooth space, objective functions may lack differentiability properties. Traditional gradient-based optimization is invalid [76].
- Combinatorial complexity: The discrete nature leads to immense search spaces growing

exponentially with problem size. Even moderately sized instances can become intractable to solve exactly [56].

Popular examples of discrete problems include the traveling salesman problem (TSP) [47], knapsack problem [103], job shop scheduling [16], and vehicle routing [142].

2.6 The optimization concept

Optimization permeates our daily lives, driven by the interplay between predicting outcomes and pursuing specific objectives. Success in any optimization endeavor hinges upon clearly defining the problem and desired goals.

An optimization problem can be characterized as follows ([19], [113]):

Minimize (or Maximize):

$$\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_k(\mathbf{x}))$$

Subject to:

$$g_i(\mathbf{x}), \quad i = 1, 2, \ldots, m$$
$$h_j(\mathbf{x}), \quad j = 1, 2, \ldots, p$$

Where:

- $\mathbf{f}(\mathbf{x})$ represents the vector of objective functions to be minimized or maximized. Each objective function $f_i(\mathbf{x})$ depends on the decision variables $\mathbf{x} = (x_1, x_2, \ldots, x_n)$.
- $g_i(\mathbf{x})$ denotes the inequality constraints that limit the feasible solutions, ensuring the solution space falls within specific bounds.
- $h_j(\mathbf{x})$ represents the equality constraints, further restricting the feasible solutions by enforcing specific relationships among the decision variables.
- m and p indicate the number of inequality and equality constraints, respectively.
- k represents the number of objective functions, indicating that the optimization problem aims to optimize multiple conflicting objectives simultaneously.

The objective of the optimization problem is to determine the values of the decision variables $\mathbf{x}$ that minimize or maximize the objective functions $f(\mathbf{x})$, all while adhering to the defined constraints.

2.7 Combinatorial optimization [119, 124]

Combinatorial Optimization (CO) addresses problems of finding the best (or worst) element in a finite, valued set. A combinatorial optimization problem is defined by a discrete space of feasible solutions Ω and an objective function $f : \Omega \to \mathbb{R}^n$. The aim is to find $s^* \in \Omega$ such that $s^* = \arg\max_{s \in \Omega}\{f(s)\}$.

For a minimization problem (or maximization), a global optimum is a solution $s^* \in \Omega$ such that $\forall s \in \Omega, f(s^*) \leq f(s)$ or $f(s^*) \geq f(s)$. If Ω has a neighborhood relation V, a local optimum is $s^* \in \Omega$ such that $\forall s \in V(s^*), f(s^*) \leq f(s)$ or $f(s^*) \geq f(s)$.

CO finds applications in diverse fields such as transportation, management, energy, and engineering. For example, optimizing delivery routes to minimize costs in a large-scale distribution network is a combinatorial optimization challenge.

2.7.1 Graph theory notations

The definitions related to graph theory are taken from [139]. Combinatorial optimization problems can be naturally modeled as graphs, with the solution space represented by the constituents of a graph such as vertices and edges [56].

Graph Definition

- A graph $G = (X, U)$ is considered complete if every pair of vertices in X is connected by an edge in U.
- A directed graph or digraph G has nodes and arcs as ordered pairs of distinct nodes.
- Multigraph specifics: $X(G)$ is the set of nodes $n = |X(G)|$, and $U(G)$ is the set of arcs $m = |U(G)|$.
- $U(i)$ is the set of adjacent edges for node i, and arc costs are associated.
- An undirected graph is a digraph where arc $(x_i, x_j) \in U$ implies $(x_j, x_i) \in U$.
- Subgraph relationship: $G_1 = (X_1, U_1)$ is a subgraph of $G_2 = (X_2, U_2)$ if $X_1 \subseteq X_2$ and $U_1 \subseteq U_2$.
- A spanning subgraph $G^{'}$ of a graph G is denoted as $G^{'} = (X^{'}, U^{'})$, where $X^{'}$ is a subset of the vertices of G, $U^{'}$ is a subset of the edges of G , and $G^{'}$ includes all vertices of G.
- A clique in G refers to a complete subgraph of G.
- A stable set in a graph is a collection of vertices that are pairwise non-adjacent.
- A graph is termed bipartite if its vertex set can be partitioned into two non-empty sets, X_1 and X_2, in a way that no pair of vertices in X_1 (or X_2) are adjacent. It is well-established that a graph is bipartite if and only if it lacks an odd cycle.
- A transversal T in G is a subset of vertices such that for every edge $(u, v) \in U$,

$$\forall (u, v) \in U, \quad (u \in T) \vee (v \in T)$$

Paths, Strings and Cycles

- A path from x_1 to x_k in G is a simple list $[x_1, \ldots, x_k]$ with (x_i, x_{i+1}) as an arc for $i \in [1..k-1]$.
- A path is called an elementary path (String) if it passes through each vertex at most

once.

- The sequence $x_1, x_2, \ldots, x_{k+1}$ forms a cycle if $k > 1$ and $k_1 = x_k + 1$.
- A cycle is Hamiltonian if it forms an elementary path encompassing every node of X.
- An Eulerian path in a graph is a path that traverses every edge exactly once.
- The cost of a path p, denoted as $w(p)$, is the sum of the costs of its constituent arcs.

Connectivity

A graph is connected if there is a path between every pair of nodes; otherwise, it is disconnected.

Trees, Spanning Trees and Components

- A tree is a connected graph without cycles.
- A tree $T = (X', U')$ is a spanning tree of G if $X' = X$ and $U' \subseteq U$. The non-tree edges of T are those in $U - U'$.
- A minimum spanning tree minimizes the cost of its tree edges.
- Connected components are the largest connected subgraphs within G.

Cuts and Cutsets

- A cut (S, T) is a partition of nodes, where $S \subseteq X$ and $T = X - S$.
- The cutset of a cut (S, T) is the set of edges $(x_i, x_j) \in U$ with $x_i \in S$ and $x_j \in T$.
- A k-cutset is a cutset with a cardinality of k.

2.7.2 Landscape definition

The concept of fitness landscape was first introduced by Wright [149] in his studies on the evolution of living beings, aiming to comprehend evolutionary drifts. This idea was later extended to various domains, including combinatorial optimization. In the context of combinatorial optimization, a landscape is defined by a triplet (Ω, V, f), comprising the search space Ω, the neighborhood relation V associated with each feasible solution, and the objective function f measuring the quality of solutions. This definition imparts a geometric structure to the optimization problem, based on a neighborhood relation and an objective function.

2.7.3 Search space

The concept of a fitness landscape, as introduced by Wright in the context of evolutionary biology and later extended to combinatorial optimization, emphasizes the importance of understanding the structure of the search space in optimization problems. In combinatorial optimization, the search space, denoted as Ω, constitutes the set of all feasible solutions to a given problem. It is the landscape upon which the optimization process unfolds, encompassing all potential solutions that can be explored. This search space is not arbitrary; rather, it is intricately defined by the problem at hand. The geometric structure of this space is shaped by the relationships between different solutions, as determined by the neighborhood relation, V.

The neighborhood relation identifies the feasible solutions that are proximate to each other, forming the basis for local exploration during the optimization journey.

2.7.4 Neighborhood relation

In combinatorial optimization, the search space grows with problem complexity, making problems challenging and often impractical to solve exhaustively. The neighborhood relation denoted V connects solutions, crucial for navigating the vast search space. Due to its size, explicitly describing all neighbors becomes impractical. The relation is formalized with operations denoted Δ, symmetric movements from one solution to another.

The relation between the initial solution s and a neighboring solution s' is characterized by the equivalence between membership in $V(s)$ and the existence of an operation δ from Δ such that $\delta(s) = s'$. This formally defined neighborhood relation describes the set of neighbors for each solution.

The concept of a path between two solutions s and s' is introduced, representing a finite sequence of movements from one solution to another. The distance between two solutions s and s' is defined as the length of the shortest path. The neighborhood of order n, denoted V_n, comprises solutions at an exact distance of n from the initial solution s.

2.7.5 Objective space

The objective space in combinatorial optimization is a crucial facet of the fitness landscape. Represented by the objective function, f, it serves as the metric for evaluating the quality of solutions within the search space. The objective function assigns a numerical value to each feasible solution, indicating its fitness or desirability concerning the optimization goals. The optimization process seeks to navigate this objective space, aiming to locate solutions that optimize the objective function. The landscape of the objective space is defined by the distribution of these fitness values across the set of feasible solutions. The interplay between the search space, neighborhood relations, and the objective function constructs a comprehensive framework for understanding and navigating the fitness landscape in combinatorial optimization.

In combinatorial optimization, the objective space, defined by the objective function, comes in three main types: mono-objective, multi-objective, and many-objective.

Mono-objective optimization involves a single criterion, simplifying the search for the global optimum. Multi-objective optimization deals with conflicting criteria, creating a multi-dimensional objective space. In multi-objective optimization, two common approaches are Pareto optimization, which seeks solutions not improvable in one objective without worsening another, and aggregation, where multiple objectives are combined into a single function for optimization.

Many-objective optimization extends this to handling a large number of conflicting objectives, demanding strategies like diversity preservation for exploring the intricate objective space. In summary, the nature of the objective function shapes the landscape, influencing the complexity of finding optimal solutions in combinatorial optimization.

2.7.6 Landscape analysis

The analysis of adaptive landscapes in combinatorial optimization problems focuses on three complementary axes of study: interconnected search space (Ω, V), fitness value space (Ω, f), and the complete landscape (Ω, V, f). The approach distinguishes characteristics induced by the neighborhood relation or the evaluation function. Various statistical tools are proposed to analyze each axis:

1. **Interconnected Search Space** (Ω, V)**:**
   - *Distance in a population:* Calculating distances between solutions in a population to understand interconnections.
   - *Diameter:* Measuring the maximum distance between solutions in a population.
2. **Fitness Value Space** (Ω, f)**:**
   - *Amplitude:* Quantifying the relative difference between the fitness values of the best and worst solutions in a population.
   - *Gap:* Providing the average relative difference between the fitness values of local optima and the best-known solution.
3. **Complete Landscape** (Ω, V, f)**:**
   - *Fitness distance correlation coefficient:* Determining the correlation between fitness values and distances to the nearest optimum.
   - *Autocorrelation:* Assessing the correlation between fitness values of neighboring solutions.
   - *Length of correlation:* Describing the roughness of the landscape in terms of neighbors.
   - *Escape rate:* Representing the probability of leaving the attraction basin of a local optimum.
   - *Point of quality:* Evaluating the ability of a solution to lead to the best solutions in the search space.

These indicators provide insights into the characteristics of the landscape, including its smoothness, ruggedness, and the effectiveness of optimization. The analysis covers aspects such as solution distribution, landscape ruggedness, and the relationship between fitness values and solution distances.

2.8 Classification of combinatorial optimization methods

Combinatorial optimization techniques are systematically classified based on their methodologies into two primary categories as shown in Figure 2.3: exact (deterministic) methods, and approximate (probabilistic) methods (heuristics and meta-heuristics)[147].

Exact methods, also called deterministic algorithms, including linear, constraint, and dynamic

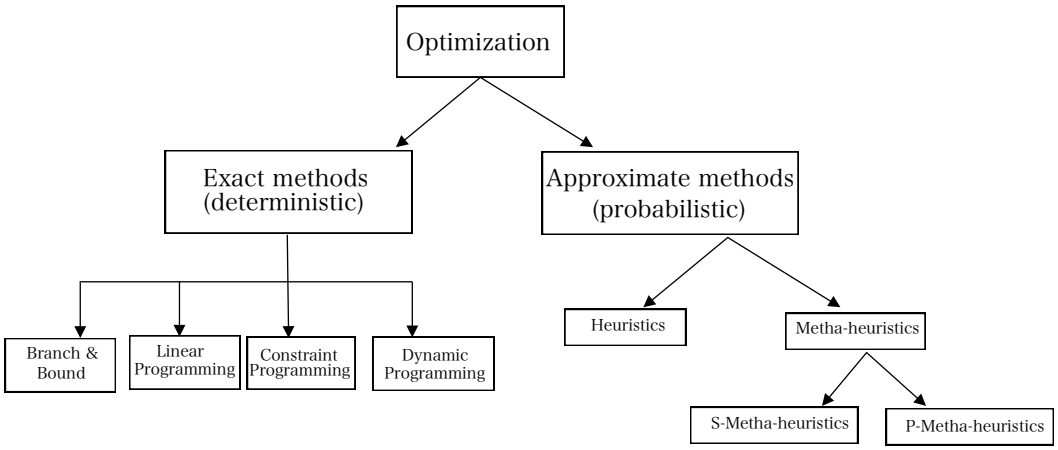


Figure 2.3: Methods classification [147].

programming, are designed to find the globally optimal solution within a finite amount of time [11]. These methods guarantee optimal solutions but are computationally intensive, and suitable for precision-critical problems without significant computational constraints. Thus, exact methods face limitations in handling problems with stochastic, discrete, dynamic, or diverse variable aspects. Additionally, their execution time for large-scale NP-complete problems grows exponentially. Moreover, adapting them to specific problems can be cumbersome and time-consuming.

Approximate optimization methods, also known as probabilistic algorithms, which are partially based on random reasoning, do not guarantee reaching the optimum during the optimization process. However, they efficiently optimize complex problems within reasonable timeframes, especially those belonging to the NP-complete class. These methods are well-suited for discrete, dynamic, stochastic, or nonlinear problems. Additionally, due to their natural inspiration, they are easily understandable and adaptable to various problem types. Some of the most widely employed techniques in this category include Genetic Algorithms [73], Simulated Annealing [88], and Particle Swarms [84].

2.9 Metha-heuristic for combinatorial optimization [153]

There are two categories of metaheuristics: single-solution methods also known as local searches, and methods employing a population of solutions. Local searches tend to intensify exploration by exploiting a portion of the search space, while population-based methods lean towards diversification, exploring different regions of the search space. In the following, we specifically introduce some metaheuristics utilized in this thesis.

2.9.1 Solution Representation

Creating an effective iterative metaheuristic involves a crucial step: designing a suitable solution encoding. The efficiency and effectiveness of the metaheuristic heavily depend on this choice. The encoding must align with the optimization problem and significantly influence the

performance of search operators. Four prevalent encodings in the literature are binary (e.g., knapsack), discrete value vectors (e.g., location, assignment problems), permutations (e.g., traveling salesman, scheduling problems), and real value vectors (e.g., continuous functions).

2.9.2 S-solution methods

The search space for combinatorial optimization problems is often too vast to enumerate within a reasonable time. To address this, connections between solutions are established, allowing the exploration of one solution to lead to another. A crucial aspect is defining a neighborhood relation, mapping each solution to its neighboring set (excluding itself). Local searches hinge on this relation, employing a procedure to exploit the neighborhood. Different local searches vary in how they exploit the neighborhood, with the neighborhood serving as a parameter in this process.

Simulated annealing (SA)

Simulated annealing is an S-solution optimization technique inspired by the physical annealing process (statistical mechanics). This approach incorporates a temperature parameter, which undergoes dynamic adjustments throughout the search process.

The simulated annealing procedure applied to optimization problems considers a neighborhood exploration that allows moving to a less-quality neighboring solution with a non-zero probability. This helps escape local optima.

Initially, a temperature T is set and decreases throughout the algorithm tending towards 0. The probability p of accepting deteriorating solutions decreases as T decreases.

Simulated annealing provides asymptotic convergence proof, so under certain temperature reduction schemes, it guarantees finding the optimal solution. However, the theoretical parameter settings are impractical, requiring parameter tuning.

Acceptance Criterion

The acceptance of a new state depends on whether the change in cost, denoted as ΔC, is negative. In cases where $\Delta C \geq 0$, the new state may be accepted with a probability determined by the expression $e^{-\Delta C/T}$, where T is the temperature.

This probabilistic acceptance strategy allows for occasional exploration of suboptimal solutions, particularly in the early stages. The temperature, T, gradually decreases in incremental steps, focusing the search on regions associated with better solutions.

The acceptance criterion strikes a balance between intensifying the exploration of favorable solutions and promoting diversification through occasional acceptance of worse solutions, controlled by the temperature parameter. This approach facilitates escaping local optima and directs the search towards converging on high-quality solution regions.

2.9.3 Population solution methods

Unlike local searches, population-based methods improve a population of solutions over iterations. The advantage of these methods is to use the population as a diversity factor. There are several evolution strategies for this population, leading to methods such as ant colonies [33] and genetic algorithms [73].

In this research, we exclusively explore the use of genetic algorithms. Genetic algorithms simulate the process of evolution within a population. Starting from an initial set of N solutions representing individuals of the problem, we apply operators such as crossover or mutation to simulate genomic interventions. This process leads to obtaining a population of solutions increasingly adapted to the problem, with adaptation being evaluated through the objective function associated with the optimization problem.

A classical genetic algorithm unfolds in several successive steps:

1. creation of a random population,
2. evaluation of each individual in the population using the objective function,
3. selection of individuals through a selection strategy to form a parent population,
4. application of the crossover operator with a probability P_c to individuals in the parent population to obtain a child population,
5. application of the mutation operator with a probability P_m to individuals in the child population,
6. use of a replacement strategy to form the new population,
7. return to step 2 if the termination criterion is not met.

Concurrently, other aspects of our research focus on exploring simulated annealing and ant colony optimization (ACO) algorithms. As demonstrated in section 2.6.1, simulated annealing draws inspiration from the cooling process of molten metals to attain a stable crystalline configuration. Similarly, solutions are adjusted over time to seek optimal configurations. ACO, on the other hand, is inspired by the collective behavior of ants in solving optimization problems. Ants communicate by depositing pheromones, thereby influencing the selection of optimal paths in their quest for food.

A classical ant colony optimization method unfolds in several successive steps:

1. initialization of ant colony positions, pheromone levels, and configuring parameters that control the dynamics of pheromone deposition and evaporation (α, β, ρ, and Q).
2. iteration through each ant's movement:
   (a) probabilistic selection of the next move based on pheromone levels and heuristic information,
   (b) updating pheromone levels on the paths taken by the ants,
3. evaluation of the solutions constructed by the ants using the objective function,

4. global updating of pheromone levels based on the quality of solutions,
5. return to step 2 for additional iterations if the termination criterion is not met.

2.10 Choice of Metha-heuristic

Metaheuristics constitute a category of algorithmic techniques aimed at solving or approximating the best solution for optimization problems. The choice of an appropriate metaheuristic depends on the specific nature of the problem and the objectives pursued. The definition of the landscape is intricately tied to the dynamics of metaheuristics. Indeed, within the scope of a metaheuristic, the objective function plays a pivotal role in making choices among encountered solutions, and the neighborhood relation is a crucial element in local search. Therefore, the study of three key questions arise:

- How should the problem be modeled? This includes the representation of solutions, the definition of the objective function, the establishment of neighborhood relations, and the consideration of problem constraints.
- Which metaheuristic is most suitable? This involves determining the most favorable method in terms of local search, neighborhood exploration, or the use of methods based on a population of solutions.
- What is the optimal parameterization? This question concerns the appropriate configuration of metaheuristic parameters to achieve optimal performance.

Table 2.3 provides a detailed comparison of Simulated Annealing, Ant Colony Optimization (ACO), and Genetic Algorithm (GA) with a focus on their application to combinatorial problems. Each criterion provides insights into the unique characteristics of these algorithms, allowing for a nuanced evaluation of their strengths and weaknesses.

In problem modeling, Simulated Annealing represents states in state space, ACO utilizes pheromone trails, and GA employs a population of solutions. This reflects their distinct approaches to representing and exploring the solution space. Simulated Annealing employs a temperature-guided objective function, ACO relies on pheromone level-based mechanisms, and GA employs a fitness function, illustrating the diverse optimization strategies employed by each algorithm.

The neighborhood relations criterion highlights the methods through which solutions are explored. Simulated Annealing transitions through temperature, ACO defines relations through pheromone updates, and GA utilizes crossover and mutation operations. These differences underscore the unique exploration strategies employed by each metaheuristic.

Constraints are handled differently as well. Simulated Annealing controls acceptance through temperature adjustments, ACO adapts pheromone levels, and GA utilizes selection operators. This reflects their diverse approaches to handling constraints within combinatorial problems.

The table also addresses metaheuristic selection, optimal parameterization, strengths, challenges, applications, local search strategy, neighborhood exploration technique, convergence, exploration vs. exploitation balance, robustness, and sensitivity to parameters. Each criterion

Criteria	Simulated Annealing	ACO (Ant Colony Optimization)	Genetic Algorithm
Problem Modeling	States in state space	Pheromone trails	Population of solutions
Objective Function	Temperature-guided	Pheromone level-based	Fitness function
Neighborhood Relations	Transition through temperature	Defined by pheromone updates	Crossover and mutation operations
Constraint Handling	Temperature controls acceptance	Pheromone adjustments	Selection operators
Metaheuristic Selection	Rugged landscapes, exploration	Combinatorial optimization	Versatile, global search
Optimal Parameterization	Cooling rate, initial temperature	Pheromone decay, α, β, ρ, and Q	Population size, crossover rate, mutation rate
Strengths	Escape local optima through randomness	Efficient for complex combinatorial problems	Powerful for global search
Challenges	Dependence on cooling schedule	Sensitivity to parameter settings	Sensitivity to parameter tuning
Applications	Function optimization, TSP	TSP, Job Scheduling, Routing	Scheduling, Machine Learning
Local Search Strategy	Stochastic exploration	Use of pheromones	Genetic mutations and recombinations
Convergence	Depends on cooling and initial solutions	Rapid convergence due to pheromones	Convergence depends on selection and recombination
Exploration vs Exploitation	Balanced by temperature	Balanced by pheromones(α, β, ρ, and Q)	Balance between genetic exploration and exploitation
Robustness	Sensitive to parameters and initial conditions	Robust and adaptive	Robust, but dependent on parameters and operators
Sensitivity to Parameters	Sensitive to cooling and temperature parameters	Sensitive to parameters related to pheromones	Sensitive to selection and recombination parameters

Table 2.3: Comprehensive Comparison of Metaheuristics: Simulated Annealing, ACO, and Genetic Algorithm

contributes to a holistic understanding of the algorithms' behaviors and capabilities.

Choosing the best metaheuristic for a combinatorial problem involves a careful consideration of these criteria. The nature of the problem, such as its complexity, size, and specific constraints, plays a crucial role. If the problem exhibits a rugged landscape and requires effective exploration to escape local optima, Simulated Annealing might be a suitable choice. For combinatorial optimization tasks with a focus on efficient exploration, ACO could be preferred. If a global search is essential, and the problem involves optimization, scheduling, or machine learning, GA might be the most versatile option.

Parameter tuning is critical, as the performance of these algorithms is sensitive to various parameters. Therefore, an iterative approach involving experimentation and fine-tuning is often necessary to identify the optimal set of parameters for a specific problem.

2.11 Parallel Metha-heuristic [4]

Efficiently addressing complex optimization problems with metaheuristics remains challenging due to the time-intensive nature of evaluating objective functions and constraints, especially in resource-intensive scenarios with vast search spaces. The computational demands increase significantly when employing S-metaheuristics on large neighborhoods or executing P-metaheuristic reproductive cycles on extended individuals or populations. To tackle these challenges, researchers focus on algorithmic enhancements, such as devising new move operators, hybrid algorithms, and parallel models. The parallel and distributed computing paradigm emerges as a valuable strategy to enhance metaheuristic performance. Leveraging parallelism **accelerates the search process**, enabling real-time and interactive optimization methods. Additionally, it **enhances solution quality** by facilitating information exchange between cooperative metaheuristics. **The robustness of parallel metaheuristics** is evident in their effective resolution of various optimization problems and instances, along with improved sensitivity to parameters. Furthermore, parallelism facilitates **solving large-scale problems** and more accurate mathematical models that surpass the capabilities of sequential machines.

In this goal, three primary parallel models stand out: solution-level, iteration-level, and algorithmic-level. The solution-level parallel model centers on concurrently evaluating individual solutions, particularly advantageous when the evaluation function is computationally intensive or IO-demanding. This model parallelizes problem-dependent operations on solutions and views the function as an aggregation of partial functions. The iteration-level parallel model, operating as a low-level Master-Worker model, maintains the heuristic's behavior while parallelizing solution evaluations within each iteration. At the outset of every iteration, the master duplicates solutions for parallel nodes, facilitating efficient execution, especially when evaluating each solution is resource-intensive. Finally, the algorithmic-level parallel model involves simultaneously launching multiple metaheuristics, whether heterogeneous or homogeneous, independent or cooperative. These metaheuristics may commence from the same or different solutions, configured with similar or distinct parameters, with the overarching goal of collectively computing improved and robust solutions through concurrent execution.

2.11.1 Decomposition for Parallel Metha-heuristic

Parallel metaheuristics harness the computational capabilities of multiple processors or units to significantly improve optimization processes. One prevalent and effective strategy employed in parallelization is decomposition. This approach entails breaking down the original optimization problem into smaller, more manageable subproblems that can be solved concurrently.

The rationale behind decomposition lies in the principle of dividing and conquering the optimization challenge. By subdividing the problem into independent components, each computational unit can focus on solving its assigned subproblem simultaneously with others. This concurrent processing enables a more efficient exploration of the solution space and accelerates the overall optimization process.

In the decomposition process, careful consideration is given to how the problem can be partitioned into smaller, logically separated tasks. This may involve dividing the solution space spatially, allocating specific regions to different processors. Alternatively, it might include partitioning the search space, where each processor handles a distinct subset of the optimization problem.

Decomposition in parallel metaheuristics is not only about dividing the workload but also about facilitating effective communication and coordination between the parallel processes. Information exchange mechanisms are implemented to share relevant data, such as solutions, fitness values, or intermediate results, ensuring that the parallel search remains coherent and globally informed.

Moreover, the decomposition strategy allows for dynamic load balancing, where the workload distribution among processors is adjusted dynamically based on the evolving computational load. This adaptability ensures that each processor remains optimally utilized, preventing bottlenecks and optimizing overall computational efficiency.

2.11.2 Decomposition methods

Decomposition methods break large optimization problems down into smaller subproblems that can be solved in parallel, allowing metaheuristics to take advantage of distributed computing architectures. One popular approach is problem space decomposition, where the variable space is partitioned among processors. Each processor then runs a metaheuristic independently on its subset of variables while communicating intermediate solutions.

Clustering-based decomposition involves partitioning candidate solutions into coherent groups using unsupervised machine-learning algorithms like k-means, affinity propagation, density peaks clustering, and others. Local metaheuristics are then run simultaneously within the clusters' search spaces. Reassembly techniques like genetic recombination merges high-quality cluster results into improved global solutions.

Decomposition can also happen spatially by dividing a physical problem domain like a traveling salesman into distinct geographical regions. Heuristics search within sub-tours that are dynamically joined across boundaries. Parallelization occurs naturally from smaller independent subproblems.

For problems modeled as graphs, partitioning algorithms split nodes among sets to minimize edge cut sizes.

K-means [67]

The initialization phase of the K-means algorithm begins by randomly selecting K initial centers. To compute the distance between a sample, denoted as x_j, and a center, denoted as c_i, the algorithm employs the Euclidean distance formula:

$$\text{dist}(i,j) = \sum_{k=1}^{d} (x_{jk} - c_{ik})^2 \tag{2.1}$$

In this context, d represents the dimensionality of the samples. Subsequently, each sample is assigned to the cluster center with the minimum distance. The next step involves updating the cluster centers by computing the mean of the samples assigned to each cluster:

$$c_i' = \frac{1}{m_i} \sum_{x_j \in \text{cluster } c_i} x_j \tag{2.2}$$

Here, m_i denotes the total count of samples belonging to the cluster associated with the center c_i. The process iterates, recalculating distances between samples and cluster centers until the algorithm converges.

Affinity Propagation [52]

Affinity Propagation (AP) begins by constructing a similarity matrix s. The similarity $s(i,j)$ between data points i and j is defined as:

$$s(i,j) = -\|x_i - x_j\|^2 \tag{2.3}$$

Additionally, the diagonal preference $s(i,i)$ is typically set to a value, often the median of the off-diagonal similarities, controlling the number of exemplars (clusters) yielded.

Responsibility $r(i,j)$ reflects the suitability of point j as the exemplar for point i compared to other candidate exemplars for i:

$$r(i,j) = s(i,j) - \max_{k \neq j} \{s(i,k) + a(i,k)\} \tag{2.4}$$

where $a(i,j)$ is the availability of point j for point i.

Availability $a(i,j)$ indicates how suitable it is for point i to choose point j as its exemplar, considering support from other points for j as an exemplar:

$$a(i,j) = \min \left\{ 0, r(j,j) + \sum_{k \neq i,j} \max \{0, r(k,j)\} \right\} \tag{2.5}$$

$$a(i,i) = \sum_{k \neq i} \max \{0, r(k,i)\} \tag{2.6}$$

Both responsibility and availability matrices are typically initialized to zero and then updated iteratively using the above formulas. After several iterations (either a set number or until a convergence criterion is met), exemplars are identified.

For each data point i, an exemplar is chosen based on the combined responsibility and availability:

$$\text{exemplar}_i = \arg\max_j \{r(i,j) + a(i,j)\} \tag{2.7}$$

If i is its exemplar, it represents the center of a cluster. Other points selecting i as their exemplar belong to the cluster represented by i.

Density peaks clustering [125]

The density peaks clustering (DPC) algorithm typically follows three main steps:

1. **Compute Local Density for Each Point**:
   The local density $\rho(i)$ of a data point i is determined based on the distance between point i and other points in the dataset. A common method for computing local density is given by:
   $$\rho(i) = \sum_j \chi(d_{ij} - d_c) \tag{2.8}$$
   where d_{ij} represents the distance between point i and point j, d_c is a cutoff distance, and χ is the characteristic function, equal to 1 if the condition inside is true and 0 otherwise.

2. **Compute the Distance from Higher Density Points**:
   For each point, the distance $\delta(i)$ from points with higher density is calculated. Specifically, it is defined as:
   $$\delta(i) = \min_{j:\rho(j)>\rho(i)} d_{ij} \tag{2.9}$$
   For the point with the highest density, $\delta(i)$ is defined as the maximum distance to any other point.

3. **Select Cluster Centers**:
   By plotting each point with $\rho(i)$ on the x-axis and $\delta(i)$ on the y-axis, cluster centers can be identified as points with both high ρ and high δ values, typically situated in the top right corner of the plot.

4. **Assign Points to Clusters**:
   Starting from points with the highest density, each point is assigned to the same cluster as its nearest neighbor with higher density.

2.11.3 Evaluating decomposition methods

Evaluating decomposition methods, including clustering-based decomposition, spatial decomposition, and graph partitioning, necessitates a multifaceted analysis using tailored criteria for each approach. For clustering-based decomposition, metrics such as **intra-cluster cohesion**, **inter-cluster separation**, and **cluster homogeneity** gauge the quality of formed clusters,

assessing their tightness, distinctiveness, and internal consistency. **Scalability** and **cluster stability** measures are crucial to ensure the method's adaptability to varying data sizes and its robustness across multiple runs. In the context of spatial decomposition, attention turns to **spatial coherence**, **load balancing**, and **communication overhead**. Effective spatial decomposition methods exhibit alignment with the inherent spatial structure of the problem, ensure balanced workloads among spatial regions, and minimize communication costs. **Adaptability to irregular geometries** is an additional criterion, emphasizing the importance of versatility. Meanwhile, graph partitioning evaluation relies on **cut size**, **balance**, and **connectivity** metrics to appraise the efficiency of partitioning in reducing communication costs, achieving load balance, and maintaining desired connectivity levels within and between partitions. **Scalability**, **adaptability to graph characteristics**, and **computational complexity** further contribute to a comprehensive assessment, addressing the method's ability to handle varying graph sizes, structures, and computational demands. The combination of these criteria provides a nuanced understanding of the strengths and limitations of each decomposition method, aiding in their application to diverse optimization challenges.

2.11.4 Index measures

In evaluating the efficacy of decomposition methods for parallel metaheuristics, a suite of key index measures is employed to assess various aspects of the partitioning outcomes. These measures offer a comprehensive perspective on the quality, separation, balance, and structural characteristics of the decomposed components. Each index provides unique insights, collectively guiding the analysis and refinement of parallel metaheuristic approaches for diverse optimization challenges Here are some popular evaluation indices based on the considerations mentioned in 2.11.3:

Calinski-Harabasz index (C-index) [22]: Measures the ratio of between-cluster dispersion to within-cluster dispersion. Higher values are better, indicating clear separation between clusters.

Davies-Bouldin index (DBI) [32]: Computes the average similarity between each cluster and its most similar cluster. Lower values are better, indicating high intra-cluster similarity and low inter-cluster similarity.

Gini coefficient [26]: Measures the statistical dispersion of data within clusters. Ranges from 0 to 1, with lower values indicating more equal cluster sizes. Favors balanced partitioning.

Silhouette coefficient [126]: Compares within-cluster distances to distances to other clusters. Ranges from -1 to 1, with higher values indicating denser clusters that are well separated.

Dunn index [36]: Represents the ratio of the minimum inter-cluster distance to the maximum intra-cluster distance. Higher values are better, indicating separated and tightly-knit clusters.

Connectivity: Measures the density of connections between solutions within a cluster. Higher connectivity signifies a more relevant grouping.

Modularity [112]: Evaluates partitioning quality by comparing the edge density within clusters to the expected density. Higher modularity values indicate a better community structure.

Evaluating variance: Compares objective variance before and after decomposition to assess the simplification effect.

2.12 Hybrid Methaheuristic [4]

Hybrid metaheuristics have emerged as powerful optimization techniques that combine multiple metaheuristic algorithms to address complex problems more efficiently by providing robustness, adaptability to various problems, and efficient constraint handling. They compensate for individual algorithmic limitations, accelerating convergence and delivering effective solutions. There exists a dual categorization that underscores the innovative amalgamation of optimization techniques. The first category involves the intricacies of crafting solvers wherein components from one metaheuristic are seamlessly integrated into another. This symbiotic fusion of metaheuristic elements harnesses the strengths of diverse optimization approaches, culminating in a solver that exhibits enhanced performance and adaptability. The second category delves into the synergy between metaheuristics and other methodologies derived from disciplines like operations research and artificial intelligence. This collaborative integration propels the hybrid approach beyond the confines of traditional metaheuristics, extending its reach into a broader spectrum of problem-solving domains. The cross-pollination of techniques from distinct fields yields a powerful and versatile toolkit for addressing complex optimization challenges, fostering a new frontier in the pursuit of efficient and effective solutions.

2.12.1 Reasons for Hybridization

Hybrid metaheuristic methods offer several compelling advantages:

- Broader exploration of the solution space than with single techniques.
- Synergistic effects from leveraging complementary search mechanisms.
- Ability to escape local optima is enhanced through strategically alternating different neighborhood structures.
- Machine learning augmentation improves the effectiveness of heuristics on large-scale problem instances.

2.12.2 Relation between Parallel models and Hybrid Metha-heuristic

In the dynamic landscape of parallel models with heterogeneous computing, the strategic integration of metaheuristics is pivotal for achieving optimal performance. This section navigates the intricacies of *when* and *where* to deploy metaheuristics in such models, addressing the crucial considerations of *who* selects the method and *when* this selection should occur.

Determining the optimal position to deploy a metaheuristic within a parallel model with heterogeneous computing requires a nuanced understanding of the problem at hand. *who*, in

this context, refers to the decision-maker the entity responsible for selecting and placing the metaheuristic within the model. Are there specific layers, stages, or components in the model where metaheuristics are most effective? Understanding the architectural nuances is essential for informed decision-making.

The temporal dimension in the integration of metaheuristics is equally critical. Identifying the right instant (t) for introducing metaheuristics can significantly impact the overall performance of the parallel model. Who decides when to invoke the metaheuristic, and are there specific criteria or triggers that govern this decision? Delving into the temporal considerations provides insights into the strategic timing of metaheuristic utilization for optimal results.

The term *who* refers to the tools and algorithms employed for decision-making. The selection of the appropriate metaheuristic and the determination of the ideal timing for its application are automated processes guided by algorithmic strategies. The focus on the *who* shifts to the development and implementation of these automated decision-making tools, ensuring a seamless and efficient integration of metaheuristics in response to dynamic computational requirements. This automated approach not only streamlines the decision-making process but also maximizes the potential of metaheuristic techniques for superior performance and efficiency in the complex landscape of parallel heterogeneous computing.

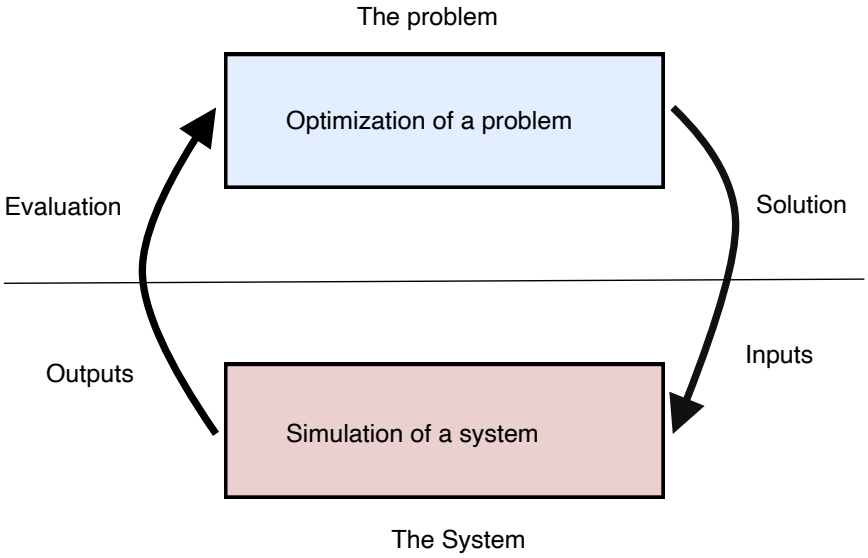


Figure 2.4: Methodologies of OvS [54].

2.12.3 Integration of machine learning techniques with Metaheuristic

The integration of machine learning (ML) techniques into metaheuristics for solving combinatorial optimization problems represents a cutting-edge frontier in research, offering substantial potential for enhancing algorithmic efficiency. The comprehensive classification proposed by Karimi-Mamaghan [83] delineates various purposes for incorporating ML within metaheuristics, including algorithm selection, fitness evaluation, initialization, parameter setting, and

cooperation.

At a high level, ML allows for algorithm selection by predicting the performance of different metaheuristics on a given problem. It can also speed up fitness evaluation, a key component of any metaheuristic, by approximating computationally expensive objective functions. ML aids the initialization process as well, generating high-quality starting solutions or decomposing large problems into smaller sub-problems. During the search evolution, ML can intelligently select variation operators, guide the generation of neighbors based on search history, and develop learnable models to evolve solution populations more effectively. ML also facilitates automated parameter control, whether setting parameters before a search or adaptively regulating them during the process. Finally, when multiple metaheuristics cooperate in solving an optimization problem sequentially or in parallel, ML helps improve their collaborative performance by dynamically adjusting their behaviors. In all of these ways, the integration of ML and metaheuristics has great potential to further enhance optimization capabilities.

2.13 Modeling, Simulation, and Optimisation

The extensive research efforts conducted have led to the emergence of a specialized research field termed *Optimization via Simulation* (OvS) as identified by Fu et al [54]. OvS primarily aims to ascertain optimal solutions through the simulation of the specific problem under consideration. It operates as a connecting link between two distinct research domains: *optimization* and *modeling and simulation* Such that, a myriad of concepts exists within each of these domains, presenting opportunities for integration. Prior research, notably the work of Fu et al [55], has categorized six specific methods at the intersection of these domains. The first involves *ranking and selection* suitable for applications with alternatives simulatable in a reasonable time. The second uses *response surface methodology* based on statistical tools to approximate the optimal relationship between inputs and outputs. The third exploits the concept of *gradient* and *stochastic approximation* using perturbation analyses and various gradient estimation procedures. The fourth group focuses on *random search* using notions of neighborhood and probabilistic traversal of the search space. The fifth gathers *sample path optimization* methods, generating a sample of results through simulations. Finally, the sixth group covers *metaheuristics* for optimizing NP-complete and NP-hard problems with a considerable number of variable combinations, avoiding lengthy execution times associated with other methods.

as shown in Figure 2.4, the connection emphasizes the intricate interplay between optimization methodologies and the modeling and simulation field. OvS establishes a nuanced connection between the optimization process and the dynamic representations furnished by modeling and simulation approaches, enabling a thorough exploration of optimal solutions within complex systems. This collaborative synergy amplifies the effectiveness of optimization methodologies by drawing on insights gleaned from the nuanced behaviors of systems as modeled through simulation.

Several studies in the field of complex systems have utilized the DEVS formalism along with various optimization methods. In 1996, B. Zeigler pioneered the use of metaheuristics to optimize the representation of fuzzy sets in the modeling of infiltration systems [157]. By 2008,

Lee et al [93] integrated branch and bound techniques with DEVS to optimize parameters for a Link-11 network gateway, determining optimal frame sizes through parallel simulation. The same year, Ntaimo et al [114] introduced DEVS-FIRE, optimizing firefighting strategies using the Vost Plus Net Value Change method. In 2009, Mittal et al [107] proposed P-DEVS and DEVS/SOA for decentralized execution of NSGA-II and SPEA2 optimization algorithms, targeting mobile devices' memory efficiency. In 2011, Bisgambiglia et al [14] employed genetic algorithms to optimize fuzzy sets conforming to DEVSFIS. Two years later, Capocchi and Santucci [23] used DEVS to optimize a hydraulic network, selecting optimal dates to maximize a micro-power plant's production and minimize operational costs. In recent years, Heinzl [69] employs a hybrid DEVS model and a multi-objective Variable Neighborhood Search (VNS) to optimize energy-efficient Production Planning and Control (PPC) in industrial plants, facilitated by a domain-specific model abstraction using Model-driven Engineering (MDE). Cárdenas et al [24] propose Mercury, a Modeling, Simulation, and Optimization framework, addresses fog computing's dimensioning and dynamic operation. Using the DEVS formalism, it offers a location-aware solution for data stream analytics, exemplified in a driver assistance case study with comparisons to other simulators. Introducing a Cloud-deployable DEVS-based framework, Bordón-Ruiz et al [18] optimizes UAV trajectories and sensor strategies in target-search missions using a multi-objective Genetic Algorithm.

2.14 Conclusion

This chapter establishes a robust foundation by elucidating global concepts in complex systems and optimization. It provides a comprehensive guide for readers to navigate complex systems, discrete events, and combinatorial optimization. The integration of parallel and hybrid meta-heuristic approaches, coupled with machine learning techniques, showcases the evolving landscape of optimization for real-world problems. This serves as a springboard for subsequent chapters to delve deeper into specific methodologies and applications.

Chapter 3

Parallel architecture and embedding

3.1 Introduction

Optimization is a critical field in operations research and artificial intelligence, aimed at solving complex problems often constrained by hardware capabilities in terms of computational power and memory. In the quest for efficient solutions, parallel computers emerge as powerful tools, utilizing various architectures such as multiprocessors and interconnection networks. Within these parallel architectures, the hypercube network stands out as one of the most renowned topologies, offering remarkable performance.

This chapter delves into optimization in the context of parallel architectures, with a focus on the hypercube interconnection network. It begins by exploring Flynn and topological classifications, detailing the different categories of parallel architectures and connection networks. The section on hypercube networks highlights their properties, advantages, disadvantages, and variations.

Crossed cubes are then introduced as a variant of the hypercube network. Finally, the concept of embedding, or graph embedding, is addressed, emphasizing associated problems, categories, the quality of graph embedding, and the embedding process.

3.2 Flynn classification

In the future, computers of all sizes will leverage parallelism more extensively, unlocking extraordinary possibilities in the coming decade. Understanding applications, algorithms, and architecture will be crucial to seize these opportunities. In 1966, Flynn [49] introduced a model for classifying computers based on parallelism (Figure 3.1), which remains relevant today. This model categorizes computers based on the number of parallel instruction and data streams. The categories include Single Instruction Stream, Single Data Streams (SISD) for monoprocessors, Single Instruction Stream, Multiple Data Streams (SIMD) for processors with one instruction stream and multiple data streams, Multiple Instructions Streams, Single Data Stream (MISD) for processors with multiple instruction streams and one data stream, and Multiple Instructions Streams, Multiple Data Streams (MIMD) for processors with multiple instruction and data streams [2].

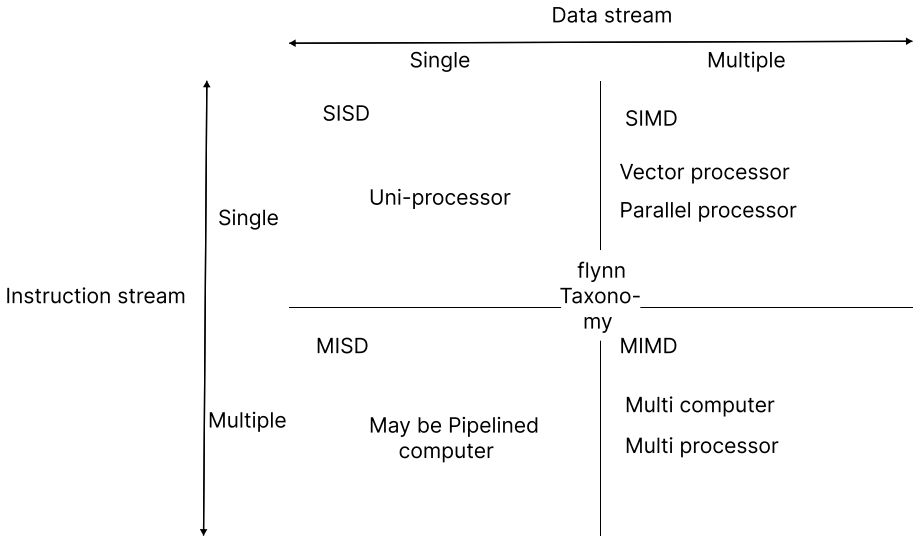


Figure 3.1: Flynn Taxonomy [49].

the SISD model, representing von Neumann architecture, is characterized by a singular memory and processor with a control unit and processing unit. This lacks parallelism, as the control unit reads program instructions from memory and directs the processing unit to operate on data stored in memory. On the other hand, the SIMD architecture involves multiple processing units supervised by a shared control unit, executing identical instructions simultaneously on distinct data sets for synchronous processor operation. The MISD architecture, theoretically capable of concurrently executing multiple instructions on the same data, has not been commercially realized. In the MIMD paradigm, a machine with N processors simultaneously executes diverse instruction sequences on separate data sets, distinguishing between shared and distributed memory models. The shared memory model allows independent processor operation with communication through shared memory. In contrast, the distributed memory model associates each processor with local memory modules interconnected by a network for message transmission, optimizing interaction among processors.

3.3 Topological classification

This categorization comprises three types of connection networks:

- Simple networks.
- Hybrid networks.
- Hypercube networks.

3.3.1 Simple connection networks

In this segment, we aim to elucidate the concept of the simple parallel architecture, delineating its two variants: the simple parallel architecture featuring a linear vector interconnection network and the one with a complete binary tree interconnection network. The fixed connections

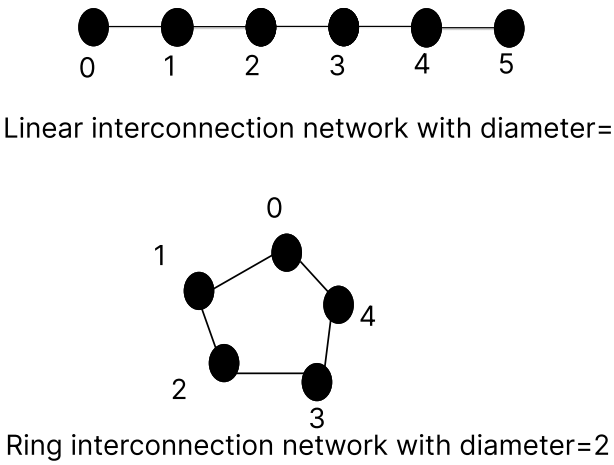


Figure 3.2: Example of vector interconnection networks.

distinguish these architectures between various pairs of processors, which remain unchanged over time. The specific types include the Vector architecture and the Complete Binary Tree architecture.

