Name Surname Salih Yalçın

A Master's Thesis

AGU Year 2022/2023

ACCELERATING COMPUTER ALGORITHMS BY USING GPU

A THESIS
SUBMITTED TO THE DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
AND THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE
OF ABDULLAH GUL UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
M.SC.

By
Salih Yalçın
June 2023

ACCELERATING COMPUTER ALGORITHMS BY USING GPU

A THESIS
SUBMITTED TO THE DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
AND THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE OF ABDULLAH GUL UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
M.SC.

By
Salih Yalçın
June 2023

SCIENTIFIC ETHICS COMPLIANCE

I hereby declare that all information in this document has been obtained in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all materials and results that are not original to this work.

Name-Surname: Salih Yalçın
Signature :

REGULATORY COMPLIANCE

M.Sc. thesis titled “Accelerating Computer Algorithms by Using GPU” has been prepared in accordance with the Thesis Writing Guidelines of the Abdullah Gül University, Graduate School of Engineering & Science.

Prepared By	Advisor
Salih Yalçın	Asst. Prof. Gülay Yalçın Alkan
Signature	Signature

Head of the Electrical and Computer Engineering Program

Assoc. Prof. Zafer Aydın

Signature

ACCEPTANCE AND APPROVAL

M.Sc. thesis titled “Accelerating Computer Algorithms by Using GPU” and prepared by Salih Yalçın has been accepted by the jury in the Electrical and Computer Engineering Graduate Program at Abdullah Gül University, Graduate School of Engineering & Science.

09/06/2023

(Thesis Defense Exam Date)

JURY:

Advisor : Asst. Prof. Gülay Yalçın Alkan

Member : Assoc. Prof. Ahmet Turan Özdemir

Member : Asst. Prof. Abdulkadir Köse

APPROVAL:

The acceptance of this M.Sc thesis has been approved by the decision of the Abdullah Gül University, Graduate School of Engineering & Science, Executive Board dated 22/01/2021 and numbered 02.

…….. /…….. / ……..

(Date)

Graduate School Dean
Prof. Dr. İrfan ALAN

ABSTRACT

ACCELERATING COMPUTER ALGORITHMS BY USING GPU

Salih Yalçın
MSc. in Electrical and Computer Engineering
Advisor: Asst. Prof. Gülay Yalçın Alkan

June 2023

Travelling Salesman Problem (TSP) is one of the significant problems in computer science which tries to find the shortest path for a salesman who needs to visit a set of cities and it involves in many computing problems such as networks, genome analysis, logistic etc. Using parallel executing paradigms, especially GPUs, is appealing in order to reduce the problem-solving time of TSP. One of the main issues in GPUs is to have limited GPU memory which would not be enough for the entire data. Therefore, transferring data from host device would reduce the performance in execution time.

In this study, we present a methodology for compressing data to represent cities in the TSP so that we include more cities in GPU memory. We implement our methodology in Iterated Local Search (ILS) algorithm with 2-opt and show that our implementation presents 29% performance improvement compared to the state-of-the-art GPU implementation.

Keywords: Travelling Salesman Problem, GPU Programming, 2-opt, CUDA

ÖZET

BİLGİSAYAR ALGORİTMALARININ GPU YARDIMI İLE HIZLANDIRILMASI

Salih Yalçın
Elektrik ve Bilgisayar Mühendisliği Anabilim Dalı Yüksek Lisans
Tez Yöneticisi: Dr. Öğr. Üyesi Gülay Yalçın Alkan

Haziran 2023

Gezgin Satıcı Problemi (TSP), bir dizi şehri ziyaret etmesi gereken bir satıcı için en kısa yolu bulmaya çalışan bilgisayar bilimlerinin önemli problemlerinden biridir ve ağlar, genom analizi, lojistik vb. gibi birçok hesaplama probleminde yer almaktadır. TSP'nin problem çözme süresini azaltmak için paralel yürütme paradigmalarını, özellikle GPU'ları kullanmak caziptir. GPU'lardaki ana sorunlardan biri, tüm veriler için yeterli olmayacak sınırlı GPU belleğine sahip olmaktır. Bu nedenle, verilerin ana cihazdan aktarılması, yürütme süresindeki performansı düşürecektir.

Bu çalışmada, TSP'deki şehirleri temsil etmek için verileri sıkıştırmak için bir metodoloji sunuyoruz, böylece GPU belleğine daha fazla şehir dahil ediyoruz. Metodolojimizi 2-opt ile Yinelemeli Yerel Arama (ILS) algoritmasında uyguluyoruz ve uygulamamızın son teknoloji GPU uygulamasına kıyasla %29 performans artışı sunduğunu gösteriyoruz.

Anahtar kelimeler: Gezgin Satıcı Problemi, GPU Programlama, 2-opt, CUDA

Acknowledgements

I would like to express my deepest gratitude to my advisor, Asst. Prof. Gülay Yalçın Alkan, for her unwavering support and invaluable guidance throughout my thesis journey. Her dedication, patience, and expertise have been instrumental in shaping the direction and quality of my work. I am also immensely grateful to my girlfriend, Fadime Demir, for her unconditional love, encouragement, and unwavering belief in my abilities. Her presence and unwavering support have been my source of strength and motivation during the challenging times. Their contributions have been invaluable to the success of this thesis, and I am honored to have had the privilege to work with them.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

TSP	Travelling Salesman Problem
CPU	Central Processing Unit
GPU	Graphics Processing Unit
FPGA	Field Programmable Gate Arrays
CUDA	Compute Unified Device Architecture
GCD	Greatest Common Divisor
ILS	Iterated Local Search
SMT	Simultaneous Multi Threading
CMP	Chip Multi Processing
HDL	Hardware Description Language
ACO	Ant Colony Optimization
ABC	Artificial Bee Colony
CS	Cuckoo Search
DE	Differential Evolution
FA	Firefly Algorithm
CLB	Configurable Logic Blocks
GA	Genetic Algorithm
PSO	Particle Swarm Optimization
SA	Simulated Annealing
WCA	Water Cycle Algorithm
DRL	Deep Reinforcement Learning

To My Family…

Chapter 1

Introduction

Travelling Salesman Problem (TSP) is one of the significant problems in computer science which tries to find the shortest path for a salesman who needs to start from an initial city, visit a set of cities by stopping by at each of them exactly once and return back to the initial city at the end [1]. This problem is involved in many real-life issues like drilling of printed circuit boards, computer wiring, design of global navigation satellite system surveying networks, genome analysis, logistics and many more [2]. Thus, finding an optimal solution for a TSP in a feasible amount of time is an essential and also challenging concern.

Travelling Salesman Problem can be represented as finding the Hamiltonian cycle in an N-vertices weighted graph (for N cities) in which cities are represented by vertices and edges are showing the distances between cities. In addition, if the distances between two cities in both directions are identical, the number of solutions is halved and it is called as a symmetric TSP which can be expressed by an undirected graph.

It is trivial that finding the exact solution for TSP with a pure brute force algorithm, in which the length of each possible tour is calculated, requires factorial time. Such that, for a selected initial city among n cities, there will be n-1 options for the second city and n-2 options for the third city and so on. Therefore, TSP is classified as an NP-hard problem in which finding the exact solution is not possible in polynomial time [3]. Due to the difficulty of finding the exact solution, other problem solving approaches utilizing heuristics are implemented to find an acceptable solution in an acceptable time such as genetic algorithm, ant algorithm, tabu search, neural network, etc. which are discussed in Section 2.1.

