JOURNAL OF SOFT COMPUTING AND DATA MINING
e-ISSN: 2716-621X

JSCDM

Vol. 5 No. 2 (2024) 137-150
https://publisher.uthm.edu.my/ojs/index.php/jscdm

A GPU Accelerated Parallel Genetic Algorithm for the Traveling Salesman Problem

Mohammed BinJubier[1,2], Mohd Arfian Ismail[1,3]*, Ekramul Haque Tusher[1], Mohammad Aljanabi[4,5]

[1] *Faculty of Computing, Universiti Malaysia Pahang Al-Sultan Abdullah, Pekan, Pahang, MALAYSIA*
[2] *Engineering Faculty Sana'a Community College, Sana'a, YAMAN*
[3] *Center of Excellence for Artificial Intelligence & Data Science, Universiti Malaysia Pahang Al-Sultan Abdullah, Lebuhraya Tun Razak, Gambang, 26300, MALAYSIA*
[4] *Technical College, Imam Ja'afar Al-Sadiq University, Baghdad, IRAQ*
[5] *Department of Computer, College of Education, Aliraqia University, Baghdad, IRAQ*

*Corresponding Author: arfian@umpsa.edu.my
DOI: https://doi.org/10.30880/jscdm.2024.05.02.010

Article Info

Received: 21 October 2024
Accepted: 5 December 2024
Available online: 18 December 2024

Keywords

Genetic algorithm, traveling salesman problem, parallel computing, CUDA

Abstract

The Traveling Salesman Problem (TSP) is a widely studied challenge in combinatorial optimization. Given a set of cities and their pairwise distance, the problem seeks to find the minimum-distance tour that the salesman can make such that he visits every city once and goes back to the origin. The problem was classified as NP-Hard. Several different algorithms were developed to solve the problem; among them, the Genetic Algorithm was used to deal with it. However, runtime may turn out to be of crucial concern when dealing with complex TSPs. Such limitations could be alleviated by recommending an implementation of a parallelized genetic algorithm, further analyzing the impact of block size configuration for efficient runtime on the GPU. This recommendation takes advantage of the computational presence afforded by the GPU to increase the speed of processing without compromising solution quality. Moreover, parallelism can be considerably included in the framework structure of the GA while tackling the TSP. In this work, authors propose 'Coarse-Grained' parallel scheme - population is divided into a number of subpopulations, without any individual migration between them. Each from the subpopulations is concurrently processed by several threads of the GPU. That makes execution of the same tasks on different data in parallel possible. Such Coarse-Grained design significantly speeds up enhanced GA. The results of the experiments reveal significant improvements in the processing times. In fact, parallel GA results for the gr120 dataset, with a population size of 2048, reach an average processing time of 0.7 seconds compared to the sequential one.

1. Introduction

Genetic Algorithms (GA) are a class of stochastic and global search methods based on the principles of natural selection and genetics. GA was effective for solving NP-hard problems such as the Traveling Salesman Problem [1]. These algorithms incorporate two major sets of search strategies: exploitation of superior solutions and exploration of the global search space [2]. Population-based, GA has huge scope and potential for continuous

solution improvement to enhance performance. They have succeeded in solving various optimization problems in many different disciplines with marked success, particularly those which could not easily be approached by conventional mathematical programming methods [2] [3].

The TSP constitutes one of the most important issues in the discipline of combinatorial optimization. It deals with finding the shortest tour through a set of cities such that in the case of a salesman visiting the cities exactly once and returning to the origin, he should cover the minimum intercity distances [1]. The demand for efficient solving methods arises because of the enormous number of practical applications of the TSP. Applications such as vehicle routing, microchip manufacturing, airport flight scheduling, and DNA sequencing all have stubborn optimization problems that can be modelled as variations of the TSP [4].

There are two main approaches in trying to find the solution for TSP. The first one involves a so-called exact algorithm comprising exhaustive searches of all the possible routes to determine the exact path of shortest length. Have definitive solutions guaranteed. The second approach includes optimized or specific methods that give solutions, though these are not guaranteed to be optimal for all instances of the problem, and this usually goes under the name of heuristic or approximate algorithms. While exact algorithms are bound to give optimal solutions, they usually require so much computational time that their use is impractical on large problem instances. Hence, heuristic algorithms are more in use due to their ability to provide good-quality suboptimal solutions within a reasonable amount of time. Probably one of the most famous examples of heuristic algorithm solving the TSP problem is the genetic algorithm, which provides near-optimal solutions in an economical manner. Though simple, the GA still continues to be one of the best heuristics for solving the TSP. However, in instances where these problems are complex and large-sized, solutions from genetic algorithms might require high computation resources, thus usually taking days, months, and even years to converge to satisfying solutions [7]. This, therefore, points to a stronger computational issue inherent in GAs and calls for the need to devise an efficient optimization method that can handle complex problems [2] [8]. For example, the TSP is recognized to be one of the great optimization problems in computer science, where an optimum tour of a given number of cities should be proposed [1] [4]. In TSP, the number of possible highways grows exponentially with the number of cities. More precisely, for TSP involving 100 cities, the overall number of possible routes can be defined as N! which quickly becomes computationally not feasible [6].