Vector architecture

Vectors exhibit a structured arrangement where a set of processors is aligned. Each processor has a bidirectional connection with its right neighbor (predecessor) and left neighbor (successor), except for those at the ends that have a single connection and serve as I/O in the network. This configuration is convenient as processors only require two ports for connections to the rest of the network, allowing for flexible expansion without being constrained by the number of ports. Linear and ring network topologies have been widely utilized in early distributed memory multiprocessors. In a linear network, nodes are ordered from 0 to $(p-1)$, with each node having two neighbors except the first and last. The diameter of a linear network is $(p-1)$. In a ring, each node has exactly two neighbors, with the first and last nodes connected. The diameter of a ring is $p/2$. Example of vector interconnection network is shown in Figure 3.2 [94].

Mesh (Vector Composition)

By combining N linear vectors of k-cells, we create a $k * N$ vector (or mesh) by applying the mirror effect. Applying this mirror effect to a k-cell vector yields a similar copy. By prefixing node labels of the first copy with 0 and the other copy with 1 connections between the two copies are established only between k nodes of each copy, under the condition that node labels differ by a single bit. Three cases are to be studied: The first case (Figure 3.3) uses a horizontal generator to obtain a horizontal rectangular $k * N$ mesh. The second case (Figure 3.4) uses a vertical generator to obtain a vertical rectangular $k * N$ mesh. Finally, the third case (Figure 3.4) uses a hybrid generator (vertical and horizontal) to obtain a square $N * N$ mesh [94].

A grid of size $(N * M)$ consists of N rows of M processors. Each processor has four neighbors,

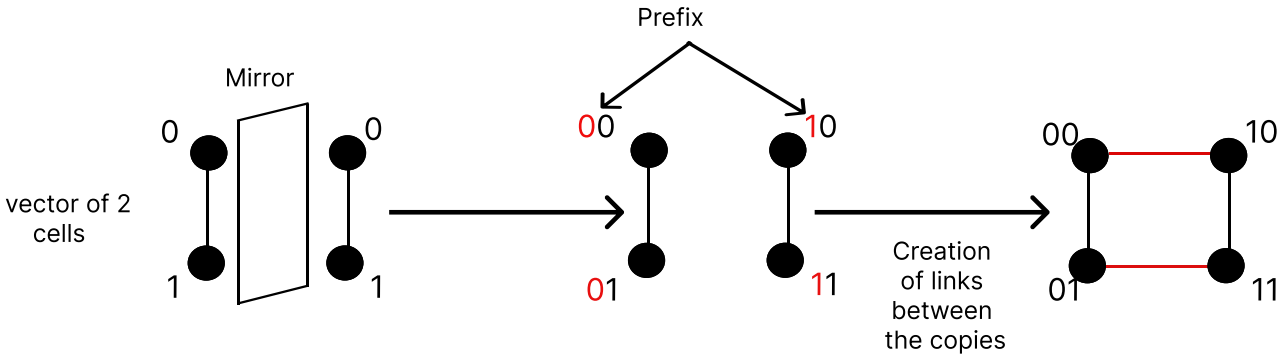


Figure 3.3: Horizontal generator for generating horizontal rectangular of 2*2 mesh (first case).

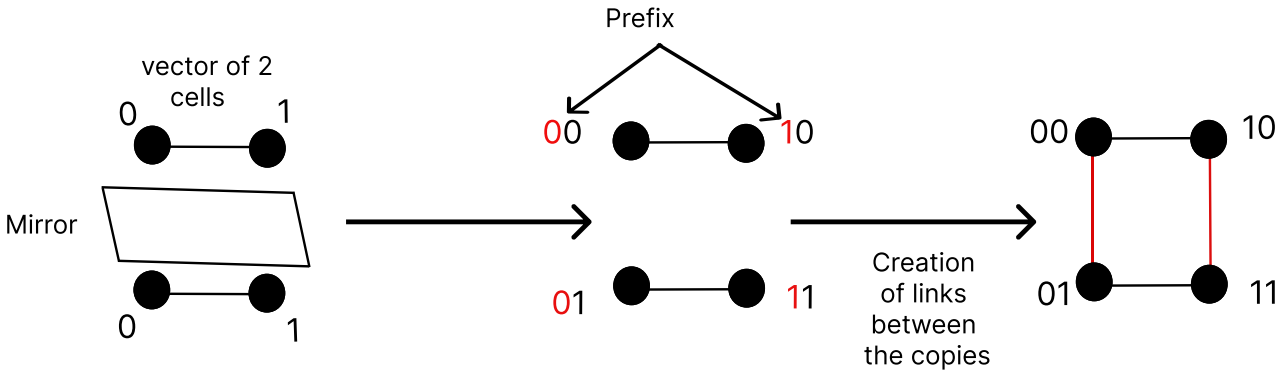


Figure 3.4: Vertical generator for generating vertical rectangular of 2*2 mesh (second case).

except for those located on the first and last rows, as well as the first and last columns. Using the index pair notation (i, j) to represent the jth processor of the ith row (starting from $(0, 0)$), the connections for different vertices are as follows:

- For the interior vertices, $i = 1, 2, \ldots, N-2$ and $j = 1, 2, \ldots, M-2$:

$$V(i,j) = \{(i-1,j), (i+1,j), (i,j-1), (i,j+1)\}$$

- For the vertices of the first row:

$$\begin{aligned} V(0,0) &= \{(1,0), (0,1)\} \\ V(0,M-1) &= \{(1,M-1), (0,M-2)\} \\ \text{For } 1 \le j \le M-2: \quad V(0,j) &= \{(1,j), (0,j-1), (0,j+1)\}. \end{aligned}$$

- For the vertices of the last row:

$$\begin{aligned} V(N-1,0) &= \{(N-1,1), (N-2,0)\} \\ V(N-1,M-1) &= \{(N-2,M-1), (N-1,M-2)\} \\ \text{For } 1 \le j \le M-2: \quad V(N-1,j) &= \{(N-2,j), (N-1,j-1), (N-1,j+1)\}. \end{aligned}$$

- For the interior vertices of the first column:

$$\text{For } 1 \le i \le N-2: \quad V(i,0) = \{(i,1), (i-1,0), (i+1,0)\}.$$

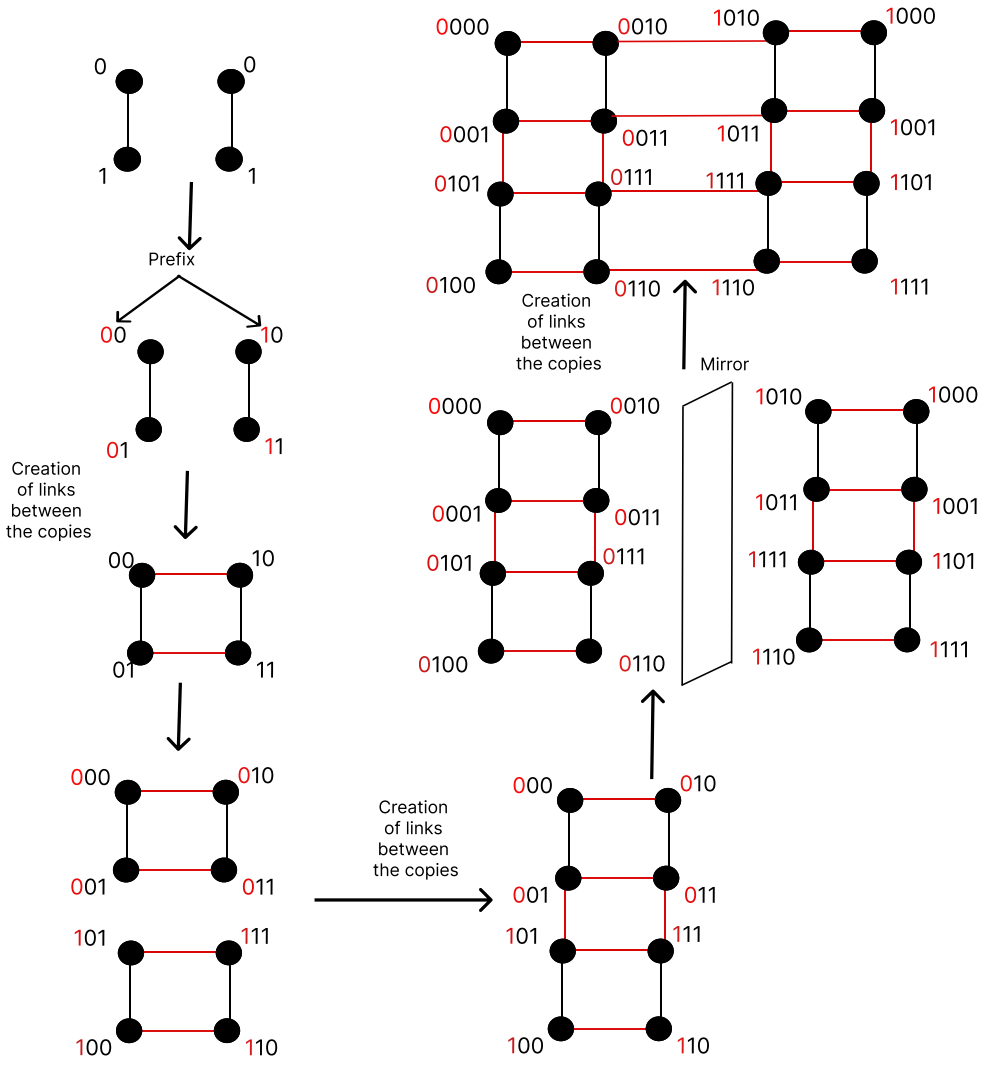


Figure 3.5: Hybrid generator for generating hybrid rectangular of 4*4 mesh (third case).

- For the interior vertices of the last column:

$$\text{For } 1 \leq i \leq N-2: \quad V(i, M-1) = \{(i, M-2), (i-1, M-1), (i+1, M-1)\}.$$

Communication between two opposite vertices of the grid requires traversing $N + M - 2$ links.

In mesh networks, certain properties characterize their structural features. Firstly, the large diameter is defined as the maximum distance between any pair of processors within the network. The distance between two processors is the smallest number of links that must be traversed to move from one processor to another. For a vector N*M mesh, the diameter is specifically determined to be $(N + M - 2)$ according to [94]. Secondly, the small bisection width is another property denoting the minimum number of links that need to be removed to disconnect the network into two identical parts. In the case of a mesh of dimensions $(N * M)$, the bisection width is expressed as the minimum of N and M if the maximum of N and M is even. However, if the maximum of N and M is odd, the bisection width is the minimum of N and M plus one. These properties provide insights into the connectivity and partitioning characteristics of mesh networks. Additionally, mesh networks exhibit a recursive construction methodology. The creation of a mesh $(N * M)$ involves the connection of two copies resulting from the application of the mirror effect, where one is prefixed by 0 and the other by 1. This recursive approach contributes to the scalability and self-replicating nature of mesh networks [94].

Trees

A tree is defined as a connected graph without cycles, where the notion of orientation is not considered. Various theorems define the characteristic properties of trees. The first theorem states that for a tree $G = (X, U)$ with $|X| = n \geq 2$, the following properties are equivalent: 1) G is connected and acyclic, 2) G is acyclic and has $n-1$ edges, 3) G is acyclic and maximal in this property (adding an edge creates a unique cycle), 4) G is connected and has $n-1$ edges, 5) G is connected and minimal for this property (removing any edge makes it disconnected), and 6) There exists a unique chain in G that connects all pairs of vertices. The second theorem states that if G is a connected graph with at least one edge, the following conditions are equivalent: 1) G is strongly connected, 2) every edge in G belongs to a circuit, and 3) G does not contain a co-circuit. These theorems establish fundamental relationships between connectivity, absence of cycles, and other structural properties in the context of trees [94].

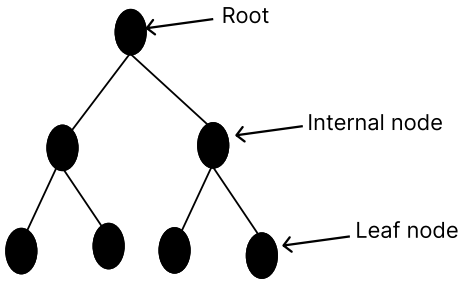


Figure 3.6: Complete Binary tree of n=3 .

Binary trees and Complet binary trees
Analogous to representing hierarchical structures, trees, particularly binary trees, offer a natural way to depict hierarchical machines. In a binary tree, each node can have at most two children, distinguished as a left child and a right child (Figure 3.6). The concept of completeness in a binary tree, denoted as a Complete Binary Tree (CBT), involves a connected graph without cycles containing $(2^n - 1)$ vertices, where non-terminal vertices have a depth less than n, each possessing exactly two children (left child and right child). The average degree of a node is 3, with the root having a degree of 2, and the leaf nodes having a degree of 1. The tree's diameter is $2\log_2(N)$, where N is the number of vertices, ensuring that all nodes are at a distance less than $\log_2(N)$ from the root. Various convenient numbering schemes exist for processors in a complete binary tree, with popular representations being binary and integer-based. These numbering principles maintain a left-to-right increasing order within the same level. The advantages of a complete binary tree include its application in divide-and-conquer algorithms. However, drawbacks include a large diameter $2\log_2(N)$ and a narrow bisection width of 1, meaning that removing a single edge results in network failure [94]. The tree lacks symmetry in nodes and arcs due to the unique characteristics of the root and leaves, posing challenges in simultaneous communication between processors [63]. These factors make it crucial to consider alternatives to address potential bottlenecks in communication flow within the tree structure.

The Complete Binary Tree with a Double Root (CBDRT) graph closely resembles the CBT. The CBDRT, specifically designed with N nodes, is essentially a complete binary tree where the root is replaced by a path of length two, connecting two nodes [94]. See Figure 3.7.

Quadtrees [130, 132] and Octrees [128]
Quadtrees and octrees are hierarchical data structures that extend the concepts of binary trees into 2D and 3D space. In a quadtree, each node can have up to four children, representing quadrants, while an octree allows up to eight children for octants. The number of vertices in these structures grows exponentially with depth, providing a hierarchical spatial organization. The diameter of quadtrees and octrees exhibits logarithmic growth, ensuring efficient spatial indexing with a bound on the distance between points. The degree, representing the number of children per node, is four for quadtrees and eight for octrees, influencing spatial granularity. Bisection width, indicating the minimum edges to split a structure into identical halves, is proportional to the square root and cube root of the total nodes for quadtrees and octrees, respectively. These structures offer efficient spatial representation and manipulation in 2D and 3D, with characteristics that make them valuable for various applications.

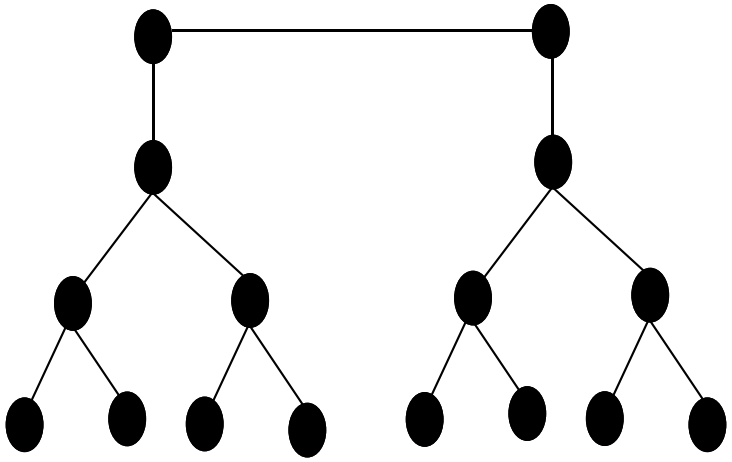


Figure 3.7: Complete Binary tree of double root.

3.3.2 Hybrid connection networks

The two-dimensional tree mesh $(n * n)$ is formed by a grid of $n \times n$ processors with the addition of complete binary trees in each row and column. The tree leaves are the nodes of the grid, and the internal nodes of the trees are additional nodes. Processors at leaves and roots have a degree of 2, while other internal processors have a degree of 3, totaling $3N^2 - 2N$ processors. This structure has a small diameter of $4 \log_2 N$, a large bisection width, and a recursive decomposition into 4 copies of tree meshes $(\frac{N}{2} \times \frac{N}{2})$ when removing the roots of the $2N$ row and column. The two-dimensional tree mesh provides a derivation of $K_{N,N}$, the complete graph, by replacing each N-degree node with a complete binary tree [94].

The N-dimensional tree meshes are formed by adding trees to an N-face and r-dimensional grid. The grid's leaves become the tree leaves, and the r-dimensional tree mesh contains $|v| = (r+1)N^r - rN^{r-1}$ nodes and $|E| = 2rN^r - rN^{r-1}$ edges distributed among rN^{r-1} complete binary trees of N leaves. The network's diameter is $2r \log_2 N$, and the bisection width is $N^{r-1} = \frac{|v|}{rN}$. These structures exhibit important characteristics for parallel computing, including a small diameter and a large bisection width, particularly useful when r is large and $N = 2$, enabling the construction of hypercubes of dimension $-r$ [94].

3.3.3 Hypercube connection networks

The hypercube denoted Q_d, a network with $N = 2^d$ nodes and $d \cdot 2^{(d-1)}$ edges, is characterized by nodes labeled with d-bit binary numbers. Connectivity is established between nodes differing in exactly one bit. See Figure 3.9. In a three-dimensional setting $(d = 3, N = 8)$, three families of connections (C_0, C_1, and C_2) exist based on variations in the least significant, intermediate, and most significant bits, respectively. For instance, in the C_0 family, node 0 (binary 000) connects to node 1 (binary 001), forming connections with a Hamming distance of 1. Gray codes provide a practical numbering scheme, allowing each node to be encoded with d binary bits. Nodes in the 3-cube, for example, are connected if their Gray code representations differ in one bit. This systematic approach ensures a structured network where edges can be classified based on the dimensions they traverse, with a K-dimensional edge connecting nodes

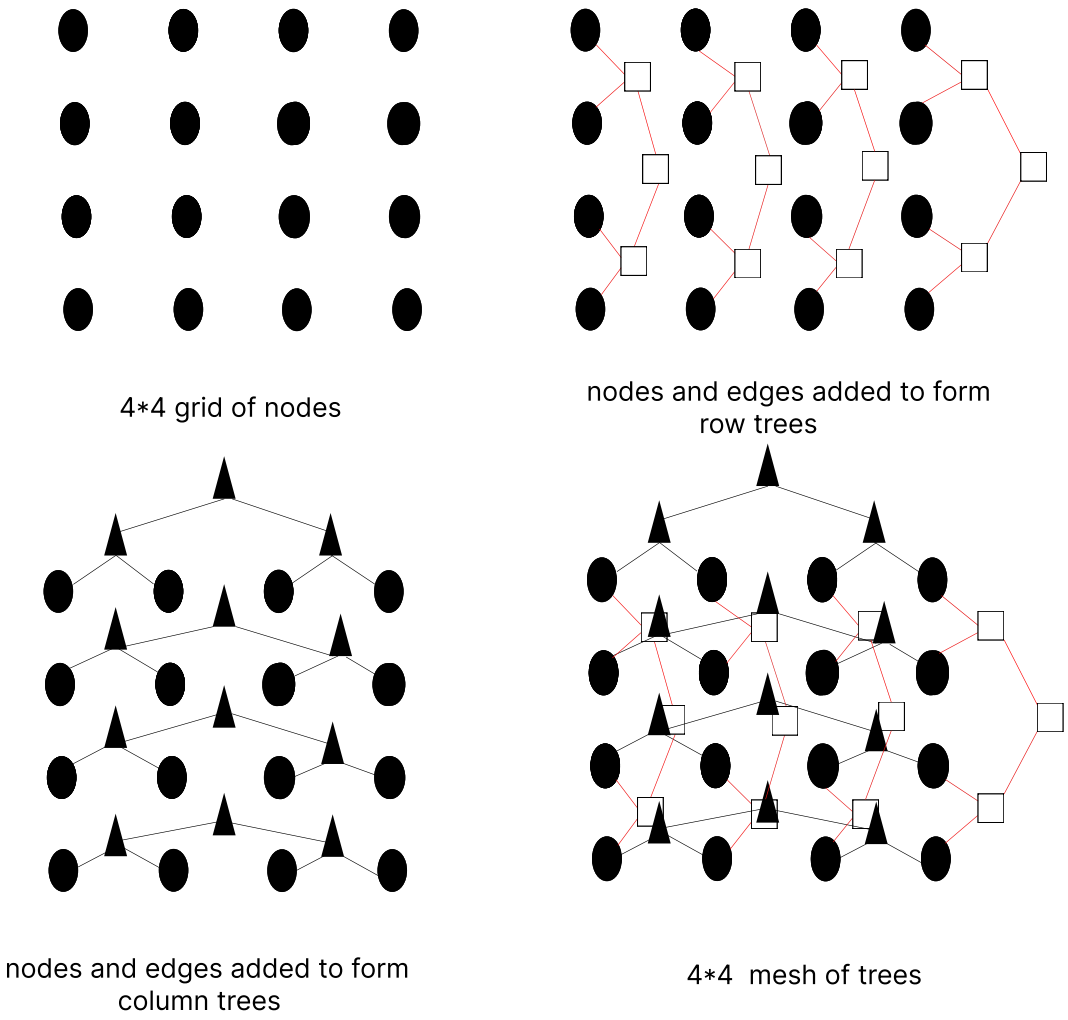


Figure 3.8: construct a two-dimensional tree mesh.

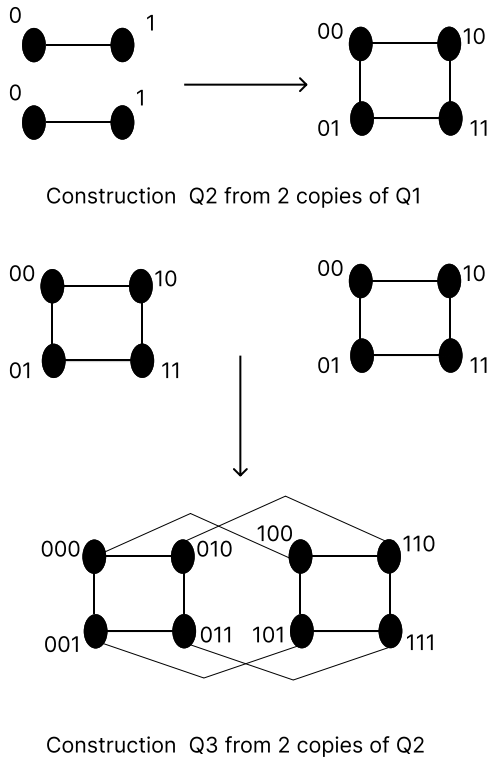


Figure 3.9: Construction hypercube of dimension 2 and 3.

differing in the K-th bit position [94].

Properties of the Hypercube

The hypercube possesses crucial properties that make it a favored topology in modern computer architectures, endowing it with power, flexibility, and efficiency. Firstly, the hypercube is strongly connected, ensuring that there is a path between any two nodes in the graph. Additionally, the hypercube exhibits a small degree compared to other architectures, with the degree being equal to the dimension of the hypercube. The regular topology of the hypercube ensures that all nodes have the same degree. Notably, the hypercube demonstrates various symmetries, being node and edge symmetric. It can be mapped such that any node corresponds to another node, and any edge corresponds to another edge. Furthermore, the hypercube is a Cayley graph, making it Hamiltonian. In addition to its simple and recursive structure, the hypercube possesses other important properties, such as a small diameter ($\log_2 N$) when N is small(Figure 3.11) and a large bisection width ($N/2$) as shown in Figure 3.10. The connectivity of nodes concerning diameter is easily demonstrated by observing that any two nodes $U = U_1U_2...U_{\log N}$ and $V = V_1V_2...V_{\log N}$ are connected by a specific path. Finally, the construction of a hypercube is recursive, as a hypercube of N nodes can be constructed from two hypercubes of $N/2$ nodes by connecting corresponding nodes between them [94, 9, 156, 1, 117, 155].

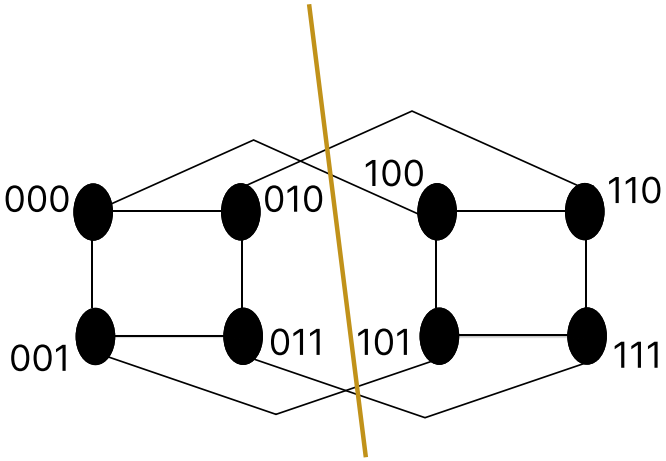


Figure 3.10: Bisection width of Q_3. Bisection width=8/2.

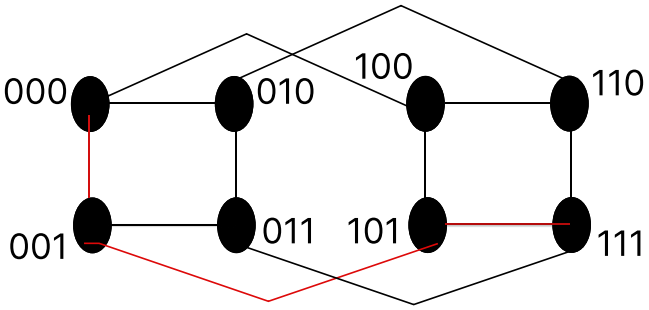


Figure 3.11: Diameter of Q_3. Diameter=3

The Networks Within a Hypercube

The hypercube with N nodes encompasses various subnetworks, each possessing distinct properties and applications. Notably, it contains a linear vector of N nodes, representing a compelling feature of the N-node hypercube. This is supported by the lemma stating that the hypercube of N nodes includes a linear vector of N cells as a subgraph, a property valid for $N \geq 4$. Additionally, the hypercube easily gives rise to rings, where an N-power-of-2-sized ring is a partial graph of the corresponding hypercube. Another valuable result involves toroidal grids, as square toroidal grids with a power-of-2 number of processors are shown to be partial graphs of the hypercube. Emphasizing the significance of tree meshes, considered the most powerful for parallel processing, they are highlighted as essential networks within the hypercube. Lastly, an interesting inclusion is the complete binary tree with a double root, recognized as a partial graph of the N-node hypercube. These diverse networks within the hypercube play crucial roles in parallel computing and contribute to its versatility and computational power[94].

Advantages and disadvantages

The hypercube stands out as an excellent choice for parallel computing interconnection networks due to its numerous advantages. It features a small number of connections per

processor, a high vertex count allowing for massive hardware parallelism, a small diameter, and a substantial bisection width (bisection width = N/2). Additionally, it exhibits fault tolerance, making it resilient to processor failures as replacements can be seamlessly integrated. The hypercube adapts well to various architectures such as vectors, trees, rings, grids, and tree meshes, facilitating a broad range of computations. Despite its computational prowess, there are notable drawbacks, including the node degree increasing with size, hindering backward compatibility, and challenges in hardware construction when dealing with networks of different sizes and link lengths. While the hypercube dominated the market for distributed memory MIMD machines in the 1980s, its design constraints and potential scalability issues warrant consideration when choosing an architecture for parallel computing [94].

Importance of the hypercube

Extensive research has focused on the hypercube, highlighting its rich interconnection structure, logarithmic degree, straightforward routing, fault tolerance, and ability to simulate other networks with a low load factor [109, 87, 152, 45, 160]. Various studies have demonstrated its computational power, showcasing its capability to simulate networks such as the mesh, tree, and star with minimal load [12, 90, 13]. The hypercube is recognized as a robust and fault-tolerant machine, capable of reconfiguring itself in the presence of faults in links or nodes [151, 44, 121]. Research has also explored modifications to its structure to enhance its computational capabilities [109, 87].

Variations of the hypercube

Despite the considerable computing power of the hypercube, it has drawbacks, notably the increase in node degree and diameter as it scales in size. To address these limitations, computer scientists have proposed variations of the hypercube aimed at mitigating these challenges associated with node degree and diameter. Several variations have been developed, among which the most significant include the Butterfly network, the Cube-Connected Cycle, the Benes network, and the Crossed cubes [94].

The Butterfly network of dimension d has $(d+1)2^d$ nodes and $d2^{d+1}$ edges. Nodes are pairs (w, i), where i is the level ($0 \leq i \leq d$) and w is a d-bit binary number denoting the row. Nodes (w, i) and (w', i') are connected if $i' = i + 1$ and w and w' satisfy specific conditions. The Butterfly network can simulate N-node hypercube computations in $\log_2 N$ steps, resembling the hypercube's structure. It exhibits a simple recursive nature, comprising two copies of a $d-1$-dimensional Butterfly network, and a unique path of length d between level 0 in any row w and level d in any row w'. Like the hypercube, it has a substantial bisection width of $\theta = (N/\log_2 N)$ [94].

The Benes network, obtained by concatenating Butterfly networks, comprises $2d+1$ levels, each with $2d$ nodes. The first and last $(d+1)$ levels constitute the Butterfly network of dimension d, sharing the central level. Similar to the Butterfly network, each node in level 0 can have two inputs, and each node in level $2d$ can have two outputs. The network facilitates connections to ensure every input permutation is linked to outputs through disjoint edge paths [94].

The Cube-Connected Cycles (CCC) is an N-dimensional hypercube variant where each vertex is replaced by a cycle of N Processor Elements (PEs). CCC maintains a fixed degree of three, two in the cycle and one in the hypercube, offering an advantage over Hypercubes as its degree doesn't grow with the dimension or the number of processors. Represented by pairs (w, i), nodes connect in CCC based on conditions: 1) $w = \hat{w}$ and $i = \hat{i} = \pm 1 \mod N$, or 2) $i = \hat{i}$ and w differs from $\hat{w}$ in the i-th bit position. CCC has $N \cdot 2N$ vertices and a diameter of $d_{CCC} = 2N - 2 + \frac{N}{2}$ for $N > 3$ [94].

3.4 Crossed Cubes

The n-dimensional hypercube, denoted as Q_n, and its counterpart, the crossed cube CQ_n, both share a set of vertices, each associated with a binary string of length n. In Q_n, vertices are considered adjacent if their binary labels differ in precisely one-bit position. On the contrary, CQ_n defines adjacency differently. For binary strings of length two, $x = x_1x_0$ and $y = y_1y_0$, x and y are deemed pair-related only if they belong to the set $\{(00, 00), (10, 10), (01, 11), (11, 01)\}$ [37].

The properties of the Crossed cubes (CQ_n) play a crucial role in understanding its topological characteristics. To facilitate the discussion of these properties, a mechanism for identifying subgraphs is introduced. Specifically, $\Gamma A, B(G)$ represents a subgraph of G in which each node is prefixed by either A or B. Similarly, $\Gamma A(G)$ is a subgraph of G with nodes prefixed by A [37].

In terms of composition and decomposition, the construction of CQ_n is recursive. For $n \geq 2$, the isomorphism rules for the Crossed Cube network CQ_n can be summarized as follows: $\Gamma_0(CQ_n) = CQ_{n-1}$ and $\Gamma_1(CQ_n) = CQ_{n-1}$. The isomorphism involves removing the prefix 0 for nodes belonging to $\Gamma_0(CQ_n)$ and the prefix 1 for those in $\Gamma_1(CQ_n)$; see Figure 3.12. Additionally, for any $K \geq 1$, the specific rules $\Gamma_{00,10}(CQ_{2K}) = CQ_{2K-1}$ and $\Gamma_{01,11}(CQ_{2K}) = CQ_{2K-1}$ are defined, with the isomorphism achieved by removing the bit at position $(2K-2)$ for each node label (Figure 3.12). Furthermore, for any $K \geq 1$, the rule $\Gamma_{A,B}(CQ_{2K+1}) = CQ_{2K-1}$ is established for each pair A, B belonging to $\{(001, 111), (011, 101), (000, 100), (010, 110)\}$. The isomorphism is performed by removing the bits at positions $(2K - 1)$ and $(2K - 2)$ for each node label in $\Gamma_{A,B}(CQ_{2K+1})$; see Figure 3.14 [37].

Moving on to composition as shown in /Figure 3.13, CQ_n with N nodes can be constructed from two copies of CQ_{n-1}, each containing $N/2$ nodes, interconnected according to a defined pattern. The graph exhibits a small diameter (Figure 3.15), specifically $\lceil (n + 1)/2 \rceil$, and maintains the same diameter for CQ_n and CQ_{n-1} for $K \geq 1$ [37]. CQ_n demonstrates high connectivity (Figure 3.17), effectively preventing congestion, with a connectivity function $K(CQ_n) = n$ for $n \geq 1$. The graph also possesses a wide bisection width $(N/2)$ (Figure 3.16) and regular topology, where all nodes have the same degree equal to n. Additionally, CQ_n features Hamiltonian cycles for $n \geq 2$ (Figure 3.18) and exhibits symmetry up to $n = 4$, while nodes in larger graphs become dissimilar [37, 28, 42, 163, 91, 143].

In general, CQ_n with 2^n nodes contains multiple classes of nodes, each class exhibiting similarities among its members. The graph is strongly connected, ensuring the existence of a

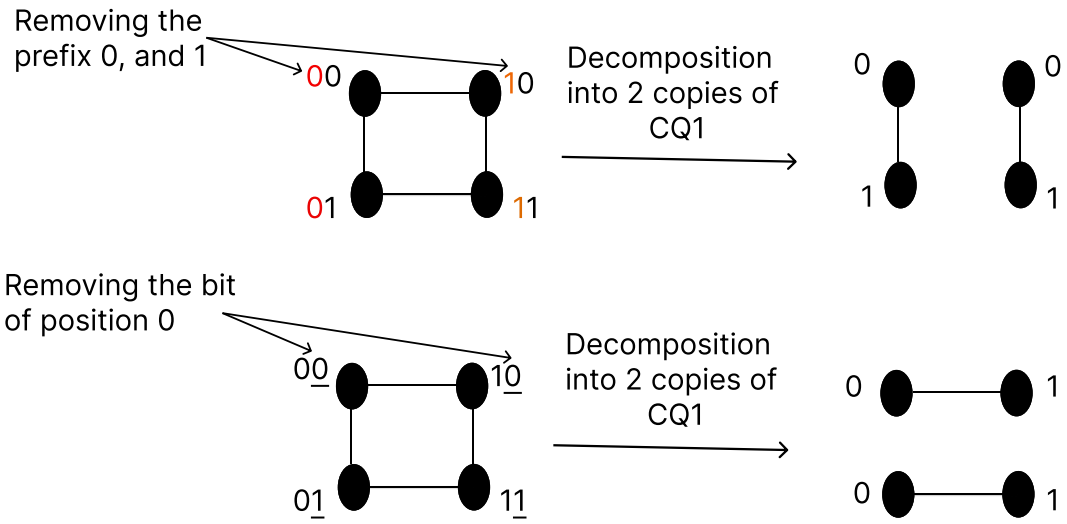


Figure 3.12: An example illustrating the operation of decomposition (rule 1, and 2).

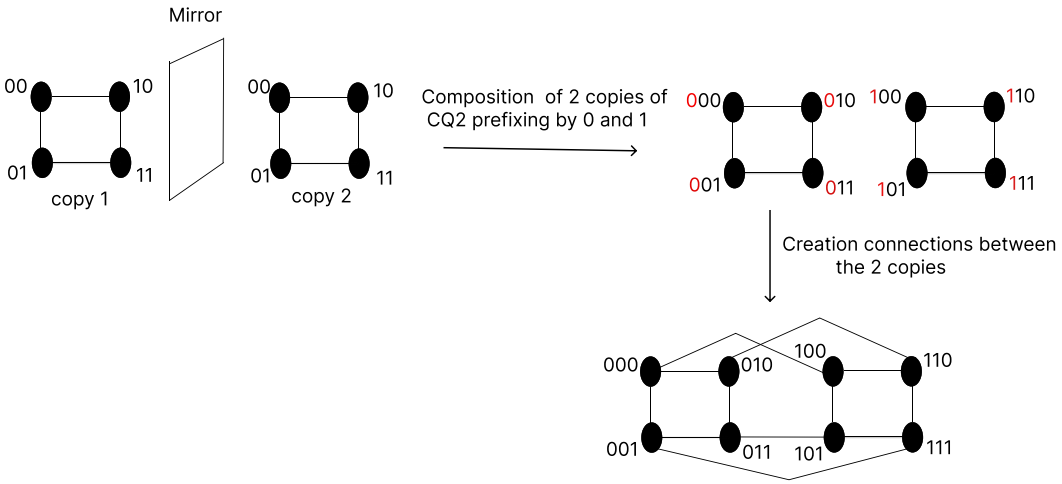


Figure 3.13: An example illustrating the operation of composition.

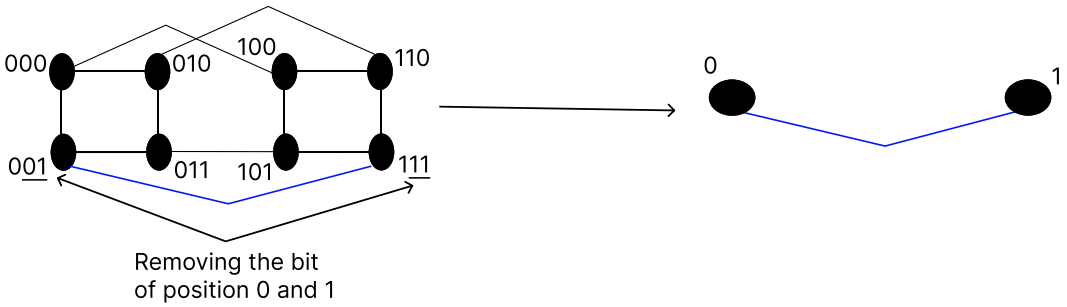


Figure 3.14: An example illustrating the operation of decomposition (rule 3).

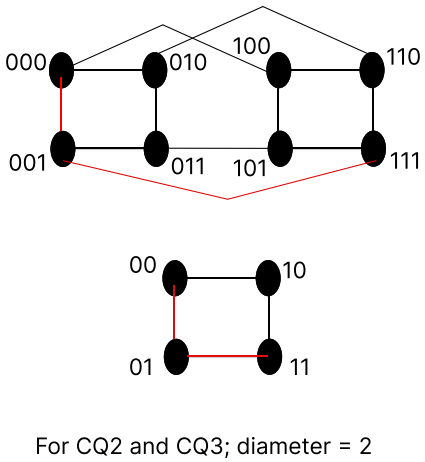


Figure 3.15: Diameter of CQ_2 and CQ_3.

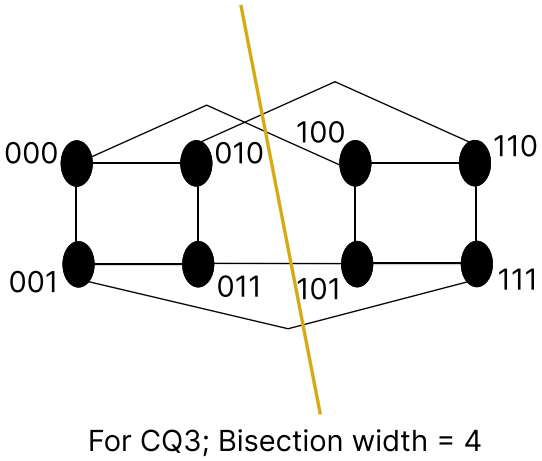
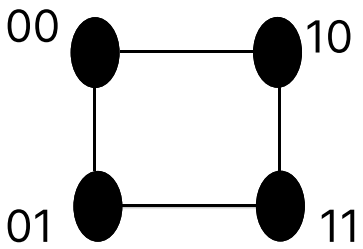


Figure 3.16: Bisection width of CQ_3.

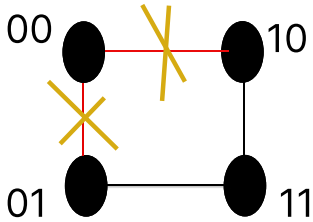
path connecting all nodes.

The in-depth study of the crossed cubes, introduced in [38], reveals its ability to preserve many attractive properties of the hypercube. The most significant is the reduction of the diameter by a factor of two. This reduction has a positive impact on the communication time between processors and the number of parallel computing steps, respectively halving the data transmission time and the number of steps in addition and prefixing operations used in scientific calculations [162]. The study also demonstrates that the crossed cubes is fault-tolerant due to its regular topology, high degree of connectivity, and node symmetry organized into five classes where nodes are all similar. Finally, the study shows that the crossed cubes has a significant capability to simulate other architectures such as the complete binary tree, the double-rooted tree, the hypercube, the quadtree, and the octree.



For CQ2; The Hamiltonian cycle is: 00-01-11-01-00

Figure 3.18: Hamiltonian cycle in CQ_2.



For CQ2; eliminating two edges
disconnecting the network

Figure 3.17: An example illustrating the connectivity of CQ_2.

3.5 Embedding

The implementation of parallel algorithms on distributed memory multiprocessor architectures has led to the development of the concept of embedding a source graph G into a host graph H. Frequently, a distributed algorithm A is described assuming the existence of a logical topology S on which A is defined, with commonly used logical topologies including trees and meshes. The method used to deploy an algorithm A and its associated logical structure S onto a network R is the simulation of S in R. This involves embedding only the structure of the source graph S into the host graph R, facilitating an effective adaptation of algorithm A to the topology of R.

3.5.1 Embedding problems

Graph embedding encompasses a spectrum of challenges, including the organization of processes on processor networks, and the simulation of one network architecture on another. For process organization, the efficient embedding of a graph representing the network of original processes into the graph representing the processor network leads to effective process coordination. This involves nodes representing processes and edges symbolizing communications between sub-processes. Similarly, in simulating one architecture on another, both architectures

are represented by graphs. This versatility extends to various parallel computing applications, from sorting and matrix computation to image processing. The power lies in deploying efficient algorithms across diverse architectures, ensuring portability and adaptability in parallel computing [40].

3.5.2 Definitions

Definition 1: An embedding of graph G into graph H, denoted as (P, R), consists of an injective mapping P from the vertices of G to the vertices of H, and an injective mapping R associating each edge $[u, v]$ of G with a path $R(u, v)$ connecting $P(u)$ and $P(v)$.

Definition 2: Alternatively, an embedding of graph G into graph H can be defined by an injective mapping Qq or σ from the set of vertices of G to the set of vertices of H, and a mapping PQq from the set of edges of G to the set of edges of H. PQq associates each edge (x, y) of G with a path connecting the vertices $Qq(x)$ and $Qq(y)$ in H [40].

3.5.3 Embedding types

There are three types of embeddings for a source graph G into a host graph H:

Many by One: This type is defined by mapping several nodes of G to a single node in H. It is typically employed when the cardinality of G is strictly greater than that of H:

$$|V(G)| > |V(H)|$$

Consider graphs $G = (V(G), E(G))$ and $H = (V(H), E(H))$. Example shown in Figure 3.19

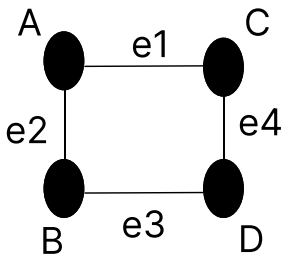
One by Many: This type is defined when the cardinality of G is strictly less than that of H: $|V(G)| < |V(H)|$. In this embedding, the concept of adjacency plays a crucial role. Example shown in Figure 3.20

One by One: This embedding type is often used when the cardinality of the set of nodes is equal or nearly equal to that of H. It establishes a correspondence between one node in G and one node in H. Example shown in Figure 3.21

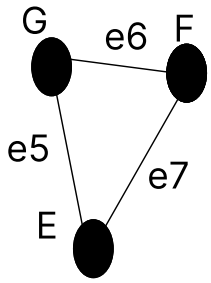
3.5.4 Quality of graph embedding

Evaluating the quality of graph embeddings involves various measures tailored to different optimization goals [75]. Key metrics include dilation, which gauges the spread of images in the destination graph for neighboring vertices in the source graph [75, 46]. Congestion assesses how many chains in the destination graph contain a specific edge, while expansion quantifies the ratio of vertices in the destination graph to the source graph. The node load factor characterizes the distribution of processes from the source to the destination, defining the maximum nodes in the source corresponding to a single node in the destination. These measures collectively provide insights into the efficiency and characteristics of diverse graph embeddings in network-based architectures [75].

Dilation: $dil(Qq) = \max\{dist(Qq(x), Qq(y))\}$ represents the dilation of the embedding Qq, where $dil(Qq)$ is the maximum length of distances between the images of vertices x and y



Graph G



Graph H

Nodes Embedding

σ(A) ↪ G

σ(B) ↪ G

σ(C) ↪ F

σ(D) ↪ E

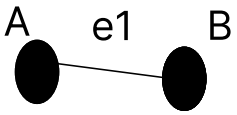
Edges Embedding

σ(e1) ↪ e6

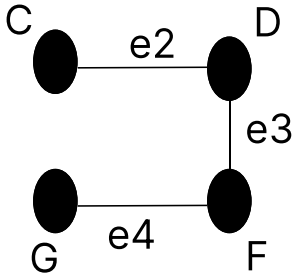
σ(e3,e2) ↪ e5

σ(e4) ↪ e7

Figure 3.19: Example of the many by one embedding .



Graph G



Graph H

Nodes Embedding

σ(A) ↪ C

σ(B) ↪ G

Edges Embedding

σ(e1) ↪ e2,e3,e4

Figure 3.20: Example of the one by many embedding .

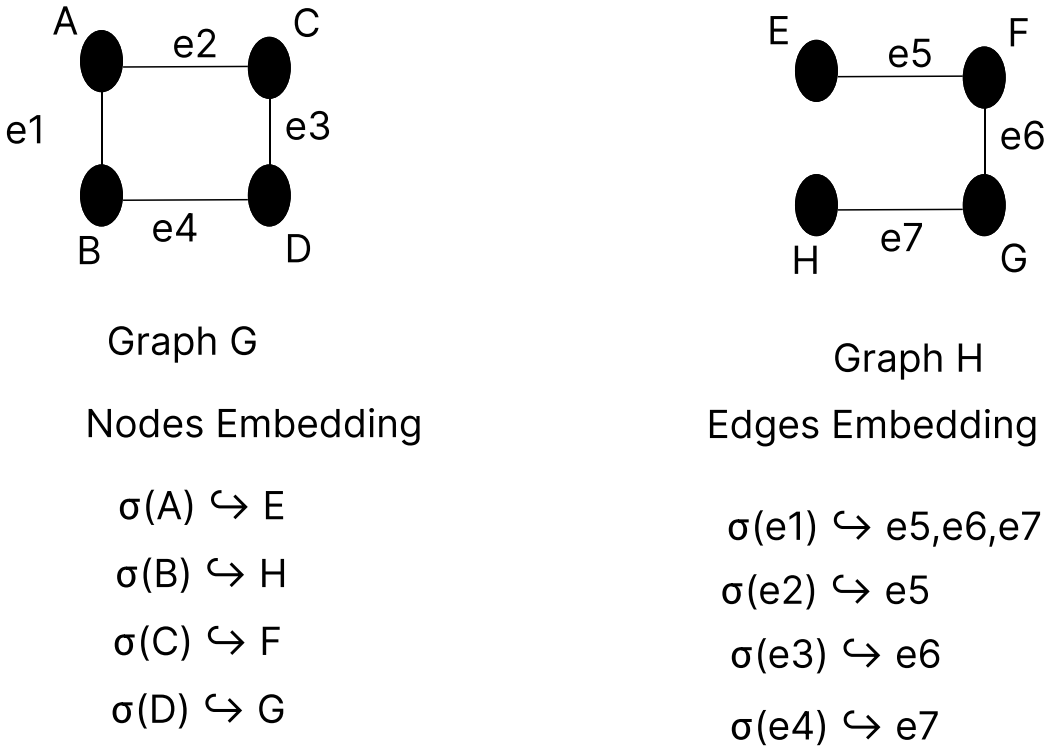


Figure 3.21: Example of the one by one embedding .

in the host graph H. In simpler terms, it measures the maximum stretch or elongation of edges in the embedding, indicating how much the distances between connected vertices in the source graph G are expanded when embedded into the host graph H. See the example in Figure 3.22.