Although several algorithmic methods are implemented to find an optimum solution for TSP, it is still time-consuming for an average microprocessor especially when the problem size is relatively large like in many real-life applications. In order to reduce the

execution time of those algorithms, it is possible to use parallel execution paradigms provided in the hardware level such as multiple threads in CPUs, Field Programmable Gate Arrays (FPGAs) or Graphical Processing Units (GPUs). Those hardware improvements can provide great amount of performance efficiency in the execution time which is discussed in Section 2.2 and Section 2.3.

Graphical Processing Units are used to increase the performance of applications by using Data Level Parallelism in which different small execution units operate on different portion of the data with the same instruction. GPUs are first developed for graphical operations specifically in the game industry in which performance is important, to make graphical operations faster. For instance, computation of each pixel in an image is done in a different execution unit, a.k.a a processing element in a streaming multi-processor. However, due to their data-parallel feature, GPUs have also been used in many different areas besides the game industry such as machine learning applications, bioinformatics, blockchain applications. Applications can be programmed by using CUDA (Compute Unified Device Architecture) [4] to be able to distribute the data parallel sections to the processing elements. It is shown that the speedup provided by GPUs can be tremendously high since many processing elements operates in parallel while they are more power efficient due to the simplicity of processing elements compared to CPUs. For instance, it is claimed that Hopper, NVIDIA's newest GPU architecture, can provide 7x Dynamic Programming Performance[1] with its new instruction set named DPX. Thus, Hopper DPX instructions will speed up optimization algorithms up to 40 times [5]. GPUs are used to reduce the execution time of TSP in several studies which is discussed in Section 2.4.

The memory wall is one of the major limitations of high performance computing, especially for highly parallel architectures due to the fact that many computing units are requesting data simultaneously. Because GPUs have high execution power and memory requirements, their shared memory is also limited, which prevents them from achieving a high speed up. In this study, our goal is to provide a methodology for compressing data to represent cities in the Traveling Salesman Problem in order to reduce the memory usage of GPUs and fit more cities in the shared memory area. To this end, we represent each

[1] Dynamic Programming breaks complex problems down to simpler subproblems that are solved recursively, therefore, it reduces complexity and time to polynomial scale.

city with smaller numbers (i.e. 16-bit numbers) instead of 32-bit integer numbers after applying three main steps. First, we shift cities to (0,0) center in the coordinate system to avoid big numbers and negative numbers. Second, we find Greatest Common Divisor (GCD) for both X and Y coordinates of all cities and divide coordinates of each city to those GCD values. Third, we split the entire area into grids and represent each city with respect to the base coordinate of the grid. We present the details of our design in Section 3. Our implementation presents 29% performance improvement compared to [6] the state-of-the-art GPU implementation of the Travelling Salesman Problem.

We use Iterated Local Search (ILS) algorithm [7] with 2-opt [8] for solving Travelling Salesman Problem in our implementation as in [6]. ILS is an algorithmic solution which presents a heuristic to find an acceptable solution for TSP in a shorter amount of time. After generating an initial solution, ILS makes a local search on close combinations and finds the local minimum value. Then, it perturbates the current local minimum value by hoping there is a better configuration with a shorter path in another local minimum value. After the perturbation, ILS searches again for the next local minimum value in the modified solution. In summary, ILS finds a sequence of locally optimal solutions after the initial by iterating on 1) perturbation 2) finding current local optimum and it takes the best as the solution for the given TSP. Using k-opt algorithm is proposed for the perturbation step. For instance, the 2-opt algorithm basically removes two edges from the tour, and reconnects the two new sub tours created. This is often referred to as a 2-opt move. There is only one way to reconnect the two sub-tours so that the tour remains valid. The 2-opt method returns local minimal in polynomial time [3]. We explain the details of ILS and 2-opt algorithm in Section 2.1.1.

The contribution of this study is as following:

• We present a comprehensive review on hardware methods (i.e. FPGAs, Multi-threads and GPUs) used to speed up the TSP solving algorithms.

• We present a data compression methodology for representing cities to reduce memory usage of GPUs

• We implement our methodology in ILS algorithm with 2-opt and show that our GPU implementation, on average, presents 29% performance improvement compared to the state-of-art GPU implementation.

Chapter 2

Review on Studies Accelerating TSP Solving

2.1 Algorithm-Based Methods

TSP (Traveling Salesman Problem) is a widely studied problem in the field of optimization. Over the years, many algorithm-based methods have been developed to solve this problem. Some of the popular algorithm-based methods are; Ant Colony [9 - 14], Artificial Bee Colony [15 – 18], Cuckoo Search [12, 19, 20], Differential Evolution Algorithm [21], Firefly Algorithm [22], Genetic Algorithm [23 - 29], Particle Swarm Optimization [30, 31], Simulated Annealing [32], Tabu Search [33, 34], Water Cycle Algorithm [35], and Deep Reinforcement Learning [36].

Each of these approaches has benefits and drawbacks. For instance, the Ant Colony algorithm is inspired by the behavior of ants and can find a good solution to the TSP problem in a relatively short amount of time.

The Genetic Algorithm, on the other hand, mimics the process of natural selection and can find an optimal solution to the problem. The Deep Reinforcement Learning approach is relatively new and has shown promising results in solving TSP with large-scale inputs.

Overall, the selection of an algorithm-based method for solving TSP depends on various factors such as the size of the problem, the required level of accuracy, the available computational resources, and the time constraints.

2.1.1 Ant Colony Optimization

Ant Colony Optimization (ACO) is a metaheuristic algorithm that has been widely used for solving the Traveling Salesman Problem (TSP). ACO is inspired by the behavior of real ant colonies, where ants communicate with each other by depositing pheromones to mark the shortest path to food sources. In ACO, a similar pheromone-based mechanism is used to construct candidate solutions to the TSP. The algorithm iteratively builds a set of solutions by simulating the behavior of ants that follow paths with higher pheromone levels, while also exploring new paths with a certain probability.

One of the main advantages of ACO is its ability to produce near-optimal solutions within a reasonable time frame, even for large-scale TSP problems. ACO is also able to handle asymmetric TSP instances, where distances between cities may be different in each direction, whereas other heuristic algorithms such as the 2-opt and 3-opt algorithms may not perform as well in such situations. Moreover, ACO can easily incorporate domain-specific knowledge such as tour length constraints and precedence relationships between cities, making it a flexible and customizable approach to solving the TSP.

However, ACO also has some drawbacks. The algorithm requires tuning of several parameters, such as the pheromone evaporation rate and the probability of choosing a new path, which can greatly affect the performance of the algorithm. Moreover, the algorithm is sensitive to the initial pheromone values and can easily get trapped in local optima. Additionally, ACO is a stochastic algorithm, which means that the quality of the solution can vary for each run.

2.1.2 Artificial Bee Colony

Artificial Bee Colony (ABC) is a nature-inspired algorithm that has gained considerable popularity in solving optimization problems, particularly the Traveling Salesman Problem (TSP). The algorithm emulates the foraging behavior of honeybees, and its effectiveness has been demonstrated in solving TSP instances of varying magnitudes. ABC is distinguished by its capacity to effectively balance search space exploration and exploitation, rendering it suitable for solving complex optimization problems.