Congestion: The congestion of a host edge is defined as the maximum number of images of source edges passing through a single edge in the host graph. See the example in Figure 3.22.

Expansion: The expansion $exp(Qq)$ is defined as the ratio of the number of vertices in the embedding graph H to the number of vertices in the guest graph G. Mathematically, it is expressed as $exp(Qq) = \frac{|X(H)|}{|X(G)|}$. See the example in Figure 3.23.

Load Factor: The load factor F_g is defined as the maximum number of guest graph nodes assigned to a single vertex in the host graph. It is represented by $(F_g = \max(Qq(x)), \forall x \in V(G))$, where Qq is an injective application of the embedding. See the example in Figure 3.24.

3.5.5 Process of embedding

The process of embedding a source graph G into a host graph H involves several essential steps. Firstly, the cardinality of the source graph is calculated to determine the size of the graph to be embedded. Next, the choice of the embedding method comes into play, with two distinct options. The first method, embedding by excess, involves selecting a destination graph whose cardinality is equal to or slightly greater than that of the source graph. Conversely,

Figure 3.22: Embedding with dilation and congestion equal to 2 .

Figure 3.23: Embedding with expansion equal to 3/2 .

Figure 3.24: Embedding with load factor equal to 2.

the second method, embedding by default, selects a destination graph with a slightly lower cardinality than that of the source graph. The process of embedding nodes follows, ensuring the integration of nodes from the source graph into the destination graph while preserving neighborhood relationships. Finally, the embedding of edges relies on the prior embedding of nodes, where a source edge can correspond to an edge or a path in the host graph. These detailed steps contribute to the creation of an effective and structured embedding of the source graph into the host graph.

3.5.6 The need for embedding

The significance of embedding techniques becomes pronounced with the emergence of new architectures. Researchers often leverage embedding methodologies to integrate established architectures into these novel architectures. This approach not only streamlines the transition to new platforms but also leverages the collective knowledge and optimizations embedded within the existing architectures. By embracing embedding techniques, we economize time and resources, bypassing the need to create representations from scratch and instead focusing on embedding solutions into host architectures. In essence, embedding techniques serve as a bridge between legacy architectures and the next generation of computing, facilitating seamless integration and innovation in computational paradigms.

Recent research exemplifies this trend, with scholars such as H. Zhang pioneering the parallel construction of independent spanning trees on Folded Crossed Cubes [159]. Similarly, Wang's work focuses on constructing completely independent spanning trees within a family of line-graph-based data center networks [144]. Yang's contributions include the construction of multiple independent spanning trees on burnt pancake networks. Additionally, H. Liu proposes innovative modifications to l-embedded edge-connectivity within enhanced hypercubes [99], while Kao introduces a parallel algorithm for constructing multiple independent spanning trees in bubble-sort networks [82]. This collective body of research underscores the vital role of embedding techniques in advancing the capabilities and versatility of computing architectures, driving forward progress in computational science and engineering.

3.6 Conclusion

In conclusion, this chapter has delved into the critical domain of optimization within parallel architectures, with a specific focus on the hypercube interconnection network. By exploring classifications, properties, and variations of the hypercube, as well as introducing crossed cubes and the concept of embedding, we have gained valuable insights into the powerful tools and challenges within this field. The hypercube's remarkable performance positions it as a standout solution for complex problems in parallel computing. This chapter sets the stage for further advancements and applications at the intersection of optimization, parallel computing, and interconnection networks.

Chapter 4

Case of study: The Traveling Salesman Problem (TSP)

4.1 Introduction

In this chapter, we embark on a detailed examination of solving combinatorial problems, focusing on a prominent case study: the Traveling Salesman Problem (TSP). Combinatorial problems, characterized by the need to find an optimal arrangement or selection from a finite set of elements, pose significant challenges across various domains, including logistics, scheduling, and network optimization.

The TSP, a quintessential combinatorial optimization problem, involves determining the shortest possible route that visits a set of cities exactly once and returns to the origin city. This problem encapsulates the complexities inherent in combinatorial optimization, requiring efficient algorithms and strategies to find optimal or near-optimal solutions.

This chapter begins with an overview of the TSP, including its definition, formal model, and datasets utilized for experimentation. It then proceeds to examine classical methods for solving the TSP, encompassing exact solving techniques, heuristic approaches, and metaheuristic strategies such as tour construction and improvement algorithms.

Furthermore, the chapter explores hybrid TSP-solving methods, which integrate optimization techniques such as metaheuristics and machine learning. Additionally, parallel TSP solving is discussed, focusing on problem formalization for parallelization, decomposition, and alternative representations.

The chapter provides a detailed overview of the proposed model methodology, emphasizing the feature selection, processing, and visualization modules. Each module is essential to the comprehensive solution framework developed for addressing the TSP. Various methodologies and techniques are explored throughout the chapter, laying the groundwork for a thorough investigation into solving the TSP.

4.2 The problem

4.2.1 Introduction

The Traveling Salesman Problem (TSP), whose origins remain uncertain, emerged from the practical need to optimize routes for 19th-century salesmen traveling between cities. Over time, it evolved into a fundamental problem in combinatorial optimization. In the 1950s, researchers recognized its mathematical significance, applying it to diverse fields such as school bus routing, genome sequencing, industry, astronomy, and scheduling problems. The TSP represents the challenge of finding the shortest tour that visits a set of cities exactly once, reflecting real-world problems faced by salesmen and modern applications alike, where efficient routes are crucial.

4.2.2 Definition

The Traveling Salesman Problem (TSP) involves finding a Hamiltonian cycle $HC \in G$ that minimizes the total weight $w(HC)$ in a given undirected graph G. It is assumed, for simplicity, that each edge in the graph appears at most once, and there are no self-loops (i.e., arcs of the form (u, v) where $u = v$).

4.2.3 Formal Model

The Traveling Salesman Problem (TSP) can be formulated in a graph as the task of finding the minimum-cost cycle that traverses all nodes exactly once. A standard integer programming model captures this objective as follows:

$$
\begin{aligned}
&\text{Minimize } \sum_{e \in U} w(e) \cdot x_e \quad (2.1) \\
&\text{Subject to the degree constraint: } \sum_{e \in U(i)} x_e = 2 \quad (2.2) \\
&\text{Subtour constraint: } \sum_{i \in S, j \in S, i<j} x_{(i,j)} \leq |S| - 1 \quad (2.3) \\
&\text{Decision variables: } x_e \in \{0, 1\} \quad (2.4)
\end{aligned}
\tag{4.1}
$$

In this formulation, a solution with $x_e = 1$ indicates that edge e belongs to the minimum-cost cycle, while $x_e = 0$ implies exclusion. The degree constraint (2.2) enforces that each node participates in exactly two edges, ensuring the formation of a cycle. The subtour constraint (2.3) prevents the existence of cycles within node subsets, except for the complete set of nodes. The objective function (2.1) seeks to minimize the overall cost of the solution. Solving this integer programming problem provides an optimal solution comprising a unique cycle that visits all nodes in the graph.

4.2.4 The euclidean TSP

In the Euclidean version of the TSP, the cities are positioned in a Euclidean space such as on a map. The distance between each city pair is the Euclidean distance between their coordinates. This adds geometric constraints compared to more abstract TSP problem formulations where distances are not based on physical locations in a space. This makes the Euclidean TSP applicable to routing problems where terrain or physical infrastructure must be navigated efficiently.

The ability to represent distances as exact Euclidean calculations makes the problem simpler to define computationally compared to more abstract TSP problems. However, it retains the NP-hard optimization challenge, making the Euclidean TSP a standard benchmark for developing new heuristics approaches that can exploit geometric properties. The regular problem structure also facilitates the theoretical analysis of approximation algorithms.

4.2.5 Datasets

In the context of experiments on TSP landscapes, it is recommended to leverage the benchmark TSP instances available in the comprehensive TSPLIB library. This repository encompasses a wide range of TSP instances, varying in size from small to large problem instances.

4.2.6 The Classic Modeling of the problem

Representation

The solutions of the TSP belong to the space 1, 2, ..., N. They are represented using a vector of N integers, where each integer corresponds to a city.

Evaluation

The evaluation of a solution involves calculating the total distance or cost of the TSP tour. It is the sum of the distances between consecutive cities in the order specified by the solution vector.

Neighborhood

The neighborhood relation chosen for the TSP is based on the concept of swapping two cities. For any solution in 1, 2, ..., N, the neighboring solutions are those obtained by swapping the positions of two cities. The size of the neighborhood for a solution is equal to N choose 2, as there are N(N-1)/2 possible swaps.

Initialization

We create a random permutation of cities to initialize a solution for the TSP. This involves randomly shuffling the order of cities in the solution vector.

4.3 The Classic TSP solving methods

4.3.1 Exact solving

Exact solving methods for the TSP aim to find the optimal solution, that is, the shortest possible tour that visits each city exactly once. These methods guarantee an optimal solution.

Brute force is the most straightforward method for solving the TSP. It involves examining all possible permutations of cities and calculating the total distance for each permutation, which means $O(n!)$ combinations. Dynamic programming techniques, such as Held-Karp algorithm [70], exploit the inherent substructure and overlapping subproblems in the TSP.

In the early exploration of the Traveling Salesman Problem (TSP) during the 1950s-60s, Branch-and-Bound emerged as one of the pioneering exact approaches [98]. Its enduring significance lies in its simplicity, providing a foundational methodology for tackling the TSP.

A major leap in the late 1980s brought forth the Branch-and-Cut technique, a potent combination of Linear Programming (LP) relaxation bounds and dynamically generated cutting planes derived from problem-specific inequalities [116]. This innovation significantly improved the traversal of search trees, marking a crucial advancement in exact TSP solving methods.

Concorde, introduced in 2006-2007 by Applegate et al [6], stands as the contemporary state-of-the-art exact TSP solver. Its prowess lies in incorporating problem-tailored dynamic cut generation schemes within an advanced Branch-and-Cut algorithm. Remarkably, Concorde has demonstrated optimality in solving TSP instances approaching 10,000 cities.

Further diversification in algorithmic paradigms is evident in CP-based approaches [15] and ongoing refinements in Integer Linear Programming formulations. These methods hold the potential to extend the reach of exact solving methods for the TSP, offering new avenues for tackling larger instances.

In recent years, Isoart [78] makes notable strides in solving the Traveling Salesman Problem through constraint programming with Lagrangian relaxation, introducing novel structured constraints and other techniques for optimizing TSP.

Despite their exponential time complexity, well-engineered exact algorithms remain indispensable tools for benchmarking heuristics and establishing optimally proven solutions, particularly on problems of practical relevance and size. The continuous evolution of exact solving methods showcases the resilience and adaptability of these approaches in addressing the challenges posed by the TSP over the decades.

4.3.2 Heuristic solving

In contrast to deterministic algorithms that ensure optimal solutions, heuristics offer a more expedient but non-guaranteed approach to computing TSP tours. While lacking a certainty of optimality, heuristics boast significantly faster computation times. Consequently, when an acceptable solution suffices, heuristics prove to be a more fitting choice. TSP heuristics are categorized into three types: tour construction algorithms, which progressively incorporate

nodes into the current tour; tour improvement algorithms, which refine a tour through exchanging city orders; and composite algorithms, which seamlessly blend both approaches. In this section, we introduce widely adopted heuristics employed in addressing the Traveling Salesman Problem (TSP). Further elaboration on additional heuristics can be found in [65].

Tour Construction Algorithms

One category of TSP heuristics comprises tour construction algorithms. These methods progressively build a tour by iteratively adding nodes to the current solution. Prominent examples include the Nearest Neighbor Algorithm (NN), among the earliest heuristics for solving the TSP, which constructs a tour by iteratively visiting the nearest unexplored node from the current location. Although its time complexity is $O(n^2)$, and it can be deemed somewhat greedy, Rosenkrantz et al. demonstrated that under the triangular inequality, the tour is at most $O(log(V))$ longer than the optimal one.

Expanding upon the NN approach, Christofides' algorithm, an early approximation algorithm, takes the tour construction process a step further. This algorithm ensures that the resulting TSP tour is at most 3/2 times longer than the optimal one for metric graphs satisfying the triangular inequality.

Tour improvement algorithms

Complementing the tour construction algorithms in solving the Traveling Salesman Problem (TSP), tour improvement algorithms focus on refining existing solutions to approach optimality. Unlike tour construction methods that progressively build a solution from scratch, tour improvement algorithms enhance the quality of an initial tour. These iterative techniques iteratively modify the sequence of nodes in the tour to reduce its length and improve efficiency. Local search methods, particularly 2-opt and 3-opt, stand out as the prevailing approaches for the improvement of TSP solutions.

The 2-opt heuristic operates by systematically examining pairs of edges (e_1, e_2) within the given tour T. Should the replacement of this pair with another set of edges (e_3, e_4) result in a more compact and connected tour, the algorithm executes the replacement, defining this procedure as a move. It is noteworthy that certain heuristics adopt an approach where they identify the optimal improving move before executing the replacement. Importantly, for each pair of edges, only one move is available to reconnect the graph, eliminating the possibility of a null move. The iterative process of considering pairs gives rise to a time complexity of $O(n^2)$. Illustrated in Figure 2.5 is an example where $\mathrm{e}_1 = (x_1, x_2)$, $e_2 = (y_1, y_2)$, and the corresponding move involves $e_3 = (x_1, y_1)$ and $e_4 = (x_2, y_2)$.

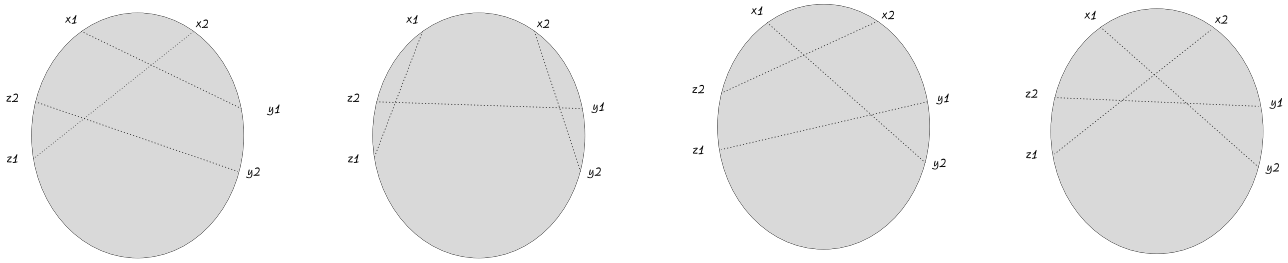


Figure 4.2: Visualizing 3-Opt Heuristic: Tour Configuration with Proposed Moves for the Triplet of Edges $((x_1, x_2), (y_1, y_2), (z_1, z_2))$.

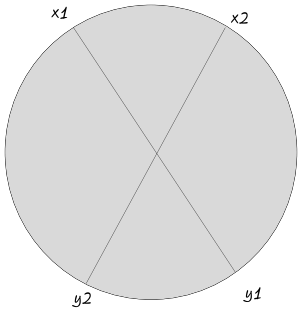


Figure 4.1: Visualization of 2-Opt Heuristic: A Tour Representation with Proposed Moves for the Edge Pair $((x_1, x_2), (y_1, y_2))$.

In 3-opt algorithm, the methodology diverges from selecting pairs of edges, instead opting for triplets of edges within the given tour. Similar to the 2-opt approach, the algorithm systematically explores moves that decrease the overall cost of the tour. Notably, in this scenario, there are seven possible ways to reconnect the graph. Intriguingly, three of these options boil down to simple 2-opt moves, involving a combination with one edge of the triplet remaining stationary. Consequently, the 3-opt algorithm delves into more intricate combinations than its 2-opt counterpart, potentially uncovering superior moves. However, this increased complexity results in an algorithm with a time complexity of $O(n^3)$. Figure 2.6 visually represents an instance showcasing all 3-opt moves distinct from the 2-opt variations.

As a natural extension, the 2-opt and 3-opt algorithms find generalization in the form of the k-opt algorithm, where k represents the number of edges involved in each move. The time complexity of this generalized approach is $O(n^k)$. Empirical investigations have revealed that augmenting the value of k enhances the tour quality but concurrently hampers solving times. Consequently, certain methodologies, as proposed by researchers such as [115] and [10], adopt a pragmatic approach by incorporating selective 3-opt and/or 4-opt moves rather than considering all possibilities. This strategic choice aims to mitigate time complexity, thereby expediting the solving times without compromising on the overall quality of the generated tours.

Lin-Kernighan

Lin and Kernighan [97] introduced a variable k to their TSP algorithm, enhancing tour quality and solving times. Their strategic rules focused on promising permutations and allowed for refining k-opt moves constructed from sequences of 2-opt moves. Helsgaun later refined these rules, creating the Lin-Kernighan-Helsgaun (LKH) algorithm [71]. LKH is recognized as a highly efficient TSP heuristic, often integrated into exact methods for its ability to produce tours close to optimality, especially in multiple runs.

4.3.3 Metha-heuristic solving

Metaheuristics provides general algorithmic frameworks to solve combinatorial optimization problems like the TSP. These methods offer a more flexible, expedient but non-guaranteed approach to solving the TSP optimistically. While lacking the certainty of optimality, metaheuristics excel at providing substantially faster computation times. When the primary objective is to obtain an acceptable solution within a reasonable time frame, metaheuristics prove to be a more fitting choice.

In the chronological evolution of solving the Traveling Salesman Problem (TSP) with single-solution-based metaheuristic methods, Simulated Annealing (SA) emerged prominently in the early 1980s. The foundational work of Kirkpatrick, Gelatt, and Vecchi (1983)[88] introduced SA as a versatile optimization algorithm, with subsequent contributions by Cerny [27] showcasing its adaptability to the TSP. The late 1980s saw Glover [59] extend SA to combinatorial optimization problems, emphasizing temperature schedules and neighborhood structures (Glover, 1986). Tabu Search (TS), introduced by Glover in 1986, marked a significant milestone, incorporating memory structures to escape local optima. TS gained widespread recognition in the 1990s, as demonstrated in Glover's "TS Paradise" and the comprehensive book "Tabu Search" by Glover and Laguna [60]. Variable Neighborhood Search (VNS), conceptualized by Mladenović and Hansen in the late 1990s, expanded the horizon by dynamically altering neighborhood structures. Its application to the TSP by Mladenović et al. [108] showcased the method's ability to explore diverse solution spaces.

As highlighted in Chapter 1, the application of population-based metaheuristic methods to solve combinatorial problems, such as the Traveling Salesman Problem (TSP) has yielded significant advancements over the years. Genetic Algorithms (GA) entered the scene in the 1980s, with Holland's pioneering work [73] laying the foundation. Genetic algorithms have since become a cornerstone in TSP research, with applications ranging from traditional GAs to advanced variants. The advent of Ant Colony Optimization (ACO) in the early 1990s, proposed by Dorigo [33], introduced the concept of decentralized agents mimicking ant foraging behavior. ACO's application to the TSP, particularly through the Max-Min Ant System [84], marked a significant breakthrough. Another notable population-based approach is Particle Swarm Optimization (PSO), which Kennedy and Eberhart introduced in 1995 [84]. The TSP has successfully incorporated PSO's cooperative behavior, which is social interaction-inspired.

The comparison between single-solution-based metaheuristics on the one hand and population-based metaheuristics on the other reveals distinctive characteristics in terms of neighborhood

relations, convergence, complexity, problem modeling, constraint handling, and sensitivity to parameters.

Simulated Annealing and Tabu Search exhibit a strategy of explicitly searching neighborhoods through local changes. These methods require a well-defined neighborhood structure, often tied to alterations in the tour sequence. The theoretical convergence of SA and TS to optimal solutions is guaranteed given infinite time, but their practical convergence is notably slower. Additionally, the complexity of SA and TS increases for large-scale TSP instances due to the intricacies of defining an effective neighborhood.

On the other hand, Genetic algorithms and Ant Colony Optimization adopt a different approach for TSP. These methods explore solution spaces through probabilistic operators without explicit reliance on a neighborhood concept. Although lacking theoretical guarantees, GA and ACO have proven effective in practice when properly parameterized for the TSP. Their inherent population-based nature allows for parallelization, mitigating the complexity associated with larger TSP instances.

When modeling the TSP problem, SA and TS need to encode solutions as states and set distance-based acceptance criteria, which usually involve changing the order of city sequences. GA and ACO, on the other hand, use genetically inspired mechanisms and pheromone communication to model solutions in a more natural way, without the need for problem-specific encoding.

Constraint handling in the TSP presents distinctions. SA and TS require the introduction of repair or rejection mechanisms for infeasible moves or states, addressing constraints such as capacity limitations or time constraints in the tour. Conversely, GA and ACO apply operators that inherently preserve feasibility, drawing inspiration from genetics and decentralized pheromone communication to navigate constraint space more effectively.

Sensitivity to parameters is nuanced in the TSP context. SA and TS are heavily reliant on schedule, acceptance thresholds, and initial states, making them sensitive to perturbations. On the other hand, GA and ACO are less affected by changes, but they need to be carefully tuned in terms of population size and operators to work best, especially when solving the TSP.

4.4 Hybrid TSP solving methods

The complexity of the Traveling Salesman Problem has fueled the exploration of innovative approaches, leading to the emergence of hybrid solving methods. As shown in Chapter 1, hybridization combines multiple optimization techniques, harnessing their complementary strengths to enhance solution quality, convergence speed, and overall algorithmic performance. The rationale behind employing hybrid methods lies in leveraging the diverse strategies of different algorithms to address specific challenges posed by the TSP. Hybrid approaches aim to strike a balance between exploration and exploitation, harnessing the benefits of various methods to achieve superior performance compared to standalone algorithms.

In Chapter 1, we delved into the intricate challenges of combinatorial problems. Hybridization,

the strategic combination of optimization methods, is a key method that is becoming more popular for improving problem-solving abilities. Two primary types emerge: hybridization between Optimization Methods and Integration of Machine Learning Methods with Metaheuristics. The former combines diverse optimization algorithms, each contributing unique strengths. The latter harnesses the synergy between metaheuristics and machine learning, creating a harmonious alliance. This chapter explores these hybridization types, delving into their methodologies, significance, and remarkable advancements in TSP problem-solving. Through an examination of related works, we unveil the evolutionary trajectory and contributions of hybrid TSP-solving methods, shedding light on the nuanced strategies propelling the field forward.

4.4.1 Hybridization between Optimization methods

Combining Metaheuristics

Hybrid algorithms for solving the TSP encompass both route construction and route improvement phases, utilizing a combination of population-based heuristics. One of the earliest forms of hybridization, combining metaheuristics has remained prevalent. Or-opt and k-opt local search were integrated within genetic algorithms in the 1990s to refine solutions [148]. Notable algorithms include Artificial Bee Colony (ABC), Ant Colony Optimization, Genetic Algorithm, Simulated Annealing, Tabu Search, and Particle Swarm Optimization. ACO-based hybrid approaches, like PACO-3Opt[64], use ACO for building the initial population and 3-Opt to optimize each individual, which improves the quality and reliability of the solutions. The PSO-ACO-3Opt algorithm [102] uses PSO to optimize ACO parameters and 3-Opt to improve solutions, giving better performance in terms of being accurate and stable. The Adaptive Simulated Annealing algorithm with Greedy Search (ASA-GS) [57] integrates three mutation-based operations, coupled with greedy search, demonstrating a commendable balance between running time and accuracy. Ant Colony Extended (ACE) [39], a new ACO-based algorithm, uses dual-task ants and a regulation policy to solve TSPs more efficiently. The C-PSO-ACO-kOpt algorithm [85] combines ACO and PSO, utilizing ACO to generate initial solutions and enhancing them with k-Opt, showcasing competitive performance in accuracy and running time. The Fruit Fly Optimization Algorithm (FOA) [118] is enhanced through the Improved FOA (IFOA) [77], addressing convergence and precision issues. The Discrete Symbiotic Organisms Search (DSOS) hybrid algorithm [41] uses three mutation-based local search operators to find the best possible routes. It quickly finds the best solutions and has been shown to work on TSPLIB datasets.

Integrating exact methods with heuristics

since exact algorithms become intractable even for moderately sized instances. Heuristics can find good solutions efficiently, but they provide no quality guarantees. Research has explored hybrid approaches leveraging both.

Early work combined traveling salesman heuristics with branch-and-bound (BB) search to iteratively improve solutions [34]. A study introduces a hybrid approach for solving TSP, combining Constraint Programming propagation algorithms for path feasibility with Opera-

tions Research techniques for optimization [50]. Another study assesses the BB algorithm's efficacy for TSP and the benefits of hybridizing it with local search algorithms [80]. Isoart presents novel constraint programming models for solving the Traveling Salesman Problem (TSP), incorporating three new constraints and filtering algorithms based on graph structure. Additionally, it adds the SSSA algorithm to make computing more efficient and parallelizes the search process using Embarrassingly Parallel Search (EPS). This cuts down on solving times and makes the system more scalable [78]. Several studies have studied the hybridization of dynamic programming with local search algorithms [141] such as Yongliang Lu et al [101], this research introduces a hybrid approach combining dynamic programming and memetic algorithm for solving a variant of TSP with Hotel Selection (TSPHS), addressing real-life constraints on maximum travel time.

4.4.2 Machine learning methods with Meta-heuristics for solving TSP

In Chapter 1, we talked about the integration of machine learning techniques into meta-heuristics for solving combinatorial optimization problems and he comprehensive classification proposed by Karimi-Mamaghan [83] including algorithm selection, fitness evaluation, initialization, parameter setting, and cooperation. In this section, we go deeper into related works that study the integration of ML methods in metaheuristics for solving TSP. Notable studies exemplify the efficacy of ML-in-metaheuristics, such as Kanda's [81] utilization of label ranking algorithms for algorithm selection in solving the Traveling Salesman Problem (TSP). To address the computational cost associated with assessing candidate tours, researchers like Fan and Li [43] employ surrogate models within genetic algorithms, while Golabi et al [61] introduce an extreme learning machine approach hybridized with a genetic algorithm. Moreover, recent advancements explore ML-driven evolution in algorithms, as seen in Drori et al's [35] reinforcement learning-based method for customized TSP heuristics. The integration of ML spans parameter tuning, algorithm selection, and initialization strategies, with innovative approaches such as Wang et al's [146] symbiotic organisms search (SOS) and ACO for ACO parameter optimization. Notably, ML-aided initialization, encompassing complete and partial tour generation, problem decomposition, and clustering, emerges as a pivotal aspect in optimizing large-scale problem instances. Approaches like Alipoor et al's [5] use of Multiagent Reinforcement Learning (MARL) and Miki et al's [104] convolutional neural network-based partial generation showcase the versatility of ML in guiding metaheuristics through vast and complex search spaces, contributing to their efficacy in tackling real-world optimization challenges.

4.5 Parallel TSP solving

The scale of real-world TSP instances encountered in various applications can be staggering, often involving millions or even billions of nodes representing cities or locations. As a result, traditional sequential solving approaches face formidable challenges due to the immense time and space requirements associated with exploring such vast solution spaces. Consequently, parallelization emerges as a compelling strategy to address these limitations by harnessing the

computational power of multiple processors or computing nodes concurrently. By distributing the computational workload across multiple processing units, parallel solving of the TSP offers a promising avenue to overcome the scalability barriers inherent in sequential approaches.

Parallelizing TSP solving involves problem decomposition, assigning subproblems to separate processors for independent solving. Asynchronous search allows metaheuristics to explore subproblems concurrently, periodically merging partial solutions. Cooperative search fosters synergy among processors by sharing information and solutions. Exact solvers employ synchronous branching on different parts of the partitioning tree. Hybrid approaches combine heuristic and exact methods, leveraging a parallel architecture for efficient solution construction. Overall, these strategies distribute computational workload effectively, facilitating an efficient exploration of large TSP solution spaces.

4.5.1 Formalization of TSP for Parallelization

TSP Instance:

$$I = (V, E, c)$$

Where:

$$\begin{aligned} V &= \text{set of nodes } \{v_1, v_2, ..., v_n\} \\ E &= \text{set of edges between nodes in } V \\ c &= \text{cost function } c(v_i, v_j) \text{ that returns the distance between nodes} \end{aligned}$$

Partitioning:

$$P = \{P_1, P_2, ..., P_k\}$$

Where:

$$\begin{aligned} P &= \text{partitioning of } I \text{ into } k \text{ subsets (subproblems)} \\ P_i &= (V_i, E_i, c_i) \text{ is a sub-TSP containing:} \\ V_i &= \text{subset of nodes from } V \\ E_i &= \text{subset of edges from } E \text{ connecting nodes in } V_i \\ c_i &= \text{restriction of } c \text{ to edges in } E_i \end{aligned}$$

Properties:

$$\begin{aligned} V_i \cap V_j &= \emptyset \text{ for all } i \neq j \\ \bigcup_i V_i &= V \\ \bigcup_i E_i &\subseteq E \end{aligned}$$

This represents decomposing the original TSP instance into distinct sub-TSPs that cover all nodes/edges and can be solved independently in parallel.

4.5.2 TSP problem decomposition

Decomposing the variable decision space of the TSP for parallelization involves partitioning the set of cities or decision variables into smaller subsets. that can be solved independently or concurrently.

Clustering is an indispensable methodology in the field of data analytics, especially when tackling complex problems like the TSP. By effectively grouping similar data points, clustering reveals hidden patterns that can greatly enhance our understanding of the underlying problem landscape in TSP scenarios.

In the context of TSP, clustering empowers analysts to customize optimization strategies tailored to the specific characteristics of different clusters of cities. This segmentation of cities into clusters not only simplifies the overall problem but also reduces its dimensionality, making it more manageable for further analysis and solution development.

Moreover, clustering plays a pivotal role in quickly generating initial solutions for the TSP, thereby expediting the convergence process towards finding an optimal route. By efficiently organizing cities into clusters, clustering algorithms can provide valuable insights into potential routes and help refine the search for the shortest path.

The versatility of clustering techniques extends to their ability to optimize various aspects of the TSP solution, such as minimizing travel distance or time, maximizing resource utilization, or balancing workload distribution. These diverse applications of clustering in TSP optimization have far-reaching consequences, impacting industries ranging from logistics and transportation to manufacturing and supply chain management.

There are a lot of algorithms that allow the partitioning of the TSP points. The clustering approach, e.g., k-means [67], affinity propagation algorithm [52], and density peaks clustering [125].

Table 4.1 presents a comparative analysis of three prominent clustering techniques K-means, Affinity Propagation, and Density Peaks Clustering in the context of partitioning TSP points. Each technique is evaluated based on its distinct advantages and disadvantages.

k-means is highlighted for its simplicity, scalability, and ease of implementation, making it suitable for convex-shaped clusters. However, it requires a predefined number of clusters (k) and is sensitive to initial centroids, potentially leading to convergence issues.

Affinity Propagation stands out for its ability to automatically determine cluster centroids and identify exemplar points, while being robust to noise and outliers. Yet, it may consume high memory and pose challenges in tuning parameters like damping factor.

Density Peaks Clustering offers the advantage of discovering clusters of arbitrary shapes without requiring a predefined number of clusters. It is robust to varying cluster densities and effective in identifying clusters with different sizes. However, it necessitates parameter tuning and can be computationally expensive, particularly for large datasets and those with high dimensionality.

Hierarchical clustering [68, 138] stands out as a versatile algorithm that introduces a hierar-

chical structure into the clustering process, making it a robust preprocessing technique with significant potential for optimizing complex problems. This hierarchical clustering method offers several advantages in the context of TSP. Firstly, it provides a systematic means of structuring the TSP problem, allowing for a clearer understanding of its complexities. Additionally, the hierarchical nature of the clustering process enables the application of tailored optimization strategies to finer-grained clusters, thus enhancing the efficiency and effectiveness of the overall solution approach.

Through the utilization of hierarchical clustering, analysts can gain deeper insights into the underlying structure of TSP instances, uncovering patterns and relationships that may not be immediately apparent. This, in turn, enables the development of more nuanced and targeted optimization techniques, ultimately leading to improved solutions for the TSP.

Many studies have studied hierarchical clustering for solving TSP problem. Jiang et al. [79] suggest a hierarchical method for dealing with large-scale TSP. It starts by using k-means and affinity propagation to group cities into smaller groups. Another work by Liao et al. [96] came up with a hierarchical hybrid algorithm for the TSP that uses density peaks clustering for decomposition and ant colony optimization for local resolution. In a real-world application, Jadhav et al. propose an automated travel itinerary generation system utilizing K-Means clustering and the TSP.

4.5.3 The need for alternative problem representations

To enhance the exploration of solution spaces, it is advantageous to broaden our perspective beyond conventional node-based methods. Techniques such as cluster-based abstractions [86, 140], hierarchical organizations [53, 62], and graph embeddings [130] offer novel approaches to understanding the intricacies of complex combinatorial problems. By adopting these alternative problem representations, researchers can potentially unlock new avenues for developing more effective optimization strategies.

For TSP, hierarchical representations offer a structured approach but struggle with large instances due to their inherent parallel limitations. As problem size grows, hierarchical trees experience difficulties efficiently processing vast computations and data for optimal solutions.

Moreover, although hierarchical representations offer efficient capture of clustering hierarchy and localized optimization, they exhibit limitations in fault tolerance and hierarchical depth variation across different problem scales. This topology relies heavily on well-defined clustering of cities, making it vulnerable to disruption by noise or outliers in the data, which can distort neighborhood relationships. Moreover, the hierarchical structure is sensitive to perturbations in cluster organization during initialization, often requiring complete restructuring. To mitigate these challenges, incremental update strategies could be adopted, selectively reoptimizing affected regions rather than restructuring the entire hierarchy. While hierarchical representations provide valuable insights, hybrid approaches integrating flexible graph-based elements may offer greater resilience in handling real-world TSP problems with varying complexity over extended optimization periods.

Interconnection network representations provide an alternative parallel architecture to more

Algorithm	Advantages	Disadvantages
k-means	- Simple and computationally efficient - Scales well with large datasets - Easy to implement and interpret - Suitable for convex-shaped clusters	- Requires a predefined number of clusters (k) - Sensitive to initial cluster centroids - May converge to local optima
Affinity Propagation	- Does not require a predefined number of clusters - Automatically determines cluster centroids - Can identify exemplar points representing clusters - Robust to noise and outliers	- High memory consumption, especially for large datasets - Complexity in tuning the damping factor - May converge to suboptimal solutions in some cases
Density Peaks Clustering	- Can discover clusters of arbitrary shapes - Does not require a predefined number of clusters - Robust to different cluster densities - Effective in identifying clusters with varying sizes	- Requires tuning of parameters like density threshold - Sensitive to noise and parameter settings - Computationally expensive, especially for large datasets - May struggle with datasets of high dimensionality

Table 4.1: Comparison of Clustering Techniques for TSP Partitioning

effectively tackle large-scale TSP. Interconnection networks depict processors and routers as nodes, connected via links in a topology like a mesh or hypercubes. This framework distributes computational tasks across multiple processing units in a more flexible and scalable manner.

Interconnection networks offer several key advantages for parallel TSP solutions. One significant advantage lies in their small network diameters, which remain constant even as the number of nodes increases. This ensures efficient communication between processors, allowing for constant-time message routing as the problem size scales. Additionally, interconnection networks boast high bisection widths in topologies like meshes or hypercubes. This characteristic facilitates balanced partitioning of workloads across halves of the network, optimizing parallelism, and load balancing among processors. Another crucial benefit is the fault tolerance provided by distributed, redundant paths between nodes. In the event of link or processor failures, the network can continue operating without partitioning, ensuring that computations proceed uninterrupted. Moreover, interconnection networks offer scalability by allowing elastic expansion of nodes and links on demand to accommodate growing TSP instances. This flexibility contrasts with hierarchical trees, which may struggle to adjust their structure efficiently at large scales. Overall, these advantages make interconnection networks a powerful framework for parallel TSP solutions, enabling efficient communication, load balancing, fault tolerance, and scalability as problem sizes increase.

4.6 Proposed model methodology

Previous chapters and sections highlighted the state of the art for solving combinatorial problems, especially TSP as a case study with a focus on metaheuristics and exact algorithms. While both families of methods demonstrate successes, limitations also exist in terms of runtimes, scalability and suboptimality of solutions. To further optimize TSP solving, more advanced approaches integrating hybridization, parallelism, and machine learning have emerged.

Hybridization aims to leverage the respective strengths of heuristics and exact methods within a unified, parallel framework. The benefits of combining intensification and diversification strategies have motivated the incorporation of machine learning to autonomously control algorithms. Parallelization distributes computational work aligned with hierarchical network topologies. However, existing parallel implementations face challenges in mapping dynamically evolving combinatorial problems. Therefore, we scrutinize the limitations of these topologies, particularly concerning their adaptability in dynamic environments.

To address such issues, this section introduces a dynamic, hybrid parallel solving framework for the Euclidean TSP. The framework integrates heterogeneous algorithms, architectures and dataflows to flexibly scale across diverse hardware. Representing the problem hierarchically facilitates decomposition into parallelizable sub-units explored cooperatively. Advanced partitioning enables workload balancing amid ongoing network adaptations. Machine learning optimizes partitioning, guides cooperative search, and tunes algorithms in real-time.

At the heart of our discussion lies the investigation of interconnection network systems and how they play a crucial role in making combinatorial optimization more efficient. By explaining

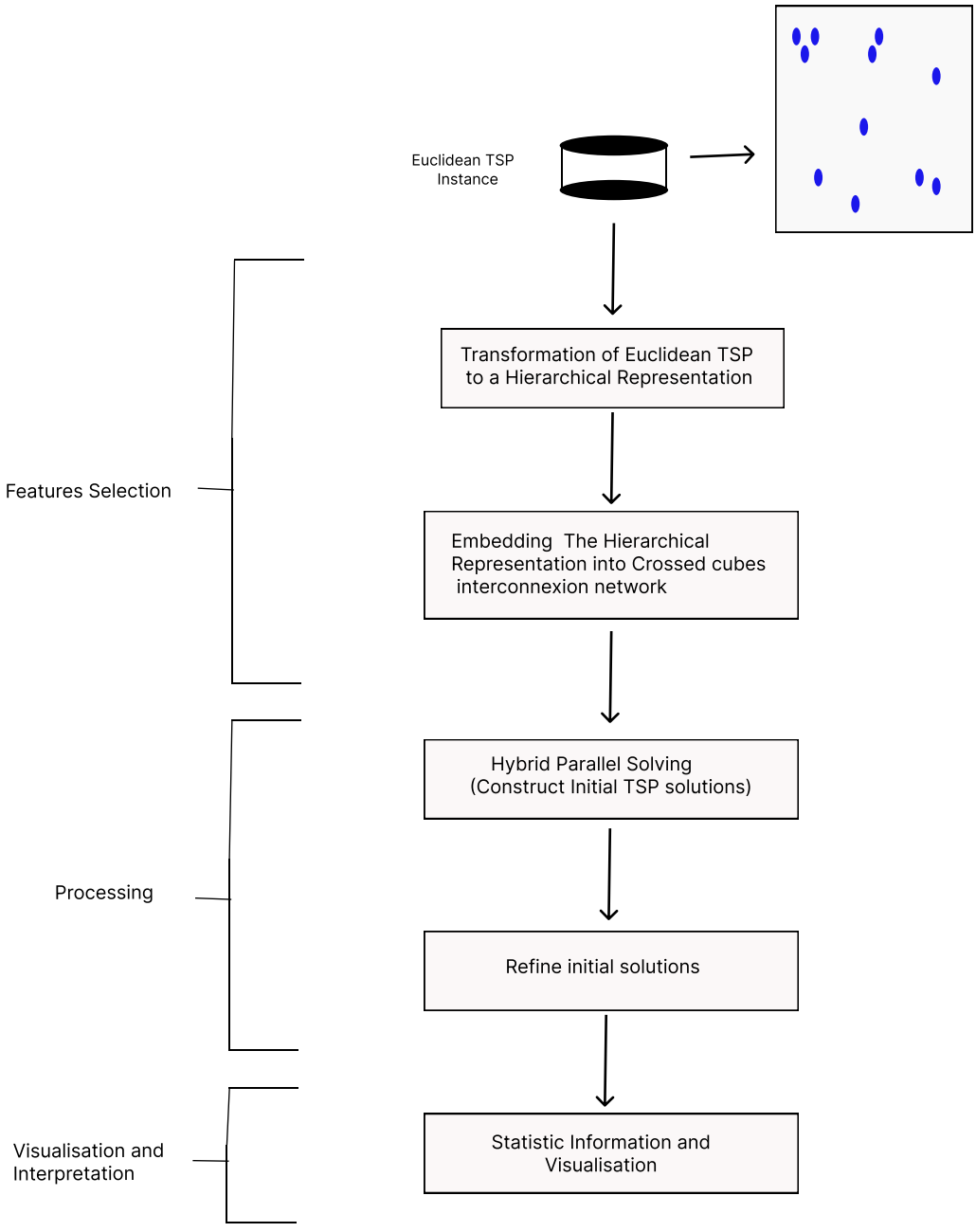


Figure 4.3: Global Proposed Model.

why using these networks is advantageous, we set the stage for coming up with new and creative ways to solve problems.

This system design aims to overcome the limitations of static approaches for large-scale, time-varying TSP instances. By exploiting parallelism through hybridization and machine intelligence, the framework pursues an optimal balance of solution quality, generality, and computational efficiency. The components are described in detail, outlining goals for exploring new frontiers in dynamic optimization through cooperative parallelism.

In Figure 4.3, we present the methodology we've developed for solving the Euclidean Traveling Salesman Problem (TSP). Our approach consists of three primary modules: the Feature Selection module, the Optimization module, and the Visualization and Interpretation module. In the upcoming sections, we'll delve into each module's process, providing a detailed explanation of its functionality and contributions to our overall solution framework.

4.6.1 Feature selection module

The feature selection module is pivotal in identifying relevant spatial, geometric, and topological attributes of the problem. It adapts to the dynamic and hierarchical nature of the input graph, extracting salient features while ensuring dimensionality reduction to retain only those influencing solution quality. This iterative process streamlines problem representation and enhances subsequent optimization, facilitating the discovery of high-quality solutions to the TSP.

In this context, this module is composed of 2 steps: the transformation of Euclidean TSP to a hierarchical representation, and the embedding of The hierarchical representation into crossed cubes interconnexion network steps.

Transformation of Euclidean TSP to a Hierarchical Representation

We introduce an innovative hierarchical representation to revolutionize conventional methods for addressing the TSP. Our approach integrates recursive hybrid clustering techniques within a multi-layered structure. Each clustering layer undergoes thorough evaluation using distinct metrics to gauge both intra-cluster and inter-cluster dissimilarity, as well as the equitability of point distribution within clusters. Through iterative refinement at each hierarchical layer, our method dynamically optimizes for cluster homogeneity and separation. This results in a highly adaptable solution to the TSP, capable of navigating complex data landscapes. Our hybrid hierarchical representation enhances optimization transparency and facilitates parallel computing.

Embedding the Hierarchical Representation into Crossed cubes interconnection network

Utilizing the inherent hierarchical characteristics of the datasets, we aim to leverage a hierarchical representation for the solution of the TSP. Specifically, we intend to employ a compressed hierarchical structure tailored for parallel computational tasks. This strategic approach seeks to enhance optimization outcomes while effectively managing computational

resources, particularly for handling large-scale datasets. However, it's essential to note that hierarchical structures have inherent limitations, such as fault tolerance constraints characterized by a bisection width and connectivity of 1 [94], as well as limitations related to diameter.

Due to these limits, embedding the hierarchical organization of optimization problems into interconnection networks offers numerous advantages, particularly in scalability, fault tolerance, and resource management. This strategic embedding not only enables handling larger-scale problems but also enhances system resilience against faults while optimizing resource allocation effectively.

Addressing the previously identified challenge, we introduce a notable contribution: the dilation 2 one-by-one embedding of hierarchical-based recursive hybrid clustering (in both 2D and 3D representations) into m-dimensional crossed cubes. The primary goal of this embedding is to facilitate efficient resource coordination and distribution within the guest topology. This deliberate integration aims to ensure seamless compatibility between the host and guest architectures, thereby optimizing resource utilization and effectively organizing the operational environment.

4.6.2 Processing module

Initial solutions construction

Upon establishing the parallel environment, our approach constructs a hybrid hierarchical representation included in the crossed cubes interconnexion network for the TSP. This technique strategically decomposes the TSP into manageable subproblems by clustering cities into coherent groups across multiple granularities. This multi-tiered abstraction partitions the optimization landscape into localized clusters. Next, we quickly solve the TSP in each cluster at the same time using efficient construction heuristics like nearest neighbor and ant colony optimization. Our goal is to make a large group of different, high-quality paths for each cluster.

To integrate these intra-cluster solutions and form unified inter-cluster routes, we employ a genetic networking heuristic. This bio-inspired technique selects the fittest individuals from each cluster population to establish judicious inter-cluster connections. By concurrently solving decomposed subproblems and combining the solutions, our approach adeptly scales to large problem sizes. The clustering-based decomposition, coupled with the parallel construction of local solutions, facilitates rapid exploration compared to conventional monolithic methods.

Optimization

To assess the effectiveness of our proposed clustering-driven initialization method, we conducted experiments by employing the generated solutions to seed a simulated annealing metaheuristic for solving the TSP. The hierarchical hybrid clustering approach yields a diverse set of tours that accurately capture the intricacies of the problem structure. From this population, we select the top-performing individuals based on tour length to serve as initial solutions for the simulated annealing optimization.

The localized clusters facilitate rapid feasibility-preserving adjustments during the annealing process, allowing for refinement of the tours. We compared our approach against random restarts and alternative construction heuristics like farthest insertion and k-opt on TSPLIB instances. By striking a careful balance between global exploration through clustering and local intensification within clusters, we can generate high-quality starting solutions.

Our technique significantly accelerates the simulated annealing process across various problem instances, demonstrating its efficacy as a well-informed initialization mechanism.

4.6.3 Visualisation and interpretation module

After obtaining optimal or near-optimal TSP solutions, visualizing and interpreting the results is crucial for assessing algorithm effectiveness. Hierarchical clustering yields meaningful city partitions at each tree level, validated by maps showcasing tight local groupings. Visualizing graph embeddings illustrates how the hierarchical TSP representation aligns with crossed cube topology. The representation of solution paths on Euclidean planes allows direct comparison of lengths and patterns. Analyzing cluster connections aids in understanding problem interfaces and guides further preprocessing. Rigorous statistical analysis validates observed differences in mean, standard deviation, and relative error, supplemented by runtime analyses with graphical visualizations for nuanced interpretation.

4.7 Conclusion

In conclusion, this chapter has provided a comprehensive exploration of solving the TSP, establishing a foundation for comprehending the complexities inherent in tackling this combinatorial optimization challenge. The subsequent chapter will delve deeper into our proposed methodology, where we will detail our contributions and innovations in feature selection, optimization, and visualization modules. Our focus will particularly be on our contribution to feature selection, aimed at enhancing the efficacy and efficiency of TSP-solving techniques.

Chapter 5

Proposed Modeling Paradigm for Solving TSP

5.1 Introduction

In this chapter, we present our proposed modeling paradigm aimed at enhancing the efficiency and effectiveness of solving the Traveling Salesman Problem (TSP). Building upon the comprehensive exploration of TSP-solving techniques in the previous chapter, we detail our contributions and innovations in feature selection, optimization, and visualization modules. Our focus is particularly on our novel approach to feature selection, which is designed to address the inherent complexities of the TSP and improve solution quality.

We explore feature selection, including enhancements to the clustering techniques. We then delve into representation, detailing the TSP-Compressed Quadtree/Octree-based recursive hybrid clustering. Next, we discuss optimization, highlighting hybrid parallel-solving techniques. Finally, we focus on visualization and interpretation.

5.2 Feature Selection: TSP-Compressed Quadtree/Octree-based recursive hybrid clustering representation

In this section, we describe our contributions, which allow the representation of the Euclidean space of the TSP problem in 2D/3D hierarchical representation. Given the intrinsic 2D/3D spatial nature of the TSP datasets, we propose leveraging a hierarchical tree-based representation for solving the problem in a parallel fashion. Specifically, we intend to utilize a compressed quadtree/octree structure that is well-suited for distributed optimization tasks.

This quadtree/octree layout [130, 128], referred to as a fragmented design, provides a novel approach where each node's child elements are dispersed within the overall tree structure. This kind of topological organization tries to find a balance between making solutions better and carefully managing the amount of work that needs to be done, which is very important for massive problem instances.

By modeling the input space in a decomposed quadtree/octree format, the dataset can be stratified into disjoint sub-regions processed simultaneously across compute resources. This partitioning scheme offers benefits such as workload distribution, reduced communication overhead, and flexible granularity of problem decomposition.

When integrated with cooperative parallel algorithms operating at different tree levels, it can efficiently solve large-scale TSP instances while exploiting the intrinsic dimensionality of the inputs. The tree framework is thus a promising representation to unleash the full potential of distributed optimization for Euclidean TSP solutions.

To do this operation, we use a hybridization of clustering algorithms such as K-means, Affinity propagation, and Density peaks clustering. In this context, we aim to adapt these methods to our specific problem. As such, we introduce several contributions outlined in the following sections.

5.2.1 Enhancing k-means with post-redistribution [129]

K-means is a popular choice for clustering tasks because of its simplicity and computational efficiency. However, one drawback is its tendency to converge to local optima, potentially resulting in suboptimal solutions. The algorithm aims to minimize the sum of squared distances between data points and cluster centroids (inertia or sum of squares errors). Despite its effectiveness, K-means may not always find the best overall solution due to its susceptibility to getting stuck in local optima, impacting cluster balance and compactness. It's important to be mindful of this characteristic when utilizing K-means for clustering purposes.

To overcome this limitation, we propose an enhanced version of the K-means clustering algorithm by introducing a novel post-processing redistribution step. This step is designed to address the issue of cluster imbalance and improve the overall quality and compactness of the clusters. By adding the postprocessing redistribution technique-based diameter, we were able to greatly lower the evaluation metrics that were used to judge the performance of clustering. This enhancement significantly impacted the balance within the clusters, leading to more well-organized and tightly grouped data points within each cluster.

To address this limitation, we introduce an improved version of the K-means clustering algorithm, which incorporates a novel post-processing redistribution step. This step aims to mitigate cluster imbalance issues and enhance the quality and compactness of clusters. The proposed enhancement is described as follows:

The proposed algorithm, depicted in Figure 5.1, is a modified iteration of the K-means algorithm with iterative refinement. In standard K-means, the objective is to partition a dataset into K clusters, with each data point assigned to the cluster with the nearest mean. Initially, we set the value of K and apply the standard K-means algorithm to the dataset. Subsequently, we refine the cluster formation through two variants.

In the first variant, called SSE-Based Cluster Splitting (SSE-SPLITTING_KMEANS), we compute the sum of squared errors (SSE) for each cluster to identify the one with the highest SSE. We then select a subset of points closest to the center of this cluster to define the first cluster, while the remaining points become residual points.

For the second variant, known as Iterative Diameter-Based K-Means (DIAMETER_KMEANS), we focus on the diameter of each cluster. After sorting the points within each cluster by their distance from the center, we select a subset of points nearest to the center to calculate the new center for each cluster. We then identify the point with the maximum distance to the new center for each cluster and select the cluster that minimizes the distances between these maximum distances. We repeat the process for this selected cluster, aiming to refine its structure.

Next, we merge the residual points from the selected cluster with other clusters, strategically assigning each residual point to the cluster that minimizes the increase in SSE and diameter. This merging process aims to optimize both SSE and diameter, resulting in enhanced compactness and spatial spread within clusters. Consequently, the number of clusters is reduced by setting $K = K - 1$, and the K-means algorithm is applied again to the combined points from residual clusters and merged points.

This iterative process continues until K becomes 0, indicating that all clusters have been merged. By iteratively merging data points exhibiting the highest distance within clusters, the algorithm aims to refine the clustering solution for the dataset. The choice of the parameter r significantly influences the clustering outcomes.

Rationale for SSE and diameter criteria in redistribution

The adoption of SSE (Sum of Squared Errors) and diameter as criteria for redistribution in clustering algorithms stems from their distinct strengths and complementary functions in evaluating cluster quality. SSE, which gauges the compactness of clusters, promotes the formation of tightly-knit groups, ensuring close alignment between data points and their respective centroids. This metric provides an intuitive and straightforward assessment of intra-cluster cohesion. Conversely, Diameter, used as a redistribution criterion in clustering algorithms, evaluates the spatial dispersion within clusters. It measures the maximum distance between data points, offering insights into the overall spread. This metric is particularly useful for accommodating clusters with irregular shapes and contributes to the creation of well-rounded, spatially balanced clusters.

The complexity of enhanced K-means algorithm

We conduct a thorough examination of the computational costs incurred at each step of the standard and enhanced K-means clustering approaches.

For the basic K-means algorithm, the initialization phase scales as $O(n \cdot K \cdot d \cdot I)$ - where n, K, d, and I denote the number of data points, initial clusters, dimensionality, and iterations respectively.

The SSE-SPLITTING_KMEANS variant introduces an additional $O(n \cdot K \cdot I)$ complexity for calculating SSE values and selecting points per cluster.

DIAMETER_KMEANS incurs $O(n \cdot K \cdot \log(n) \cdot I)$ due to sorting distances within clusters.

Merging residual clusters takes $O(n \cdot I)$ to update memberships and K.

Considering the iterative execution until $K = 0$ over T iterations, the total asymptotic runtime is:

$O(T \cdot (n \cdot K \cdot D \cdot d \cdot I + n \cdot K \cdot I + n \cdot I))$ for SSE-SPLITTING_KMEANS

$O(T \cdot (n \cdot K \cdot d \cdot I + n \cdot K \cdot I + n \cdot K \cdot \log(n) \cdot I + n \cdot I))$ for DIAMETER_KMEANS.

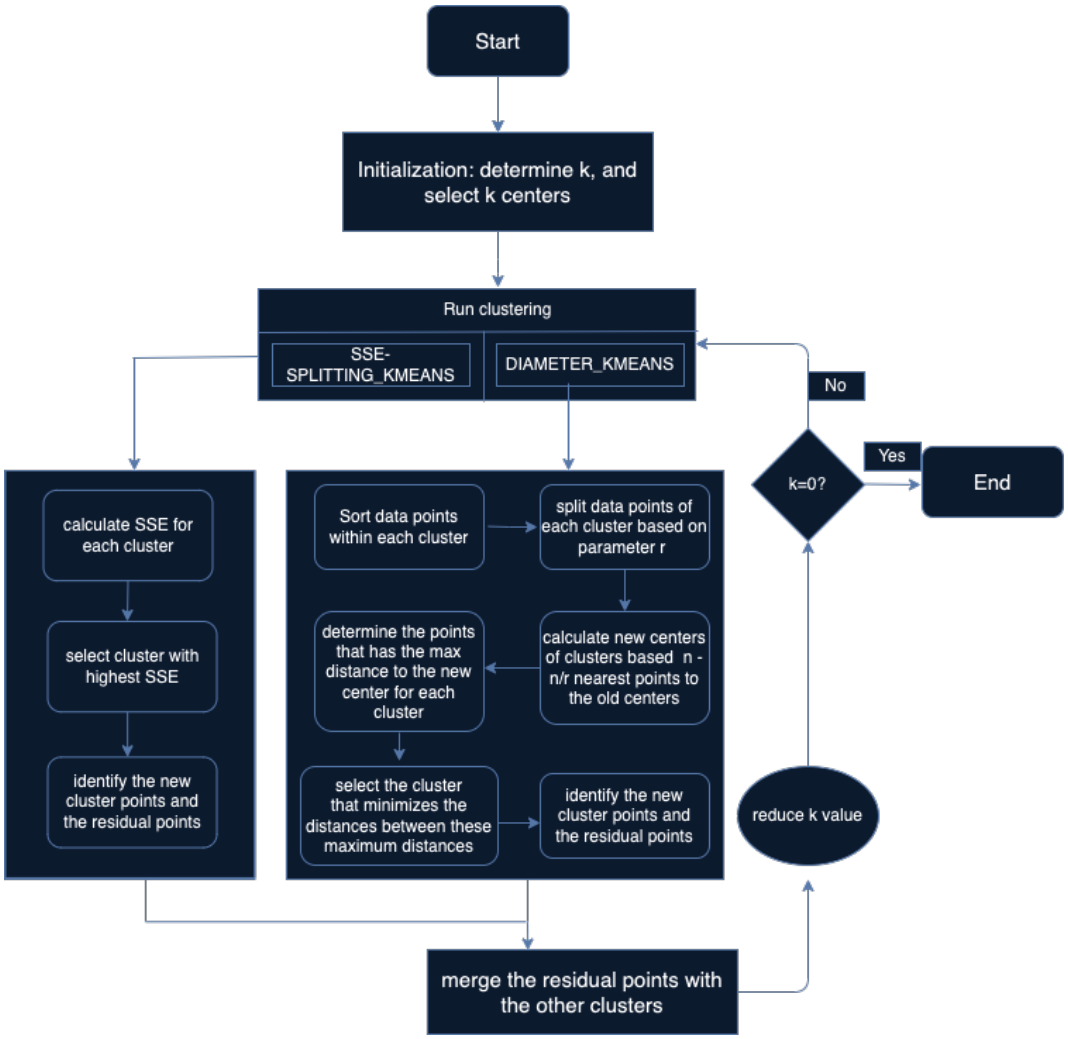


Figure 5.1: The process of the proposed algorithm: Variant 1 (SSE-SPLITTING_KMEANS), Variant 2 (DIAMETER_KMEANS) [129].

5.2.2 K-Affinity Propagation and K-Density Peaks Clustering

Since we used 2D/3D hierarchical representation the number of cluster K for a clustering algorithm is determined. While algorithms like Affinity Propagation (AP), and Density Peaks clustering (DPC) do not have a predefined number of clusters, we propose a hybridization between k-means and AP as well as a variant of DPC that allows the determination of K.

K-Affinity Propagation (AP)

K-affinity propagation (K-AP) represents a fusion of K-means and Affinity Propagation clustering algorithms, aiming to harness the benefits of both approaches. Initially, the data undergoes traditional K-means clustering, dividing it into K initial clusters. Subsequently, a phase inspired by the Affinity Propagation algorithm ensues, where cluster centers are iteratively refined using a pre-calculated similarity matrix. Each cluster center is updated to reflect the most representative point within the cluster, as dictated by the similarity relationships among data points. This iterative refinement process continues until the cluster centers stabilize. The final clusters are then determined based on their proximity to these refined cluster centers, resulting in a solution that capitalizes on the initial organization provided by K-means while benefiting from the adaptive refinement capabilities inherent in Affinity propagation.

K-Density Peaks Clustering (K-DPC)

The K-DPC modifies the Density Peaks clustering algorithm, focusing on the identification of K clusters with high density. Similar to Density Peaks clustering, it prioritizes density as a criterion for determining cluster centers and assigns points to clusters based on their proximity to these centers. The key distinction lies in its emphasis on selecting K clusters with the highest density, while the remaining clusters are distributed among these K clusters based on their nearest neighbors. This adaptation allows for the extraction of K clusters that exhibit significant density, enhancing the clustering process's effectiveness and providing a more refined clustering solution.

5.2.3 Evaluation clustering methods

The assessment of these clustering techniques is grounded on two fundamental metrics: the Davies-Bouldin Index (DBI) and the Gini coefficient. Below, we outline the rationale behind opting for these metrics, offering a detailed explanation for their inclusion in evaluating the proposed algorithmic improvements.

The rationale behind selecting the Davies-Bouldin Index (DBI) as our clustering metric is rooted in its superior performance compared to external validation metrics, particularly in real-world scenarios where prior knowledge may be unavailable. Unlike external indexes, which necessitate prior data information, DBI operates as an internal validation metric, offering adaptability across diverse contexts [123]. Previous studies [100, 66] have underscored DBI's robust discriminatory ability and its suitability for practical applications, especially in scenarios with multiple clusters. DBI consistently scores higher than other internal validation metrics in studies [100]. This shows that it is more reliable and works better, especially when compared to metrics like the Dunn Index [36], which may be affected by boundary points. Additionally, DBI's consistent performance across various clustering experiments highlights its effectiveness as a comprehensive evaluation metric.

On the other hand, the Gini coefficient is a widely adopted metric in clustering evaluations due to its capacity to quantify diversity or inequality within a dataset [26]. When utilized in clustering analyses, the Gini coefficient serves as a measure of clustering quality and the

uniformity of clusters [136]. Its sensitivity to cluster compactness enables the assessment of data point distribution and cohesion within clusters [136]. Additionally, its adaptability to clusters of varying shapes and sizes renders it versatile for evaluating clustering outcomes across diverse datasets. The single-scalar output of the Gini coefficient ensures straightforward interpretation and enables comparisons across different clustering experiments [26].

5.2.4 TSP-Compressed Quadtree/Octree-based recursive hybrid clustering representation process

We present our innovative algorithm tailored to intricately address the specified problem using a hierarchical framework. We use a methodical combination of recursive hybrid clustering methods, such as Enhanced k-means, k-affinity propagation, and k-density peaks clustering. This orchestrated strategy results in a series of progressive recursive partitions tailored to the TSP instance. Within this hierarchical structure, intermediary nodes represent distinct clusters at varying levels of recursion or granularity. At the lower levels of this hierarchy, leaf nodes correspond to individual cities or smaller clusters, marking the culmination of the recursive partitioning process.

The proposed hierarchical clustering algorithm for solving the Euclidean TSP problem is outlined in Algorithm 1 and functions as follows:

Input Set of TSP points.

Initialization: The algorithm begins by considering the entire dataset as a unified partition. Key parameters such as k, min, and max are initialized alongside the collection of TSP cities. The value of k, set to represent 3, or 7 clusters, aligns with the three branches within each internal node of the compact quadtree layout. By defining the parameters min and max, thresholds are established to regulate the permissible range of city counts within sub-partitions. Notably, the depth (d) of the compressed quadtree/octree structure is determined by the number of recursive partitioning iterations. These values assigned to min, max, and d act as termination criteria, effectively guiding the execution of the algorithm.

Initial Clustering: Utilize k-means, affinity propagation, and density peaks clustering algorithms to generate an initial segmentation of the TSP points into k clusters.

Evaluation, Select Best Clustering:

1. Compute the Davies-Bouldin Index and Gini Coefficient for the clusters obtained from each clustering method.
2. Determine a composite score for each clustering method using a weighted approach: $MethodScore = w_{DBI} \times Score_{DBI} + w_{Gini} \times Score_{Gini}$.
3. Compare the scores of each method to identify the best-performing one.
4. Select the clustering results from the method with the highest score.

Recursive Partitioning and repeat:

1. Verify whether the current partition meets the specified stopping criteria (minimum,

maximum, or depth). If not, proceed to the subsequent step.

2. Iterate through the aforementioned steps until the stopping criteria are met.

Output: Cease the recursive process upon reaching the specified stopping criteria. The resulting optimized clustering solution denoted as C^*, is then produced as the output.

By assigning weights to individual clustering methods and aggregating evaluation metrics into an overall score, you can tailor the influence of each method in the decision-making process. This method enables you to harness the unique strengths of each approach and make a more well-rounded and informed selection of the optimal clustering method.

Algorithm 1: TSP-compressed quadtree/octree-based recursive hybrid clustering algorithm

Data: Set of TSP points
Result: Final clustering solution
Parameters : k, min, max, d
Initialization:;
Set the values of k;
Set the values of min and max;
Initialize the set of TSP points;
Set the value of d;
Initial Clustering:;
Apply K-means, affinity propagation, and density peaks clustering to the set of TSP points, creating initial clusters C;
while *stopping criteria not achieved* **do**
  Evaluation, Select Best Clustering: Calculate Davies-Bouldin Index and Gini Coefficient for each method.
  Calculate a global score for each method:
  $MethodScore = w\_DBI \times Score\_DBI + w\_Gini \times Score\_Gini.$
  Choose the cluster results of the best method C_i.
  Apply K-means, affinity propagation, and density peaks clustering to the set of sub-clusters.
end
Output the final clustering solution C^* ;

5.3 Feature Selection: Enhancing optimization via the embedding TSP-Compressed Quadtree/Octree into Crossed Cubes Interconnection Network

The TSP-Compressed Quadtree/Octree ($TSP\text{-}CQT_n/TSP\text{-}COT_n$) partitions the space into sub-regions, efficiently capturing the geometric distribution of points. The m-Dimensional Crossed Cubes topology is commonly used in parallel computing systems for interconnection. In the context of embedding $TSP\text{-}CQT_n/TSP\text{-}COT_n$ into the m-Dimensional Crossed Cubes (CQ_m) network, a systematic procedure emerges.

Initially, the dimensionality m of CQ_m is determined. Then, a sequential one-by-one vertex embedding of *TSP-CQT*$_n$/*TSP-COT*$_n$ onto CQ_m takes place. This careful embedding strategy ensures a seamless integration while preserving intrinsic geometric properties.

Next, the dilation-two embedding technique is applied by systematically mapping each *TSP-CQT*$_n$/*TSP-COT*$_n$ edge onto corresponding paths within CQ_m, realizing a dilation-two embedding.

This results in a cohesive structure, fusing the spatial efficiency of the TSP-Compressed Quadtree/Octree with the organization of the m-dimensional Crossed Cubes. The output structure, termed *TSP-CQ*$_m$, capitalizes on advantageous traits of both components to enable efficient data distribution and communication within the interconnected system.

5.3.1 Definition

In the crossed cubes Q_m, vertices are considered adjacent if their labels differ in only one-bit position. Conversely, in the crossed cube CQ_m, adjacency is defined differently for pairs of 2-bit strings $x = x_1x_0$ and $y = y_1y_0$, where adjacency holds if (x, y) is one of the specified pairs: $\{(00, 00), (10, 10), (01, 11), (11, 01)\}$ [37].

The m-dimensional crossed cube CQ_m is recursively defined as follows: CQ_1 forms a complete graph with 2 vertices. For $m > 1$, CQ_m comprises subcubes $0CQ_{m-1}$ and $1CQ_{m-1}$. Vertices $u = 0u_{m-2}...u_0 \in 0CQ_{m-1}$ and $v = 1v_{m-2}...v_0 \in 1CQ_{m-1}$ are considered adjacent under specific conditions: $u_{m-2} = v_{m-2}$ when m is even, and for odd m, $u_{2i+1}u_{2i}$ and $v_{2i+1}v_{2i}$ are pair-related.

5.3.2 Notations

A *TSP-CQT*$_n$ structure is constructed by combining three instances of *TSP-CQT*$_{n-1}$, each prefixed by 01, 02, and 03 respectively, along with a root prefixed by 0. This composition forms a hierarchical tree-like structure.

A *TSP-COT*$_n$ structure is constructed by combining seven instances of *TSP-COT*$_{n-1}$, each prefixed by 01, 02, 03, 04, 05, 06 and 07 respectively, along with a root prefixed by 0. This composition forms a hierarchical tree-like structure.

For each instance, denoted as *TSP(name of the instance)-CQT*$_n$/*-COT*$_n$, the *TSP-CQT*$_n$/*-COT*$_n$ can be represented as follows:

Consider a graph structure represented as either *TSP-CQT*$_n$ or *TSP-COT*$_n$, where:

- Each vertex is a string of length p denoted as *AV*, where $AV = Aa_{p-1}suff_i$.
- In *TSP-CQT*$_n$, the format of the string A_{p-1} is $a_1a_2 \dots a_j \dots a_{p-1}$, where a_j, and $suff_i$ ranges from 1 to 3.
- In *TSP-COT*$_n$, the format of the string A_{p-1} is $a_1a_2 \dots a_j \dots a_{p-1}$, where a_j, and $suff_i$ ranges from 1 to 7.
- The root is represented by the address $a_1 = 0$.

- An edge between a parent vertex and one of its children is represented as $A_{p-1} - A_{p-1}suff_i$.

$$\text{TSP-CQT}_n = 01\text{TSP-CQT}_{n-1} \parallel 02\text{TSP-CQT}_{n-1} \parallel$$
$$03\text{TSP-CQT}_{n-1} \parallel 0\text{TSP-CQT}_n$$

$$\text{TSP-COT}_n = 01\text{TSP-COT}_{n-1} \parallel 02\text{TSP-COT}_{n-1} \parallel$$
$$03\text{TSP-COT}_{n-1} \parallel 04\text{TSP-COT}_{n-1} \parallel 05\text{TSP-COT}_{n-1}$$

$$06\text{TSP-COT}_{n-1} \parallel 07\text{TSP-COT}_{n-1} \parallel 0\text{TSP-COT}_n$$

Here, $C^* = C^0 0, C^1 0 \dots C^k_{0j}$ denotes the set of clusters, where:

0 represents the root. j indicates the path from the root to the target cluster. k ranges from 1 to 3 or 1, 7.

Crossed Cubes

Let $\text{CQ}_m = (\text{O}, \text{E})$ where O represents the set of vertices and E represents the set of edges.

Case TSP-CQT:

Given an edge $B \in \text{E}$ where $B = C\text{pref}jX3X_2X_1X_0$:

- $C = b_{r-1}, \bar{b}r - 1, \phi$ represents the set of possible values for the control bit.
- Address $(adrr)$ components: X, Y, Z
- The $adrr$ has a length of two and consists of $X1X_0$.
- Either (X, Y) or (Y, Z) must be pair-related.
- Pref$j$ is defined as $bk \dots b_{m-4}$, where j ranges from 0 to 3 respectively if $b_r b_{r+1}, b_r \bar{b}_{r+1}$, $\bar{b}_r b_{r+1}$, $\bar{b}_r \bar{b}_{r+1}$..

Construction of CQ_m

Case 1: When $C = \phi$, CQ_m is constructed by four instances of CQ_{m-2} each prefixed by 00, 01, 10, and 11 respectively.

$$\text{CQ}_m = 00\text{CQ}_{m-2} \parallel 01\text{CQ}_{m-2} \parallel 10\text{CQ}_{m-2} \parallel 11\text{CQ}_{m-2}$$

Case 2: When $C \neq \phi$, CQ_m is constructed by two instances of CQ_{m-1} each prefixed by 0 and 1 respectively.

$$\text{CQ}_m = 0\text{CQ}_{m-1} \parallel 1\text{CQ}_{m-1}$$

Case TSP-COT:

Let $B \in E$ such that: $B = pref_j X_2 X_1 X_0$ where $pref_j = b_r \dots b_{m-3}$ / $j = \overline{0,7}$ respectively if $b_r b_{r+1} b_{r+2}$, $b_r b_{r+1} \bar{b}_{r+2}$, $b_r \bar{b}_{r+1} b_{r+2}$, $b_r \bar{b}_{r+1} \bar{b}_{r+2}$, $\bar{b}_r b_{r+1} b_{r+2}$, $\bar{b}_r b_{r+1} \bar{b}_{r+2}$, $\bar{b}_r \bar{b}_{r+1} b_{r+2}$, $\bar{b}_0 \bar{b}_1 \bar{b}_2$.

The number of super node CQ_3 is equal to 2^{m-3}.

CQ_m is produced by eight copies of CQ_{m-3} prefixed respectively by 000, 001, 010, 011, 100, 101, 110, and 111.

$$\mathrm{CQ}_m = 000\mathrm{CQ}_{m-3} \parallel 001\mathrm{CQ}_{m-3} \parallel 010\mathrm{CQ}_{m-3} \parallel 011\mathrm{CQ}_{m-3} \parallel$$

$$100\mathrm{CQ}_{m-3} \parallel 101\mathrm{CQ}_{m-3} \parallel 110\mathrm{CQ}_{m-3} \parallel 111\mathrm{CQ}_{m-3}$$

The symbol $\parallel$ signifies the simultaneous construction of multiple copies of $TSP\text{-}CQT_n$ and CQ_m.

The column denoted as 'Dil' in the tables refers to dilation.

5.3.3 Dimension of CQ_m

The dimension of the crossed cubes m related to the height of a compressed quadtree/octree. In the case of $TSP\text{-}CQT_n$, we can calculate m as follows:

- Where $n \leq 8$: $m \simeq log_2((3^n - 1)/2))$
- Where $n > 8$: $m = (n - 8) * 2 + 12$

While in the case of $TSP\text{-}COT_n$:

$$m = (n - 1) * 3$$

5.3.4 One-by-one vertex embedding [130, 128]

One-by-one vertex embedding of $TSP\text{-}CQT_n$

The process of the *one-by-one vertex embedding* of $TSP\text{-}CQT_n$ into CQ_m occurs in the following way:
For $n = 3$: The basic one-by-one vertex embedding function f is defined as follows:

- $Prem(0) := pref_0 00$
- $f(A_{p-1} suff_1) := Pref_1 X$
- $f(A_{p-1} suff_2) := Pref_2 Y$
- $f(A_{p-1} suff_3) := Pref_3 Z$

For $n > 3$, the one-by-one vertex embedding occurs in two scenarios. The first scenario arises when $C = \phi$, where we employ the basic function f (see Figure 5.2).

The second scenario arises when $C \neq \phi$, where a function f_1 is employed for this one-by-one vertex embedding. In this situation, there are three cases. The rules of Case 1, as depicted in

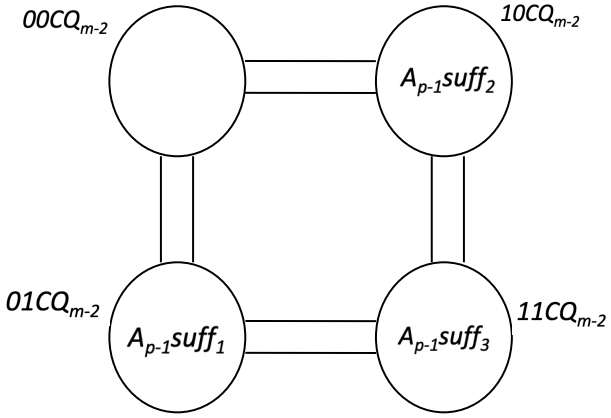


Figure 5.2: Vertex embedding Situation 1 [130]

Figure 5.3, define f_1.

- $f_1(A_{p-1}suff_1) := 0Pref_1 00X$
- $f_1(A_{p-1}suff_2) := 1Pref_1 00Y$
- $f_1(A_{p-1}suff_3) := 1Pref_0 00Z$

OR

- $f_1(A_{p-1}suff_1) := \bar{0}Pref_1 00X$
- $f_1(A_{p-1}suff_2) := \bar{1}Pref_1 00Y$
- $f_1(A_{p-1}suff_3) := 1Pref_0 00Z$

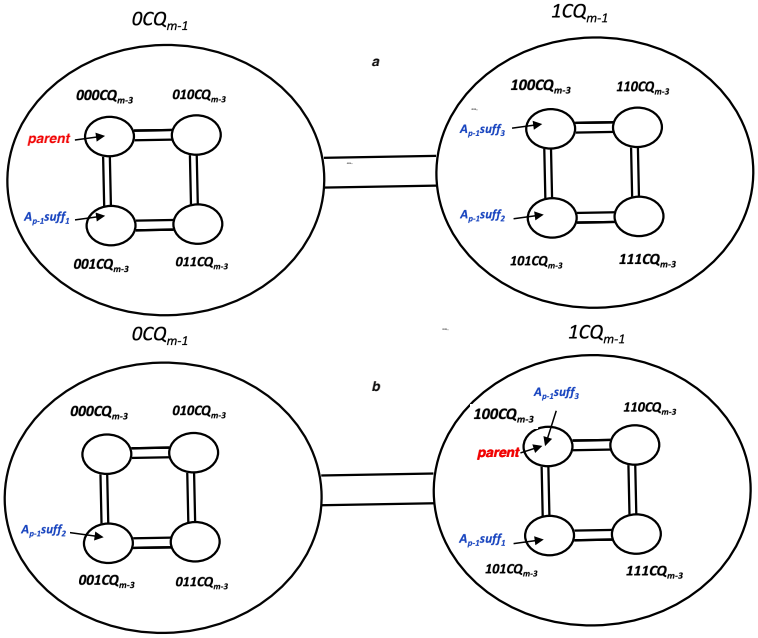


Figure 5.3: Vertex embedding situation 2, case 1 [130].

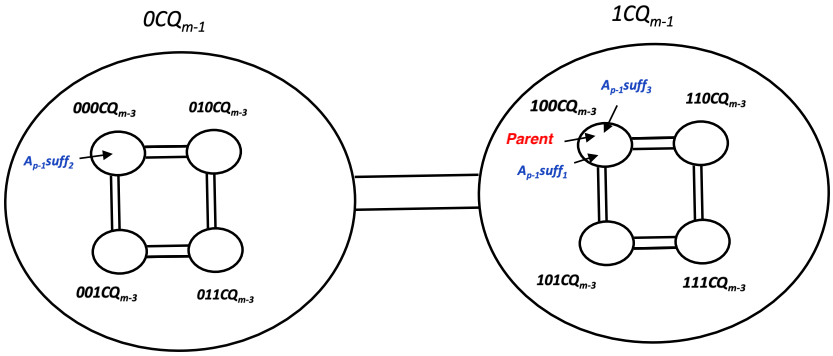


Figure 5.4: Vertex embedding situation 2, case 2 [130].

The guidelines outlined in Case 2, depicted in Figure 5.4, generate the function f_1 for this vertex embedding:

- $f_1(A_{p-1}suff_1) := \bar{0}Pref_0 00X$
- $f_1(A_{p-1}suff_2) := \bar{1}Pref_0 00Y$
- $f_1(A_{p-1}suff_3) := 1Pref_0 00Z$

The regulations presented in Case 3, as illustrated in Figure 5.5, yield the function f_1 for this vertex embedding when $t = 2$ or $t = 3$.

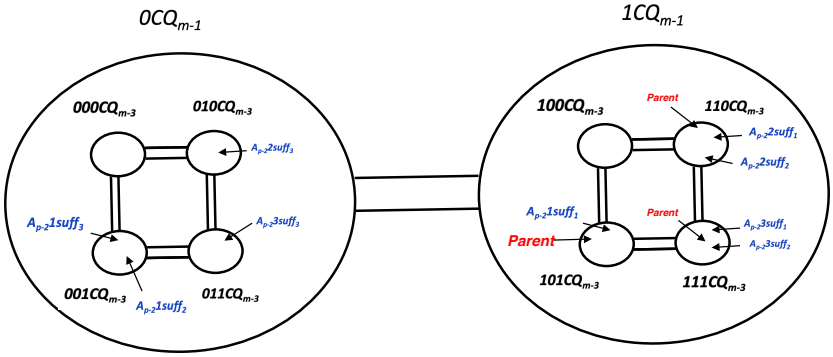


Figure 5.5: Vertex embedding situation 2, case 3 [130].

- $f_1(A_{p-1}suff_1) := \bar{0}Pref_1 00X$
- $f_1(A_{p-1}suff_2) := \bar{1}Pref_1 00Y$
- $f_1(A_{p-1}suff_3) := \bar{1}Pref_1 00Z$

OR

- $f_1(A_{p-1}suff_1) := \bar{0}Pref_t 00X$
- $f_1(A_{p-1}suff_2) := 1Pref_t 00Y$
- $f_1(A_{p-1}suff_3) := \bar{1}Pref_t 00Z$

One-by-one vertex embedding of $TSP\text{-}COT_n$

The procedure for one-by-one vertex embedding $TSP\text{-}COT_n$ into CQ_m involves two scenarios. The first occurs when all nodes $A_{p-1}suff_i$ are internal nodes.

For the case where $n = 3$, the one-by-one vertex embedding process is depicted in Figure 5.6. Here, the basic function f is defined as follows:

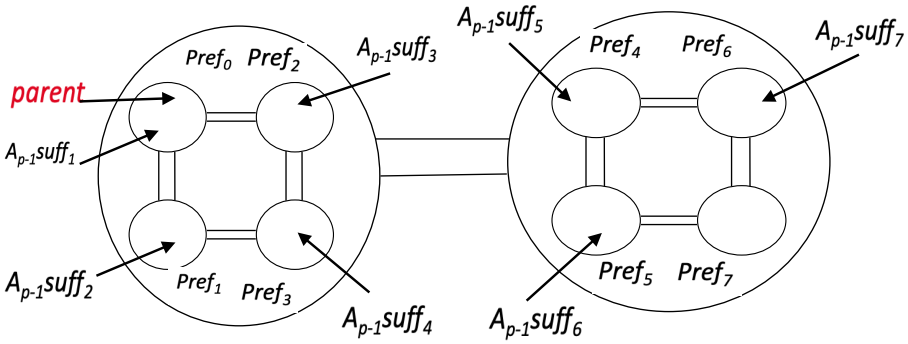


Figure 5.6: Situation 1, rules of the basic function for $n \geq 3$ [128].

- $Prem(0) := Pref_0000$
- $f(A_{p-1}suff_1) := Pref_0001$
- $f(A_{p-1}suff_2) := Pref_1000$
- $f(A_{p-1}suff_3) := Pref_2000$
- $f(A_{p-1}suff_4) := Pref_3000$
- $f(A_{p-1}suff_5) := Pref_4000$
- $f(A_{p-1}suff_6) := Pref_5000$
- $f(A_{p-1}suff_7) := Pref_6000$

For $n > 3$, the one-by-one vertex embedding process is illustrated in Figures 5.6 and 5.7. The following set of rules ($g = \overline{1,3}$) generates the function f:

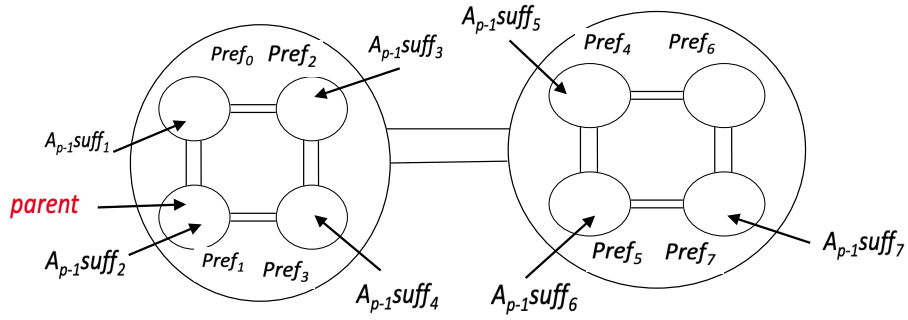


Figure 5.7: Situation 1, rules of the basic function for $n > 3$ [128].

- $f(A_{p-1}suff_1) := b_r b_{r+1} b_{r+2} b_{r+3} b_{r+4} \bar{b}_{r+5} \ldots b_{m-3}000$

- $f(A_{p-1}suff_2) := Pref_1000$
- $f(A_{p-1}suff_3) := Pref_2000$
- $f(A_{p-1}suff_4) := Pref_3000$
- $f(A_{p-1}suff_5) := Pref_4000$
- $f(A_{p-1}suff_6) := Pref_5000$
- $f(A_{p-1}suff_7) := Pref_6000$

OR

- $f(A_{p-1}suff_1) := b_r b_{r+1} b_{r+2} b_{r+3} b_{r+4} \bar{b}_{r+5} \dots b_{m-3}000$
- $f(A_{p-1}suff_2) := b_r b_{r+1} \bar{b}_{r+2} b_{r+3} b_{r+4} \bar{b}_{r+5} \dots b_{m-3}000$
- $f(A_{p-1}suff_3) := Pref_2000$
- $f(A_{p-1}suff_4) := Pref_3000$
- $f(A_{p-1}suff_5) := Pref_4000$
- $f(A_{p-1}suff_6) := Pref_5000$
- $f(A_{p-1}suff_7) := Pref_7000$

OR

- $f(A_{p-1}suff_1) := Pref_0001$
- $f(A_{p-1}suff_2) := Pref_1001$
- $f(A_{p-1}suff_3) := Pref_2000$
- $f(A_{p-1}suff_4) := Pref_3000$
- $f(A_{p-1}suff_5) := Pref_4000$
- $f(A_{p-1}suff_6) := Pref_5000$
- $f(A_{p-1}suff_7) := Pref_7000$

In the second scenario, where all nodes $Asuff_i$ are leaf nodes, there are two potential cases to consider. In the first case, all nodes of any sub-$TSP\text{-}COT_2$ are embedded into the same component of dimension $m = 3$. The function f for this case is derived as follows:

- $f(A_{p-1}suff_1) := Pref_j001$
- $f(A_{p-1}suff_2) := Pref_j010$
- $f(A_{p-1}suff_3) := Pref_j011$
- $f(A_{p-1}suff_4) := Pref_j100$
- $f(A_{p-1}suff_5) := Pref_j101$
- $f(A_{p-1}suff_6) := Pref_j110$

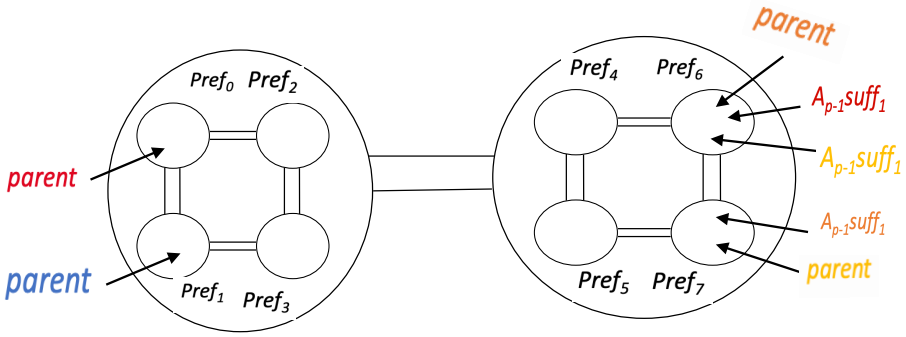


Figure 5.8: Situation 2, case 2 [128].

- $f(A_{p-1}suff_7) := Pref_j111$

In Case 2, when all nodes $A_{p-1}suff_1$ are embedded into separate components of dimension $m = 3$, we derive the rules for f as illustrated in Figure 5.8.

- $f(A_{p-1}suff_1) := Pref_6001$

  OR

- $f(A_{p-1}suff_1) := Pref_7000$

  OR

- $f(A_{p-1}suff_1) := Pref_7000$

  OR

- $f(A_{p-1}suff_1) := Pref_6000$

5.3.5 Dilation two one-by-one edges embedding [130, 128]

Dilation two one-by-one edges embedding of $TSP\text{-}CQT_n$

Dilation-two embedding of one-by-one edges from $TSP\text{-}CQT_n$ onto CQ_m is performed as described below: For the case when $n = 3$, the basic function R governing this dilation two embedding of edges is outlined as follows:

- $R(A_{p-1}\text{-}A_{p-1}suff_1) := Pref_000\text{-}Pref_100$
- $R(A_{p-1}\text{-}A_{p-1}suff_2) := Pref_000\text{-}Pref_200$
- $R(A_{p-1}\text{-}A_{p-1}suff_3) := Pref_000\text{-}Pref_200\text{-}Pref_300$

For $n > 3$, the dilation two embedding of one-by-one edges is executed in two scenarios. In the first scenario, there are two cases to consider. The first case arises when $C = \phi$, in which case we employ the basic function R. An illustrative example can be observed in Figure 5.9.

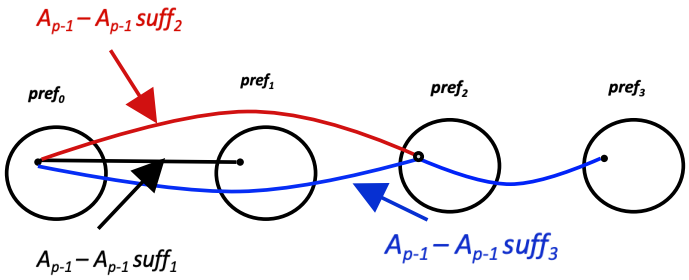


Figure 5.9: Edges embedding situation 1, case 1 [130].

In the second case as shown in Figure 5.10, when $C \neq \phi$, and we utilize only one copy $b_0 CQ_{m-1}$ or $\bar{b}0CQ_{m-1}$, the embedding follows a similar approach to situation 1, case 1. Alternatively, R can be generated as follows:

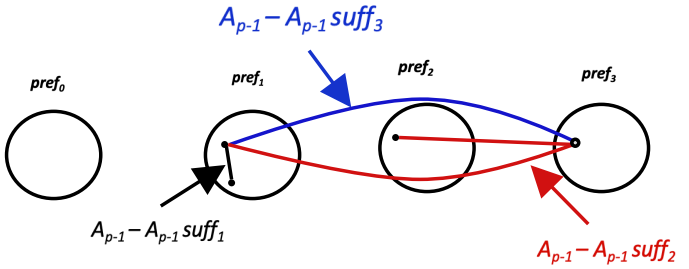


Figure 5.10: Edges embedding situation 1, case 2 [130].

- $R(a_{p-1}\text{-}A_{p-1}suff_1) := Pref_1 00X\text{-}Pref_1 00Y$
- $R(A_{p-1}\text{-}A_{p-1}suff_2) := Pref_1 00X\text{-}Pref_3 00Y\text{-}Pref_2 00Z$
- $R(A_{p-1}\text{-}A_{p-1}suff_3) := Pref_1 00X\text{-}Pref_3 00Y$

In the second scenario where $C \neq \phi$, both copies $b_0 CQ_{m-1}$ and $\bar{b}0CQ_{m-1}$ are employed. This leads to the derivation of a function R_1 for the dilation two one-by-one edges embedding. Within this context, three distinct cases arise, with the first illustrated in figure 5.11. The rules governing case 1 for generating R_1 are as follows:

- $R_1(A_{p-1}\text{-}A_{p-1}suff_1) := 0Pref_0 00X\text{-}0Pref_1 00Y$
- $R_1(A_{p-1}\text{-}A_{p-1}suff_2) := 0Pref_0 00X\text{-}0Pref_1 00Y\text{-}1Pref_1 00Z$
- $R_1(A_{p-1}\text{-}A_{p-1}suff_3) := 0Pref_0 00X\text{-}1Pref_0 00Y$

OR

- $R_1(A_{p-1}\text{-}A_{p-1}suff_1) := \bar{0}Pref_0 00X\text{-}\bar{0}Pref_1 00Y$
- $R_1(A_{p-1}\text{-}A_{p-1}suff_2) := \bar{0}Pref_0 00X\text{-}\bar{0}Pref_1 00Y\text{-}\bar{1}Pref_1 00Z$
- $R_1(A_{p-1}\text{-}A_{p-1}suff_3) := \bar{0}Pref_0 00X\text{-}1Pref_0 00Y$

Case 2, illustrated in figure 5.12, outlines the rules that generate R_1.

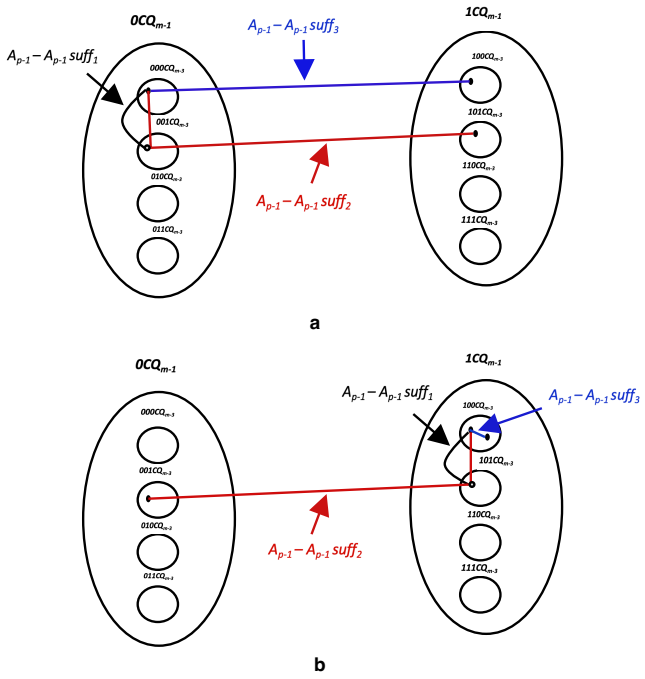


Figure 5.11: Edges embedding situation 2, case 1 [130].