One of the main benefits of using ABC for TSP is its ability to handle large problem instances with high accuracy and fast convergence. Moreover, ABC is a population-based algorithm, which allows it to explore multiple solutions simultaneously, improving the chances of finding the optimal solution. Additionally, ABC is a parallelizable algorithm, making it suitable for implementation on high-performance computing platforms such as GPUs.

However, like any optimization algorithm, ABC has some drawbacks that should be considered when using it for TSP. One of the main drawbacks is the dependence on the initialization of the algorithm's parameters, which can impact the convergence rate and quality of the solution. Moreover, the algorithm may get trapped in local optima, which can lead to suboptimal solutions. Finally, the implementation of ABC for TSP requires careful tuning of the algorithm's parameters, which can be time-consuming.

In summary, ABC is a promising method for solving TSP, offering benefits such as the ability to handle large problem instances, parallelization, and exploration of multiple solutions. However, it also has some drawbacks related to parameter initialization, local optima, and tuning. Overall, ABC can be a useful addition to the toolbox of methods for solving TSP, complementing other approaches such as the use of GPUs with CUDA.

2.1.3 Cuckoo Search

Cuckoo Search (CS) is a metaheuristic algorithm that is inspired by the reproductive behavior of cuckoo birds. It has been applied to various optimization problems, including the Traveling Salesman Problem (TSP). CS is a population-based algorithm that uses Levy flights to explore the search space and has been shown to be effective in finding near-optimal solutions for TSP instances of varying sizes. The benefits of CS for TSP include its simplicity, ability to handle multiple objectives, and ability to escape local optima.

However, the algorithm's main drawbacks include its slow convergence rate and sensitivity to its parameters. By considering Cuckoo Search alongside other algorithms such as CUDA, one can find the most effective optimization method for TSP.

2.1.4 Differential Evolution Algorithm

Differential Evolution (DE) is a population-based optimization algorithm that has been applied to various optimization problems, including the Traveling Salesman Problem (TSP). DE works by evolving a population of candidate solutions through the use of mutation, crossover, and selection operations. DE has shown promising results in solving TSP instances of different sizes and complexities, and its performance can be further improved by using advanced strategies such as self-adaptation and hybridization with other optimization algorithms.

One of the main advantages of DE is its simplicity, which allows for easy implementation and parameter tuning.

However, the algorithm may suffer from premature convergence and lack of diversity, which can be addressed by adjusting the mutation and crossover rates and using suitable selection mechanisms.

Overall, DE is a promising method for solving TSP and can complement other optimization techniques such as CUDA-based algorithms.

2.1.5 Firefly Algorithm

Firefly Algorithm (FA) is a nature-inspired optimization algorithm that has been successfully applied to solve complex problems such as the Traveling Salesman Problem (TSP). FA simulates the flashing behavior of fireflies and their attraction to brighter ones to find optimal solutions. The algorithm's effectiveness in solving TSP is due to its ability to handle large search spaces and the presence of multiple local optima. FA has shown promising results in minimizing the total distance traveled by the salesman on a given TSP instance.

However, FA also has some drawbacks, such as the tendency to converge to suboptimal solutions and sensitivity to parameter settings. Despite these limitations, FA remains a useful tool for solving TSP and other optimization problems.

2.1.6 Genetic Algorithm

Genetic Algorithm (GA) is a heuristic optimization algorithm that simulates the natural selection process in genetics. It has been widely used to solve the Traveling Salesman Problem (TSP). GA starts by encoding the solutions to the TSP problem as chromosomes in a population, where each chromosome represents a potential tour. The algorithm then uses various genetic operators, such as mutation, crossover, and selection, to evolve the population over successive generations. Through this process, GA aims to find the optimal tour with the shortest distance. GA is known for its versatility and ability to handle various types of optimization problems, including TSP.

However, the performance of GA is highly dependent on the choice of parameters and the initial population. Additionally, it may suffer from the issue of premature convergence, which can lead to suboptimal solutions.

2.1.7 Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a well-known metaheuristic optimization algorithm inspired by the social behavior of bird flocks and fish schools. PSO has been applied to solve various optimization problems including the Traveling Salesman Problem (TSP). In PSO, candidate solutions are represented as particles that move in the search space based on their personal and global best positions. PSO has demonstrated its effectiveness and efficiency in solving TSP instances of different sizes. The main advantage of PSO is its ability to quickly converge to a good solution, particularly in high-dimensional search spaces.

However, PSO can suffer from premature convergence and can be sensitive to its parameters, which can affect its performance.

2.1.8 Simulated Annealing

Simulated Annealing (SA) is a popular metaheuristic algorithm that has been applied to solve various optimization problems, including the Traveling Salesman Problem (TSP). SA mimics the physical annealing process of metals by iteratively adjusting the temperature and searching for lower energy states. SA has the advantage of

being able to escape local optima and search the global optima, making it a useful method for TSP instances of different sizes.

However, SA requires extensive computational resources, as it involves a large number of iterations and function evaluations. Additionally, the performance of SA is heavily dependent on the initial temperature and cooling rate, which require careful tuning to achieve optimal results. Overall, SA is a powerful algorithm for TSP, but it requires careful parameter tuning and significant computational resources.

2.1.9 Tabu Search

Tabu Search is a well-known metaheuristic algorithm that has been used to solve combinatorial optimization problems, including the popular Traveling Salesman Problem (TSP). The algorithm works by iteratively exploring a neighborhood of candidate solutions, while preventing the search from revisiting recently explored solutions by keeping a tabu list. The tabu list is a crucial mechanism that helps Tabu Search avoid getting trapped in local optima and guides the search towards better solutions. By effectively balancing exploration and exploitation of the search space, Tabu Search has proven to be a powerful optimization algorithm for TSP and other challenging problems.

One of the benefits of Tabu Search is its ability to efficiently search through large search spaces and to find high-quality solutions. However, Tabu Search has some drawbacks, such as the sensitivity of its performance to the choice of tabu tenure and the difficulty in determining the stopping criterion. Overall, Tabu Search is a promising optimization algorithm that can complement CUDA-based solutions to the TSP, particularly when dealing with large-scale instances.

2.1.10 Water Cycle Algorithm

Water Cycle Algorithm (WCA) is a novel metaheuristic algorithm inspired by the natural process of water cycle. WCA has been applied to solve different optimization problems including the Traveling Salesman Problem (TSP). The WCA algorithm is based on the movement of water droplets from higher to lower potential energy, which simulates the flow of water in the water cycle process. The algorithm is characterized by its ability to balance exploration and exploitation, making it suitable for solving complex optimization problems like TSP. WCA can handle the constraints of TSP such as visiting

all cities only once and returning to the starting point. The benefits of WCA include its simplicity, flexibility, and fast convergence rate, while its drawbacks include its sensitivity to parameter tuning and premature convergence to sub-optimal solutions.

2.1.11 Deep Reinforcement Learning

Deep Reinforcement Learning (DRL) is a relatively new approach to solving optimization problems, including the Traveling Salesman Problem (TSP). DRL models use neural networks to learn from experience and make decisions that lead to the best solution. The advantage of DRL is that it can find optimal solutions without relying on heuristics or hand-designed features. Furthermore, DRL can handle complex and large-scale TSP instances that other methods may struggle with. However, DRL requires a large amount of computational resources and a considerable amount of training data to achieve high performance. Additionally, DRL is known to suffer from the problem of exploration-exploitation trade-off, where it may get stuck in sub-optimal solutions or converge to local optima.