- $R_1(A_{p-1}\text{-}A_{p-1}suff_1) := \bar{0}Pref_0 00X\text{-}\bar{0}Pref_0 00Y$
- $R_1(A_{p-1}\text{-}A_{p-1}suff_2) := \bar{0}Pref_0 00X\text{-}\bar{0}Pref_0 00Y\text{-}\bar{1}Pref_0 00Z$
- $R_1(A_{p-1}\text{-}A_{p-1}suff_3) := \bar{0}Pref_0 00X\text{-}1Pref_0 00Y$

OR

- $R_1(A_{p-1}\text{-}A_{p-1}suff_1) := \bar{0}Pref_0 00X\text{-}\bar{0}Pref_0 00Y$
- $R_1(A_{p-1}\text{-}A_{p-1}suff_2) := \bar{0}Pref_0 00X\text{-}\bar{1}Pref_0 00Y$
- $R_1(A_{p-1}\text{-}A_{p-1}suff_3) := \bar{0}Pref_0 00X\text{-}1Pref_0 00Y\text{-}1Pref_0 00Z$

The rules depicted in figure 5.13 outline the generation of R_1 for the last case, specifically when t equals 2 or 3:

- $R_1(A_{p-1}\text{-}A_{p-1}suff_1) := \bar{0}Pref_1 00X\text{-}\bar{0}Pref_1 00Y$
- $R_1(A_{p-1}\text{-}A_{p-1}suff_2) := \bar{0}Pref_1 00X\text{-}\bar{1}Pref_1 00Y$
- $R_1(A_{p-1}\text{-}A_{p-1}suff_3) := \bar{0}Pref_1 00X\text{-}0Pref_1 00Y\text{-}\bar{1}Pref_1 00Z$

OR

- $R_1(A_{p-1}\text{-}A_{p-1}suff_1) := \bar{0}Pref_t 00X\text{-}\bar{0}Pref_t 00Y$
- $R_1(A_{p-1}\text{-}A_{p-1}suff_2) := \bar{0}Pref_t 00X\text{-}\bar{0}Pref_t 00Y\text{-}1Pref_1 00Z$
- $R_1(A_{p-1}\text{-}A_{p-1}suff_3) := \bar{0}Pref_t 00X\text{-}\bar{1}Pref_t 00Y$

OR

- $R_1(A_{p-1}\text{-}A_{p-1}suff_1) := \bar{0}Pref_2 00X\text{-}\bar{0}Pref_2 00Y$
- $R_1(A_{p-1}\text{-}A_{p-1}suff_2) := \bar{0}Pref_2 00X\text{-}1Pref_2 00Y$
- $R_1(A_{p-1}\text{-}A_{p-1}suff_3) := \bar{0}Pref_2 00X\text{-}\bar{1}Pref_2 00Y$

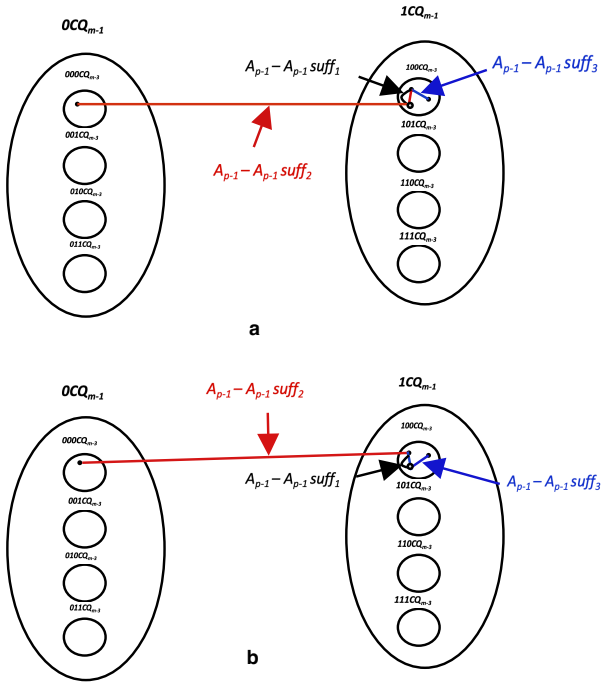


Figure 5.12: Edges embedding situation 2, case 2 [130].

Dilation two one-by-one edges embedding of $TSP\text{-}COT_n$

The procedure for embedding $TSP\text{-}COT_n$ onto CQ_m via dilation of two one-by-one edges involves two scenarios:

When the edge connects two internal nodes. For the case where $n = 3$, the basic function R for this dilation process, illustrated in Figure 5.14, is formulated as follows:

- $R(A_{p-1}\text{-}A_{p-1}suff_1) := Pref_0 000\text{-}Pref_0 001$
- $R(A_{p-1}\text{-}A_{p-1}suff_2) := Pref_0 000\text{-}Pref_1 000$
- $R(A_{p-1}\text{-}A_{p-1}suff_3) := Pref_0 000\text{-}Pref_2 000$
- $R(A_{p-1}\text{-}A_{p-1}suff_4) := Pref_0 000\text{-}Pref_2 000\text{-}Pref_3 000$
- $R(A_{p-1}\text{-}A_{p-1}suff_5) := Pref_0 000\text{-}Pref_4 000$
- $R(A_{p-1}\text{-}A_{p-1}suff_6) := Pref_0 000\text{-}Pref_4 000\text{-}Pref_5 000$
- $R(A_{p-1}\text{-}A_{p-1}suff_7) := Pref_0 000\text{-}Pref_4 000\text{-}Pref_6 000$

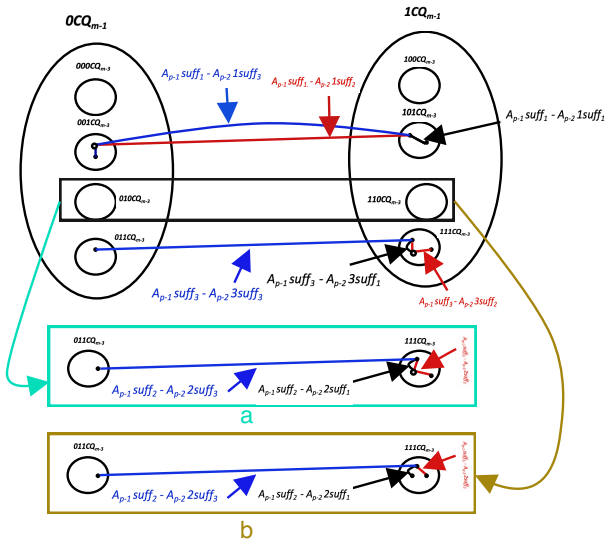


Figure 5.13: Edges embedding situation 2, case 3 [130].

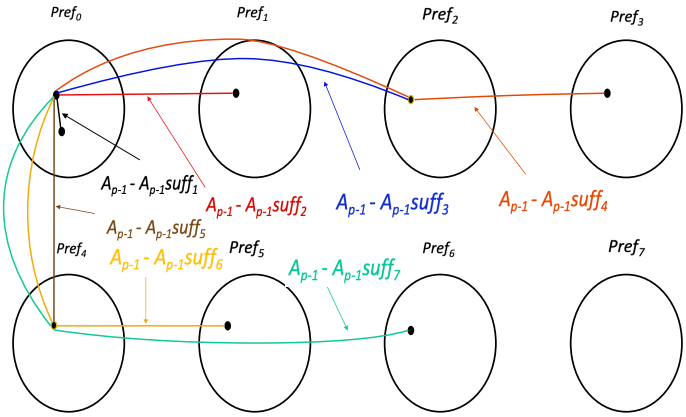


Figure 5.14: Edges embedding situation 1 for $n \geq 3$ [128].

For $n > 3$, the formulation of R for the dilation two one-by-one edges embedding, depicted in Figures 5.14 and 5.15, is achieved through the application of the following group rules ($g = \overline{1,3}$):

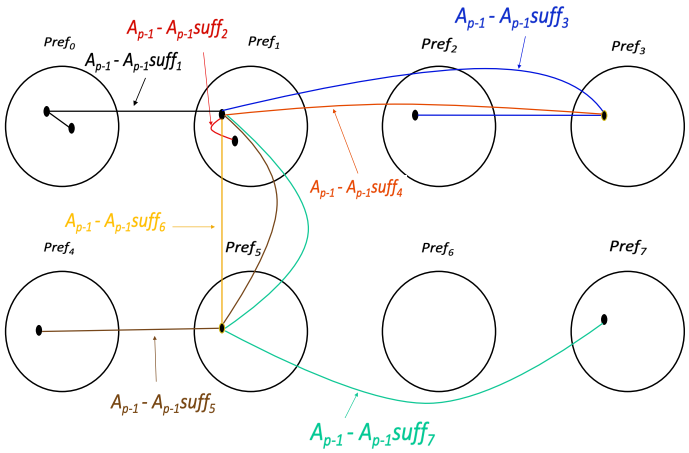


Figure 5.15: Edges embedding situation 1 for $n > 3$ [128].

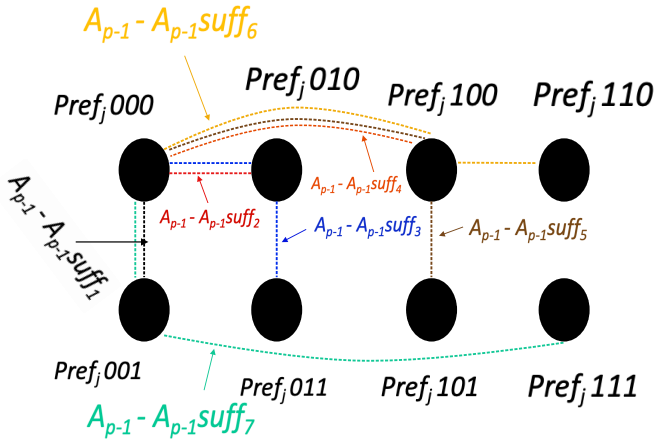
- $R(A_{p-1}\text{-}Asuff_1) := Pref_0000$
  $\text{-}b_r b_{r+1} b_{r+2} b_{r+3} b_{r+4} \bar{b}_{r+5} \ldots b_{m-3}000$
- $R(A_{p-1}\text{-}A_{p-1}suff_2) := Pref_0000\text{-}Pref_1000$
- $R(A_{p-1}\text{-}A_{p-1}suff_3) := Pref_0000\text{-}Pref_2000$
- $R(A_{p-1}\text{-}A_{p-1}suff_4) := Pref_0000\text{-}Pref_2000\text{-}Pref_3000$
- $R(A_{p-1}\text{-}A_{p-1}suff_5) := Pref_0000\text{-}Pref_4000$
- $R(A_{p-1}\text{-}A_{p-1}suff_6) := Pref_0000\text{-}Pref_4000\text{-}Pref_5000$
- $R(A_{p-1}\text{-}A_{p-1}suff_7) := Pref_0000\text{-}Pref_4000\text{-}Pref_6000$

  OR
- $R(A_{p-1}\text{-}Asuff_1) := Pref_1000\text{-}$
  $Pref_0000\text{-}b_r b_{r+1} b_{r+2} b_{r+3} b_{r+4} \bar{b}_{r+5} \ldots b_{m-3}000$
- $R(A_{p-1}\text{-}A_{p-1}suff_2) := Pref_1000\text{-}b_r b_{r+1} \bar{b}_{r+2} b_{r+3} b_{r+4} \bar{b}_{r+5} \ldots b_{m-3}000$
- $R(A_{p-1}\text{-}A_{p-1}suff_3) := Pref_1000\text{-}Pref_3000\text{-}Pref_2000$
- $R(A_{p-1}\text{-}A_{p-1}suff_4) := Pref_1000\text{-}Pref_3000$
- $R(A_{p-1}\text{-}A_{p-1}suff_5) := Pref_1000\text{-}Pref_5000\text{-}Pref_4000$
- $R(A_{p-1}\text{-}A_{p-1}suff_6) := Pref_1000\text{-}Pref_5000$
- $R(A_{p-1}\text{-}A_{p-1}suff_7) := Pref_1000\text{-}Pref_5000\text{-}Pref_7000$

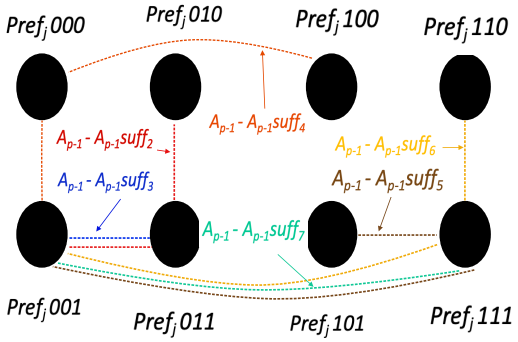
  OR
- $R(A_{p-1}\text{-}A_{p-1}suff_1) := Pref_1000\text{-}Pref_0000\text{-}Pref_0001$
- $R(A_{p-1}\text{-}A_{p-1}suff_2) := Pref_1000\text{-}Pref_1001$
- $R(A_{p-1}\text{-}A_{p-1}suff_3) := Pref_1000\text{-}Pref_3000\text{-}Pref_2000$

- $R(A_{p-1}\text{-}A_{p-1}suff_4) := Pref_1000\text{-}Pref_3000$
- $R(A_{p-1}\text{-}A_{p-1}suff_5) := Pref_1000\text{-}Pref_5000\text{-}Pref_4000$
- $R(A_{p-1}\text{-}A_{p-1}suff_6) := Pref_1000\text{-}Pref_5000$
- $R(A_{p-1}\text{-}A_{p-1}suff_7) := Pref_1000\text{-}Pref_5000\text{-}Pref_7000$

The second scenario arises when the edge is between an internal node and a leaf node. In this situation, there are two cases to consider. In case 1, all edges of any sub-$TSP\text{-}COT_2$ are embedded in the same component of dimension $m = 3$ (Figure 5.16). The basic function R for this case is derived from the following group rules:



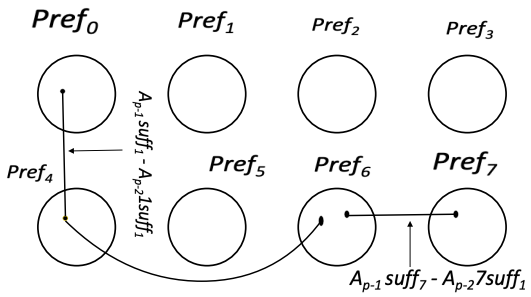
(a)



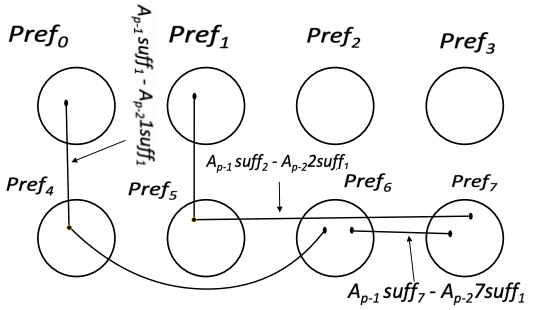
(b)

Figure 5.16: Edges embedding situation 2 case 1 [128].

- $R(A_{p-1}\text{-}A_{p-1}suff_1) := Pref_j000\text{-}Pref_j001$
- $R(A_{p-1}\text{-}A_{p-1}suff_2) := Pref_j000\text{-}Pref_j010$
- $R(A_{p-1}\text{-}A_{p-1}suff_3) := Pref_j000\text{-}Pref_j010\text{-}Pref_j011$
- $R(A_{p-1}\text{-}A_{p-1}suff_4) := Pref_j000\text{-}Pref_j100$
- $R(A_{p-1}\text{-}A_{p-1}suff_5) := Pref_j000\text{-}Pref_j100\text{-}Pref_j101$
- $R(A_{p-1}\text{-}A_{p-1}suff_6) := Pref_j000\text{-}Pref_j100\text{-}Pref_j110$



(a)



(b)

Figure 5.17: Edges embedding situation 2 case 2 [128].

- $R(A_{p-1}\text{-}A_{p-1}suff_7) := Pref_j000\text{-}Pref_j001\text{-}Pref_j111$

  OR
- $R(A_{p-1}\text{-}A_{p-1}suff_2) := Pref_j001\text{-}Pref_j011\text{-}Pref_j010$
- $R(A_{p-1}\text{-}A_{p-1}suff_3) := Pref_j001\text{-}Pref_j011$
- $R(A_{p-1}\text{-}A_{p-1}suff_4) := Pref_j001\text{-}Pref_j000\text{-}Pref_j100$
- $R(A_{p-1}\text{-}A_{p-1}suff_5) := Pref_j001\text{-}Pref_j111\text{-}Pref_j101$
- $R(A_{p-1}\text{-}A_{p-1}suff_6) := Pref_j001\text{-}Pref_j111\text{-}Pref_j110$
- $R(A_{p-1}\text{-}A_{p-1}suff_7) := Pref_j001\text{-}Pref_j111$

In Case 2, all edges $A_{p-1}\text{-}A_{p-1}suff_1$ are embedded between different components of dimension $m = 3$ (Figure 5.17). The first rule for R is formulated as follows:

- $R(A_{p-1}\text{-}A_{p-1}suff_1) := Pref_0001\text{-}Pref_4011\text{-}Pref_6001$

  OR
- $R(A_{p-1}\text{-}A_{p-1}suff_1) := Pref_1001\text{-}Pref_5011\text{-}Pref_7001$

  OR
- $R(A_{p-1}\text{-}A_{p-1}suff_1) := Pref_6000\text{-}Pref_7000$

  OR

- $R(A_{p-1}\text{-}A_{p-1}suff_1) := Pref_7000\text{-}Pref_6000$

5.3.6 Theoretical Validation of Embeddings [131]

Lemma 1. For $n < 5$, $TSP\text{-}CQT_n$ is one-by-one vertex embedded into m-dimensional crossed cubes CQ_m.
Proof. We will prove this by mathematical induction on n.
Base Case (n = 2) and (n = 3, 4). See tables 6.5, Table 6.6.

Induction hypothesis

Suppose that for $k \leq n-1$, $TSP\text{-}CQT_k$ is one-by-one vertex embedding into CQ_l with $l < m$ is true.
Let us now prove that it is true for $k = n$.

The root 0 is embedded into $b_0b_1\,CQ_{l-2}$ by using *Prem(root)*. Nodes $A_{k-1}suff_1$, $A_{k-1}suff_2$, $A_{k-1}suff_3$ are respectively embedded into $b_0\bar{b}_1\,CQ_{l-2}$, $\bar{b}_0b_1\,CQ_{l-2}$, $\bar{b}_0\bar{b}_1\,CQ_{l-2}$ using the basic function f (induction hypothesis, shown in figure 5.2).

Lemma 2. For $n = 5$, $TSP\text{-}CQT_n$ is one-by-one vertex embedding into m-dimensional crossed cubes CQ_m.
Proof. We prove lemma 2 by induction on n.
Base (n=5). See Table 6.8.

Induction hypothesis

Suppose that for $k \leq n-1$, $TSP\text{-}CQT_k$ is one-by-one vertex embedding into CQ_l with $l < m$ is true.
Let us now prove that it is true for k=n.

The root 0 is embedded into $b_0b_1\,CQ_{l-2}$ by using the basic function *Prem(root)*. Nodes of $01\,TSP\text{-}CQT_{k-1}$ are embedded into $b_0\,CQ_{l-1}$ such that: $0suff_1$ is embedded into $0pref_10000$ of $b_0b_1\,CQ_{l-2}$ using f_1 of situation 2, case 1 (figure 5.3, induction hypothesis). Other nodes of $011\,TSP\text{-}CQT_{k-2}$, $012\,TSP\text{-}CQT_{k-2}$, $013\,TSP\text{-}CQT_{k-2}$ are respectively embedded into the root embedded component, $b_0\bar{b}_1\,CQ_{l-2}$ of $b_0\,CQ_{l-1}$, and $b_0\bar{b}_1\,CQ_{l-2}$ of $b_0\,CQ_{l-1}$ using f (figure 5.2, induction hypothesis).

Nodes of $02\,TSP\text{-}CQT_{k-1}$ are embedded into $\bar{b}_0\,CQ_{l-1}$ such that: $0suff_2$ is embedded into $1pref_10000$ of $\bar{b}_0b_1\,CQ_{l-2}$ by **Definition** using f_1 of situation 2, case 1 (figure 5.3). Other nodes of $021\,TSP\text{-}CQT_{k-2}$, $022\,TSP\text{-}CQT_{k-2}$, $023\,TSP\text{-}CQT_{k-2}$ are respectively embedded into the same parent component ($\bar{b}_0b_1\,CQ_{l-2}$), $\bar{b}_0\bar{b}_1\,CQ_{l-2}$ of $\bar{b}_0\,CQ_{l-1}$, and $\bar{b}_0\bar{b}_1\,CQ_{l-2}$ of $\bar{b}_0\,CQ_{l-1}$ using f (figure 5.2, induction hypothesis).

Nodes of $03\,TSP\text{-}CQT_{k-1}$ are embedded into CQ_l such that: $0suff_3$ is embedded into $1pref_00000$ of $\bar{b}_0b_1\,CQ_{l-2}$ by **Definition** using f_1 of situation 2, case 1 (figure 5.3); and $03suff_1$ is embedded into this same component $\bar{b}_0b_1\,CQ_{l-2}$ using f_1 of situation 2, case 2 (figure 5.4, induction hypothesis). Node $03suff_2$ is embedded into $b_0b_1\,CQ_{l-2}$ by **Definition** using f_1 of situation 2, case 2 (figure 5.4). Nodes of $031\,TSP\text{-}CQT_{k-2}$, $032\,TSP\text{-}CQT_{k-2}$ are

embedded into the same parent component, respectively $\bar{b}_0 b_1 CQ_{l-2}$, $b_0 b_1 CQ_{l-2}$ using f (figure 5.2, induction hypothesis).

Nodes of $033\,TSP\text{-}CQT_{k-2}$ are embedded into CQ_l such that: $03suff_3$ is embedded into the same parent component ($\bar{b}_0 b_1 CQ_{l-2}$) using f_1 of situation 2, case 2 (figure 5.4, induction hypothesis). For nodes of $0331\,TSP\text{-}CQT_{k-3}$ are embedded into $\bar{b}_0 b_1 CQ_{l-2}$ and $b_0 b_1 CQ_{l-2}$ such that:

- $033suff_1$ is embedded into the same parent component ($\bar{b}_0 b_1 CQ_{l-2}$) using f (figure 5.2, induction hypothesis);
- $0331suff_1$ is embedded into the same parent component ($\bar{b}_0 b_1 CQ_{l-2}$) using f_1 of situation 2, case 3 (figure 5.5, induction hypothesis);
- nodes $0331suff_2$ or $0331suff_3$ are embedded into $b_0 b_1 CQ_{l-2}$ by **Definition** using f_1 of situation 2, case 3 (figure 5.5).

For nodes of $0332\,TSP\text{-}CQT_{k-3}$ are embedded into $\bar{b}_0 \bar{b}_1 CQ_{l-2}$ and $b_0 \bar{b}_1 CQ_{l-2}$ such that:

- $033suff_2$ is embedded into $\bar{b}_0 \bar{b}_1 CQ_{l-2}$ of $\bar{b}_0 CQ_{l-1}$ using f (figure 5.2, induction hypothesis);
- nodes $0332suff_1$ or $0332suff_2$ are embedded into the same parent component ($\bar{b}_0 \bar{b}_1 CQ_{l-2}$) using f_1 of situation 2, case 3 (figure 5.5, induction hypothesis);
- node $0332suff_3$ is embedded into $b_0 \bar{b}_1 CQ_{l-2}$ of $b_0 CQ_{l-1}$ by **Definition** using f_1 situation 2, case 3 (figure 5.5).

For nodes of $0333\,TSP\text{-}CQT_{k-3}$ are embedded into $\bar{b}_0 \bar{b}_1 CQ_{l-2}$ and $b_0 \bar{b}_1 CQ_{l-2}$ such that:

- $033suff_3$ is embedded into $\bar{b}_0 \bar{b}_1 CQ_{l-2}$ of $\bar{b}_0 CQ_{l-1}$ using f (figure 5.2, induction hypothesis);
- nodes $0333suff_1$ or $0333suff_2$ are embedded into the same parent component ($\bar{b}_0 \bar{b}_1 CQ_{l-2}$) using f_1 of situation 2, case 3 (figure 5.5, induction hypothesis);
- node $0333suff_3$ is embedded $b_0 \bar{b}_1 CQ_{l-2}$ of $b_0 CQ_{l-1}$ by **Definition** using f_1 of situation 2, case 3 (figure 5.5).

Theorem 1. For $n > 5$, a compressed quadtree $TSP\text{-}CQT_n$ is one-by-one vertex embedding into m-dimensional crossed cubes CQ_m.
Proof. We prove theorem 1 by induction on n.
Base (n=6,8). See Tables 6.7, and 6.9.

Induction hypothesis

Suppose that for $k \leq n-1$, $TSP\text{-}CQT_k$ is one-by-one vertex embedding into CQ_l with $l < m$ is true.
Let us now prove that it is true for $k = n$.
There are two cases:
Case a: $C = \phi$

One-by-one vertex embedding of $01\,TSP\text{-}CQT_{k-1}$, $02\,TSP\text{-}CQT_{k-1}$, $03\,TSP\text{-}CQT_{k-1}$, and

$0\,TSP\text{-}CQT_k$ respectively into $b_0\bar{b}_1\,CQ_{l-2}$, $\bar{b}_0 b_1\,CQ_{l-2}$, $\bar{b}_0\bar{b}_1\,CQ_{l-2}$, and $b_0 b_1\,CQ_{l-2}$; in this case, we use the same actions as lemma 1.
Case b: $C \neq \phi$

One-by-one vertex embedding of $01\,TSP\text{-}CQT_{k-1}$, $02\,TSP\text{-}CQT_{k-1}$, $03\,TSP\text{-}CQT_{k-1}$, and $0\,TSP\text{-}CQT_k$ respectively into $b_0\,CQ_{l-1}$, $\bar{b}_0\,CQ_{l-1}$, $b_0\,CQ_{l-1}$ and $\bar{b}_0\,CQ_{l-1}$, and the root 0 into $b_0 b_1\,CQ_{l-2}$; in this case, we use the same actions as lemma 2 except the *TSP-CQT*: $0111\,TSP\text{-}CQT_{k-3}$, $0311\,TSP\text{-}CQT_{k-3}$, and $0321\,TSP\text{-}CQT_{k-3}$ are embedded as situation 2, case 1, b as shown in figure 5.3.

lemma 3. For any $n \geq 2$, nodes in the same sub-$TSP\text{-}COT_2$ of any $TSP\text{-}COT_n$ are one-by-one vertex embedding into the same sub-CQ_3 of any supernode of CQ_m.
Proof. We prove lemma 3 by induction on n.
Base.
For $n = 2$: as shown in Figure 6.13.

Induction hypothesis
Suppose that for $k \leq n-1$, any sub-$TSP\text{-}COT_2$ of $TSP\text{-}COT_k$ is one-by-one vertex embedding into any sub-CQ_3 of CQ_l with $l < m$ is true.
Is it true for $k = n$?.
For any sub-$TSP\text{-}COT_{k'}$, sub-$CQ_{l'}$ with $k' = 2$, $l' = 3$; the root A_{p-1} of sub-$TSP\text{-}COT_{k'}$ are embedded into $pref_j000$; one-by-one embedding vertex of $A_{p-1}suff_1$, $A_{p-1}suff_2$, $A_{p-1}suff_3$, $A_{p-1}suff_4$, $A_{p-1}suff_5$, $A_{p-1}suff_6$, and $A_{p-1}suff_7$ are respectively embedded using rules of situation 2, case 1 (figure 6.13) into:

- The same parent component $pref_j 0\,CQ_{l'-1}$ (induction hypothesis): $pref_j001$, $pref_j010$, $pref_j011$
- The component $pref_j 1\,CQ_{l'-1}$ by **Definition** : $pref_j100$, $pref_j101$, $pref_j110$, $pref_j111$.

lemma 4. For any $n \geq 3$, nodes in the same sub-$TSP\text{-}COT_3$ of any $TSP\text{-}COT_n$ is one-by-one vertex embedding into the same sub-CQ_6 of any supernode of CQ_m.
Proof. We prove lemma 4 by induction on n.
Base (n=3). See Tables 6.14 and 6.15.

Induction hypothesis
Suppose that for $k \leq n-1$, any sub-$TSP\text{-}COT_3$ of $TSP\text{-}COT_k$ is one-by-one vertex embedding into any sub-CQ_6 of CQ_l with $l < m$ is true.
Is it true for $k = n$?.
For any sub-$TSP\text{-}COT_{k'}$, sub-$CQ_{l'}$ with $k' = 3$, $l' = 6$. The root 0 is embedded into $000\,CQ_{l'-3}$ by using the basic function, *Prem(root)*. Nodes $0suff_1$, $0suff_2$, $0suff_3$, $0suff_4$, $0suff_5$, $0suff_6$, and $0suff_7$ are respectively embedded using the basic function f (figure 5.6) into:

- The same parent component $0\,CQ_{l'-1}$ (induction hypothesis): $pref_0001$, $pref_1000$, $pref_2000$, and $pref_3000$.
- The component $1\,CQ_{l'-1}$ by **Definition** : $pref_4000$, $pref_5000$, and $pref_6000$.

All nodes of $01\,TSP\text{-}COT_{k'-1}$, $02\,TSP\text{-}COT_{k'-1}$, $03\,TSP\text{-}COT_{k'-1}$, $04\,TSP\text{-}COT_{k'-1}$, $05\,TSP$-

$COT_{k'-1}$, $06\,TSP\text{-}COT_{k'-1}$, $07\,TSP\text{-}COT_{k'-1}$, and $0\,TSP\text{-}COT_{k'}$ are embedded as lemma 5; except the nodes $01suff_1$ and $07suff_1$ are respectively embedded using rules of situation 2, case 2 (figure 5.18) into the component $1CQ_{l'-1}$ by **Definition** : $pref_6001$, and the same parent component $1CQ_{l'-1}$ (induction hypothesis): $pref_7000$.

Theorem 2. A TSP-compressed octree of dimension n $TSP\text{-}COT_n$ is one-by-one vertex embedding into m-dimensional crossed cubes CQ_m for any $n > 3$.
Proof. We prove Theorem 2 by induction on n.
Base (n=4). See Tables 6.16, and 6.17.

Induction hypothesis
Suppose that for $k \leq n-1$, $TSP\text{-}COT_k$ one-by-one vertex embedding into CQ_l with $l < m$ is true.
Let us now prove that it is true for $k = n$, $l = m$.
The root 0 is embedded into $000CQ_{l-3}$ by using the basic function, *Prem(root)*. Then, nodes $A_{p-1}suff_1$, $A_{p-1}suff_2$, $A_{p-1}suff_3$, $A_{p-1}suff_4$, $A_{p-1}suff_5$, $A_{p-1}suff_6$, $A_{p-1}suff_7$ are respectively embedded using rules of situation 1, group 1 (figure 5.6) into:

- The same parent component $0CQ_{l'-1}$ (induction hypothesis): $b_rb_{r+1}b_{r+2}b_{r+3}b_{r+4}\bar{b}_{r+5}\ldots b_{k-3}000$, $pref_1000$, $pref_2000$, and $pref_3000$.
- The component $1CQ_{l'-1}$ by **Definition** : $pref_4000$, $pref_5000$, and $pref_6000$.

Except for all nodes $A_{p-2}1suff_2$, $A_{p-2}1suff_7$ are respectively embedded using rules of situation 1, group 2 (figure 5.19) into the same parent component $0CQ_{l'-1}$ (induction hypothesis): $b_rb_{r+1}\bar{b}_{r+2}b_{r+3}b_{r+4}\bar{b}_{r+5}\ldots b_{k-3}000$ and the component $1CQ_{l'-1}$ by **Definition** : $pref_7000$. Nodes of $TSP\text{-}COT_{k-t}$ where $k-t=3$ are embedded as lemma 6; except nodes $A_{p-2}1suff_2$, $A_{p-2}1suff_7$ of $A_{p-2}1\,TSP\text{-}COT_{k-t}$ are embedded using rules of situation 1, group 3 into the same parent component $0CQ_{l'-1}$ (induction hypothesis): $pref_1001$, and the component $1CQ_{l'-1}$ by **Definition** : $pref_7000$; and $A_{p-3}12suff_1$, $A_{p-3}17suff_1$ of $A_{p-2}1\,TSP\text{-}COT_{k-t}$ are embedded using rules of situation 2, case 2 (figure 5.18) into the component $1CQ_{l'-1}$ by **Definition** : $pref_7001$, and the same parent component $1CQ_{l'-1}$ (induction hypothesis): $pref_6000$.

Lemma 5. For any $n < 5$, a compressed quadtree $TSP\text{-}CQT_n$ of dimension n is dilation two one-by-one edges embedding onto m-dimensional crossed cubes CQ_m.
Proof. We prove lemma 5 by induction on n.
Base (n=2,4). See Tables 6.10, and 6.11.

Induction hypothesis

Suppose that for $k \leq n-1$, $TSP\text{-}CQT_k$ is dilation two one-by-one edges embedding onto CQ_l with $l < m$ is true.
Is it true for $k = n$?.

Edges between the root 0 and $01\,TSP\text{-}CQT_{k-1}$, $02\,TSP\text{-}CQT_{k-1}$, and $03\,TSP\text{-}CQT_{k-1}$ are embedded respectively onto paths between $b_0b_1\,CQ_{l-2}$ and $b_0\bar{b}_1\,CQ_{l-2}$, $b_0b_1\,CQ_{l-2}$ and $\bar{b}_0b_1\,CQ_{l-2}$,

and $b_0b_1\,CQ_{l-2}$ and $\bar{b}_0b_1\,CQ_{l-2}$ and $\bar{b}_0\bar{b}_1\,CQ_{l-2}$. Edges in this lemma are embedded using R (induction hypothesis, situation 1, case 1, as shown in figure 5.9).

Lemma 6. For any $n \geq 5$, all edges in the same $TSP\text{-}CQT_5$ of any $TSP\text{-}CQT_n$ is dilation two one-by-one edges embedding onto paths in the same CQ_7 of any supernode of CQ_m.
Proof. We prove lemma 6 by induction on n.
Base (n=5). See Tables 6.12, 6.13.

Induction hypothesis

Suppose that for $k \leq n-1$, any $TSP\text{-}CQT_5$ of $TSP\text{-}CQT_k$ is dilation two one-by-one edges embedding onto any CQ_7 of CQ_l with $l < m$ is true.
Is it true for $k = n$?.

For any $TSP\text{-}CQT_{k'}$, $CQ_{l'}$ with $k' = 5$, $l' = 7$; the edge between $0\,TSP\text{-}CQT_{k'}$, $01\,TSP\text{-}CQT_{k'-1}$ is embedded onto a path in the same component $b_0b_1\,CQ_{l'-2}$ using R_1 of situation 2, case 1 (figure 5.11, induction hypothesis). For edges of $01\,TSP\text{-}CQT_{k'-1}$ are embedded onto paths between $b_0b_1\,CQ_{l'-2}$, $b_0\bar{b}_1\,CQ_{l'-2}$, and the same $b_0b_1\,CQ_{l'-2}$, $b_0\bar{b}_1\,CQ_{l'-2}$ of $b_0\,CQ_{l'-1}$ using R (figures 5.9, 5.10, induction hypothesis).

The edge between $0\,TSP\text{-}CQT_{k'}$, $02\,TSP\text{-}CQT_{k'-1}$ is embedded onto a path in the same component $b_0b_1\,CQ_{l'-2}$ (induction hypothesis), and between $b_0b_1\,CQ_{l'-2}$, $\bar{b}_0b_1\,CQ_{l'-2}$ by **Definition** using R_1 of situation 2, case 1 (figure 5.11). For edges of $02\,TSP\text{-}CQT_{k'-1}$ are embedded onto paths between $\bar{b}_0b_1\,CQ_{l'-2}$, $\bar{b}_0\bar{b}_1\,CQ_{l'-2}$, and the same $\bar{b}_0b_1\,CQ_{l'-2}$, $\bar{b}_0\bar{b}_1\,CQ_{l'-2}$ of $\bar{b}_0\,CQ_{l'-1}$ using R (figures 5.9, 5.10, induction hypothesis).

The edge between $0\,TSP\text{-}CQT_{k'}$, $03\,TSP\text{-}CQT_{k'-1}$ is embedded onto a path between $b_0b_1\,CQ_{l'-2}$, $\bar{b}_0b_1\,CQ_{l'-2}$ by **Definition** using R_1 of situation 2, case 1 (figure 5.11). For edges of $03\,TSP\text{-}CQT_{k'-1}$ are embedded as follows: the edge between $03\,TSP\text{-}CQT_{k'-1}$, $031\,TSP\text{-}CQT_{k'-2}$ is embedded onto a path in the same component $\bar{b}_0b_1\,CQ_{l'-2}$ using R_1 of situation 2, case 2 (figure 5.12, induction hypothesis); the edge between $03\,TSP\text{-}CQT_{k'-1}$, $032\,TSP\text{-}CQT_{k'-2}$ is embedded onto a path in the same component $\bar{b}_0b_1\,CQ_{l'-2}$ (induction hypothesis), and between $\bar{b}_0b_1\,CQ_{l'-2}$, $b_0b_1\,CQ_{l'-2}$ by **Definition** using R_1 of situation 2, case 2 (figure 5.12). Edges of $031\,TSP\text{-}CQT_{k'-2}$, $032\,TSP\text{-}CQT_{k'-2}$ are embedded onto paths in the same $b_0b_1\,CQ_{l'-2}$, $\bar{b}_0b_1\,CQ_{l'-2}$ using R of situation 1, case 1(figure 5.9, induction hypothesis).

The edge between $03\,TSP\text{-}CQT_{k'-1}$, $033\,TSP\text{-}CQT_{k'-2}$ is embedded onto a path in the same component $\bar{b}_0b_1\,CQ_{l'-2}$ using R_1 of situation 2, case 2 (figure 5.12, induction hypothesis). For edges of $033\,TSP\text{-}CQT_{k'-2}$ are embedded as follows: the edge between $033\,TSP\text{-}CQT_{k'-2}$, $0331\,TSP\text{-}CQT_{k'-3}$ is embedded onto a path in the same component $\bar{b}_0b_1\,CQ_{l'-2}$ using R of situation 1, case 1 (figure 5.9, induction hypothesis). Edges of $0331\,TSP\text{-}CQT_{k'-3}$ are embedded onto paths in the same component $\bar{b}_0b_1\,CQ_{l'-2}$ (induction hypothesis), between $\bar{b}_0b_1\,CQ_{l'-2}$, $b_0b_1\,CQ_{l'-2}$ by **Definition** , and in the same component $b_0b_1\,CQ_{l'-2}$ (induction hypothesis) using R_1 of situation 2, case 3 (figure 5.13).

Edges between $033\,TSP\text{-}CQT_{k'-2}$, $0332\,TSP\text{-}CQT_{k'-3}$ or $0333\,TSP\text{-}CQT_{k'-3}$ are embedded onto paths between $\bar{b}_0b_1\,CQ_{l'-2}$, $\bar{b}_0\bar{b}_1\,CQ_{l'-2}$ of $\bar{b}_0\,CQ_{l'-1}$ using R of situation 1, case 1 (figure

5.9, induction hypothesis).

Edges of $0332\,TSP\text{-}CQT_{k'-3}$ or $0333\,TSP\text{-}CQT_{k'-3}$ are embedded onto paths in the same component $\bar{b}_0\bar{b}_1\,CQ_{l'-2}$ (induction hypothesis), and between $\bar{b}_0\bar{b}_1\,CQ_{l'-2}$, $b_0\bar{b}_1\,CQ_{l'-2}$ by **Definition** using R_1 of situation 2, case 3 (figure 5.13).

Theorem 3. For any $n > 5$, a compressed quadtree $TSP\text{-}CQT_n$ is dilation two one-by-one edges embedding onto m-dimensional crossed cubes CQ_m.
Proof. We prove theorem 3 by induction on n.
Base. See Tables 6.11, and 6.13.

Induction hypothesis

Suppose that for $k \leq n-1$, $TSP\text{-}CQT_k$ is dilation two one-by-one edges embedding onto CQ_l with $l < m$ is true.
Is it true for $k = n$?.
There are two cases:
Case a: $C = \phi$

Dilation two one-by-one edges embedding between the root 0 and respectively $01\,TSP\text{-}CQT_{k-1}$, $02\,TSP\text{-}CQT_{k-1}$, and $03\,TSP\text{-}CQT_{k-1}$ onto paths respectively between $b_0b_1\,CQ_{l-2}$ and $b_0\bar{b}_1\,CQ_{l-2}$, $b_0b_1\,CQ_{l-2}$ and $\bar{b}_0b_1\,CQ_{l-2}$, $b_0b_1\,CQ_{l-2}$ and $\bar{b}_0\bar{b}_1\,CQ_{l-2}$. In this case, we use the same actions as lemma 5.
Case b: $C \neq \phi$

Dilation two one-by-one edges embedding between the root 0 and respectively $01\,TSP\text{-}CQT_{k-1}$, $02\,TSP\text{-}CQT_{k-1}$, and $03\,TSP\text{-}CQT_{k-1}$ onto paths in the same supernode $b_0b_1\,CQ_{l-2}$, and between $b_0b_1\,CQ_{l-2}$, $\bar{b}_0b_1\,CQ_{l-2}$ (situation 2, case 1, a figure 5.11). Dilation two one-by-one edges of $01\,TSP\text{-}CQT_{k-1}$, $02\,TSP\text{-}CQT_{k-1}$, and $03\,TSP\text{-}CQT_{k-1}$ respectively onto paths in $b_0\,CQ_{l-1}$, $\bar{b}_0\,CQ_{l-1}$, and both $b_0\,CQ_{l-1}$, $\bar{b}_0\,CQ_{l-1}$.

In this case, we use the same actions as lemma 6 except the edges of $TSP\text{-}CQT$: $0111\,TSP\text{-}CQT_{k-3}$, $0311\,TSP\text{-}CQT_{k-3}$, and $0321\,TSP\text{-}CQT_{k-3}$ are embedded like situation 2, case 1, b (figure 5.11). Moreover, edges of $TSP\text{-}CQT$: $01113\,TSP\text{-}CQT_{k-4}$, $03113\,TSP\text{-}CQT_{k-4}$, and $03213\,TSP\text{-}CQT_{k-4}$ are embedded as situation 2, case 2, b (figure 5.12).

lemma 7. For any $n \geq 2$, all edges in the same sub-$TSP\text{-}COT_2$ of any $TSP\text{-}COT_n$ is dilation two one-by-one edges embedding onto the same sub-CQ_3 of any supernode of CQ_m.
Proof. We prove lemma 7 by induction on n.
Base . See Table 6.18.

Induction hypothesis
Suppose that for $k \leq n-1$, any sub-$TSP\text{-}COT_2$ of $TSP\text{-}COT_k$ is dilation two one-by-one edges embedding onto any sub-CQ_3 of CQ_l with $l < m$ is true.
Let us now prove that it is true for $k = n$.
For any sub-$TSP\text{-}COT_{k'}$, sub-$CQ_{l'}$ with $k' = 2$, $l' = 3$. All edges between the root A_{p-1} and $A_{p-1}suff_1$, $A_{p-1}suff_2$, $A_{p-1}suff_3$, $A_{p-1}suff_4$, $A_{p-1}suff_5$, $A_{p-1}suff_6$, and $A_{p-1}suff_7$ are embedded

using R (rules of situation 2, case 1, group 1, figure 5.16, (a)) respectively onto paths:

- In the same component $0CQ_{l'-1}$ (induction hypothesis): $pref_j$000-$pref_j$001, $pref_j$000-$pref_j$010, and $pref_j$000-$pref_j$010-$pref_j$011.
- By **Definition** : $pref_j$000-$pref_j$100.
- By **Definition** and in the same component $1CQ_{l'-1}$ (induction hypothesis): $pref_j$000-$pref_j$100-$pref_j$101, and $pref_j$000-$pref_j$100-$pref_j$110.
- In the same component $0CQ_{l'-1}$ (induction hypothesis) and by **Definition** : $pref_j$000-$pref_j$001-$pref_j$111.

lemma 8. For any $n \geq 3$, all edges in the same sub-$TSP\text{-}COT_3$ of any $TSP\text{-}COT_n$ are dilation two one-by-one edges embedding onto the same sub-CQ_6 of any supernode of CQ_m.
Proof. We prove lemma 8 by induction on n.
Base ($n \leq 4$). Tables 6.20, Table 6.19

Induction hypothesis
Suppose that for $k \leq n-1$, any sub-$TSP\text{-}COT_2$ of $TSP\text{-}COT_k$ is dilation two one-by-one edges embedding onto any sub-CQ_3 of CQ_l with $l < m$ is true.
Let us now prove that it is true for $k = n$.
For any sub-$TSP\text{-}COT_{k'}$, sub-$CQ_{l'}$ with $k^{'} = 3$, $l^{'} = 6$. All edges between the root 0 and $01\,TSP\text{-}COT_{k'-1}$, $02\,TSP\text{-}COT_{k'-1}$, $03\,TSP\text{-}COT_{k'-1}$, $04\,TSP\text{-}COT_{k'-1}$, $05\,TSP\text{-}COT_{k'-1}$, $06\,TSP\text{-}COT_{k'-1}$, and $07\,TSP\text{-}COT_{k'-1}$ are respectively embedded using R (figure 5.14) onto paths:

- In the same component $0CQ_{l'-1}$ (induction hypothesis): in the same component $000CQ_{l'-3}$, between $000CQ_{l'-3}$ and $001CQ_{l'-3}$, between $000CQ_{l'-3}$ and $010CQ_{l'-3}$, and between $000CQ_{l'-3}$ and $010CQ_{l'-3}$ and $011CQ_{l'-3}$.
- By **Definition** : between $000CQ_{l'-3}$ and $100CQ_{l'-3}$.
- By **Definition** and in the same component $1CQ_{l'-1}$ (induction hypothesis): between $000CQ_{l'-3}$ and $100CQ_{l'-3}$ and $101CQ_{l'-3}$, and between $000CQ_{l'-3}$ and $100CQ_{l'-3}$ and $110CQ_{l'-3}$.

Edges of $02\,TSP\text{-}COT_{k'-1}$, $03\,TSP\text{-}COT_{k'-1}$, $04\,TSP\text{-}COT_{k'-1}$, $05\,TSP\text{-}COT_{k'-1}$, and $06\,TSP\text{-}COT_{k'-1}$ are embedded as lemma 7. Edges of $01\,TSP\text{-}COT_{k'-1}$, and $01\,TSP\text{-}COT_{k'-7}$ are embedded using R (rules of situation 2, case 1, figure 5.16, (b)) onto paths:

- In the same component $pref_j0CQ_2$ (induction hypothesis): $pref_j$001-$pref_j$011-$pref_j$010, and $pref_j$001-$pref_j$011.
- In the same component $pref_j0CQ_2$ (induction hypothesis) and by **Definition** : $pref_j$001-$pref_j$000-$pref_j$100.
- By **Definition** and in the same component $pref_j1CQ_2$ (induction hypothesis): $pref_j$001-$pref_j$111-$pref_j$101, and $pref_j$001-$pref_j$111-$pref_j$110.
- By **Definition** : $pref_j$001-$pref_j$111.

Except for $0suff_1$-$01suff_1$, $0suff_7$-$07suff_1$ are embedded using R (rules of situation 2, case 2, figure 5.17, (a)) onto paths:

- By **Definition** and in the same component $1CQ_{l'-1}$ (induction hypothesis): between $000CQ_{l'-3}$ and $100CQ_{l'-3}$ and $110CQ_{l'-3}$.
- In the same component $1CQ_{l'-1}$ (induction hypothesis): $110CQ_{l'-3}$ and $111CQ_{l'-3}$.

Theorem 4. A TSP-compressed octree $TSP\text{-}COT_n$ of dimension n is one-by-one dilation two embedding into m-dimensional crossed hypercube CQ_m for any $n > 3$.
Proof. We prove Theorem 4 by induction on n.
Base. See Table 6.22, and 6.21.

Induction hypothesis
Suppose that for $k \leq n-1$, $TSP\text{-}COT_k$ one-by-one dilation two embedding into CQ_l with $l < m$ is true.
Let us now prove that it is true for $k = n$.
All edges between A_{p-1}-$A_{p-1}suff_1$, A_{p-1}-$A_{p-1}suff_2$, A_{p-1}-$A_{p-1}suff_3$, A_{p-1}-$A_{p-1}suff_4$, A_{p-1}-$A_{p-1}suff_5$, A_{p-1}-$A_{p-1}suff_6$, and A_{p-1}-$A_{p-1}suff_7$ are respectively embedded using R (rules of situation 1, group 1, figure 5.14) onto paths between:

- In the same component $0CQ_{l-1}$ (induction hypothesis): in the same component $000CQ_{l-3}$, between $000CQ_{l-3}$ and $001CQ_{l-3}$, between $000CQ_{l-3}$ and $010CQ_{l-3}$, and between $000CQ_{l-3}$ and $010CQ_{l-3}$ and $011CQ_{l-3}$.
- By **Definition** : between $000CQ_{l-3}$ and $100CQ_{l-3}$.
- By **Definition** and in the same component $1CQ_{l-1}$ (induction hypothesis): between $000CQ_{l-3}$ and $100CQ_{l-3}$ and $101CQ_{l-3}$, and between $000CQ_{l-3}$ and $100CQ_{l-3}$ and $110CQ_{l-3}$.