2.2 Multi-threaded Methods

In Multi-threaded execution algorithms can be divided into threads using programming methods such as OpenMP [37] and PThread [38], and these threads are executed in parallel in the hardware. This mechanism is called as Thread Level Parallelism in which different threads may have different instruction sequences. During the execution, multiple threads can be scheduled to the same core as in Simultaneous Multi Threading (SMT) [39] or the threads can be executed in different cores as in Chip Multiprocessing (CMP) [40]. While SMT provides benefit of using the same cache area for different threads potentially increasing the cache hit performance, it is generally not scalable when thread number increases more than 8 threads. According to Amdahl's Law [41], in which it is presented that the minimum execution time can be as low as Sequential Execution Time/n when n threads are used, only up to linear speedup can be provided to the applications by using multi-threaded execution as long as the same algorithm is parallelized and executed with equal amount of resources other than the thread numbers. Multi-threaded execution is used to reduce the execution time of TSP in several studies as well PThread [42] and OpenMP [43-45].

2.3 FPGA Methods

Field Programmable Gate Arrays (FPGAs) are semiconductor devices that are based around a matrix of configurable logic blocks (CLBs) connected via programmable interconnects. FPGAs can be reprogrammed to desired application or functionality requirements after manufacturing. The FPGA configuration is generally specified using a hardware description language (HDL) FPGA is used to increase the performance of TSP in FPGA [46, 47].

2.4 GPU Methods

Tiling is a technique used in parallel computing to optimize memory usage and reduce data movement between shared and global memory in GPUs. The basic idea behind tiling is to partition the input data into smaller, more manageable chunks called tiles, and process each tile independently in parallel. This reduces the amount of data that needs to be stored in shared memory and allows multiple threads to access the same data simultaneously, thereby reducing data movement and increasing computation efficiency.

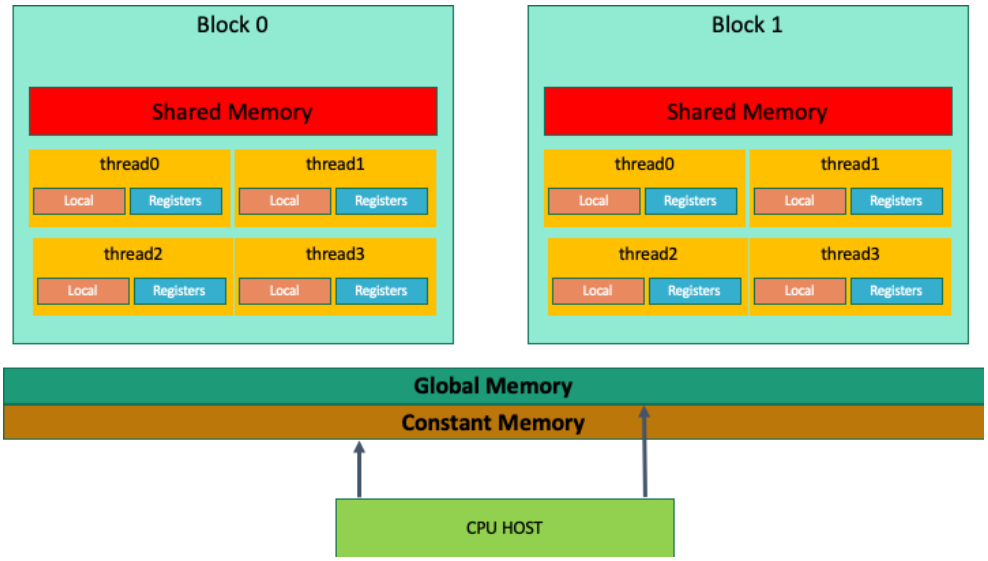


Figure 2.1 Memory Spaces in GPU

Chapter 3

Our Method: CompreCity

Graphical Processing Units (GPUs) may provide a substantial speedup due to high number of parallel execution units included in GPUs. one of the main limitation hindering to achieve highest speedup is the memory requirement of the parallel executions for many algorithm. In order reduce the memory access time in GPU, CUDA supports several low-capacity, high-performance memories to keep the data closer to the execution cores rather that accessing DRAM slowly.

In Figure 2.1, we present memory spaces in GPUs. Global memory is relatively slower memory in GPU (it is still faster than DRAM) which can be accessed by all running threads as well as the host CPU. Global memory is allocated and deallocated by the host by using cudaMalloc, cudaFree, cudaMemcpy and cudaMemset commands. Shared Memory is very fast (it can be in the speed of a register) compared to global memory and it can be accessed by all the threads in the same block.

In this study, our goal is; while solving Travelling Salesman Problem in GPU, to be able to fit as much data as possible in the GPU memory, possibly in shared memory, so that the DRAM access is reduced and the overall performance is improved. To this end, we apply three-step comparison to the input file to represent cities with less number of bits. In this section, we present these compression steps.

3.1 Using Greatest Common Divisor

In many data analysis and visualization tasks, it is common to encounter datasets with varying scales. When dealing with datasets that have coordinates, such as those in the Cartesian plane, it can be useful to scale the values to a common scale. One way to

achieve this is by using the greatest common divisor (GCD) of the x-coordinates and y coordinates.

The GCD can be calculated using a simple algorithm, and once obtained, it can be used to scale the x and y coordinates to a common factor. This ensures that the dataset is scaled proportionally, and that the relationships between the points in the dataset are preserved. By using the GCD for scaling, it becomes easier to analyze and visualize datasets with varying scales, and to compare datasets with one another.

In our analysis, we followed this approach and computed the GCD of the x-coordinates and y-coordinates separately for our dataset. We then divided each x-coordinate and y- coordinate by their respective GCD values. This scaling process ensured that our dataset was proportionally scaled, and the relationships between the points were preserved.

In Table 3.1, we demonstrate how GCD is applied for a map with five cities. In the example, the first table represents the raw data which has the (x,y) coordinates of cities. In our compression algorithm, we first find the GCD of all x coordinates and GCD of all y coordinates. In the given example both GCD_x and GCD_y is calculated as 25. Then, each x coordinate is divided to GCD_x and each y coordinate is divided to GCD_y. The values after applying gcd is presented in the second table. In the example, in the row data, the maximum value for x coordinates was 52000 which can be represented by 19 digits in binary while after applying GCD, the new maximum value becomes 20800 which can be represented by 15 digits.

In GPU, after applying GCD method to all coordinates, the host needs to pass GCD_x and GCD_y values to GPU together with city coordinates. Note that the overhead GCD values is negligable compared to the entire city coordinates.

3.2 Shifting City

Another useful technique for data scaling is shifting. Shifting involves finding the minimum x-coordinate and y-coordinate values in a dataset and subtracting them from all

the x-coordinates and y-coordinates in the dataset. This process translates the entire dataset so that the minimum x-coordinate and y-coordinate values are at the origin of the coordinate system. By shifting the dataset in this way, we can create a more standardized and consistent dataset that is easier to analyze and compare.

Instead only the values becomes smaller the entire map shifts in the coordinate system to the origin. Also note that the minimum values in the row data can be a negative value and shifting helps eliminating negative values in the coordinates and we can use unsigned integer to represent cities.

In our analysis, we applied the shifting technique by computing the minimum x-coordinate and y-coordinate values in our dataset and subtracting them from all the x-coordinates and y-coordinates. This approach ensured that our dataset was centered at the origin of the coordinate system, making it easier to interpret and analyze. By using both GCD scaling and shifting techniques, we were able to create a more standardized and comparable dataset for our analysis.

In Table 3.1, in the third table, we showed the data values after applying shift operation. In the example, the minimum value for x in the second table was 8155 while the minimum value for y was 20277 that subtract those values from the x coordinates and y coordinates of all the cities respectively. After the shifting operation, the minimum values for x and y coordinates becomes zero. In the example, the maximum x value becomes 12645 which can be represented by 14 bits in binary (it was 15 bits in the second table) while the maximum value for y becomes 29 which needs only 5 bits which is a substantial drop in the data size compared to 15 bits in the second table.

3.3 Splitting Surface to the Grids

In addition to scaling and shifting techniques, another important aspect of data preprocessing is the splitting of surfaces into grids. This technique involves dividing a surface into a set of smaller regions, or grids, to enable more granular analysis and modeling. In our analysis, we split our surface into a set of 16x16 grids, with each grid assigned a unique number ranging from 0 to 255, from left to right. To determine the

range for each grid, we used a simple formula that divided the difference between the maximum and minimum x-coordinate values by 16.

In Figure 3.1, we present an example for representing a city after splitting the surface into grids. In the example, we assume that the maximum x value is 4000 and the maximum y value is 16000 after the shift operation and we divide the surface into 4x4 grid for simplicity. The city A, which is located in cell 10, has the coordinates of 2100 and 10500 in the example. After the splitting operation, we now represent cities with three components such as ($X_{relative}$, $Y_{relative}$, cell_number). In the example, the base coordinates of cell 10 is (2000,8000) which can be calculated as;

$$(x_{max}/4 \text{ x (int) } (10/4), y_{max}/4 \text{ x } (10 \text{mod} 4) \quad (3.1)$$

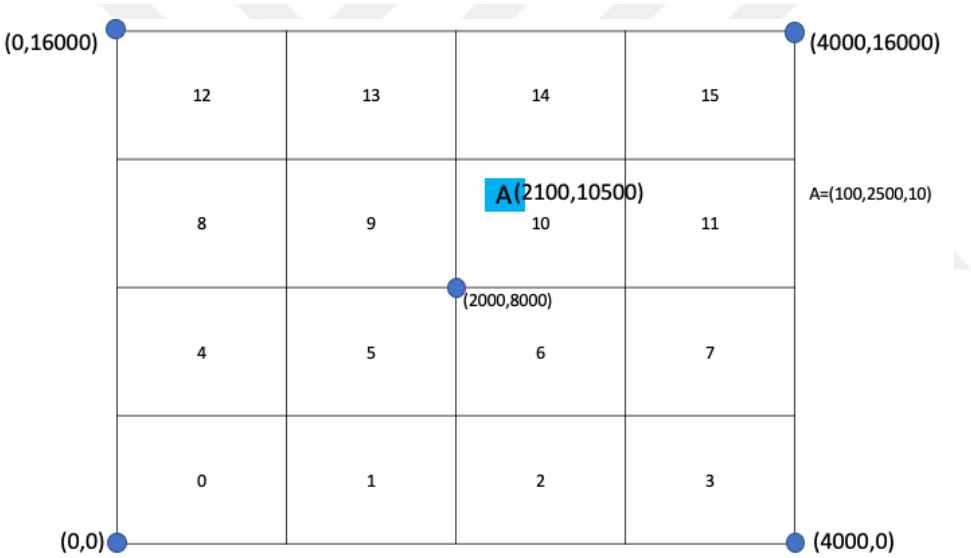
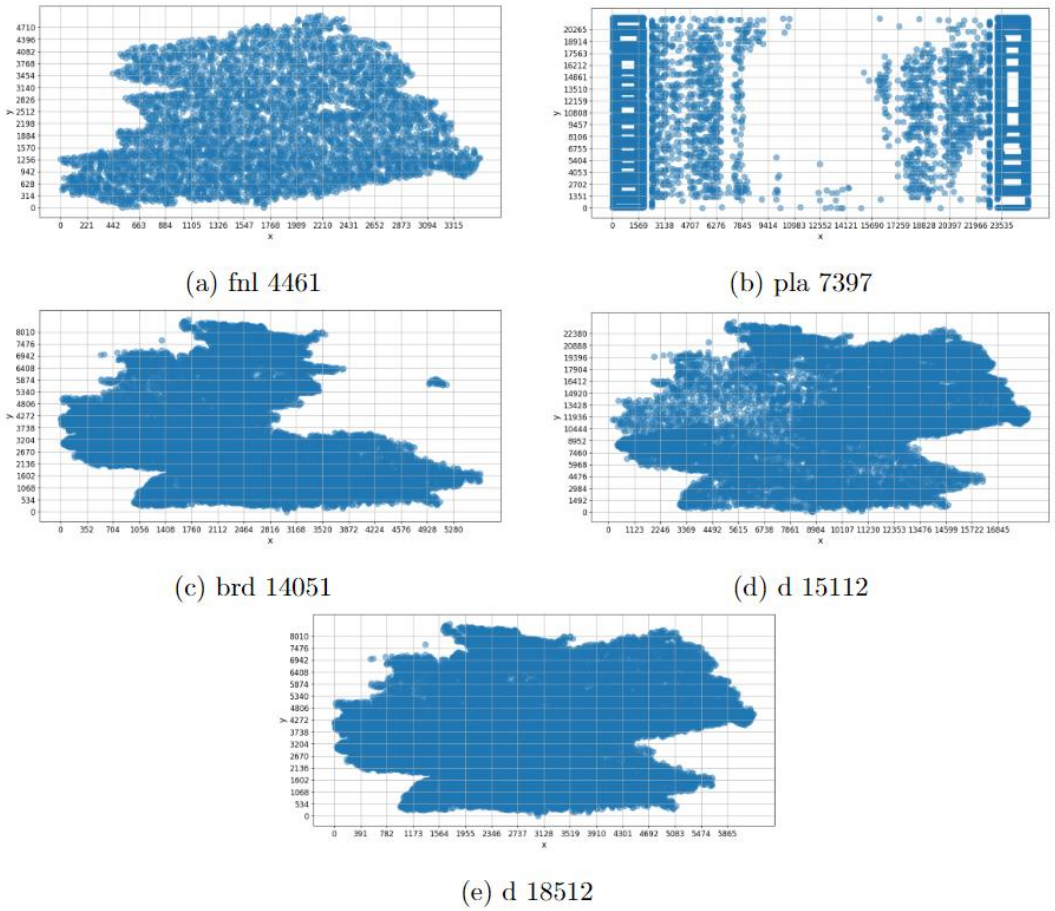


Figure 3.1 Example of Splitting to the Grids

This approach ensured that each grid covered an equal range of x-coordinate values. We applied a similar formula to determine the range for each grid in the y-coordinate values. By splitting our surface into grids, we were able to have more scalable and compressed data.

Overall, we summarized the all the compression steps on Table 3.1 and showed the compression results on Figure 3.2.