Except for edges between $A_{p-1}suff_1$-$A_{p-2}1suff_1$, $A_{p-1}suff_1$-$A_{p-2}1suff_2$, $A_{p-1}suff_1$-$A_{p-2}1suff_3$, $A_{p-1}suff_1$-$A_{p-2}1suff_4$, $A_{p-1}suff_1$-$A_{p-2}1suff_5$, $A_{p-1}suff_1$-$A_{p-2}1suff_7$, and $A_{p-1}suff_1$-$A_{p-2}1suff_6$ are respectively embedded using R (rules of situation 1, group 2, figure 5.15) onto paths:

- In the same component $0CQ_{l-1}$ (induction hypothesis): between $001CQ_{l-3}$ and $000CQ_{l-3}$ and in the same component $000CQ_{l-3}$, in the same component $001CQ_{l-3}$, between $001CQ_{l-3}$ and $011CQ_{l-3}$ and $010CQ_{l-3}$, between $001CQ_{l-3}$ and $011CQ_{l-3}$.
- By **Definition** and in the same component $1CQ_{l'-1}$ (induction hypothesis): between $001CQ_{l-3}$ and $101CQ_{l-3}$ and $100CQ_{l-3}$, and between $001CQ_{l-3}$ and $101CQ_{l-3}$ and $111CQ_{l-3}$.
- By **Definition** : $001CQ_{l-3}$ and $101CQ_{l-3}$.

For edges of any $A_{p-1}1TSP\text{-}COT_{k-t}$ where $k - t = 3$; edges between level's 1, 2 nodes are embedded using R (rules of situation 1, group 3, figure 5.15); all edges $A_{p-2}1suff_2$-$A_{p-3}12suff_1$, $A_{p-2}1suff_7$-$A_{p-3}17suff_1$ are embedded using R (rules of situation 2, case 2, figure 5.17, (b)). All edges of $A_{p-1}12TSP\text{-}COT_2$ are embedded using R (rules of situation 2, case 1, figure 5.16, (b)).

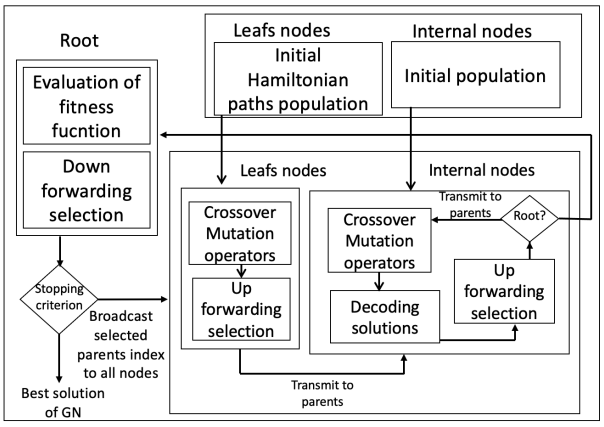


Figure 5.18: Process of generating the Initial Population [133].

5.4 Processing: Hybrid Parallel solving [133]

Following the preprocessing stage, where we strategically decompose the TSP into manageable subproblems through clustering, we proceed to the processing step. In this phase, we employ optimization algorithms tailored to handle the localized clusters generated earlier. These algorithms are designed to efficiently solve the smaller subproblems, taking advantage of the partitioning introduced during preprocessing.

The process functions within a crossed cubes interconnection network, as outlined in Algorithm 2. This framework consists of leaf nodes (basic nodes), internal nodes (leader nodes), and a root node, as depicted in Figure 5.18. Each leaf node autonomously generates an initial population of concise open Hamiltonian paths for small-scale TSP. This generation is facilitated by employing effective heuristics such as nearest neighbor and ant colony optimization, which construct individual solutions efficiently.

Next, at the internal nodes, the process combines subpath solutions using a genetic representation coding approach. The individuals at these nodes are composed of two key components: firstly, an arrangement of the subpaths from the child nodes, where each subpath's position is determined (e.g., left child first, middle child second, right child third); secondly, a binary string that signifies the connection points between these subpaths (0 indicating the starting point and 1 indicating the ending point). For a visual example illustrating the encoding and decoding of an individual at internal nodes, refer to Figure 5.19.

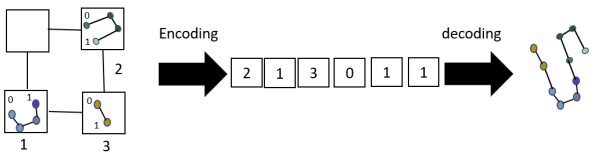


Figure 5.19: Encoding/decoding an individual in internal nodes [133].

At the root node, the process conducts an evaluation and downward forwarding selection of

Algorithm 2: Generate Initial Population

Data: Hierarchical topology, Maximum iterations
Result: Initial population of solutions
Parameters : hierarchicalTopology, maxIterations
Initialization:; Set the values of hierarchicalTopology; Set the value of maxIterations;
Generate Initial Population:;
for *each leafNode* ***in*** *hierarchicalTopology* **do**
    ▷ Leaf Node Operation
    paths ← generateHeuristicPaths() ▷ e.g., nearest neighbor or ant colony optimization
    addPathsToPopulation(leafNode, paths)
end
for *each internalNode* ***in*** *hierarchicalTopology* **do**
    ▷ Internal Node Operation
    childPaths ← getPathsFromChildren(internalNode)
    individual ← encodeIndividual(internalNode, childPaths)
    addIndividualToPopulation(internalNode, individual)
end
for *each node* ***in*** *hierarchicalTopology* **do**
    ▷ Up Forwarding Selection, Crossover, and Mutation
    if *node isNotRootNode* **then**
        if *node isLeafNode* **then**
            upForwardingSelection(node)
            mutateSubpaths(node)
            addPathsToPopulation(leafNode, paths)
        end
        else
            upForwardingSelection(node)
            crossoverAndMutate(node)
            paths ← decodeIndividuals(internalNode, childPaths)
            addPathsToPopulation(InternalNode, paths)
        end
    end
    else
        ▷ Root Node Operation
        evaluateAndDownSelect(hierarchicalTopology)
    end
end
    ▷ Stopping Criteria
if *iterations* < *maxIterations* **then**
    GenerateInitialPopulation(hierarchicalTopology, maxIterations) ▷ Recursive call for the next iteration
end

solutions, focusing on minimizing the TSP tour distance objective. This selection process is carried out using a tournament selection method. Subsequently, the root node shares the indices of the selected solutions with all other nodes through broadcasting. For a detailed procedure, refer to Algorithm 3.

Algorithm 3: Evaluate and Down Forwarding Selection

Data: Hierarchical topology
Result: Indices of selected solutions
Evaluation and Down Forwarding Selection:;
for *each rootChild* ***in*** *rootChildren* **do**
    ▷ Calculate and store tour distances
    tourDistance ← calculateTourDistance(rootChild.solution)
    storeTourDistance(rootChild, tourDistance)
end
selectedIndices ← tournamentSelection(rootChildren)
broadcastSelection(hierarchicalTopology, selectedIndices)

Crossover and mutation operations are executed in a distributed manner at each node, starting from the leaf nodes and progressing upwards to the internal nodes. Initially, leaf nodes handle these operations on their respective subpaths, and the results are subsequently propagated to the internal nodes, where recombination of subpath orderings and connections occurs.

Before transmitting updated solutions to parent nodes, a process called Up forwarding selection takes place to refine the population and enhance diversity. Each node selects one of three methods randomly: best-fit (favoring minimum distance tours), worst-fit (favoring maximum distance tours), or tournament selection. The iteration count serves as the stopping criterion. Refer to Algorithm 4 for a detailed explanation.

Algorithm 4: Up Forwarding Selection

Data: Node
Result: Selected individuals
Up Forwarding Selection:;
Procedure upForwardingSelection(node) ▷ Randomly choose one of three methods: best-fit, worst-fit, or tournament selection method ← randomlyChooseMethod()
if *method* == *"best-fit"* **then**
    bestFitSelection(node)
end
if *method* == *"worst-fit"* **then**
    worstFitSelection(node)
end
if *method* == *"tournament-selection"* **then**
    tournamentSelection(node)
end

The primary objective of generating the initial population is to explore the search space

comprehensively, aiming to identify a diverse array of candidate solutions covering a broad spectrum of objective functions. This approach provides a solid foundation for the subsequent local search algorithm in the second stage.

5.5 Processing: Optimization for refinement initial solutions [133]

After obtaining the initial population through the hybrid parallel algorithm, a simulated annealing metaheuristic is employed to refine and optimize the solutions further. The localized clusters created during the initial population generation process play a crucial role in this refinement phase by facilitating efficient perturbation moves, which enhance exploration capabilities.

Various neighborhood search structures, such as k-opt, insertion, relocate, inverse, and swap operators, are utilized to generate new solutions.

The temperature schedule for the simulated annealing process is meticulously designed to ensure effective convergence. It decreases linearly according to the following formula:

$$T = \frac{T_0}{1 + \lambda \left(\frac{\text{iteration}}{\text{tot\_iter}}\right)^{\beta}} \tag{5.1}$$

Here, T_0 represents the initial temperature, serving as a scaling factor that determines the overall rate of decrease for T. λ controls the rate of decrease as iterations progress, while β governs the overall behavior of the temperature schedule. The parameters iteration and tot_iter denote the current iteration number and the total number of iterations to be performed, respectively.

5.6 Visualisation and Interpritation

Once optimal or near-optimal solutions to the TSP instances have been acquired, it becomes essential to visualize and interpret the outcomes. This step offers valuable insights into how effectively the proposed algorithm leverages problem structures across various scales.

5.6.1 Illustrating the structures of clusters through visualization

Mapping the hierarchical clustering process highlights the significant divisions of cities at every level of the tree. By visualizing these clusters on instance maps, we uncover closely-knit local groupings that accurately represent geometric similarities. This process serves to confirm the effectiveness of the quadtree construction approach.

5.6.2 Illustrating the graph embedding

Through visual representations, we aim to offer a comprehensive understanding of how the hierarchical TSP representation aligns with the interconnected nodes within the crossed cube topology. Our goal is to create visual narratives that effectively convey the synergy between the hierarchical representation approach and the distinctive characteristics of the crossed cubes network.

5.6.3 Illustrating the solution paths

Visualizing the optimized TSP routes produced by each algorithm provides a straightforward understanding of their effectiveness and arrangement. By employing advanced visualization methods, we illustrate these paths on Euclidean planes, facilitating a direct assessment of their lengths, patterns, and overall performance.

5.6.4 Interpreting the boundary interactions

Examining the connections established between clusters helps to comprehend the interfaces and interaction hubs within the problem. This insight directs additional preprocessing efforts, such as considering merging boundary cities, to optimize the problem further.

5.6.5 Evaluating of solutions, and runtime: Statistical examination

Thorough statistical analyses, encompassing measures such as mean, standard deviation, and relative error, are undertaken to validate the significance of observed disparities. These evaluations enable us to draw confident conclusions regarding algorithmic superiority across varying conditions. Moreover, to enhance the depth of understanding regarding the influence of parallelization versus congestion levels, we complemented statistical analyses of runtimes with insightful visual representations. By integrating graphical visualization with statistical analysis, we enrich the interpretation of experiments, providing nuanced and visual confirmation of our findings.

5.7 Conclusion

In this chapter, we have presented a comprehensive framework for addressing the Traveling Salesman Problem (TSP) through advanced modeling techniques. We explored feature selection, highlighting enhancements to clustering algorithms like Enhanced k-means, K-Affinity Propagation (AP), and K-Density Peaks Clustering (K-DPC). Additionally, we detailed the process of TSP-Compressed Quadtree/Octree-based recursive hybrid clustering, as well as the embedding of the TSP-Compressed Quadtree/Octree into a Crossed Cubes Interconnection Network. This embedding aspect is crucial for optimizing the TSP representation and exploiting the unique properties of the crossed cubes network for efficient traversal. Emphasis was placed on hybrid parallel solving techniques to refine initial solutions and enhance optimization efficacy. Visualization and interpretation provided insights into cluster structures, graph embedding, and solution paths. This framework sets the stage for subsequent chapters

containing simulations, results, and discussions, aiming to contribute to the advancement of optimization methodologies for the TSP.

Chapter 6

Experimental Results of Implementation of Proposed Paradigm and Evaluation

6.1 Introduction

In this chapter, we delve into the simulation of our proposed modeling paradigm, aimed at enhancing the efficiency and effectiveness of solving the Traveling Salesman Problem (TSP). Building upon the comprehensive exploration of TSP-solving techniques in the previous chapter, we detail our contributions and innovations in feature selection, optimization, and visualization modules. Our focus is particularly on our novel approach to feature selection, which is designed to address the inherent complexities of the TSP and improve solution quality. This chapter serves as a practical demonstration of our proposed paradigm, where we apply our techniques to simulated TSP instances to evaluate their performance and effectiveness in TSPLIB instances. Through rigorous simulations, we aim to validate the efficacy of our approach and provide insights into its potential applications in solving real-world TSP instances.

6.2 Environment

For our implementation and experimental evaluation, we utilized high-performance computing resources equipped with an Intel Xeon processor and 64 GB of RAM. Our custom Python-based algorithm made extensive use of widely-used open-source packages, including Anaconda for data science tasks and Scikit-learn for its efficient machine learning tools such as clustering. To adapt Scikit-learn's implementations to our approach, we customized certain functionalities. Additionally, we harnessed the mpi4py library for parallel and distributed simulations. While the MPI framework inherently supports Cartesian topologies using functions like Create_cart, our proposed crossed cubes topology posed a challenge as existing MPI functions didn't directly support it. To overcome this, we developed a new custom function within our mpi4py implementation, enabling the establishment of the crossed cube interconnection between

processing elements as required by our algorithm.

6.3 Datasets

To evaluate the efficiency and scalability of our intelligent algorithm, we conducted tests using benchmark TSP instances sourced from the TSPLIB library [122]. This library offers a wide range of problem sizes, spanning from small to large instances, enabling us to validate our method across a spectrum of complexities. Our selection of instances encompassed a diverse sample, including variations in size (small, medium, and large) and structural properties, ensuring comprehensive validation of our approach.

6.4 Analyzing the performance of enhanced K-means

We conducted experiments on five distinct instances in assessing the proposed enhancement and its two variants (Eil51, Berlin52, Eil76, KroA100, and Eil101). Furthermore, we compared the performance of enhanced K-means with standard K-means. To evaluate clustering quality and compactness, we employed the Davies-Bouldin Index (DBI), while the Gini coefficient was used to gauge the distribution of results among clusters. The parameter r in our proposition holds significant importance as it directly influences both the clustering quality and the distribution of points across clusters.

6.4.1 Comparative performance analysis (standard K-means Vs Enhanced variants)

The comparative analysis of clustering techniques, as depicted in Table 6.1, involved evaluating standard K-means and its enhanced variants (1 and 2) across multiple TSP instances. The findings consistently show that the enhanced variants outperformed standard K-means regarding the DBI metric, with DIAMETER_KMEANS exhibiting superior cluster separation and compactness. Although Gini values remained relatively consistent across all methods, the enhanced variants displayed promise in achieving a more balanced distribution of data among clusters.

Solely focusing on DBI values, our enhanced K-means variants (1 and 2) exhibit slightly better performance than the standard K-means algorithm in terms of clustering quality and cluster compactness. Specifically, DIAMETER_KMEANS consistently achieves the most favorable results in terms of DBI, as illustrated in Table 6.2. For example, in the Eil51 instance, DIAMETER_KMEANS achieves the lowest DBI of 1.16, compared to 1.20 for K-means and 1.17 for SSE-SPLITTING_KMEANS. Similar trends are observed across other instances such as Berlin52, Eil76, and KroA100, where DIAMETER_KMEANS consistently produces the lowest DBI values, underscoring its superior clustering performance. Refer to Figure 6.1 for visualization.

According to the Gini values, both SSE-SPLITTING_KMEANS and DIAMETER_KMEANS demonstrate a slightly more balanced distribution of data points among clusters compared to the standard K-means algorithm. Notably, DIAMETER_KMEANS consistently achieves

Instance	Method	DBI	Gini
Eil51	K-means	1.20	0.23
	SSE-SPLITTING_KMEANS	1.19	0.21
	DIAMETER_KMEANS	1.17	0.22
Berlin52	K-means	1.38	0.26
	SSE-SPLITTING_KMEANS	1.27	0.23
	DIAMETER_KMEANS	1.36	0.24
Eil76	K-means	1.17	0.24
	SSE-SPLITTING_KMEANS	1.16	0.23
	DIAMETER_KMEANS	1.16	0.24
kroA100	K-means	1.30	0.30
	SSE-SPLITTING_KMEANS	1.30	0.26
	DIAMETER_KMEANS	1.26	0.29
Eil101	K-means	1.19	0.24
	SSE-SPLITTING_KMEANS	1.18	0.23
	DIAMETER_KMEANS	1.17	0.24

Table 6.1: DBI and Gini Values for Different Instances and Methods in K-means and Enhanced K-means Variants.

optimal results in terms of Gini values, as outlined in Table 6.3. For instances such as KroA100, Eil76, and Eil101, DIAMETER_KMEANS achieves the lowest Ginis, with values of 0.26, 0.22, and 2.21, respectively, indicating its superior performance compared to both KMEANS and SSE-SPLITTING_KMEANS. However, it's worth noting that KroA100 exhibits higher Gini indices across all methods, suggesting that clustering performance may be more challenging for this particular instance. Refer to Figure 6.2 for visualization.

6.4.2 Influence of Parameter r to enhanced K-means

Our thorough experimental investigations shed light on how two crucial parameters, labeled r_1 and r_2, impact the optimization effectiveness of our proposed enhanced K-means variants across a range of problem instances. By fixing K at 3 clusters, we meticulously analyze the effects of varying r_1 and r_2 values on the performance of SSE-SPLITTING_KMEANS and DIAMETER_KMEANS using key evaluation metrics. The optimal parameter settings depend on the specific optimization objectives, as summarized in Table 6.4.

For maximizing cluster separation and compactness measured by the DBI, optimal r_1 and r_2 values for DIAMETER_KMEANS typically range between 3.5-6, while SSE-SPLITTING_KMEANS performs best at approximately 1.5-8. These carefully chosen parameters enable our enhanced K-means variants to form distinct, tightly-knit clusters, overcoming the limitations of traditional K-means. Conversely, if creating clusters with balanced data distributions is paramount, as measured by the Gini coefficient, both variants excel when r_1 and r_2 are tuned between 1.25-4. This parameterization facilitates the creation of evenly populated clusters, addressing

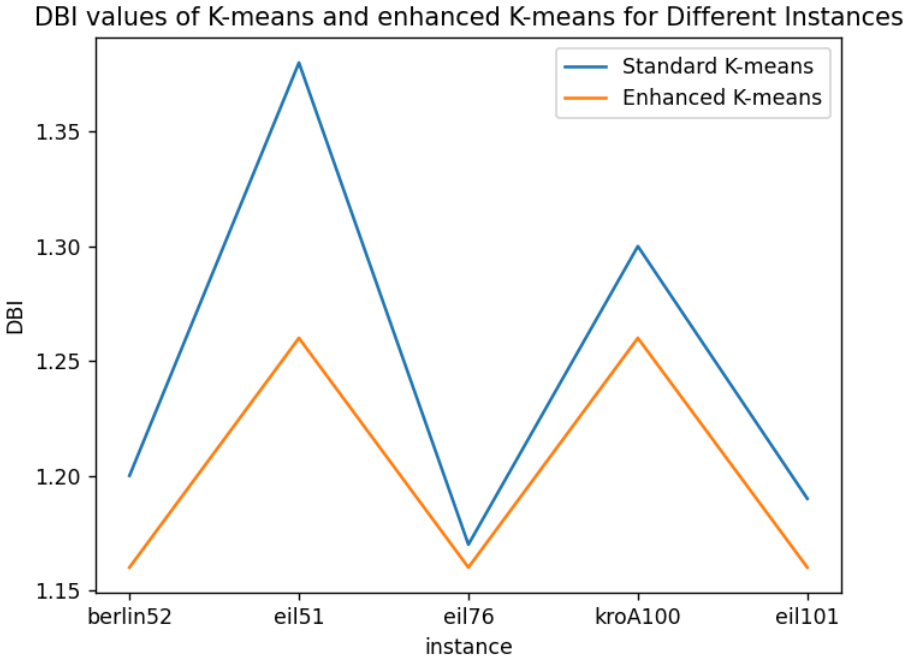


Figure 6.1: Ilustration of DBI values of K-means and enhanced K-means for Different Instances [129].

Instance	Method	DBI
Eil51	K-means	1.20
	SSE-SPLITTING_KMEANS	1.17
	DIAMETER_KMEANS	1.16
Berlin52	K-means	1.38
	SSE-SPLITTING_KMEANS	1.26
	DIAMETER_KMEANS	1.35
Eil76	K-means	1.17
	SSE-SPLITTING_KMEANS	1.16
	DIAMETER_KMEANS	1.16
kroA100	K-means	1.30
	SSE-SPLITTING_KMEANS	1.29
	DIAMETER_KMEANS	1.26
Eil101	K-means	1.19
	SSE-SPLITTING_KMEANS	1.16
	DIAMETER_KMEANS	1.17

Table 6.2: DBI values of K-means and Enhanced K-means Variants for Different Instances.

Instance	Method	Gini
Eil51	K-means	0.23
	SSE-SPLITTING_KMEANS	0.18
	DIAMETER_KMEANS	0.20
Berlin52	K-means	0.26
	SSE-SPLITTING_KMEANS	0.21
	DIAMETER_KMEANS	0.23
Eil76	K-means	0.24
	SSE-SPLITTING_KMEANS	0.23
	DIAMETER_KMEANS	0.22
kroA100	K-means	0.30
	SSE-SPLITTING_KMEANS	0.26
	DIAMETER_KMEANS	0.26
Eil101	K-means	0.24
	SSE-SPLITTING_KMEANS	0.22
	DIAMETER_KMEANS	0.21

Table 6.3: Gini values of K-means and Enhanced K-means Variants for Different Instances.

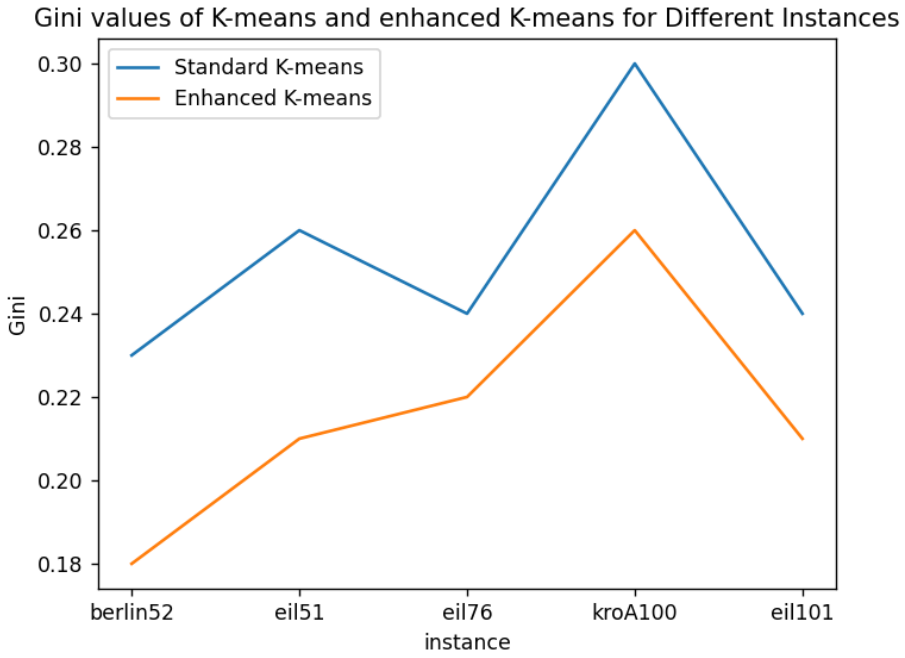


Figure 6.2: Ilustration of DBI values of K-means and enhanced K-means for Different Instances [129].

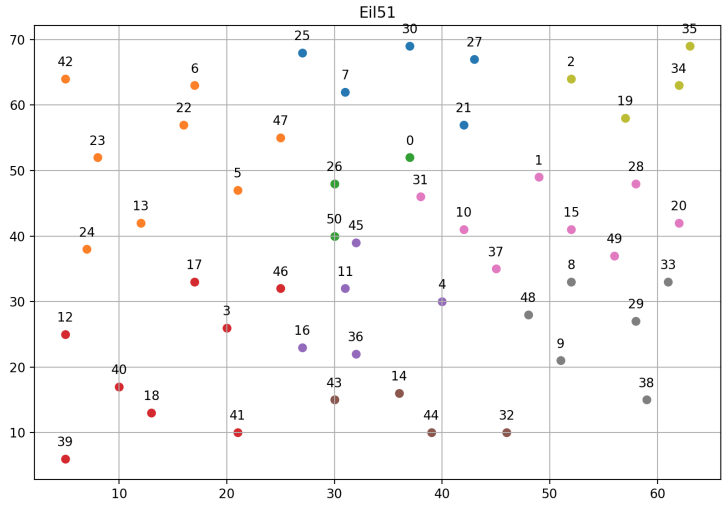


Figure 6.3: Result of the proposed top-down hybrid clustering for the instance Eil51 [133]

any imbalances. Considering the composite metric that combines Gini and DBI, our variants demonstrate robust performance across various instances. In situations where reducing DBI through enhanced cluster cohesion is prioritized, optimal r_1 and r_2 values for DIAMETER_KMEANS and SSE-SPLITTING_KMEANS typically range around 3.5-6 and 1.5-8, respectively.

6.5 Constructing the Compressed Quadtree/Octree

In detailing our experimental setup and methodology for constructing a compressed quadtree/octree (with a primary focus on the compressed quadtree), we outline the procedural steps for our experiments. This critical phase in our research involves defining the intricate process of constructing the compressed quadtree, wherein we carefully specify the algorithmic steps and parameters chosen to ensure accurate and meaningful experimentation.

Our innovative approach to hierarchical clustering operates in a top-down manner, iteratively dividing cities into a compressed quadtree structure that captures multi-scale groupings. By employing techniques such as K-means, affinity propagation, and density peaks clustering at each split, we optimize the decomposition process to create cohesive clusters across different levels of granularity. This methodology is effectively demonstrated on various TSP instances of varying scales, selected from the TSPLIB dataset, as depicted in Figures 6.3, 6.4, and 6.5.

Our method involves constructing compressed quadtrees for TSP instances including Eil51, Berlin52, KroA200, and Pr1002. Figures 6.8, 6.7, and 6.6 provide visual representations of sample quadtrees for Eil51, Berlin52, and KroA200, respectively. Initially, all cities are grouped within the root node, forming a single cluster. Through recursive partitioning, internal nodes are split into child subgroups, eventually resulting in leaf nodes containing tightly clustered cities. The depth of the tree is indicated by the length of the address strings, with leaf nodes typically having the longest addresses. These trees demonstrate optimal spans and compact representation, facilitated by compression techniques.

Instance	Method	Metric	r1	r2
Eil51	SSE-SPLITTING_KMEANS	DBI	6	2.5
		Gini	2	1.75
		Gini+ DBI	5	2
	DIAMETER_KMEANS	DBI	5.25	2.25
		Gini	6	2
		Gini+ DBI	5	2
Berlin52	SSE-SPLITTING_KMEANS	DBI	8	4
		Gini	4	1.5
		Gini+ DBI	4	4
	DIAMETER_KMEANS	DBI	3	1.5
		Gini	4	1.5
		Gini+ DBI	3	1.75
Eil76	SSE-SPLITTING_KMEANS	DBI	1.75	3
		Gini	1.75	3
		Gini+ DBI	1.75	3
	DIAMETER_KMEANS	DBI	3.5	6
		Gini	4	1.5
		Gini+ DBI	3.5	6
KroA100	SSE-SPLITTING_KMEANS	DBI	2.75	1.75
		Gini	3	1.5
		Gini+ DBI	3	1.5
	DIAMETER_KMEANS	DBI	1.5	4
		Gini	1.25	4
		Gini+ DBI	1.5	4
Eil101	SSE-SPLITTING_KMEANS	DBI	1.75	5
		Gini	1.75	1.75
		Gini+ DBI	1.75	3
	DIAMETER_KMEANS	DBI	2.5	3.5
		Gini	2.5	1.25
		Gini+ DBI	2.5	3.5

Table 6.4: Ideal r values to Optimize Performance of Enhanced K-means Variants for Different Instances.

The TSP-compressed quadtree takes on the form of a spanning tree, where cities or clusters serve as vertices interconnected by edges representing parent-child relationships within the tree structure. This connectivity, combined with the multi-resolution clustering approach, facilitates swift extraction of localized neighborhoods for parallel optimization. The trees also exhibit resilience to faults, as demonstrated by their ability to bypass failed nodes. This resilience is illustrated in the sample figures (Figures 6.8, 6.7, and 6.6), which depict scenarios of one, two, or three-node failures, respectively.

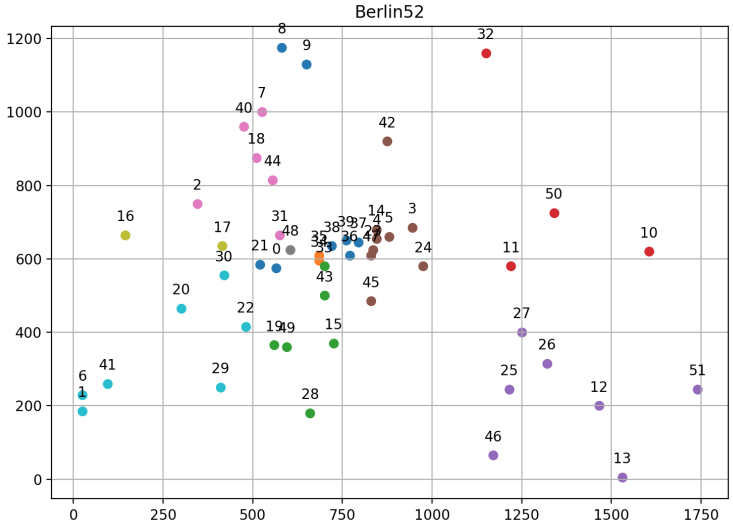


Figure 6.4: Result of the proposed top-down hybrid clustering for the instance Berlin52 [133].

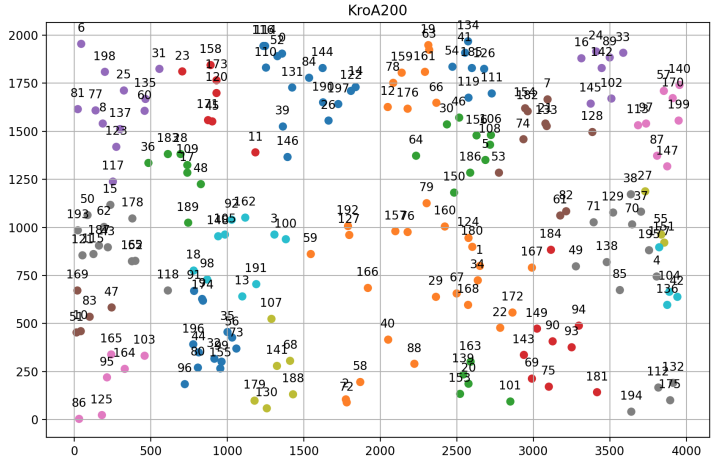


Figure 6.5: Result of the proposed top-down hybrid clustering for the instance KroA200 [133].

Our study presents a comprehensive analysis of three prominent clustering methods: Enhanced k-means, k-affinity propagation (K-AP), and k-density peaks clustering (K-DPC), applied to various TSP instances including Eil51 (small-scale), Berlin52 (small-scale), KroA200 (medium-scale), and Pr1002 (large-scale). We focus on evaluating the clustering quality using metrics

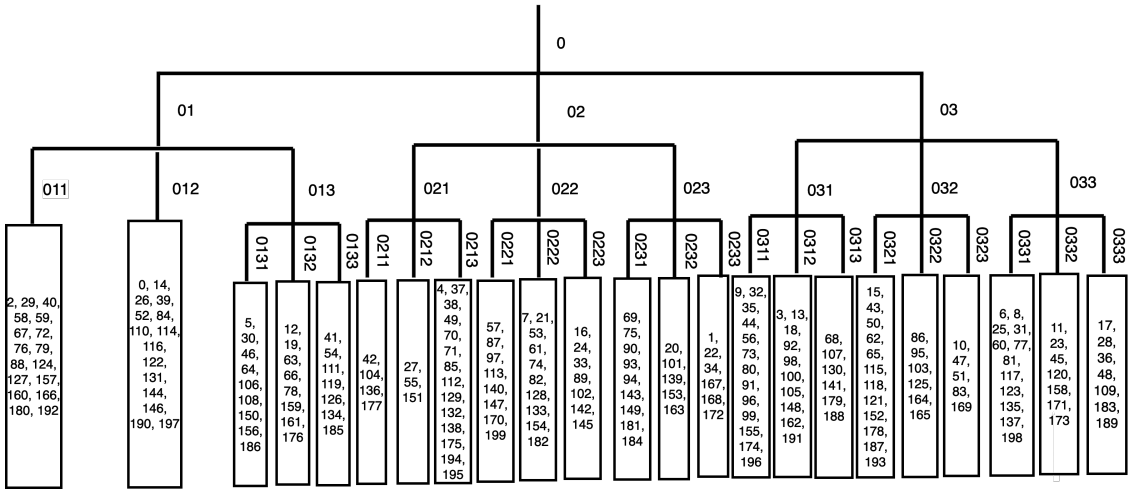


Figure 6.6: Hierarchical representation of KroA200 TSP instance based on our proposed algorithm [133].

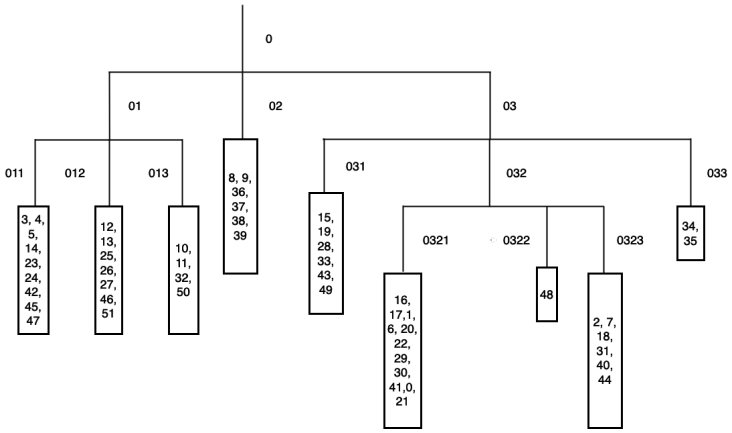


Figure 6.7: Hierarchical representation of Berlin52 TSP instance based on our proposed algorithm [133].

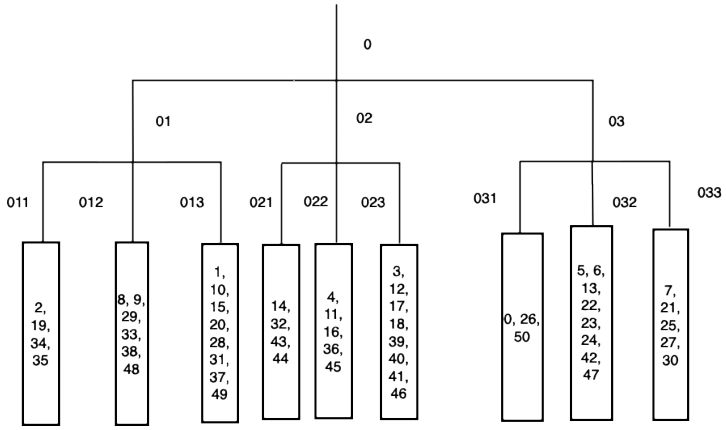


Figure 6.8: Hierarchical representation of Eil51 TSP instance based on our proposed algorithm [133].

such as the Davies-Bouldin Index (DBI) and Gini coefficients. By examining the trends and insights derived from these metrics, we gain valuable insights into the effectiveness of each clustering approach in constructing TSP-compressed quadtree representations.

6.6 Construct the graph one-by-one embedding of TSP-Compressed Quadtree/Octree representations into Crossed Cubes interconnection network

Our objective is to establish a direct mapping of hierarchical clusters, formed through quadtree decomposition of the TSP instance, onto the processing elements of a crossed cubes topology. This mapping process consists of two main steps: one-by-one vertex embedding and one-by-one edges embedding. Initially, the vertex embedding involves creating a mapping function that assigns each node in the compressed quadtree to a unique vertex in the crossed cube. This prioritizes linking connected nodes in the quadtree to nearby vertices in the crossed cube, based on their binary string labels. Subsequently, we refine the edges embedding function to ensure that every edge of the compressed quadtree corresponds to a path in crossed cubes with dilation 2. In the following sections, we provide examples of simulations based on the rules defined in the previous chapter, followed by simulations demonstrating the embedding of hierarchical TSP instances.

6.6.1 Simulation rules of embedding 2D/3D representation into Crossed Cubes

Embedding 2D representations into Crossed Cubes: Simulation rules

There are examples of the implementation of the basic function f for the one-by-one vertex embedding:

Embedding for Base Cases with Dimensions (n = 2). As depicted in Fig. 6.9.
Embedding for Base Cases with Dimensions (n = 3, 4, 6).

In this scenario, we extend the embedding process to include nodes at levels 1 and 2 of the hierarchical compressed quadtree structures $TSP\text{-}CQT_3$, $TSP\text{-}CQT_4$ and $TSP\text{-}CQT_6$, respectively. These nodes are mapped into corresponding crossed cubes topologies (CQ_4, CQ_6 and CQ_9) following the guidelines outlined in Table 6.5, Table 6.6, and Table 6.7. This mapping process is visually represented in Fig. 6.10 and Fig. 6.11.

There are examples of the implementation of the basic function f_1 for the one-by-one vertex embedding:

Embedding for Base Case with Dimension (n = 5, 8).

In this context, we extend the embedding process to encompass nodes at levels 1 and 2 of the hierarchical compressed quadtree structure $TSP\text{-}CQT_5$, and $TSP\text{-}CQT_8$. These nodes are integrated into the corresponding crossed cube topology CQ_7 following the guidelines

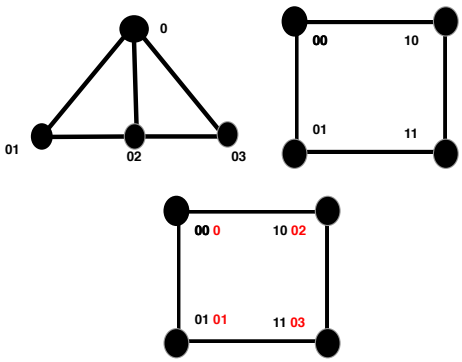


Figure 6.9: Nodes embedding graph of $TSP\text{-}CQT_2$ into CQ_2.

$Root$	$Prem(root)$	$0suff_1$	$01CQ_2$
0	0000	01	0100
$0suff_2$	$10CQ_2$	$0suff_3$	$11CQ_2$
0	1000	03	1100

Table 6.5: Level's 1, 2 nodes embedding of $TSP\text{-}CQT_3$ into CQ_4.

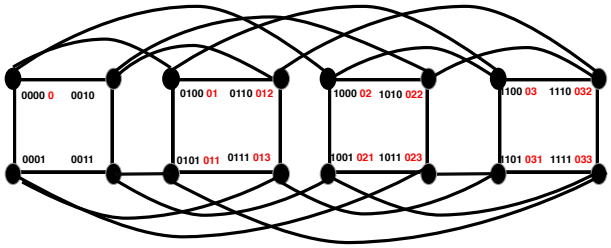


Figure 6.10: Nodes embedding graph of $TSP\text{-}CQT_3$ into CQ_4.

$Root$	$Prem(root)$	$0suff_1$	$01CQ_4$
0	000000	01	010000
$0suff_2$	$10CQ_4$	$0suff_3$	$11CQ_4$
02	100000	03	110000

Table 6.6: Level's 1, 2 nodes embedding of $TSP\text{-}CQT_4$ into CQ_6.

$Root$	$Prem(root)$	$0suff_1$	$pref_1CQ_4$
0	000000000	01	010000000
$0suff_2$	$pref_2CQ_4$	$0suff_3$	$pref_3CQ_4$
02	100000000	03	110000000

Table 6.7: Level's 1, 2 nodes embedding of $TSP\text{-}CQT_6$ into CQ_9.

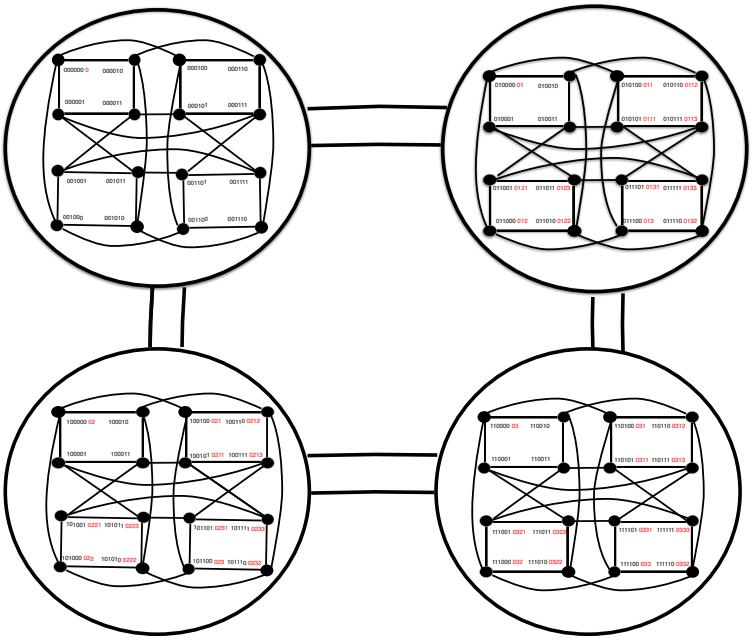


Figure 6.11: Nodes embedding graph of $TSP\text{-}CQT_4$ into CQ_6.

$Root$	$Prem(root)$	$0suff_1$	$0pref_1 CQ_4$
0	0000000	01	0010000
$0suff_2$	$1pref_1 CQ_4$	$0suff_3$	$1pref_0 CQ_4$
02	1010000	03	1000000

Table 6.8: Level's 1, 2 nodes embedding of $TSP\text{-}CQT_5$ into CQ_7.

delineated in Table 6.8, and Table 6.9. The visual representation of this embedding process is depicted in Fig. 6.12.

There are examples of the implementation of the basic function R for the Dilation two one-by-one edges embedding:

Embedding for Base Cases (n = 2-4), (n=6).

In this context, edges connecting vertices at level 1 and level 2 of the hierarchical quadtree structure $TSP\text{-}CQT_n$ are integrated using the guidelines outlined in Tables 6.10, and 6.11.

There are examples of the implementation of the basic function R_1 for the Dilation two one-by-one edges embedding:

Embedding for Base Case (n = 5,8).

In this scenario, edges within the hierarchical quadtree structure $TSP\text{-}CQT_5$, and $TSP\text{-}CQT_8$ are integrated using the guidelines outlined in Tables 6.12, 6.13.

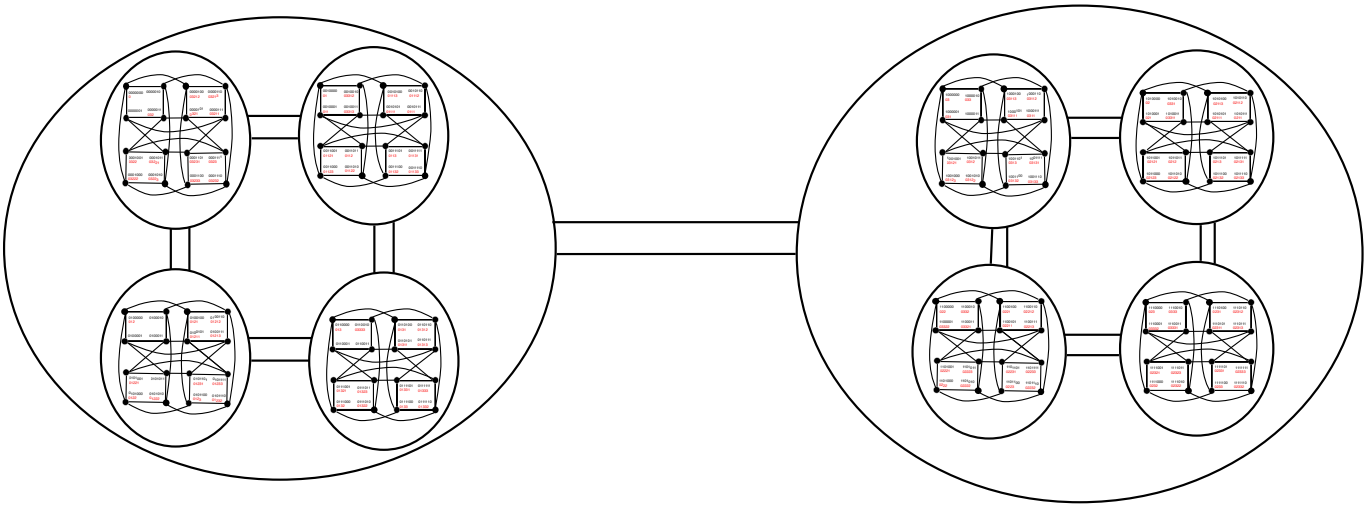


Figure 6.12: Nodes embedding graph of $TSP\text{-}CQT_5$ into CQ_7.

$Root$	$Prem(root)$	$0suff_1$	$0pref_1 CQ_4$
0	00000000000	01	01000000000
$0suff_2$	$1pref_1 CQ_4$	$0suff_3$	$1pref_0 CQ_4$
02	10100000000	03	10000000000

Table 6.9: Level's 1, 2 nodes embedding of $TSP\text{-}CQT_8$ into CQ_{12}.

	TSP-CQT	**CQ paths**	**Dil**
$n = 2$	0-01	00-01	1
	0-02	00-10	1
	0-03	00-10-11	2
$n = 3$	0-01	0000-0100	1
	0-02	0000-1000	1
	0-03	0000-1000-1100	2
$n = 4$	0-01	000000-010000	1
	0-02	000000-100000	1
	0-03	000000-100000-110000	2

Table 6.10: Edges embedding between the vertex of level 1 and level 2 of $TSP\text{-}CQT_2$, $TSP\text{-}CQT_3$, $TSP\text{-}CQT_4$.

TSP-CQT	**CQ paths**	**Dil**
0-01	000000000-010000000	1
0-02	000000000-100000000	1
0-03	000000000-100000000-110000000	2

Table 6.11: Edges embedding of $TSP\text{-}CQT_6$ into CQ_9.

	TSP-CQT	CQ paths	Dil
A	0-01	0000000-0010000	1
	0-02	0000000-0010000-1010000	2
	0-03	0000000-1000000	1
B	01-011	0010000-0010001	1
	01-012	0010000-0110000-0100000	2
	01-013	0010010-0110000	1
C	03-031	1000000-1000001	1
	03-032	1000000-1000001-0000011	2
	03-033	1000000-1000010	1
D	0331-03311	1010010-1010011	1
	0331-03312	1010010-0010010	1
	0331-03313	1010010-0010010-0010011	2

Table 6.12: Edges embedding of $TSP\text{-}CQT_5$ into CQ_7, A: example of situation 2, case 1; B: example of situation 1, case 2; C: example of situation 2, case 2; D: example of situation 2, case 3.