(a) fnl 4461 (b) pla 7397

(c) brd 14051 (d) d 15112

(e) d 18512

Figure 3.2 Maps of Benchmark Cities

Table 3.1 Example Result of Compression Steps

index	x	y		index	x	y		index	x	y
1	515725	507650		1	20629	20306		1	12474	29
2	520000	507650		2	20800	20306		2	12645	29
3	507725	507650	$\xrightarrow{GCD}$	3	20309	20306	$\xrightarrow{Shifting}$	3	12154	29
4	512000	507650		4	20480	20306		4	12325	29
5	203875	506925		5	8155	20277		5	0	0
	GCD_x	25			X_{MIN}	8155			X_{MIN}	0
	GCD_y	25			Y_{MIN}	20277			Y_{MIN}	0

3.4 ILS and 2-opt

ILS is an algorithmic solution which presents a heuristic to find an acceptable solution for TSP in a shorter amount of time. After generating an initial solution, ILS makes a local search on close combinations and finds the local minimum value. Then, it perturbates the current local minimum value by hoping there is a better configuration with a shorter path in another local minimum value. After the perturbation, ILS searches again for the next local minimum value in the modified solution.

The key advantage of ILS is that it can quickly find high-quality solutions even for large problem instances. Additionally, ILS can easily incorporate various problem specific heuristics and constraints to further improve the solution quality.

1. Generate an initial solution s0
2. Set s = s0
3. Repeat for a fixed number of iterations or until a stopping criterion is met:
   - Apply a local search procedure to s to obtain a new solution s′
   - Perturb s′ to obtain a new solution s″
   - If the objective function value of s″ is better than that of s, set s = s″

2-opt, on the other hand, is a local search algorithm that iteratively swaps two edges in a TSP tour to obtain a better solution. The basic idea behind 2-opt is that if two edges in a tour cross over each other, then swapping the endpoints of one of the edges will result in a shorter tour. This process is repeated until no further improvements can be made. 2-opt is computationally efficient and can often improve the quality of a solution significantly.

1. Generate an initial tour T
2. Set improved = true
3. Repeat until improved is false:
   - Set improved = false
   - For each pair of edges (i, j) and (k, l) such that i, j, k, l are distinct and $i \neq k$ and $j \neq l$:
     - If d(i, k) + d(j, l) < d(i, j) + d(k, l), reverse the portion of the tour between j and k
     - Set improved = true

Here, d(i, j) denotes the distance between nodes i and j, and the algorithm terminates when no more improvements can be made. We showed 2-opt algorithm in Figure 3.3.

However, it can sometimes get stuck in local optima and fail to find the global optimum. To overcome this issue, 2-opt can be combined with ILS or other metaheuristics to obtain better results.

Overall, both ILS and 2-opt are effective methods for solving combinatorial optimization problems, and their combination can often lead to even better results.

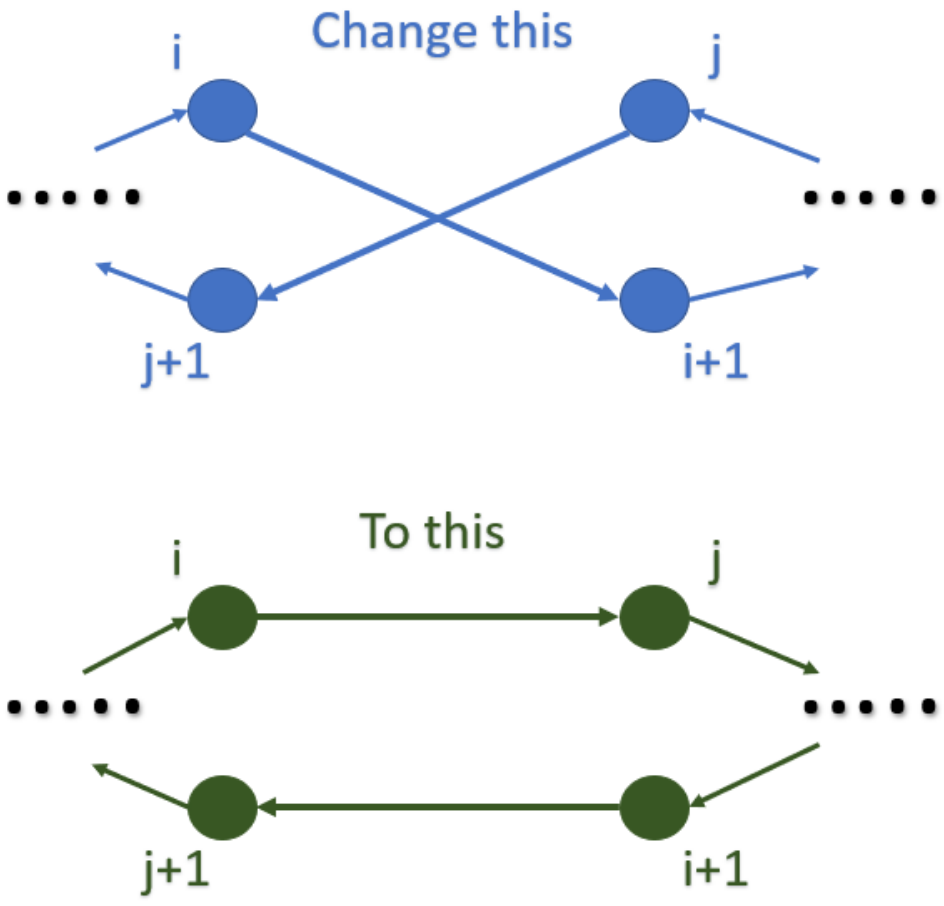


Figure 3.3 2-opt Algorithm

Chapter 4

Evaluations

In this section, we present our experimental results and demonstrate the results of our proposed method in terms of execution time. By applying our compression method, we were able to reduce the execution time significantly, as shown in Figure 4.1 and Figure 4.2. And we compared our results with Reiji Suda and Kamil Rocki [3] at Table 4.1.

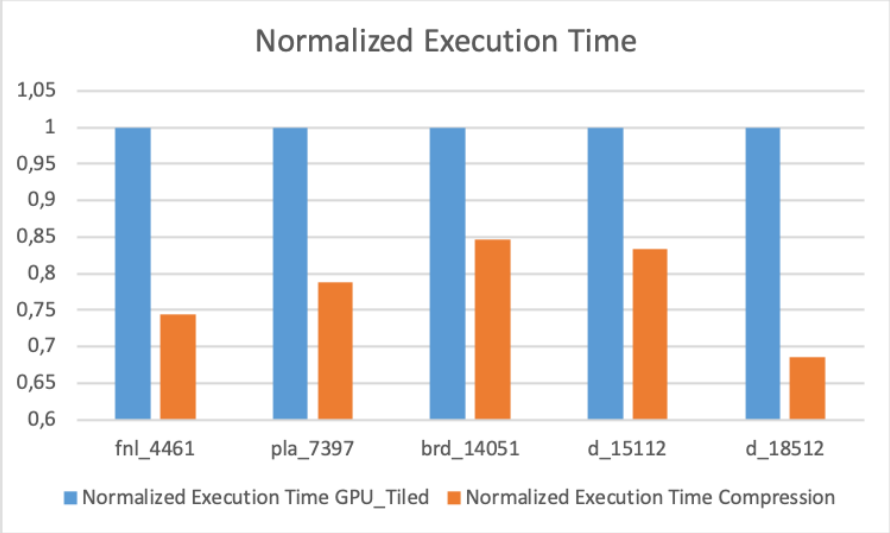


Figure 4.1 Execution Time of TSP with different inputs

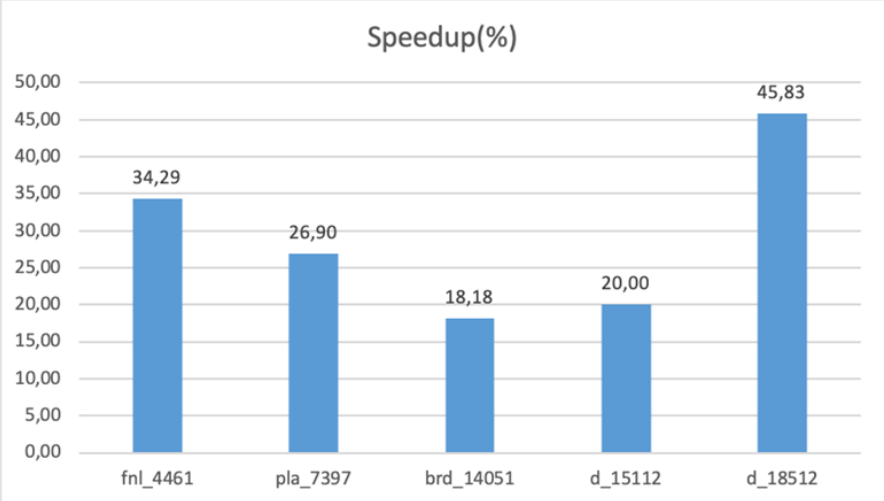


Figure 4.2 Speedup of TSP Instances

Table 4.1 Comparing the Host to Device Copy Time

Cities	Kamil Rocki, Reiji Suda (GPU: GTX 650, CPU: i7-3960)	Proposed Method (GPU : V100, CPU: Xeon)
fnl_4461	58 us	20 us
pla_7397	58 us	22 us
d_15112	69 us	25 us
d_18512	75 us	26 us

Chapter 5

Conclusions and Future Prospects

5.1 Conclusions

In this thesis, we focused on the Traveling Salesman Problem (TSP) and proposed a solution method that combines the Iterated Local Search (ILS) algorithm with the 2-opt heuristic, and utilizes the power of CUDA for parallel computation.

Our objective is to reduce the execution time of the TSP problem using data compression techniques. We have observed that the results of the compression method vary depending on the size and type of TSP instances. We believe that better results can be achieved by using different k-opt algorithms and incorporating deep learning methods into our approach.

5.2 Societal Impact and Contribution to Global Sustainability

The Traveling Salesman Problem (TSP) has numerous practical applications in a variety of fields. Some examples of practical applications of TSP include:

1. Logistics and Transportation Planning: TSP can be used to optimize delivery routes for couriers, packages, and goods. It can also help optimize the scheduling of buses, trucks, and other vehicles in public transportation systems.
2. Manufacturing and Production Planning: TSP can be used to optimize production schedules, ensuring that manufacturing processes are streamlined and efficient.

3. Circuit Design: TSP can be used to optimize the design of computer chips, ensuring that circuits are connected in the most efficient way possible.
4. DNA Sequencing: TSP can be used to optimize the sequencing of DNA, helping to identify genes and genetic mutations.
5. Computer Network Optimization: TSP can be used to optimize the routing of data packets in computer networks, improving network performance and efficiency.

By utilizing CUDA to accelerate TSP solutions, the computational time required to solve TSP instances can be reduced by orders of magnitude. This is achieved by exploiting the parallel processing power of modern GPUs, which enables a vast number of computations to be performed simultaneously. This reduction in computational time has significant implications for sustainability efforts.

By optimizing resource allocation and reducing waste, the accelerated TSP solving times enabled by CUDA can lead to more efficient routing and scheduling of transportation and logistics networks. This can help reduce fuel consumption, lower transportation costs, and ultimately contribute to a more sustainable future. Additionally, the use of CUDA to accelerate TSP solutions can enable real-time decision-making in logistics and transportation planning, which can help reduce congestion and improve the overall efficiency of these systems. Therefore, the potential benefits of using CUDA to accelerate TSP solutions are numerous, ranging from reducing carbon emissions and transportation costs to enhancing the efficiency and sustainability of complex systems in various industries.

Overall, the use of CUDA to accelerate TSP solutions is a promising research area that has the potential to contribute to the advancement of sustainable resource management practices in various fields.

5.3 Future Prospects

The use of CUDA to accelerate TSP solutions has the potential to revolutionize the field of combinatorial optimization, with numerous possibilities for further research and

development. While the use of 2-opt and ILS algorithms with CUDA has already demonstrated significant improvements in computational efficiency, there are many other algorithms that could be explored, such as k-opt algorithms.

K-opt algorithms could enable the development of even more sophisticated TSP solving techniques that are better suited to complex real-world scenarios. Additionally, the integration of machine learning algorithms such as deep learning, graph convnet, and reinforcement learning could further enhance the efficiency and effectiveness of TSP solving techniques. For example, reinforcement learning could be used to develop intelligent algorithms that learn from experience and adapt to changing conditions, while graph convnet could enable the analysis of large and complex TSP instances. These advanced techniques have the potential to revolutionize the field of TSP solving, with applications in logistics and transportation planning, manufacturing and production planning, and other areas.

Overall, the future prospects for accelerating TSP with CUDA are vast, and continued research in this area is expected to yield significant advances in computational efficiency and sustainability.

BIBLIOGRAPHY

[1] Gerhard Reinelt, "{TSPLIB}--A Traveling Salesman Problem Library," ORSA Journal on Computing, 3(4):376–384 (1991).
[2] Matai R, Singh S, Mittal ML, "Traveling Salesman Problem: an Overview of Applications, Formulations, and Solution Approaches," Traveling Salesman Problem (2010)
[3] Rocki Kamil, Suda Reiji, "Accelerating 2-opt and 3-opt Local Search Using GPU in the Travelling Salesman Problem," 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, 705-706 (2012)
[4] CUDA Toolkit, https://developer.nvidia.com/cuda-toolkit (Accessed 25th April 2023.)
[5] GTC Sept 2022 Keynote with NVIDIA CEO Jensen Huang, https://www.youtube.com/watch?v=PWcNlRI00jo (Accessed 25th April 2023.)
[6] O'Neil Molly A., Burtscher Martin, "Rethinking the Parallelization of Random-Restart Hill Climbing: A Case Study in Optimizing a 2-Opt TSP Solver for GPU Execution," Association for Computing Machinery, 99-108 (2015)
[7] Lourenco HR, Martin OC, Stutzle T, "Handbook of Metaheuristics," Springer US (2003)
[8] Croes, "A Method for Solving Traveling-Salesman Problems," Operations Research, 6, 6 (1958)
[9] Marco Dorigo, Luca Maria Gambardella, "Ant colonies for the travelling salesman problem," Biosystems, 43, 73-81 (1997)
[10] Yu, Bin, Yang, Zhong-Zhen, Yao, Baozhen, "An improved ant colony optimization for vehicle routing problem," European Journal of Operational Research, 191, 171-176 (2009)
[11] Yi Zhou, Fazhi He, Neng Hou, Yimin Qiu, "Parallel ant colony optimization on multi-core SIMD CPUs," Future Generation Computer Systems, 79, 473-487 (2018)
[12] Salal Hasan, Luna, "Solving Traveling Salesman Problem Using Cuckoo Search and Ant Colony Algorithms," Journal of Al-Qadisiyah for computer science and mathematics, 10, 59-64 (2018)
[13] Yong Wang, Zunpu Han, "Ant colony optimization for traveling salesman problem based on parameters optimization," Applied Soft Computing, 107, 107439 (2021)
[14] Skinderowicz R, "Improving Ant Colony Optimization efficiency for solving large TSP instances," Applied Soft Computing, 120, 108653 (2022)
[15] Li Li, Cheng Yurong, Tan Lijing, Niu Ben, "A Discrete Artificial Bee Colony Algorithm for TSP Problem," Springer-Verlag (2011)
[16] Bahriye Akay, "2-Opt Based Artificial Bee Colony Algorithm For Solving Traveling Salesman Problem," WCIT (2011)
[17] JIANG, "Solving Traveling Salesman Problem Using Artificial Bee Colony Algorithm," Computer Science and Technology (2017)
[18] Dong Xueshi, Lin Qing, Xu Min, Cai Yongle, "Artificial bee colony algorithm with generating neighbourhood solution for large scale coloured traveling salesman problem," IET Intelligent Transport Systems, 13, 1483-1491 (2019)