Embedding 3D representations into Crossed Cubes: Simulation rules

There are examples of the implementation of the basic function f for the one-by-one vertex embedding:

Base Case (n = 2).

For $n = 2$, the embedding process is illustrated in Figure 6.13.

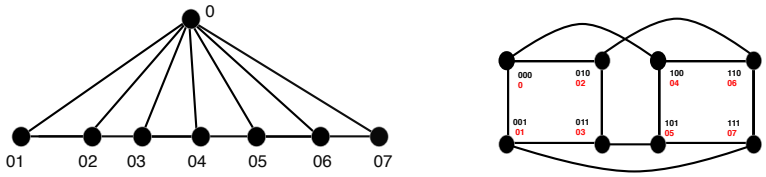


Figure 6.13: Nodes embedding graph of $TSP\text{-}COT_2$ into CQ_3.

Base Case (n = 3).

For $n = 3$, the embedding process involves nodes at levels 1 and 2 of $TSP\text{-}COT_3$, as well as the nodes with suffixes 01 and 07, utilizing the rules specified in Table 6.14 and Table 6.15 respectively. This process is depicted in Figure 6.14.

	TSP-CQT	**CQ paths**	**Dil**
A	0-01	000000000000 -001000000000	1
	0-02	000000000000-001000000000 -101000000000	2
	0-03	000000000000 -100000000000	1
B	03-031	100000000000 -100010000000	1
	03-032	100000000000- 100010000000-000010000000	2
	03-033	100000000000 -100000000001	1
C	01-011	001000000000 -001010000000	1
	01-012	001000000000- 011000000000-010000000000	2
	01-013	001000000000 -011000000000	1
D	011-0111	001010000000 -001011000000	1
	011-0112	001010000000- 001110000000-001100000000	2
	011-0113	001010000000 -001110000000	1
E	0111-01111	001011000000 -001011010000	1
	0111-01112	001011000000- 001011010000-001010010000	2
	0111-01113	001011000000 -001011000001	1

Table 6.13: Edges embedding of $TSP\text{-}CQT_8$ into CQ_{12}, A: example of situation 2, case 1; B: example of situation 2, case 2; C, D: example of situation 1, case 2; E: example of situation 2, case 1.

Root	*Prem(root)*	$0suff_1$	$000\,CQ_3$
0	000000	01	000001
$0suff_2$	$001\,CQ_3$	$0suff_3$	$010\,CQ_3$
02	001000	03	010000
$0suff_4$	$011\,CQ_3$	$0suff_5$	$100\,CQ_3$
04	011000	05	100000
$0suff_6$	$101\,CQ_3$	$0suff_7$	$110\,CQ_3$
06	101000	07	110000

Table 6.14: Level's 1, 2 nodes embedding of $TSP\text{-}COT_3$ into CQ_6.

$01suff_1$	$110CQ_3$	$07suff_1$	$111CQ_3$
011	110001	071	111000

Table 6.15: Embedding of $01suff_1$, $07suff_1$ into $110CQ_3$, $111CQ_3$.

Root	*Prem(root)*	$0suff_1$	$pref_0CQ_3$
0	000000000	01	000001000
$0suff_2$	$pref_1CQ_3$	$0suff_3$	$pref_2CQ_3$
02	001000000	03	010000000
$0suff_4$	$pref_3CQ_3$	$0suff_5$	$pref_4CQ_3$
04	011000000	05	100000000
$0suff_6$	$pref_5CQ_3$	$0suff_7$	$pref_6CQ_3$
06	101000000	07	110000000

Table 6.16: Level's 1, 2 nodes embedding of $TSP\text{-}COT_4$ into CQ_9.

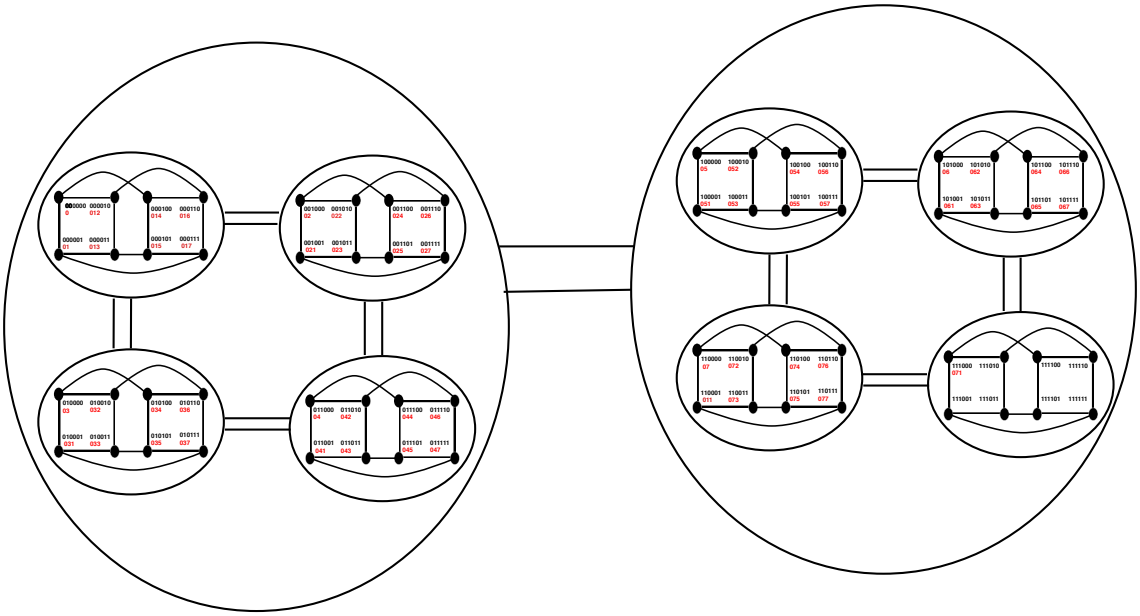


Figure 6.14: Nodes embedding graph of $TSP\text{-}COT_3$ into CQ_6.

Base Case (n = 4).

For $n = 4$, the embedding process involves nodes at levels 1 and 2 of $TSP\text{-}COT_4$, utilizing the rules specified in Table 6.16. Furthermore, nodes with suffixes 011suff$_1$, 01suff$_2$, 012suff$_1$, and 017suff$_1$ are embedded using the rules specified in Table 6.17.

There are examples of the implementation of the basic function R for the Dilation two one-by-one edges embedding:

Base Case (n $\leq$ 4). For $n \leq 4$, edges of any *sub-TSP-COT*$_2$ are embedded using the rules specified in Table 6.18.

$01suff_2$	$pref_1CQ_3$	$012suff_1$	$pref_7CQ_3$
012	000001001	0121	000111001
$01suff_7$	$pref_7CQ_3$	$017suff_1$	$pref_6CQ_3$
017	000111000	0171	000110000

Table 6.17: Embedding of $01suff_2$, $012suff_1$, $01suff_7$, $017suff_1$ into $pref_1CQ_3$, $pref_7CQ_3$, $pref_7CQ_3$, $pref_6CQ_3$.

TSP-COT	**CQ paths**	**Dil**
$0-01$	$000-001$	1
$0-01$	$000-001$	1
$0-02$	$000-010$	1
$0-03$	$000-010-011$	2
$0-04$	$000-100$	1
$0-05$	$000-100-101$	2
$0-06$	$000-100-110$	2
$0-07$	$000-001-111$	2

Table 6.18: Example of edges embedding of sub-$TSP\text{-}COT_2$ onto sub-CQ_3 for $n \leq 4$.

For $n \leq 4$, edges between level 1 and level 2 nodes of any *sub-TSP-COT*$_3$ are embedded as specified in Table 6.20. Additionally, edges such as $A_{p-1}\text{suff}_1 - A_{p-2}1\text{suff}_1$ and $A_{p-1}\text{suff}_7 - A_{p-2}7\text{suff}_1$ of *TSP-COT*$_3$ are embedded as specified in Table 6.19.

Base Case (n = 4).

For $n = 4$, edges of *TSP-COT*$_n$ are embedded as Table 6.22. Edges: $A_{p-2}1\text{suff}_2 - A_{p-3}12\text{suff}_1$ and $A_{p-2}1\text{suff}_7 - A_{p-3}17\text{suff}_1$ of *sub-TSP-COT*$_3$ are embedded as Table 6.21.

6.6.2 Graph embedding of 2D Euclidean *TSP(instance)-CQT*$_n$ into CQ$_m$

A visualization was created to illustrate how a sample compressed quadtree structure is transformed into a crossed cubes network. Each node within the crossed cubes network is depicted as a circle containing a unique binary address string. Larger circles encompass groups of four adjacent vertices, representing higher-level supernodes. To highlight the significance of the prefix portion of the address, it's displayed in red. Additionally, the position of each

TSP-COT	**CQ path**	**Dil**
$01-011$	$000001-100011$ -110001	2
$07-071$	$110000-111000$	1

Table 6.19: Example of embedding edges $A_{p-1}suff_1$-$A_{p-2}1suff_1$, $A_{p-1}suff_7$-$A_{p-2}7suff_1$ of sub-$TSP\text{-}COT_3$.

TSP-COT	**CQ paths**	**Dil**
$0-01$	$000000-000001$	1
$0-02$	$000000-001000$	1
$0-03$	$000000-010000$	1
$0-04$	$000000-010000$ -011000	2
$0-05$	$000000-100000$	1
$0-06$	$000000-100000$ -101000	2
$0-07$	$000000-100000$ -110000	2

Table 6.20: Example of embedding edges between level's 1, 2 nodes of sub-$TSP\text{-}COT_3$ onto sub-CQ_6 for $n \leq 4$.

TSP-COT	**CQ paths**	**Dil**
$01-012$	$000001000-000001001$	1
$012-0121$	$000001001-000101011$ -000111001	2
$01-017$	$000001000-000101000$ -000111000	2
$017-0171$	$000111000-000110000$	1

Table 6.21: Edges embedding of $A_{p-2}1suff_2$-$A_{p-3}12suff_1$ $A_{p-2}1suff_7$-$A_{p-3}17suff_1$ of $A_{p-2}1\,TSP\text{-}COT_3$ (example of embedding using rules of situation 2, case 2).

TSP-COT	**CQ paths**	**Dil**
$0-01$	$000000000-000001000$	1
$0-02$	$000000000-001000000$	1
$0-03$	$000000000-010000000$	1
$0-04$	$000000000-010000000$ -011000000	2
$0-05$	$000000000-100000000$	1
$0-06$	$000000000-100000000$ -101000000	2
$0-07$	$000000000-100000000$ -110000000	2

Table 6.22: Embedding edges between level's 1, 2 nodes of $TSP\text{-}COT_4$ onto CQ_9 using rules of situation 1, group 1.

node within the supernodes is encoded using the two least significant bits.

Nodes belonging to the compressed quadtree are marked in blue, with arrows indicating their positions within the crossed cubes. Straight lines represent edges, while dilation 2 paths are emphasized in red.

The graph embedding representing the *TSP(KroA200)-*CQT_4 into a 6-dimensional crossed cubes (CQ_6) is visually illustrated in the subsequent figures (6.15, 6.16, 6.18, and 6.17).

For instance, the connection between node 01 and node 013 in the compressed quadtree (represented as 01-013) corresponds to the path between node 010000, node 011000, and node 011100 in the crossed cubes topology, depicted as 010000-011000-011100.

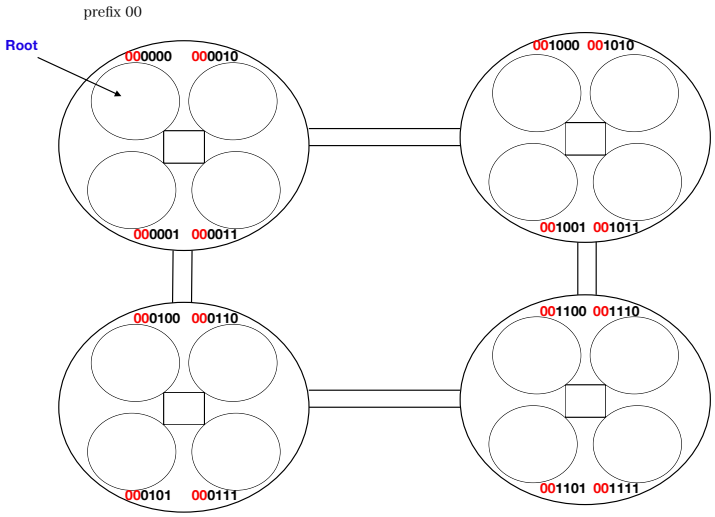


Figure 6.15: The graph embedding of the *TSP(KroA200)-*CQT_4 into CQ_6 (First copy $00CQ_4$).

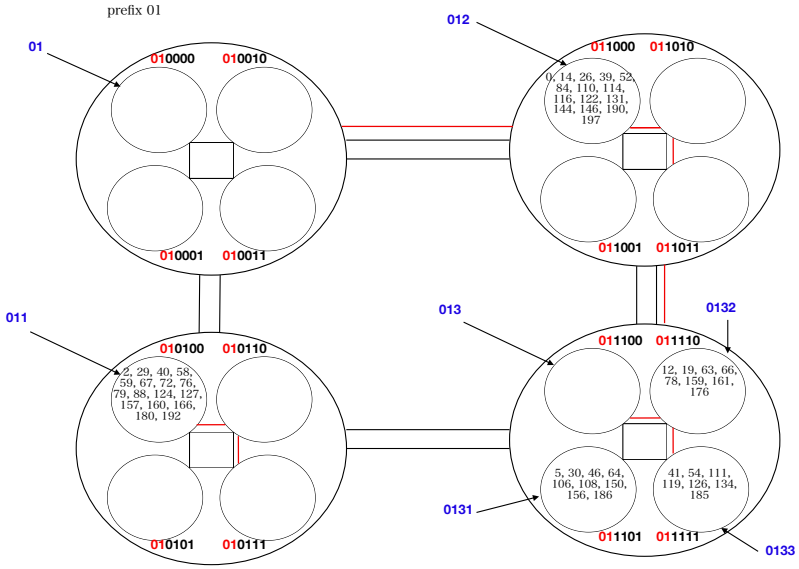


Figure 6.16: The graph embedding of the *TSP(KroA200)-*CQT_4 into CQ_6 (Second copy $01CQ_4$).

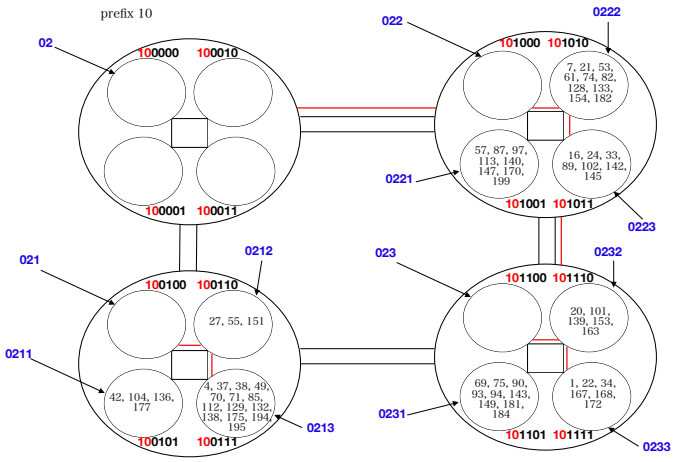


Figure 6.17: The graph embedding of the *TSP(KroA200)-CQT*$_4$ into CQ_6 (Third copy $10CQ_4$).

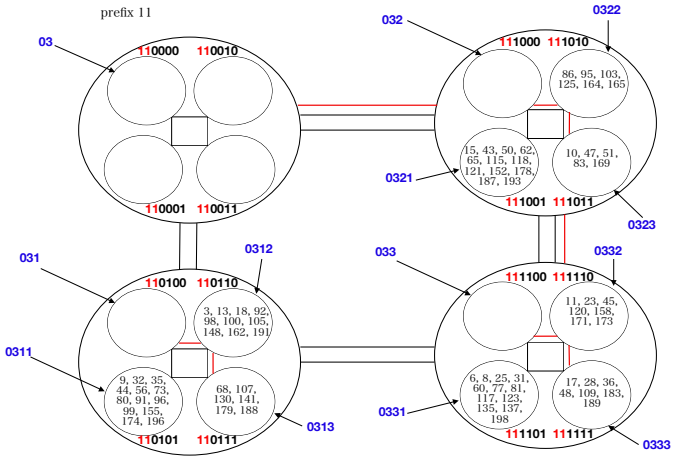


Figure 6.18: The graph embedding of the *TSP(KroA200)-CQT*$_4$ into CQ_6 (Last copy $11CQ_4$).

6.7 Processing: Construct intra-cluster solutions and inter-connection between clusters

6.7.1 Scale of each cluster and the selection of methods for local optimization

The size of clusters at the leaf nodes significantly affects the computational complexity of solving the Traveling Salesman Problem (TSP) within each cluster. When dealing with smaller clusters, typically containing around 20 or fewer cities after hierarchical decomposition, simpler and faster construction heuristics like nearest neighbor are effective. Nearest neighbor operates by sequentially connecting the closest unvisited city, resulting in a locally optimal tour. Its straightforward approach and linear time complexity of $O(n^2)$ make it well-suited for smaller clusters.

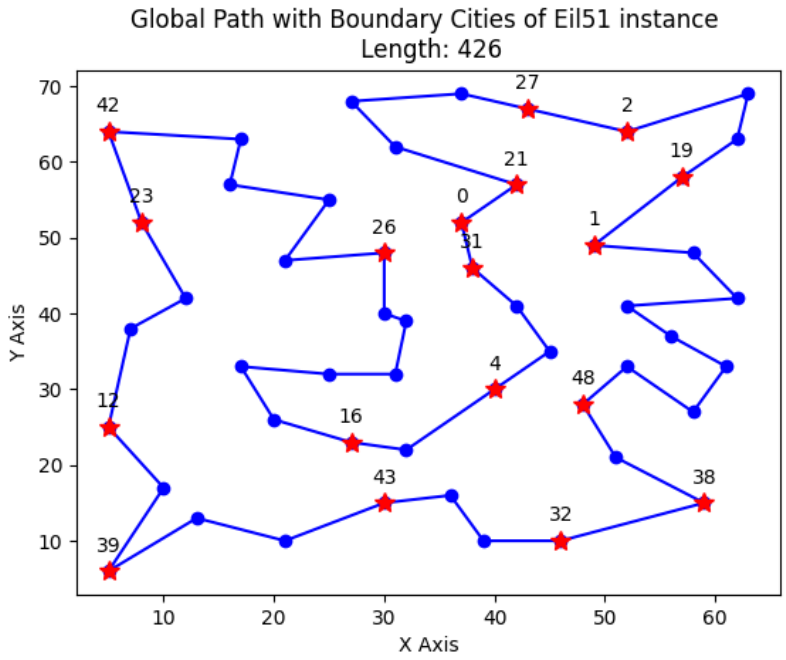


Figure 6.19: Global Path with Boundary Cities of Eil51 instance [133].

However, as the problem size increases and hierarchical partitioning leads to larger leaf node clusters, typically comprising 40-50 cities, the efficacy of nearest neighbor diminishes. In such cases, more advanced metaheuristics like Ant Colony Optimization (ACO) become necessary. ACO employs probabilistic construction guided by virtual pheromone trails to efficiently explore the solution space. Despite the increase in complexity, ACO maintains a manageable time complexity of $O(n^2)$ for these cluster sizes, while also offering the flexibility of parameter tuning to enhance search performance.

6.7.2 Determining the boundaries' cities

In this phase, we concentrate on creating a diverse initial population of TSP tours following the construction of the hierarchical decomposition. Initially, we generate open Hamiltonian paths within each leaf node cluster using both nearest neighbor and ACO metaheuristics. These paths offer a range of high-quality intra-cluster options, and they also identify candidate boundary cities located at the interfaces between clusters. By examining the endpoints of these paths and their intersections with neighboring paths, we pinpoint boundary nodes that link adjacent clusters. For instance, in the illustration of the global paths for instances Eil51, Berlin52, and KroA200, boundary cities are highlighted in Figures 6.19, 6.20, and 6.21. These boundary cities serve as promising points for potential inter-cluster routes.

Subsequently, the internal nodes combine the subpath solutions from the child nodes to generate complete global tours. To achieve this, we develop a genetic network heuristic that evolves orderings and connections between the subpath solutions through selection, crossover, and mutation operations. This process plays a crucial role in determining the key boundary cities at each level of the hierarchy.

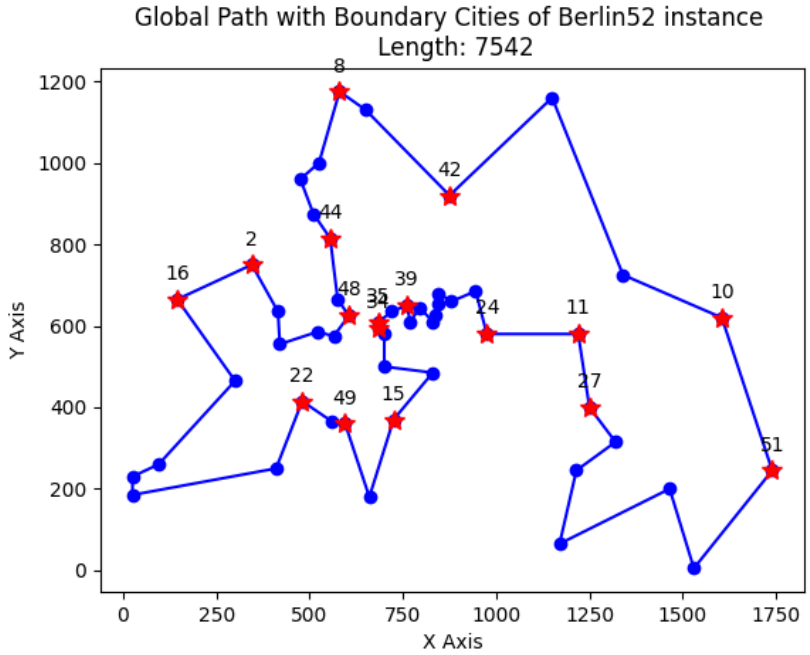


Figure 6.20: Global Path with Boundary Cities of Berlin52 instance [133].

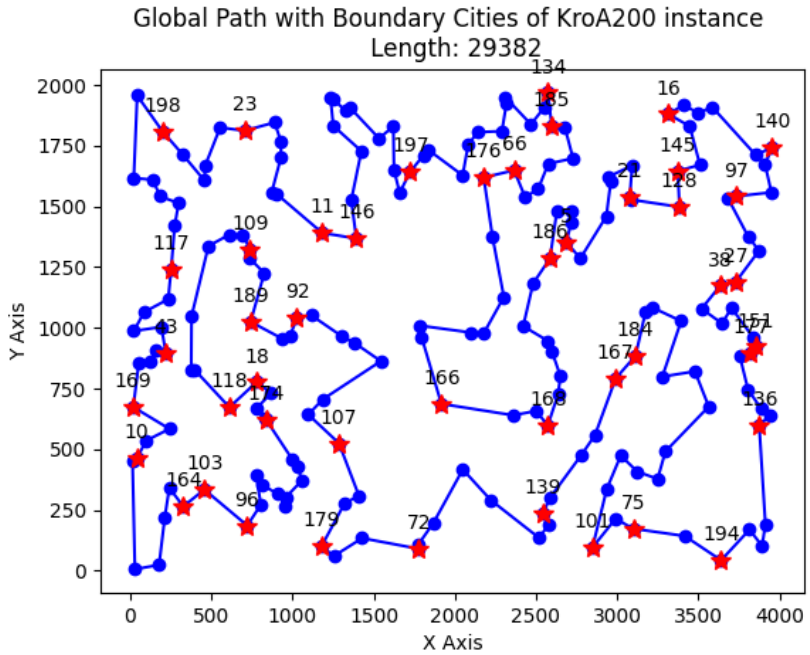


Figure 6.21: Global Path with Boundary Cities of KroA200 instance [133].

An essential aspect of this stage involves effectively selecting the boundary cities to optimize the division and recombination of subproblems. We utilize features such as graph centrality metrics to identify cities that minimize distances between subpaths when integrated globally. This approach ensures the construction of tours that are locally optimal within clusters while also maintaining good global optimality.

6.7.3 Parameters settings

To tailor the algorithm to the characteristics of each cluster, we adjust the parameters of the ACO algorithm based on the city distribution within the cluster. Table 6.23 presents the parameter values used. This customization ensures that the algorithm can generate diverse and high-quality initial solutions, effectively exploring the search space and increasing the likelihood of finding an optimal solution. Additionally, in implementing the nearest neighborhood approach, we consider tours starting from various cities within each cluster. By exploring multiple starting city options, the algorithm diversifies initial solutions and prevents convergence to local optima. For determining the parameters of the genetic network, we

α	β	ρ	Q	*Ant*	*Iteration*
0.9-1.1	1-5	0.1	1	number of cities	80-300

Table 6.23: Parameter values of ACO metaheuristic.

consider both solution quality and runtime, adapting them based on the scale of the TSP instances. The population size (n) and number of generations (g) are set according to the instance size. Typically, the population size is twice the number of cities (m), while the number of generations is calculated using a formula that considers the number of iterations (k), the height of the compressed quadtree (h), and a factor (t) related to the total number of cities. Different genetic operators are applied to leaf and internal nodes, with inversion and insertion used for leaf nodes and a combination of crossover, flip mutation, inversion, and insertion for internal nodes. These strategies leverage the hierarchical structure for an efficient search process.

The up forwarding selection phase is critical, as it identifies and preserves the best local solutions contributing to the overall global solution. This phase employs a hybrid selection strategy that balances selecting top performers with maintaining diversity. By combining exploitation of the best tours with exploration of alternative options, this technique fosters a balanced exploration-exploitation tradeoff for the TSP. It ensures that the initial population nurtures both high-performing building blocks and divergent individuals, thus avoiding entrapment in local optima.

6.8 Initial solutions refinement

Choosing the right temperature is crucial for the success of the simulated annealing phase in our algorithm. Since the initial solution obtained from the genetic network is often close to the optimal solution, it's advisable to use a low temperature during simulated annealing. This

allows for thorough exploitation of the local search space. For larger instances, fine-tuning the algorithm's exploration-exploitation balance can lead to improved performance and more accurate TSP solutions.

Understanding the neighborhood search structures is key to selecting the most effective operator for the simulated annealing phase. By combining targeted simulated annealing with specialized local search operators, our approach consistently optimizes the initial population to discover superior TSP tours. While the genetic network facilitates global exploration and quality initialization, simulated annealing and tailored operators enable in-depth local exploitation.

6.9 Results and Discussion

6.9.1 Effectiveness of clustering methods for hierarchical TSP representation

Detailed performance insights are provided in Tables 6.27, 6.26, 6.28, 6.25, and 6.24, which offer a comprehensive analysis of three clustering methods employed: Enhanced k-means, k-affinity propagation (K-AP), and k-density peaks clustering (K-DPC). These results are evaluated across various scales of TSP instances, including small-scale instances like Eil51 and Berlin52, middle-scale instance such as KroA200, and larger-scale instances like rd400 and Pr1002. Through these analyses, the effectiveness of our approach in terms of DBI and Gini metrics is thoroughly examined and highlighted.

Examining the performance of clustering methods across different levels of the compressed quadtree reveals intriguing variations. For small instances like Eil51 and Berlin52, K-DPC consistently achieves lower DBI scores and Gini coefficients compared to Enhanced k-means and K-AP. This suggests K-DPC's proficiency in creating well-separated and internally cohesive clusters across various compression levels. Such consistent performance prompts further investigation into K-DPC's robustness and suitability for datasets with similar complexities.

Conversely, for middle-scale and large-scale instances such as KroA200, rd400, and Pr1002, K-DPC's performance fluctuates across nodes and levels. Unlike smaller instances, K-DPC doesn't consistently attain the lowest DBI scores and Gini coefficients across all levels, indicating its sensitivity to underlying data characteristics.

A notable finding is the discernible performance disparities among different nodes within the same compressed quadtree level. This suggests that certain parts of the dataset may favor specific clustering methods due to unique data distributions. This variation underscores the importance of considering local data traits in clustering outcomes and raises the prospect of hybrid methods leveraging this locality for enhanced results.

The Gini coefficient, reflecting within-cluster inequality, offers additional insight into clustering outcomes. Lower Gini coefficients signify a more balanced distribution of data points within clusters. While K-DPC consistently maintains lower Gini coefficients across levels for all instances, Enhanced k-means and K-AP also demonstrate competitive performance. This suggests that Enhanced k-means and K-AP effectively manage data point distributions within

	DBI			Gini		
Level	**Enhanced K-means**	**K-AP**	**K-DPC**	**Enhanced K-means**	**K-AP**	**K-DPC**
Level 0 (root 0)	**1.24**	1.25	1.31	**0.21**	0.22	0.22
Level 1 (Node 01)	**1.36**	1.38	1.59	**0.11**	0.11	0.11
Level 1 (Node 02)	**1.18**	1.23	1.34	**0.14**	0.14	0.15
Level 1 (Node 03)	1.37	**1.33**	1.31	0.26	**0.26**	0.29
Level 2 (Node 011)	1.41	**1.35**	1.48	0.09	**0.09**	0.10
Level 2 (Node 012)	1.28	**1.25**	1.28	0.15	**0.14**	0.15
Level 2 (Node 013)	**1.19**	1.24	1.19	**0.05**	0.07	0.06
Level 2 (Node 021)	1.20	1.20	**1.17**	0.11	0.11	**0.13**
Level 2 (Node 022)	1.31	**1.25**	1.25	0.13	**0.12**	0.13
Level 2 (Node 023)	1.64	1.47	**1.19**	0.19	0.19	**0.13**
Level 2 (Node 031)	1.46	**1.28**	1.28	0.26	**0.30**	0.31
Level 2 (Node 033)	1.52	1.56	**1.43**	0.25	0.26	**0.27**
Level 3 (Node 0222)	**1.26**	1.27	1.29	**0.05**	0.05	0.05
Level 3 (Node 0231)	1.60	1.58	**1.38**	0.19	0.19	**0.16**
Level 3 (Node 0232)	1.47	1.75	**1.42**	0.13	0.12	**0.09**
Level 3 (Node 0311)	1.23	1.25	**1.17**	0.30	0.29	**0.30**
Level 3 (Node 0312)	1.37	1.49	**1.29**	0.23	0.21	**0.23**
Level 3 (Node 0313)	1.31	**1.27**	1.34	0.36	**0.37**	0.36
Level 3 (Node 0332)	**1.34**	1.89	1.82	**0.29**	0.30	0.32
Level 4 (Node 02313)	1.54	1.55	**1.45**	0.16	0.16	**0.11**
Level 4 (Node 03322)	1.24	1.22	**1.18**	0.30	0.31	**0.31**

Table 6.24: DBI and Gini values for different clustering algorithms across levels of the Pr1002 TSP instance hierarchical representation.

	DBI			Gini		
Level	**Enhanced K-means**	**K-AP**	**K-DPC**	**Enhanced K-means**	**K-AP**	**K-DPC**
Level 0 (root 0)	1.17	**1.16**	1.17	0.29	**0.29**	0.34
Level 1 (Node 01)	1.30	**1.28**	1.49	0.18	**0.19**	0.27
Level 1 (Node 02)	1.21	1.21	**1.16**	0.32	0.32	**0.34**
Level 1 (Node 03)	**1.21**	1.24	1.43	**0.26**	0.27	0.22
Level 2 (Node 012)	1.34	**1.27**	1.30	0.11	**0.11**	0.11
Level 2 (Node 013)	1.42	1.71	**1.19**	0.30	0.24	**0.35**
Level 2 (Node 021)	1.24	**1.19**	1.26	0.24	**0.25**	0.31
Level 2 (Node 023)	1.28	**1.22**	1.40	0.34	**0.33**	0.36

Table 6.25: DBI and Gini values for different clustering algorithms across levels of the Rd400 TSP instance hierarchical representation.

	DBI			Gini		
Level	**Enhanced K-means**	**K-AP**	**K-DPC**	**Enhanced K-means**	**K-AP**	**K-DPC**
Level 0 (root 0)	1.38	1.38	**1.33**	0.26	0.25	**0.21**
Level 1 (Node 01)	**1.24**	**1.24**	1.35	**0.23**	**0.23**	0.22
Level 1 (Node 03)	1.28	1.28	**1.28**	0.23	0.24	**0.16**
Level 2 (Node 032)	1.39	1.39	**1.32**	0.23	0.23	**0.15**

Table 6.26: DBI and Gini values for different clustering algorithms across levels of the Berlin52 TSP instance hierarchical representation.

Level	DBI			Gini		
	Enhanced K-means	**K-AP**	**K-DPC**	**Enhanced K-means**	**K-AP**	**K-DPC**
Level 0 (root 0)	1.20	**1.17**	1.25	0.23	**0.23**	0.22
Level 1 (Node 01)	1.36	1.36	**1.22**	0.11	0.12	**0.12**
Level 1 (Node 02)	1.23	1.23	**1.23**	0.23	0.23	**0.22**
Level 1 (Node 03)	1.27	1.27	**1.21**	0.26	0.26	**0.21**

Table 6.27: DBI and Gini values for different clustering algorithms across levels of the Eil51 TSP instance hierarchical representation.

Level	DBI			Gini		
	Enhanced K-means	**K-AP**	**K-DPC**	**Enhanced K-means**	**K-AP**	**K-DPC**
Level 0 (root 0)	**1.31**	1.32	1.46	**0.30**	0.30	0.31
Level 1 (Node 01)	**1.19**	**1.20**	1.29	**0.17**	**0.18**	0.15
Level 1 (Node 02)	**1.21**	1.24	1.35	**0.33**	0.32	0.25
Level 1 (Node 03)	1.22	1.22	1.15	0.33	0.33	0.39
Level 2 (Node 013)	**1.20**	**1.20**	1.32	**0.11**	**0.11**	0.11
Level 2 (Node 021)	1.45	1.45	**1.28**	0.37	0.37	**0.34**
Level 2 (Node 022)	1.28	**1.26**	1.37	**0.19**	0.19	0.20
Level 2 (Node 023)	**1.17**	**1.18**	1.39	**0.40**	**0.39**	0.41
Level 2 (Node 031)	**1.23**	1.24	1.29	**0.24**	0.25	0.24
Level 2 (Node 032)	**1.29**	**1.29**	1.31	**0.38**	**0.38**	0.39
Level 2 (Node 033)	**1.25**	**1.25**	1.41	**0.25**	**0.25**	0.19

Table 6.28: DBI and Gini values for different clustering algorithms across levels of the KroA200 TSP instance hierarchical representation.

	Eil51	**Berlin52**	**Eil76**	**KroA100**	**Eil101**	**Ch150**	**KroA200**	**Rd400**
BKS	426	7542	538	21282	629	6528	29368	15281
Best	426	7542	538	21282	629	6528	29382	15284
Mean	426.75	7542	540.1	21339.25	630.5	6540	29434.45	15364.16
Std	0.82	0.0	2.24	99.15	2.13	12	62.91	95.08
Error(%)	0.17	0.0	0.39	0.27	0.24	0.18	0.23	0.54
PE(%)	0.17	0.0	0.39	0.27	0.24	0.18	0.18	0.52
	D493	**Pr1002**						
BKS	35001	259045						
Best	35170	267284						
Mean	35253.14	268533.4						
Std	93.95	1452.12						
Error(%)	0.72	3.6						
PE(%)	0.24	0.47						

Table 6.29: Experimental Results of The Proposed Algorithm For Solving Euclidean TSP problem.

clusters.

These findings provide a nuanced understanding of clustering methods within TSP-compressed quadtree representations. Firstly, clustering algorithms significantly influence quality across hierarchical levels. Secondly, while quality varies across levels and nodes, Enhanced k-means showcases consistent versatility across instances. Lastly, the hierarchical structure captures varying cluster granularity, emphasizing the importance of hierarchical clustering in analyzing large, complex datasets.

These insights underscore the importance of tailoring clustering methods to data characteristics and pave the way for exploring hybrid approaches that combine the strengths of different methods.

6.9.2 Evaluation performance of processing algorithm

Table 6.29 displays the performance of our proposed approach across various benchmark TSP instances sourced from TSPLIB. For each case, the table reports the best, average, and standard deviation values obtained from 10 independent runs. Additionally, the table includes the lengths of the best-known solutions (BKS) for reference.

To facilitate comparison, we compute the relative error of the average tour length compared to the BKS, as defined in Equation 6.1. Lower relative errors indicate superior performance in terms of solution quality. Furthermore, we calculate the relative error between the average and best tour lengths among the 10 runs, as specified in Equation 6.2. This metric assesses the robustness of the approach, quantifying the variation between the averaged performance and the best individual performance across runs. Smaller values signify more consistent behavior across the different runs.

$$\text{Error } (\%) = \left(\frac{\text{Average} - \text{BKS}}{\text{BKS}} \right) \times 100\% \tag{6.1}$$

$$\text{PE } (\%) = \left(\frac{\text{Average} - \text{Best}}{\text{Best}} \right) \times 100\% \tag{6.2}$$

In smaller instances like eil51, berlin52, and eil76, our algorithm consistently discovers optimal or near-optimal solutions that closely match the best-known solution (BKS) lengths. The average tour lengths deviate by less than 0.4% from the BKS, showcasing the algorithm's effectiveness in handling small-scale cases. As we transition to medium-sized instances such as kroA100 and eil101, the average error remains below 0.3%, indicating the algorithm's scalability. Moreover, the standard deviations are minimal, indicating stable convergence.

For larger instances, the average error relative to the BKS is below 1%. Even for the pr1002 graph with over 1000 cities, the error is reasonable at 3.6%, underscoring the approach's viability for large-scale Euclidean problems. While the peak individual solutions in smaller graphs match or exceed the BKS, the best tours in larger graphs closely approximate the BKS.

Algorithm		Eil51	Berlin52	Eil76	KroA100	Eil101	Ch150	KroA200
ACO	Best	437	7547.0	562	22562.0	691.0	6771.0	31388.0
	Mean	445.5	7684.53	567.3	22884.8	694.9	6817.4	31734.2
	Std	5.85	60.56	3.1	158.55	2.34	19.03	205.01
	Error(%)	4.57	1.89	5.5	7.53	10.48	3.72	8.05
	PE(%)	1.95	1.82	0.94	1.43	0.56	0.69	1.10
GA	Best	429	7542.0	558.0	22780.0	672.0	8621.0	48296.0
	Mean	440.0	7833.43	574.836	23428.19	688.22	9113.03	53607.73
	Std	6.23	149.94	9.12	519.03	8.20	239.49	2743.08
	Error(%)	3.29	3.86	6.85	10.08	9.41	39.59	82.53
	PE(%)	2.56	1.89	5.5	7.53	10.48	3.72	8.05
SA-K-opt	Best	426	7542.0	543	21575.0	638.0	6702.0	30580.0
	Mean	432	7805.8	551.4	22227.7	651.7	6984.3	31774.4
	Std	4.40	136.34	5.57	496.88	47.81	135.17	460.93
	Error(%)	1.41	3.5	2.5	4.44	3.6	6.99	8.19
	PE(%)	4.57	1.89	1.55	3.03	2.15	4.21	3.91

Table 6.30: Experimental Results of the three metaheuristics: ACO, Genetic Algorithm(GA), hybrid simulated annealing with K-opt (SA-K-opt) For Solving Euclidean TSP problem.

Throughout the experiments, the performance error between the best and average tours consistently remains under 0.6%, demonstrating a high level of consistency and maintaining high-quality performance across runs.

Our experiments involved testing the proposed algorithm alongside three individual methods ACO, standard Genetic Algorithm (GA), and hybrid Simulated Annealing with K-opt (SA-K-opt) across several TSP instances: Eil51, Berlin52, Eil76, KroA100, Eil101, Ch150, and KroA200. The results are presented in Table 6.30 and compared with those of the proposed algorithm shown in Table 6.29.

In smaller instances like eil51 and berlin52, the proposed algorithm either matches or surpasses the best solutions found by the individual methods. Notably, the average error of 0.17-0.39% is lower than the minimum errors of 1.41-4.57% observed from ACO, GA, and SA-K-opt. As the instance size increases, our algorithm consistently delivers better average solution quality compared to the individual techniques. For instance, in kroA100, the hybrid error is 0.27% compared to the best error of 4.44% achieved among individual methods. Similarly, for kroA200, the error improves from a minimum of 8.05% (ACO) to 0.23% for the proposed algorithm.

Furthermore, the standard deviation of the proposed algorithm remains remarkably low across all instances, underscoring the stability and robustness of the solutions it produces. In contrast, the individual methods exhibit higher variances, indicating less reliability. Additionally, the performance error between the best and average solutions is the lowest for the proposed algorithm, reaffirming its consistency. By integrating multiple techniques, our approach effectively mitigates the limitations inherent in individual methods, leading to improved solution quality and reliability.

	Instance	Eil51	Berlin52	Eil76	KroA100	Eil101	Ch150	KroA200
	BKS	426	7542	538	21282	629	6528	29368
Proposed algorithm	Best	426	7542	538	21282	629	6528	29382
	Mean	426.75	7542	540.1	21339.25	630.5	6540	29434.45
	Std	0.82	0.0	2.24	99.15	2.13	12	62.91
	Error(%)	0.17	**0.0**	0.39	0.27	0.24	0.18	0.23
	PE(%)	0.17	0.0	0.39	0.27	0.24	0.18	0.18
DPC-ACO-KOpt (2018)	best	426	7542	538	21282	629	6528	29368
	Mean	426.25	7542.00	538.30	21283.65	630.35	6536.5	29368.40
	Std	0.44	0.00	0.80	5.50	2.01	14.36	39.08
	Error(%)	**0.06**	**0.0**	0.06	**0.01**	0.21	**0.13**	**0.10**
	Pe(%)	0.06	0.00	0.06	0.01	0.21	0.13	0.10
PACO-3opt (2016)	Mean	426.35	7542.00	539.85	21326.80	630.55	6601.40	29644.50
	Std	0.49	0.00	1.09	33.72	2.63	15.01	53.43
	Error(%)	0.08	**0.0**	0.34	0.21	0.25	1.12	0.94
C-PSO-ACO-Kopt (2017)	Mean	426.29	7543.29	538.15	21319.50	631.20	-	29642.00
	Std	0.46	3.90	0.65	47.79	1.5	-	154
	Error(%)	0.07	0.01	**0.02**	0.17	**0.17**	-	0.46
SA-ACO-PSO (2011)	Mean	427.27	7542.00	540.20	21370.30	635.23	6563.70	29738.73
	Std	0.45	0.00	2.94	123.36	3.59	22.45	356.04
	Error(%)	0.30	**0.0**	0.41	0.41	0.99	0.55	1.27

Table 6.31: Experimental results of different algorithms on small-medium scale Euclidean TSP instances.

Table 6.31 presents a comparative analysis of the proposed algorithm's performance against other algorithms when applied to small and medium-scale Traveling Salesman Problem (TSP) instances. The goal is to assess the reliability and consistency of our algorithm in delivering exceptional results, particularly for instances with a smaller number of vertices. We consider four algorithms for comparison: DPC-ACO-KOpt (2018) [96], PACO-3opt (2016) [64], C-PSO-ACO-Kopt (2017) [102], and SA-ACO-PSO (2011) [57]. It's important to note that these algorithms were not implemented in our study; instead, we reference the outcomes and results achieved by these algorithms when applied to the same instances.

In Table 6.32, we extend our evaluation to include three large-scale instances sourced from TSPLIB, each containing between 400 and 1002 points. Here, we compare the experimental results of our proposed algorithm with those of other algorithms, namely DPC-ACO-KOpt (2018) [96], PACO-3opt (2016) [64], HGA (2014) [145], CBA-NNM (2022) [127], and GGSC-SSA (2021) [150]. Again, it's essential to clarify that these algorithms were not implemented in our article; instead, we extract their experimental results from the original papers to facilitate comparative analysis.

For small instances, our algorithm achieves the optimal or near-optimal solutions matching the best-known solutions. This is comparable to the top techniques like DPC-ACO-KOpt. This shows its ability to solve small problems effectively. As the size increases to medium-scale instances, our approach attains lower means, and average errors than most other methods.

	Instance	Rd400	D493	Pr1002
	BKS	15281	35002	259045
Proposed algorithm	Best	**15284**	**35170**	**267284**
	Mean	15364.16	35253.14	268533.4
	Std	95.08	93.95	1452.12
	Error(%)	**0.54**	**0.72**	**3.6**
	PE(%)	0.52	0.24	0.47
DPC-ACO-KOpt(2018)	best	15333.00	35237.00	-
	Mean	15387.25	35347.5	-
	Error(%)	0.70	0.99	-
	Pe(%)	0.35	0.31	-
PACO-3opt (2016)	best	15578.00	35735.00	-
	Mean	15613.90	35841.00	-
	Error(%)	2.18	2.40	-
	Pe(%)	0.23	0.3	-
HGA(2014)	best	-	-	-
	Mean	15853.74	-	-
	Error(%)	3.74	-	-
	Pe(%)	-	-	-
CBA-NNM(2022)	best	-	35583	273825
	Mean	-	35864	277525
	Error(%)	-	2.46	7.13
	Pe(%)	-	0.78	1.35
GGSC-SSA(2021)	best	-	36470.63	271894.56
	Mean	-	-	-
	Error(%)	-	-	-
	Pe(%)	-	-	-

Table 6.32: Experimental results of different algorithms on large scale Euclidean TSP instances.

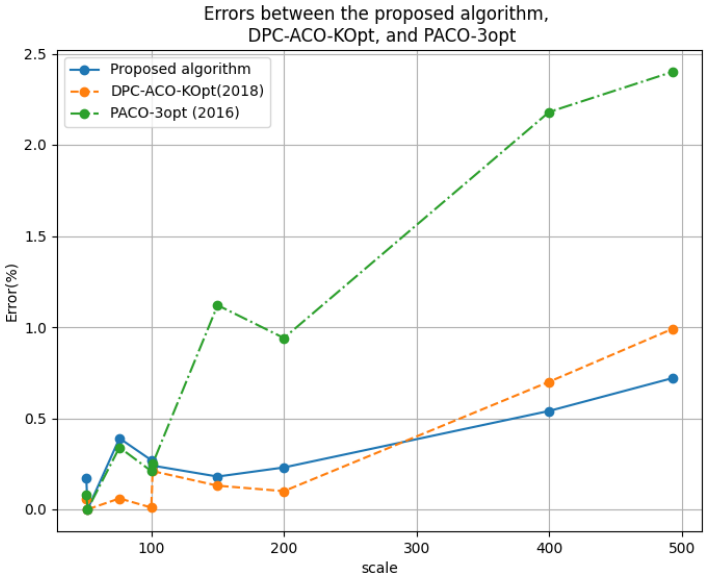


Figure 6.22: Errors between the proposed algorithm, DPC-ACO-KOpt, and PACO-3opt [133].

The performance evaluation of our algorithm was conducted by comparing its results on each small, medium-scale TSP instance with those obtained from four other algorithms. For each instance, our algorithm received a score based on its ranking concerning the quality of the solution achieved. Notably, it secured the second position among the five algorithms evaluated, indicating its competitive performance in solving these challenging small and medium-scale Euclidean TSP problems, see Fig. 6.22.

The thorough evaluation and comparison with five other state-of-the-art algorithms highlight the effectiveness of the proposed hybrid method in optimizing large-scale Euclidean Traveling Salesman Problem (TSP) instances. Our algorithm consistently achieves the best or near-best solutions, with errors ranging from 0.54% to 3.6% compared to the best-known solutions (BKS). It outperforms all other algorithms by consistently maintaining the lowest error percentages across instances.

While DPC-ACO-KOpt also performs well, it exhibits slightly higher errors ranging from 0.70% to 0.99% compared to our proposed algorithm. On the other hand, solutions generated by PACO-3opt have significantly higher errors, ranging from 2.18% to 2.4%, indicating limitations in handling complexity. Notably, our algorithm comes closest to the known optima for the largest instance (Pr1002) with a 3.6% error.

Overall, the experimental results demonstrate that our hybrid algorithm achieves the most effective solutions across all benchmark instances, irrespective of size or scale. It consistently outperforms five peer metaheuristics, as illustrated in Figure 6.22, by leveraging a synergistic integration of clustering-based decomposition with parallelized multi-strategy search. This underscores its superior ability to navigate complex and dense search landscapes. Moreover, with the lowest errors and tightest solution variances among the evaluated algorithms, our proposed hybrid optimization algorithm has established itself as a leading solver for the

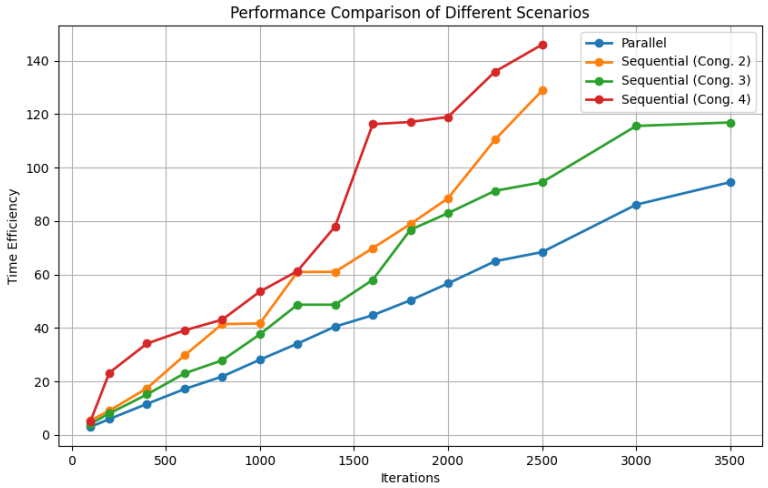


Figure 6.23: Time Efficiency Comparison Across Scenarios (Pr1002 instance) [134].

Euclidean TSP, systematically and reliably solving these nonlinear problems.

Nevertheless, challenges remain, including potential computational complexity and sensitivity to clustering parameters. Future research directions may explore additional combinations of metaheuristics, assess scalability to larger instances, and conduct further sensitivity analysis for parameter tuning. In summary, our proposed hybrid optimization algorithm represents a significant advancement in solving Euclidean TSP problems, demonstrating its efficacy and competitiveness against state-of-the-art approaches.

6.9.3 Runtime Evaluation [134]

A comprehensive evaluation was conducted to assess the computational effectiveness of our proposed approach using a benchmark set of Euclidean TSP instances spanning various sizes. We compared the runtime performance of parallel execution against sequential execution under different levels of network congestion (2, 3, and 4). Specifically, we analyzed the algorithm's execution across cores in parallel and sequentially with incremental communication contention.

In the first experiment, we selected the Pr1002 instance from the TSPLIB benchmark library as the initial test problem. This instance provided a suitable large-scale TSP formulation for evaluating runtime behavior under different execution scenarios. We aimed to discern differences in computational efficacy between leveraging full parallelism and operating under incremental channel congestion with constrained parallelism. Table 6.33 presents the runtime comparisons of our proposed hierarchical metaheuristic for the Euclidean TSP under four scenarios: parallel execution and sequential execution with congestions of 2, 3, and 4. Lower runtimes indicate better performance. Across all iterations, parallel execution consistently achieves the lowest runtimes, outperforming sequential execution by up to 2x-3x for larger iterations (see Figure 6.23). This highlights the significant benefits of parallelization in improving computational efficiency. Increasing congestion for sequential execution generally degrades performance due to higher communication costs. However, congestion 3 exhibits

Iteration	100	200	400	600	800	1000	1200	1400	1600	1800	2000	2250	2500	3000	3500
Parallel	3	5.91	11.58	17.15	21.83	28.11	34.14	40.52	44.77	50.36	56.65	64.98	68.40	86.15	94.57
Sequential (Cong. 2)	5.37	9.04	17.44	29.75	41.46	41.68	60.96	61.02	69.86	78.89	88.53	110.50	128.85	-	-
Sequential (Cong. 3)	4.15	8.01	15.1	23.02	27.89	37.6	48.71	48.71	58.00	76.73	82.97	91.34	94.52	115.61	116.92
Sequential (Cong. 4)	5.12	23.23	34.17	39.1	43.11	53.59	61.3	77.98	116.28	117.1	118.97	135.96	146.05	-	-

Table 6.33: Performance Comparison of Different Scenarios.

Scale	100	200	493	1002
Parallel	0.0048	0.032	0.20	13.03
Sequential (Cong. 2)	0.0049	0.054	0.33	24.3

Table 6.34: Performance Comparison at Different Scales.

better runtimes than congestion 2 for some mid-range iterations, possibly due to greater path diversity overcoming higher messaging overhead. The performance gap between parallel and sequential execution widens with more iterations as the scale increases.

Table 6.34 reports the runtimes for our proposed algorithm under parallel versus sequential execution with congestion 2 across four different TSP instance sizes (with 500 iterations). Runtimes scale sublinearly with problem size for both parallel and sequential executions (see Figure 6.24). However, parallelism provides significant speedup over sequential execution for all scales. For smaller instances (100 and 200 cities), runtimes are low and similar between parallel and sequential execution, suggesting computation is the dominant factor for these sizes. As the size increases, parallel runtime increases modestly while sequential runtime more than doubles, highlighting parallelism's ability to efficiently distribute computation load.

6.10 Conclusion

This chapter presents the evaluation of the proposed modeling paradigm for solving the Traveling Salesman Problem. Through a strategic blend of machine learning techniques, hybrid metaheuristics, hierarchical clustering, enhanced K-means clustering, and parallel computing paradigms, the research endeavors to unlock new avenues for efficient solution discovery across varying scales of problem instances.

The environment setup, detailed in Section 6.1, provides the foundation for conducting the experiments, ensuring consistency and reproducibility. Section 6.2 outlines the datasets employed for evaluation, essential for assessing the performance of the proposed paradigm across diverse scenarios.

The introduction of enhanced K-means clustering and its subsequent evaluation showcases promising advancements in achieving balanced cluster sizes and improving clustering quality. Such enhancements hold significant implications, particularly in domains where balanced and high-quality clustering outputs are paramount for meaningful analysis and decision-making.

Furthermore, The integration of machine learning techniques, as highlighted in the previous chapters, underscores the importance of adapting optimization strategies to the characteristics of the problem instance. By leveraging hierarchical clustering, the decomposition of the

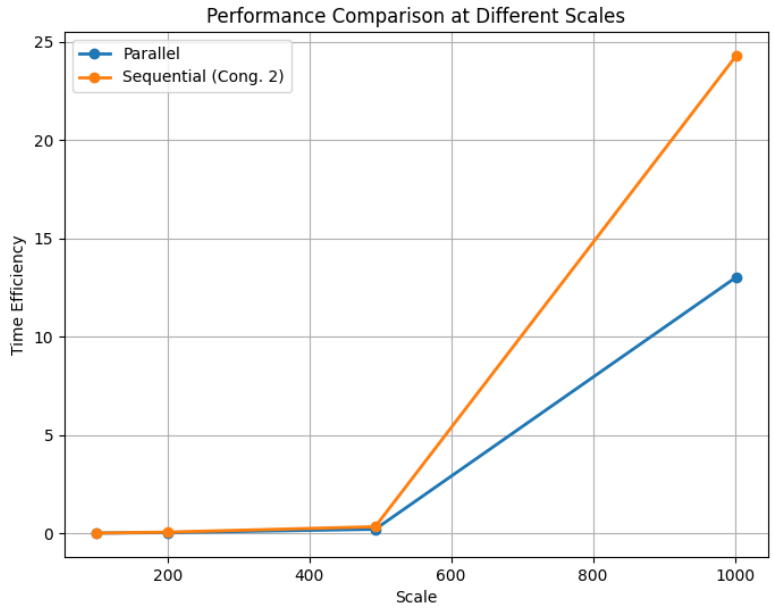


Figure 6.24: Scalability Analysis: Comparative performance across varied scales [134].

problem space into compressed 2D/3D hierarchical structures enables localized optimization and parallel exploration of subregions. However, the evaluation reveals scale-dependent differences in clustering technique performance, indicating the need for nuanced approaches across different scales.

The strategic embedding of TSP-compressed 2D/3D hierarchical structure into a crossed cubes topology with a dilation of 2. This integration serves to enhance resource allocation during parallel optimization efforts. Notably, our embedding achieves an optimal dilation of 2 and a load factor of 1, optimizing the efficiency and effectiveness of the parallel optimization process.

The hierarchical mapping presented in this chapter offers a novel perspective on distributing computational work to mitigate network congestion in TSP metaheuristics. By embedding natural city clusters into a parallel structure, the approach circumvents congestion hotspots, thereby accelerating the discovery of optimal Euclidean paths.

Chapter 7

General Conclusion and Perspectives

In conclusion, our thesis introduces a versatile platform integrating optimization tools with open and extensible capabilities, facilitated by simulation techniques like quadtree and octree topologies. This framework enhances decision-making processes from data acquisition to integration, offering deeper insights into complex problems.

In our research, we present a novel hierarchical hybrid approach designed to optimize solutions for the Euclidean Traveling Salesman Problem. A key innovation lies in our development of a hierarchical clustering-based representation, achieved through the fusion of classic clustering algorithms such as K-Means, Affinity Propagation (AP), and Density Peaks Clustering (DPC). This amalgamation, termed recursive hybrid clustering, leverages the strengths of each algorithm to create a robust framework adaptable to diverse problem domains.

We introduce Enhanced K-Means, K-Affinity Propagation, and K-Density Peaks Clustering to further tailor these methods to our specific requirements.

An enhanced K-means clustering algorithm with a post-processing step aimed at achieving balanced cluster sizes. Using SSE and a diameter-based criterion during redistribution, our algorithm consistently outperforms standard K-means, showcasing a reduction of 2.6-4% in the Davies-Bouldin Index and superior performance in achieving balanced cluster size distribution.

Balanced clusters are vital for applications such as market segmentation and fraud detection, ensuring fair representation of all subgroups and fostering more insightful analysis. This approach addresses the common challenge of imbalanced clusters, particularly crucial in domains involving predictive risk analysis like healthcare and sustainability.

Our use of 2D/3D hierarchical representation inherently determines the number of clusters (K). However, traditional algorithms like AP and DPC lack predefined cluster counts. To address this, we propose hybridizing K-means with AP and modifying DPC to allow for K determination.

The hierarchical representation we've developed offers a powerful tool for navigating and exploring the optimization landscape at multiple levels. By structuring the TSP-Compressed quadtree/octree as a spanning tree, we enable rapid identification and isolation of localized

neighborhoods, facilitating parallel optimization.

At any level within the tree structure, users can efficiently access information about sub-clusters and individual cities, along with their hierarchical relationships. This capability streamlines the partitioning of the problem into independent sub-clusters, which are ideal for simultaneous optimization.

The clusters closer to the leaves of the tree represent tightly-knit local neighborhoods, perfect for construction heuristics focused on exploiting local opportunities quickly. Meanwhile, the higher levels of the tree encompass broader spatial regions, allowing metaheuristics to explore global interactions across nearby clusters in parallel.

In essence, our hierarchical approach offers a flexible and efficient means of exploring both local and global optimization strategies within the TSP landscape.

The evaluation highlighted significant variations in clustering technique performance based on instance scale. Specifically, for smaller scales, K-DPC consistently outperformed other methods across evaluation metrics. However, as the scale increased, the effectiveness of K-DPC became more erratic, suggesting its performance is influenced by complex data characteristics. Conversely, Enhanced k-means faced challenges with large graphs. Interestingly, even at the same levels of quadtree/octree compression, distinct performances were observed among techniques, indicating that local data properties within a dataset play a crucial role in clustering outcomes, regardless of size.

Although the compressed quadtree/octree architecture efficiently captures clustering hierarchy and allows for parallel optimization of localized neighborhoods, it does have limitations, particularly concerning fault tolerance and tree depth variation across different scales, especially in medium and large instances. This architecture relies on clear and well-separated clustering of cities into cohesive groups, making it vulnerable to noise or outliers in the data, which can disrupt hierarchical partitioning and distort neighborhood relationships encoded in the tree structure. Moreover, the spanning tree is sensitive to changes in cluster structure during refinement iterations, often resulting in subtree rearrangement and the invalidation of prior optimizations.

To mitigate these issues, incremental update approaches could be implemented, selectively re-optimizing affected regions after localized changes, rather than rebuilding the entire tree from scratch. While the compressed quadtree/octree offers valuable insights, hybrid representations incorporating graph-based flexibility may enhance resilience to real-world variability in TSP problems during prolonged optimization runs.

In response to the limitations of the TSP-compressed quadtree/octree, we propose a dilation 2 one-by-one embedding of the structure into a crossed cubes topology. This strategic integration aims to optimize resource allocation during parallel optimization. Our embedding method achieves an optimal dilation of 2 and a load factor of 1.

Intra-cluster solutions were rapidly generated using efficient construction heuristics tailored to cluster sizes. These local solutions were then intelligently combined using a genetic networking heuristic, which explored connectivity across partitions. This two-phase approach capitalized on hierarchical abstraction and guided recombination to produce diverse and high-quality

initial populations.

Extensive experimentation on benchmark TSPLIB instances validated the effectiveness of our clustering-driven initialization. These solutions served as the starting point for a simulated annealing metaheuristic, ensuring a well-balanced foundation for further refinement. Our approach consistently surpassed traditional methods like the farthest insertion across graphs of varying sizes.

Comparative analysis against four and five state-of-the-art algorithms demonstrated the superior performance of our hybrid TSP optimizer across different instance scales. For instances with 200 or fewer cities, our algorithm secured the second position, closely behind the DPC-ACO-KOpt algorithm, with an impressive average error rate of 0.21%. As instance scales increased beyond 200 cities, our algorithm consistently outperformed competitors, achieving solutions closest to proven optima with error rates at least half as low. Specifically, for instances ranging from 300 to 500 cities, our algorithm exhibited an average error rate of 0.6%. Remarkably, its robust scalability was evident in solving instances containing over 1000 cities, delivering results within 3-4% of the best-known solutions.

The hierarchical mapping introduced in this research effectively manages computational workload distribution, addressing network congestion, a significant challenge faced by sequential TSP metaheuristics operating on Euclidean path structures. In the Euclidean TSP, cities are represented as points in a geometric space, and determining the optimal tour involves finding the shortest possible path connecting all cities, a task known to be NP-hard.

Metaheuristics tackle this problem by evaluating numerous candidate paths in a distributed, stochastic manner. However, executing these algorithms sequentially can lead to contention issues, particularly when dealing with long paths spanning disparate cities. As iterations accumulate, problem scales increase, and outdoor connections proliferate, the risk of excessive contention rises due to geographic dispersion of points, resulting in latency that impedes progress.

In contrast, our proposed hierarchical mapping embeds natural city clusters inherent in the Euclidean topology into a parallel structure. This approach organizes optimization to focus on geographically localized partial paths, minimizing routing conflicts (with a dilation of 2 in our model). By strategically avoiding congestion hotspots that can hinder sequential evaluations of globally dispersed candidate solutions, our method ensures smoother progress.

Through well-balanced partitioning and localized parallelism, we efficiently mitigate contention effects. This is particularly advantageous for large-scale industrial TSP formulations with tens of thousands of points spread across wide regions. By aligning algorithmic and architectural decomposition, our approach accelerates the discovery of optimal Euclidean paths.

7.1 Perspectives

This study makes the case for using machine learning, parallel computing, and understanding problem structures together to make metaheuristic optimization techniques better at solving TSP. It suggests avenues for future research such as dynamic clustering adjustments,

blending different local search methods, and developing versatile frameworks applicable to various complex problems. Although the research primarily targets the TSP, its methodology introduces an encouraging clustering-based strategy for breaking down challenging problems hierarchically. Subsequent research will continue to harness parallel computing and enhance embedding techniques within this framework.

Moreover, future research directions include investigating dynamic load balancing methods to address variations in runtime and differences in subproblem complexity. Additionally, there is a need to explore the integration of this methodology into heterogeneous High-Performance Computing (HPC) architectures, incorporating GPU and FPGA accelerators. In summary, this study paves the way for leveraging high-performance distributed computing to tackle previously insurmountable combinatorial optimization problems.

Bibliography

[1] Emad Abuelrub. Embedding interconnection networks in crossed cubes. In *Electronic Engineering and Computing Technology*, pages 141–151. Springer, 2010.

[2] Emadeddin Mohamed Abuelrub. *Interconnection networks embeddings and efficient parallel computations*. Louisiana State University and Agricultural & Mechanical College, 1993.

[3] Yusef Ahsini, Pablo Díaz-Masa, Belén Inglés, Ana Rubio, Alba Martínez, Aina Magraner, and J Alberto Conejero. The electric vehicle traveling salesman problem on digital elevation models for traffic-aware urban logistics. *Algorithms*, 16(9):402, 2023.

[4] Enrique Alba. *Parallel metaheuristics: a new class of algorithms*. John Wiley & Sons, 2005.

[5] Mir Mohammad Alipour, Seyed Naser Razavi, Mohammad Reza Feizi Derakhshi, and Mohammad Ali Balafar. A hybrid algorithm using a genetic algorithm and multiagent reinforcement learning heuristic to solve the traveling salesman problem. *Neural Computing and Applications*, 30:2935–2951, 2018.

[6] D. Applegate, R. Bixby, and V. Chvatal. Concorde tsp solver. http://www.math.uwaterloo.ca/tsp/concorde/index.html, 2006.

[7] Dmitri I Arkhipov, Di Wu, Tao Wu, and Amelia C Regan. A parallel genetic algorithm framework for transportation planning and logistics management. *Ieee Access*, 8:106506–106515, 2020.

[8] Steve Bankes. Exploratory modeling for policy analysis. *Operations research*, 41(3):435–449, 1993.

[9] L Barasch, S Lakshmivarahan, and S Dhall. Embedding arbitrary meshes and complete binary trees in generalized hypercubes. In *Proceedings of the 1st IEEE Symposium on Parallel and Distributed Processing*, pages 202–209, 1989.

[10] Jon Jouis Bentley. Fast algorithms for geometric traveling salesman problems. *ORSA Journal on computing*, 4(4):387–411, 1992.

[11] Dimitris Bertsimas and John N Tsitsiklis. *Introduction to linear optimization*, volume 6. Athena scientific Belmont, MA, 1997.

[12] Saïd Bettayeb, Bin Cong, Mike Girou, and Ivan Hal Sudborough. Embedding star

networks into hypercubes. *IEEE transactions on computers*, 45(2):186–194, 1996.

[13] Said Bettayeb, Zevi Miller, and I Hal Sudborough. Embedding grids into hypercubes. In *VLSI Algorithms and Architectures: 3rd Aegean Workshop on Computing, AWOC 88 Corfu, Greece, June 28–July 1, 1988 Proceedings 3*, pages 201–211. Springer, 1988.

[14] Paul-Antoine Bisgambiglia, Bastien Poggi, and Céline Nicolai. Models-based optimization methods for the specication of fuzzy inference systems in discrete event simulation. In *Proceedings of the 7th conference of the European Society for Fuzzy Logic and Technology*, pages 957–964. Atlantis Press, 2011.

[15] Robert E Bixby. A brief history of linear and mixed-integer programming computation. *Documenta Mathematica*, 2012:107–121, 2012.

[16] Jacek Błażewicz. Scheduling under resource constraints-deterministic models. *(No Title)*, 1986.

[17] Eric Bonabeau. Agent-based modeling: Methods and techniques for simulating human systems. *Proceedings of the national academy of sciences*, 99(suppl_3):7280–7287, 2002.

[18] Juan Bordón-Ruiz, Eva Besada-Portas, and José A López-Orozco. Cloud devs-based computation of uavs trajectories for search and rescue missions. *Journal of Simulation*, 16(6):572–588, 2022.

[19] Stephen P Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[20] Gerçek Budak and Xin Chen. A hybrid mathematical model for flying sidekick travelling salesman problem with time windows. 4(4):96, 2023.

[21] Hans-Joachim Bungartz, Stefan Zimmer, H Buchholz, and D Pfluger. Modeling and simulation. *Springer Undergraduate Texts in Mathematics and Technology. Springer Berlin Heidelberg, Berlin, Heidelberg. doi*, 10:978–3, 2014.

[22] Tadeusz Caliński and Jerzy Harabasz. A dendrite method for cluster analysis. *Communications in Statistics-theory and Methods*, 3(1):1–27, 1974.

[23] Laurent Capocchi and Jean-François Santucci. Discrete optimization via simulation of catchment basin management within the devsimpy framework. In *2013 Winter Simulations Conference (WSC)*, pages 205–216. IEEE, 2013.

[24] Román Cárdenas, Patricia Arroba, Roberto Blanco, Pedro Malagón, José L Risco-Martín, and José M Moya. Mercury: A modeling, simulation, and optimization framework for data stream-oriented iot applications. *Simulation Modelling Practice and Theory*, 101:102037, 2020.

[25] Christos G Cassandras and Stéphane Lafortune. *Introduction to discrete event systems*. Springer, 2008.

[26] Lidia Ceriani and Paolo Verme. The origins of the gini index: extracts from variabilità e mutabilità (1912) by corrado gini. *The Journal of Economic Inequality*, 10:421–443, 2012.

[27] Vladimír Černỳ. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of optimization theory and applications*, 45:41–51, 1985.

[28] Chien-Ping Chang, Ting-Yi Sung, and Lih-Hsing Hsu. Edge congestion and topological properties of crossed cubes. *IEEE Transactions on Parallel and Distributed Systems*, 11(1):64–80, 2000.

[29] Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. 1976.

[30] James P Cohoon, Shailesh U Hegde, Worthy N Martin, and Dana S Richards. Distributed genetic algorithms for the floorplan design problem. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 10(4):483–492, 1991.

[31] Stephen A Cook. The complexity of theorem-proving procedures. In *Logic, Automata, and Computational Complexity: The Works of Stephen A. Cook*, pages 143–152. 2023.

[32] David L Davies and Donald W Bouldin. A cluster separation measure. *IEEE transactions on pattern analysis and machine intelligence*, (2):224–227, 1979.

[33] Marco Dorigo. Optimization, learning and natural algorithms. *Ph. D. Thesis, Politecnico di Milano*, 1992.

[34] Stuart E Dreyfus. An appraisal of some shortest-path algorithms. *Operations research*, 17(3):395–412, 1969.

[35] Iddo Drori, Anant Kharkar, William R. Sickinger, Brandon Kates, Qiang Ma, Suwen Ge, Eden Dolev, Brenda Dietrich, David P. Williamson, and Madeleine Udell. Learning to solve combinatorial optimization problems on real-world graphs in linear time, 2020.

[36] Joseph C Dunn. A fuzzy relative of the isodata process and its use in detecting compact well-separated clusters. 1973.

[37] K. Efe. The crossed cube architecture for parallel computation. *IEEE Trans. Parallel Distrib. Syst.*, 3(5):513–524, September 1992.

[38] K Efe, P Blackwell, T Shiau, and W Slough. A reduced diameter interconnection network. In *Proceedings 2nd Symposium on the Frontiers of Massively Parallel Computation*, pages 471–472. IEEE Computer Society, 1988.

[39] Jose B Escario, Juan F Jimenez, and Jose M Giron-Sierra. Ant colony extended: experiments on the travelling salesman problem. *Expert Systems with Applications*, 42(1):390–410, 2015.

[40] Abdol-Hossein Esfahanian, Lionel M Ni, and Bruce E Sagan. On enhancing hypercube multiprocessors. Technical report, Argonne National Lab., IL (USA), 1988.

[41] Absalom El-Shamir Ezugwu and Aderemi Oluyinka Adewumi. Discrete symbiotic organisms search algorithm for travelling salesman problem. *Expert Systems with Applications*, 87:70–78, 2017.

[42] Jianxi Fan, Xiaola Lin, and Xiaohua Jia. Node-pancyclicity and edge-pancyclicity of crossed cubes. *Information Processing Letters*, 93(3):133–138, 2005.

[43] Muyao Fan and Jingpeng Li. Surrogate-assisted genetic algorithms for the travelling salesman problem and vehicle routing problem, 2020.

[44] Weibei Fan, Xuanli Liu, and Mengjie Lv. Hamiltonian cycle embedding with fault-tolerant edges and adaptive diagnosis in half hypercube. 2023.

[45] Weibei Fan, Fu Xiao, Hui Cai, Xiaobai Chen, and Shui Yu. Disjoint paths construction and fault-tolerant routing in bcube of data center networks. *IEEE Transactions on Computers*, 2023.

[46] Smain Femmam and Faouzi M Zerarka. One-by-one embedding of the twisted hypercube into pancake graph. In *Building Wireless Sensor Networks*, pages 145–169. Elsevier, 2017.

[47] Merrill M Flood. The traveling-salesman problem. *Operations research*, 4(1):61–75, 1956.

[48] Christodoulos A Floudas. *Deterministic global optimization: theory, methods and applications*, volume 37. Springer Science & Business Media, 2013.

[49] Michael J Flynn. Some computer organizations and their effectiveness. *IEEE transactions on computers*, 100(9):948–960, 1972.

[50] Filippo Focacci, Andrea Lodi, and Michela Milano. A hybrid exact algorithm for the tsptw. *INFORMS journal on Computing*, 14(4):403–417, 2002.

[51] Simon French. Sequencing and scheduling. *An Introduction to the Mathematics of the Job-shop*, 1982.

[52] Brendan J Frey and Delbert Dueck. Clustering by passing messages between data points. *science*, 315(5814):972–976, 2007.

[53] Jinyu Fu, Guanghui Sun, Jianxing Liu, Weiran Yao, and Ligang Wu. On hierarchical multi-uav dubins traveling salesman problem paths in a complex obstacle environment. *IEEE Transactions on Cybernetics*, 2023.

[54] Michael C Fu, Sigrún Andradóttir, John S Carson, Fred Glover, Charles R Harrell, Yu-Chi Ho, James P Kelly, and Stephen M Robinson. Integrating optimization and simulation: research and practice. In *Winter Simulation Conference*, volume 1, pages 610–616, 2000.

[55] Michael C Fu, Fred W Glover, and Jay April. Simulation optimization: a review, new developments, and applications. In *Proceedings of the Winter Simulation Conference, 2005.*, pages 13–pp. IEEE, 2005.

[56] Michael R Garey. computers and intractqbility. *A Guide to the Theory of NP-Completeness*, 1979.

[57] Xiutang Geng, Zhihua Chen, Wei Yang, Deqian Shi, and Kai Zhao. Solving the traveling

salesman problem based on an adaptive simulated annealing algorithm with greedy search. *Applied Soft Computing*, 11(4):3680–3689, 2011.

[58] Carlos Gershenson and Francis Heylighen. How can we think the complex. *Managing organizational complexity: philosophy, theory and application*, 3:47–62, 2005.

[59] Fred Glover. Future paths for integer programming and links to artificial intelligence. *Computers & operations research*, 13(5):533–549, 1986.

[60] Fred Glover, Manuel Laguna, Fred Glover, and Manuel Laguna. Tabu search principles. *Tabu Search*, pages 125–151, 1997.

[61] Mahmoud Golabi, Mokhtar Essaid, Muhammad Sulaman, and Lhassane Idoumghar. Extreme learning machine-based genetic algorithm for the facility location problem with distributed demands on network edges, 2023.

[62] Ruixue Gu, Mark Poon, Zhihao Luo, Yang Liu, and Zhong Liu. A hierarchical solution evaluation method and a hybrid algorithm for the vehicle routing problem with drones and multiple visits. *Transportation Research Part C: Emerging Technologies*, 141:103733, 2022.

[63] Rachid Guerraoui and Michal Kapalka. On the correctness of transactional memory. In *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming*, pages 175–184, 2008.

[64] Şaban Gülcü, Mostafa Mahi, Ömer Kaan Baykan, and Halife Kodaz. A parallel cooperative hybrid method based on ant colony optimization and 3-opt algorithm for solving traveling salesman problem. *Soft Computing*, 22:1669–1685, 2018.

[65] Gregory Gutin and Abraham P Punnen. *The traveling salesman problem and its variations*, volume 12. Springer Science & Business Media, 2006.

[66] Maria Halkidi, Yannis Batistakis, and Michalis Vazirgiannis. On clustering validation techniques. *Journal of intelligent information systems*, 17:107–145, 2001.

[67] J. A. Hartigan and M. A. Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979.

[68] Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer, 2009.

[69] Bernhard Heinzl. *Methods for hybrid modeling and simulation-based optimization in energy-aware production planning*. PhD thesis, Wien, 2020.

[70] Michael Held and Richard M Karp. A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied mathematics*, 10(1):196–210, 1962.

[71] Keld Helsgaun. An effective implementation of the lin–kernighan traveling salesman heuristic. *European journal of operational research*, 126(1):106–130, 2000.

[72] Frederick S Hillier. *Introduction to operations research*. McGrawHill, 2001.

[73] John H Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[74] John H Holland. Complex adaptive systems. *Daedalus*, 121(1):17–30, 1992.

[75] Jia-Wei Hong, Kurt Mehlhorn, and Arnold L Rosenberg. Cost trade-offs in graph embeddings, with applications. *Journal of the ACM (JACM)*, 30(4):709–728, 1983.

[76] Reiner Horst and Panos M Pardalos. *Handbook of global optimization*, volume 2. Springer Science & Business Media, 2013.

[77] Lan Huang, Gui-chao Wang, Tian Bai, and Zhe Wang. An improved fruit fly optimization algorithm for solving traveling salesman problem. *Frontiers of Information Technology & Electronic Engineering*, 18(10):1525–1533, 2017.

[78] Nicolas Isoart. *Le probleme du voyageur de commerce en programmation par contraintes*. PhD thesis, Université Côte d'Azur, 2021.

[79] Jingqing Jiang, Jingying Gao, Gaoyang Li, Chunguo Wu, and Zhili Pei. Hierarchical solving method for large scale tsp problems. In *International symposium on neural networks*, pages 252–261. Springer, 2014.

[80] Yan Jiang, Thomas Weise, Jörg Lässig, Raymond Chiong, and Rukshan Athauda. Comparing a hybrid branch and bound algorithm with evolutionary computation methods, local search and their hybrids on the tsp. In *2014 IEEE Symposium on Computational Intelligence in Production and Logistics Systems (CIPLS)*, pages 148–155. IEEE, 2014.

[81] Jorge Kanda, Andre De Carvalho, Eduardo Hruschka, Carlos Soares, and Pavel Brazdil. Meta-learning to select the best meta-heuristic for the traveling salesman problem: A comparison of meta-features. *Neurocomputing*, 205:393–406, 2016.

[82] Shih-Shun Kao, Ralf Klasing, Ling-Ju Hung, Chia-Wei Lee, and Sun-Yuan Hsieh. A parallel algorithm for constructing multiple independent spanning trees in bubble-sort networks. *Journal of Parallel and Distributed Computing*, 181:104731, 2023.

[83] Maryam Karimi-Mamaghan, Mehrdad Mohammadi, Patrick Meyer, Amir Mohammad Karimi-Mamaghan, and El-Ghazali Talbi. Machine learning at the service of meta-heuristics for solving combinatorial optimization problems: A state-of-the-art. *European Journal of Operational Research*, 296(2):393–422, 2022.

[84] James Kennedy and Russell Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95-international conference on neural networks*, volume 4, pages 1942–1948. IEEE, 1995.

[85] Indadul Khan, Manas Kumar Maiti, and Manoranjan Maiti. Coordinating particle swarm optimization, ant colony optimization and k-opt algorithm for traveling salesman problem. In *Mathematics and Computing: Third International Conference, ICMC 2017, Haldia, India, January 17-21, 2017, Proceedings 3*, pages 103–119. Springer, 2017.

[86] Iftakhar Ali Khandokar, Swakkhar Shatabda, et al. New boosting approaches for improving cluster-based undersampling in problems with imbalanced data. *Decision Analytics Journal*, page 100316, 2023.

[87] Muslim Mohsin Khudhair, Furkan Rabee, and Adil Al-Rammahi. A new fractal topologies based on hypercube interconnection network. *Al-Salam Journal for Engineering and Technology*, 2(2):128–139, 2023.

[88] Scott Kirkpatrick, C Daniel Gelatt Jr, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[89] Slawomir Koziel and Xin-She Yang. *Computational optimization, methods and algorithms*, volume 356. Springer, 2011.

[90] Priyalal Kulasinghe and Said Bettayeb. Embedding binary trees into crossed cubes. *IEEE Transactions on Computers*, 44(7):923–929, 1995.

[91] Priyalal Kulasinghe and Said Bettayeb. Multiply-twisted hypercube with five or more dimensions is not vertex-transitive. *Information Processing Letters*, 53(1):33–36, 1995.

[92] Phone Thiha Kyaw, Aung Paing, Theint Theint Thu, Rajesh Elara Mohan, Anh Vu Le, and Prabakaran Veerajagadheswar. Coverage path planning for decomposition reconfigurable grid-maps using deep reinforcement learning based travelling salesman problem. *IEEE Access*, 8:225945–225956, 2020.

[93] Hojun Lee, Bernard P Zeigler, and Doohwan Kim. A devs-based framework for simulation optimization: Case study of link-11 gateway parameter tuning. In *MILCOM 2008-2008 IEEE Military Communications Conference*, pages 1–7. IEEE, 2008.

[94] F Thomson Leighton. *Introduction to parallel algorithms and architectures: Arrays · trees · hypercubes*. Elsevier, 2014.

[95] Harry R Lewis. Michael r. πgarey and david s. johnson. computers and intractability. a guide to the theory of np-completeness. wh freeman and company, san francisco1979, x+ 338 pp. *The Journal of Symbolic Logic*, 48(2):498–500, 1983.

[96] Erchong Liao and Changan Liu. A hierarchical algorithm based on density peaks clustering and ant colony optimization for traveling salesman problem. *Ieee Access*, 6:38921–38933, 2018.

[97] Shen Lin and Brian W Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, 21(2):498–516, 1973.

[98] John DC Little, Katta G Murty, Dura W Sweeney, and Caroline Karel. An algorithm for the traveling salesman problem. *Operations research*, 11(6):972–989, 1963.

[99] Hongxi Liu, Mingzu Zhang, and Weihua Yang. On modified l-embedded edge-connectivity of enhanced hypercubes. *The Journal of Supercomputing*, pages 1–13, 2023.

[100] Yanchi Liu, Zhongmou Li, Hui Xiong, Xuedong Gao, and Junjie Wu. Understanding of internal clustering validation measures, 2010.

[101] Yongliang Lu, Una Benlic, and Qinghua Wu. A hybrid dynamic programming and memetic algorithm to the traveling salesman problem with hotel selection. *Computers & Operations Research*, 90:193–207, 2018.

[102] Mostafa Mahi, Ömer Kaan Baykan, and Halife Kodaz. A new hybrid method based on particle swarm optimization, ant colony optimization and 3-opt algorithms for traveling salesman problem. *Applied Soft Computing*, 30:484–490, 2015.

[103] Silvano Martello and Paolo Toth. *Knapsack problems: algorithms and computer implementations*. John Wiley & Sons, Inc., 1990.

[104] Shoma Miki, Daisuke Yamamoto, and Hiroyuki Ebara. Applying deep learning and reinforcement learning to traveling salesman problem, 2018.

[105] John Howard Miller and Scott E Page. Complex adaptive systems: an introduction to computational models of social life. *(No Title)*, 2008.

[106] Melanie Mitchell. *Complexity: A guided tour*. Oxford university press, 2009.

[107] Saurabh Mittal, José L Risco-Martín, and Bernard P Zeigler. Devs/soa: A cross-platform framework for net-centric modeling and simulation in devs unified process. *Simulation*, 85(7):419–450, 2009.

[108] Nenad Mladenović and Pierre Hansen. Variable neighborhood search. *Computers & operations research*, 24(11):1097–1100, 1997.

[109] Moslim Mohsin, Adil AL Rammahi, and Furkan Rabee. An innovative topologies based on hypercube network interconnection. *European Journal of Information Technologies and Computer Science*, 3(4):7–13, 2023.

[110] Tadao Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

[111] Katarzyna Nałkecz-Charkiewicz and Robert M Nowak. Algorithm for dna sequence assembly by quantum annealing. *BMC bioinformatics*, 23(1):122, 2022.

[112] Mark EJ Newman and Michelle Girvan. Finding and evaluating community structure in networks. *Physical review E*, 69(2):026113, 2004.

[113] Jorge Nocedal and Stephen J Wright. *Numerical optimization*. Springer, 1999.

[114] Lewis Ntaimo, Xiaolin Hu, and Yi Sun. Devs-fire: Towards an integrated simulation environment for surface wildfire spread and containment. *Simulation*, 84(4):137–155, 2008.

[115] Ilhan Or. *TRAVELING SALESMAN TYPE COMBINATORIAL PROBLEMS AND THEIR RELATION TO THE LOGISTICS OF REGIONAL BLOOD BANKING*. Northwestern University, 1976.

[116] Manfred Padberg and Giovanni Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM review*, 33(1):60–100, 1991.

[117] Kung-jui Pai. A parallel algorithm for constructing two edge-disjoint hamiltonian

cycles in crossed cubes. In *International Conference on Algorithmic Applications in Management*, pages 448–455. Springer, 2020.

[118] Wen-Tsao Pan. A new fruit fly optimization algorithm: taking the financial distress model as an example. *Knowledge-Based Systems*, 26:69–74, 2012.

[119] Christos H Papadimitriou and Kenneth Steiglitz. *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1998.

[120] Alexey Pyrkov, Alex Aliper, Dmitry Bezrukov, Dmitriy Podolskiy, Feng Ren, and Alex Zhavoronkov. Complexity of life sciences in quantum and ai era. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, 14(1):e1701, 2024.

[121] Hongwei Qiao, Jixiang Meng, and Eminjan Sabir. The edge fault-tolerant spanning laceability of the enhanced hypercube networks. *The Journal of Supercomputing*, 79(6):6070–6086, 2023.

[122] G. Reinelt. Discrete and combinatorial optimization. Online, july 2023. Accessed: july 30, 2023.

[123] Eréndira Rendón, Itzel Abundez, Alejandra Arizmendi, and Elvia M Quiroz. Internal versus external cluster validation indexes. *International Journal of computers and communications*, 5(1):27–34, 2011.

[124] Celso Carneiro Ribeiro. *Applications of combinatorial optimization*. Baltzer, 1994.

[125] Alex Rodriguez and Alessandro Laio. Clustering by fast search and find of density peaks. *science*, 344(6191):1492–1496, 2014.

[126] Peter J Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987.

[127] Murat Sahin. Solving tsp by using combinatorial bees algorithm with nearest neighbor method. *Neural Computing and Applications*, 35(2):1863–1879, 2023.

[128] A. Eddine Selmi, M. Faouzi Zerarka, A. Cheriet, and S. Femmam. Construction compressed octree into m-dimensional crossed cubes via the one-by-one embedding. In *2023 7th International Conference on Computer, Software and Modeling (ICCSM)*, pages 47–51, Los Alamitos, CA, USA, jul 2023. IEEE Computer Society.

[129] A.T.E. Selmi, M.F. Zerarka, and A. Cheriet. Enhancing k-means clustering with post-redistribution. *Ingénierie des Systèmes d'Information*, 29(2):429–436, 2024.

[130] Aymen Takie Eddine Selmi, Mohamed Faouzi Zerarka, and Abdelhakim Cheriet. Construction compressed quadtree into m-dimensional crossed cubes via the embedding. In *2022 International Symposium on iNnovative Informatics of Biskra (ISNIB)*, pages 1–6, 2022.

[131] Aymen Takie Eddine Selmi, Mohamed Faouzi Zerarka, and Abdelhakim Cheriet. Dilation two embedding one-by-one particular sub-quadtree into m-dimentional crossed cubes, 2022.

[132] Aymen Takie Eddine Selmi, Mohamed Faouzi Zerarka, and Abdelhakim Cheriet. Dilation two embedding one-by-one particular sub-quadtree into m-dimentional crossed cubes. *arXiv preprint arXiv:2208.11172*, 2022.

[133] Aymen Takie Eddine Selmi, Mohamed Faouzi Zerarka, and Abdelhakim Cheriet. Innovative clustering-driven techniques for enhancing initial solutions in euclidean traveling salesman problems with machine learning integration. *Arabian Journal for Science and Engineering*, May 2024.

[134] Aymen Takie Eddine Selmi, Mohamed Faouzi Zerarka, Abdelhakim Cheriet, and Smain Femmam. Improving time efficiency of a hierarchical metaheuristic for the euclidean tsp using crossed cubes interconnection networks. In *Proceedings of the 2023 6th International Conference on E-Business, Information Management and Computer Science*, EBIMCS '23, page 50–56, New York, NY, USA, 2024. Association for Computing Machinery.

[135] Michael Sipser. Introduction to the theory of computation. *ACM Sigact News*, 27(1):27–29, 1996.

[136] Zdeněk Šulc and Hana Řezanková. Evaluation of recent similarity measures for categorical data. 2014.

[137] Chaopeng Tan and Kaidi Yang. Privacy-preserving adaptive traffic signal control in a connected vehicle environment. *Transportation research part C: emerging technologies*, 158:104453, 2024.

[138] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to data mining*. Pearson Education India, 2016.

[139] Robert Endre Tarjan. *Data structures and network algorithms*. SIAM, 1983.

[140] Joe Tekli. An overview of cluster-based image search result organization: background, techniques, and ongoing challenges. *Knowledge and Information Systems*, 64(3):589–642, 2022.

[141] PHAM Dinh Thanh, Huynh Thi Thanh Binh, and BUI Thu Lam. A survey on hybridizing genetic algorithm with dynamic programming for solving the traveling salesman problem. In *2013 International Conference on Soft Computing and Pattern Recognition (SoCPaR)*, pages 66–71. IEEE, 2013.

[142] Paolo Toth and Daniele Vigo. *The vehicle routing problem*. SIAM, 2002.

[143] Xi Wang, Jianxi Fan, Shukui Zhang, and Jia Yu. Node-to-set disjoint paths problem in cross-cubes. *The Journal of Supercomputing*, pages 1–25, 2021.

[144] Yifeng Wang, Baolei Cheng, Yu Qian, and Dajin Wang. Constructing completely independent spanning trees in a family of line-graph-based data center networks. *IEEE Transactions on Computers*, 71(5):1194–1203, 2021.

[145] Yong Wang. The hybrid genetic algorithm with two local optimization strategies for traveling salesman problem. *Computers & Industrial Engineering*, 70:124–133, 2014.

[146] Yong Wang and Zunpu Han. Ant colony optimization for traveling salesman problem based on parameters optimization. *Applied Soft Computing*, 107:107439, 2021.

[147] Thomas Weise. Global optimization algorithms-theory and application. *Self-Published Thomas Weise*, 361, 2009.

[148] L Darrell Whitley, Timothy Starkweather, and D'Ann Fuquay. Scheduling problems and traveling salesmen: The genetic edge recombination operator. In *ICGA*, volume 89, pages 133–40, 1989.

[149] Sewall Wright et al. The roles of mutation, inbreeding, crossbreeding, and selection in evolution. 1932.

[150] Changyou Wu, Xisong Fu, Junke Pei, and Zhigui Dong. A novel sparrow search algorithm for the traveling salesman problem. *IEEE Access*, 9:153456–153471, 2021.

[151] Weiyan Wu and Eminjan Sabir. Embedding spanning disjoint cycles in hypercube networks with prescribed edges in each cycle. *Axioms*, 12(9):861, 2023.

[152] Liqiong Xu. Symmetric property and the bijection between perfect matchings and sub-hypercubes of enhanced hypercubes. *Discrete Applied Mathematics*, 324:41–45, 2023.

[153] Xin-She Yang. *Nature-inspired metaheuristic algorithms*. Luniver press, 2010.

[154] Xuefei Yang, Manuel Ostermeier, and Alexander Hübner. Winning the race to customers with micro-fulfillment centers: an approach for network planning in quick commerce. *Central European Journal of Operations Research*, pages 1–40, 2024.

[155] Yuxing Yang. Embedded connectivity of ternary n-cubes. *Theoretical Computer Science*, 2021.

[156] Liu Youyao, Han Jungang, and Du Huimin. A hypercube-based scalable interconnection network for massively parallel computing. *Journal of Computers*, 2008.

[157] Bernard P Zeigler, Yoonkeon Moon, Vicente L Lopes, and Jinwoo Kim. Devs approximation of infiltration using genetic algorithm optimization of a fuzzy system. *Mathematical and Computer Modelling*, 23(11-12):215–228, 1996.

[158] Bernard P Zeigler, Herbert Praehofer, and Tag Gon Kim. *Theory of modeling and simulation*. Academic press, 2000.

[159] H. Zhang, Y. Wang, J. Fan, and R. Guo. Parallel construction of independent spanning trees on folded crossed cubes. In *2021 IEEE 32nd International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pages 207–210, Los Alamitos, CA, USA, jul 2021. IEEE Computer Society.

[160] Huanwen Zhang, Yan Wang, Jianxi Fan, Yuejuan Han, and Baolei Cheng. Constructing edge-disjoint spanning trees in several cube-based networks with applications to edge fault-tolerant communication. *The Journal of Supercomputing*, pages 1–28, 2023.

[161] Fang Zhao, Bingfeng Si, Zhenlin Wei, and Tianwei Lu. Time-dependent vehicle routing

problem of perishable product delivery considering the differences among paths on the congested road. *Operational Research*, 23(1):5, 2023.

[162] Si-Qing Zheng. Simd data communication algorithms for multiply twisted hypercubes. In *Parallel Processing Symposium, International*, pages 120–125. IEEE Computer Society, 1991.

[163] Qiang Zhu, Jun-Ming Xu, Xinmin Hou, and Min Xu. On reliability of the folded hypercubes. *Information Sciences*, 177(8):1782–1788, 2007.

Publications :

SELMI, Aymen Takie Eddine, ZERARKA, Mohamed Faouzi, et CHERIET, Abdelhakim. Enhancing K-Means Clustering with Post-Redistribution. Ingénierie des Systèmes d'Information (2024)

Aymen Takie Eddine Selmi, Mohamed Faouzi Zerarka, and Abdelhakim Cheriet. Innovative clustering-driven techniques for enhancing initial solutions in euclidean traveling salesman problems with machine learning integration. Arabian Journal for Science and Engineering, May 2024

Communications:

SELMI, Aymen Takie Eddine, ZERARKA, Mohamed Faouzi, et CHERIET, Abdelhakim. Construction Compressed Quadtree into M-Dimensional Crossed Cubes via the Embedding. In : *2022 International Symposium on iNnovative Informatics of Biskra (ISNIB)*. IEEE, 2022. p. 1-6.

SELMI, Aymen Takie Eddine, ZERARKA, Mohamed Faouzi, CHERIET, Abdelhakim, *et al.* Construction Compressed Octree Into M-Dimensional Crossed Cubes Via The One-By-One Embedding. In : *2023 7th International Conference on Computer, Software and Modeling (ICCSM)*. IEEE, 2023. p. 47-51.

SELMI, Aymen Takie Eddine, ZERARKA, Mohamed Faouzi, CHERIET, Abdelhakim, *et al.* Improving Time Efficiency of a Hierarchical Metaheuristic for the Euclidean TSP using Crossed Cubes Interconnection Networks. In: 2023 6th International Conference on E-Business, Information Management and Computer Science . ACM, 2023. P. 1-7.