[19] Zhou Yongquan, Ouyang Xinxin, Xie Jian, “A discrete cuckoo search algorithm for travelling salesman problem,” International Journal of Collaborative Intelligence, 1, 68-84 (2014)
[20] Zicheng Zhang, Jianlin Yang, “A discrete cuckoo search algorithm for traveling salesman problem and its application in cutting path optimization,” 169, 108157 (2022)
[21] Ivan Zelinka, Roman Senkerik, Magdalena Bialic-Davendra, Donald Davendra, “Traveling Salesman Problem,” IntechOpen (2010)
[22] Ming-bo Wang, Qiang Fu, Nan Tong, Mengmeng Li, Yiming Zhao, “Proceedings of the 2015 4th National Conference on Electrical, Electronics and Computer Engineering,” Atlantis Press (2015)
[23] John J. Grefenstette, Rajeev Gopal, Brian J. Rosmaita, Dirk Van Gucht, “Genetic Algorithms for the Traveling Salesman Problem,” International Conference on Genetic Algorithms (1985)
[24] Oliver, I. M., Smith D. J., Holland J. R. C., “Proceedings of the Second International Conference on Genetic Algorithms on Genetic Algorithms and Their Application,” L. Erlbaum Associates Inc. (1987)
[25] Abdollah Homaifar, Shanguchuan Guan, Gunar E. Liepins, “A New Approach on the Traveling Salesman Problem by Genetic Algorithms,” International Conference on Genetic Algorithms (1993)
[26] Sena Giuseppe et. al., “Parallel and Distributed Processing,” Springer Berlin Heidelberg (1999)
[27] Chen Su et. al, “Computer and Information Science 2011,” Springer Berlin Heidelberg (2011)
[28] Li-Zhuang Tan, Yan-Yan Tan, Guo-Xiao Yun, Chao Zhang, “An improved genetic algorithm based on k-means clustering for solving traveling salesman problem,” Computer Science, Technology and Application (2016)
[29] Hussain Abid, Muhammad Yousaf shad, Sajid Nauman, Hussain Ijaz, Shoukry Alaa, Gani Showkat, “Genetic Algorithm for Traveling Salesman Problem with Modified Cycle Crossover Operator,” Computational Intelligence and Neuroscience (2017)
[30] Kang-Ping Wang, Lan Huang, Chun-Guang Zhou, Wei Pang, “Particle swarm optimization for traveling salesman problem,” IEEE, 3, 1583-1585 (2003)
[31] X.H. Shi, Y.C. Liang, H.P. Lee, C. Lu, Q.X. Wang, “Particle swarm optimization-based algorithms for TSP and generalized TSP,” Information Processing Letters (2007)
[32] Lin Xiong, Shunxin Li, “Solving TSP Based on the Improved Simulated Annealing Algorithm with Sequential Access Restrictions,” Atlantis Press (2016)
[33] Basu Sumanta, “Tabu Search Implementation on Traveling Salesman Problem and Its Variations: A Literature Survey,” American Journal of Operations Research, 2 (2012)
[34] Li Haitao, Alidaee Bahram, “Tabu search for solving the black-and-white travelling salesman problem,” Journal of the Operational Research Society, 67, 1061-1079 (2016)
[35] Eneko Osaba, Javier Del Ser, Ali Sadollah, Miren Nekane Bilbao, David Camacho, “A discrete water cycle algorithm for solving the symmetric and asymmetric traveling salesman problem,” Applied Soft Computing, 71, 277-290 (2018)
[36] da Costa Paulo, Rhuggenaath Jason, Zhang Yingqian, Akcay Alp, Kaymak Uzay, “Learning 2-Opt Heuristics for Routing Problems via Deep Reinforcement Learning,” SN Computer Science, 2, 388 (2021)
[37] Chandra Rohit, Dagum Leo, Kohr David, Menon Ramesh, Maydan Dror , McDonald Jeff, “Parallel programming in OpenMP,” Morgan kaufmann (2001)
[38] Nichols Bradford, Buttlar Dick, Farrell Jacqueline Proulx, “Pthreads programming - a POSIX standard for better multiprocessing,” O'Reilly (1996)

[39] Tullsen D.M., Eggers S.J., Levy H.M., “Simultaneous multithreading: Maximizing on-chip parallelism,” Proceedings 22nd Annual International Symposium on Computer Architecture (1995)
[40] Spracklen L., Abraham S.G., “Chip multithreading: opportunities and challenges,” 11th International Symposium on High-Performance Computer Architecture (2005)
[41] Amdahl Gene M., “Proceedings of the April 18-20, 1967, Spring Joint Computer Conference,” Association for Computing Machinery (1967)
[42] Molly A. O'Neil, Dan E. Tamir, Martin Burtscher, “A Parallel GPU Version of the Traveling Salesman Problem,” (2011)
[43] O'Neil Molly A., Burtscher Martin, “Proceedings of the 8th Workshop on General Purpose Processing Using GPUs,” Association for Computing Machinery (2015)
[44] Saxena Rahul, Jain Monika, Bhadri Sidhharth, Khemka Suyash, “Parallelizing GA based heuristic approach for TSP over CUDA and OPENMP,” ICACCI (2017)
[45] Tzy-Luen Ng, Keat Yeow Teck, Abdullah Rosni, “Parallel Cuckoo Search algorithm on OpenMP for traveling salesman problem,” ICCOINS (2016)
[46] Iouliia Skliarova, Antonio de Brito Ferrari, “FPGA-Based Implementation of Genetic Algorithm for the Traveling Salesman Problem and Its Industrial Application,” International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems (2002)
[47] Mavroidis Ioannis, Papaefstathiou Ioannis, Pnevmatikatos Dionisios, “A Fast FPGA-Based 2-Opt Solver for Small-Scale Euclidean Traveling Salesman Problem,” FCCM (2007)

CURRICULUM VITAE

2015 – 2019	B.Sc., Biomedical Engineering, Erciyes University, Kayseri, TURKIYE
2016 – 2019	B.Sc. (Double Major), Electrical-Electronics Engineering, Erciyes University, Kayseri, TURKIYE
2020 – Present	M.Sc. Student, Electrical and Computer Engineering, Abdullah Gül University, Kayseri, TURKIYE

SELECTED PUBLICATIONS AND PRESENTATIONS

B1) Nehri S, Yalcin S, "Sağlıkta Yeni Nesil Teknolojiler", ISBN: 9786052586495 Bölüm 2: Tıbbi Cihaz Teknolojileri ve Ülkemizdeki Yeri Page: 9-14 (2019)