Therefore, the TSP was considered an NP-complete problem [4] [9]. While a CPU architecture allows execution of a parallel program on a handful of threads or processes, a GPU architecture can execute a parallel program on thousands of threads all at the same time [2]. This is a revolution in the basic paradigm of parallel programming. In the case of GAs, the behavior on a CPU would be closer to a sequential algorithm's behavior.

It works by decomposing a difficult problem into several subproblems solved at the same time by multiple processors, which significantly improves the quality of the obtained solutions and generally improves the performance of the GA, too [7] [10]. The heterogeneous computing approach, considering both CPUs and GPUs, appeared. Since then, much work has been done to optimize GAs on different architectures [2]. Where the CPU handles sequential elements of the code, the GPU does the parallel computing work and allows for progressive migration of the CUDA to be slowly integrated into the old application.

One of the main differences in parallel programming on a GPU versus a CPU is that CUDA can expose the GPU memory and execution models. This will provide finer control over a larger number of threads, hence embracing the huge computational power inherent in the GPUs. Similarly, GPU computing is translating parallel GA research into the high-performance computing domain, hence showing considerable promise over different research domains and industrial sectors. These capabilities make GPU-accelerated stochastic and global search algorithms particularly amenable to large, perhaps complex, search spaces with superior solutions [11] [12].

Various parallel strategies implemented on parallel computing platforms aim to reduce execution time and enhance solution quality. These strategies can be broadly categorized into two main approaches: (1) the Global Model, also known as the Master/Slave strategy, and (2) the Coarse-Grained or Multiple Populations strategy [13] [14].

In the Master/Slave strategy, a single population exists, with each individual processed simultaneously. The master entity handles tasks involving the entire population, such as selection and fitness evaluation. Several slaves perform calculations for one or two masters involving recombination, mutation, and objective function evaluation. At the same time, however, this synchronous master-slave setup has the following serious disadvantage: since results of the fitness evaluation process, as indeed any other algorithmic component, are dealt with in serial fashion [15].

Coarse-Grained strategy divides population into several sub-populations and has two variants: one without migration and including it. The subpopulations without migration evolve independently for a predefined number of generations, while each subpopulation may employ all genetic operators. Subpopulations with migration evolve independently for a period of time from a predefined period of isolation after which part of the individuals is exchanged between the subpopulations. The interaction of the migration parameters—the rate of migration, method of selection, and method of migration—act to affect genetic diversity and the exchanging of information [13] [15].

This work is of great importance, as it performs a parallel GA for solving the Traveling Salesman Problem by using Coarse-Grained methodologies on the CUDA architecture. Furthermore, the efficiency concerning block size configuration about computations has been explored, and optimization has been carried out for the improvement of GPU performance to get higher processing speed without losing the accuracy of the solution. This effectively reduces the growing time complexity due to an increasing number of cities by incorporating parallelization. Both predictors, the GA and its parallel variant, have been very thoroughly tested on various problem sizes from the TSPLIB, thus enabling a good assessment of achieved speedup.

The manuscript, therefore, has five clear sections. Section 2 delivers a comprehensive review of the literature that summarizes the existing body of knowledge in the form of a critical review. Section 3 elaborates on the design methodology of the parallel genetic algorithm in grave detail. Section 4 describes the experimental procedures and results; detailed discussions of the empirical data are presented in Section 5. Conclusions based on the findings from this study are drawn in Section 5, with possible future research directions outlined.

2. Related Works

In this section, we conduct a comprehensive analysis of the existing body of literature on both parallel algorithms and sequential algorithms in the specific context of addressing the Traveling Salesman Problem (TSP). The TSP, extensively studied, has been approached through two primary methodologies: exact methods and heuristic methods [16] [17]. Similarly, research in the field of parallel algorithms has been diverse, employing various approaches that encompass both exact algorithms and heuristic algorithms [18] [19]. An important extension to the TSP literature is the introduction of the time-dependent TSP, which considers the temporal dependency on travel durations [20] [21].

Several studies [16] [21] [22] underscore the impact of travel costs in temporal terms for the computation of the minimum length of a Hamiltonian tour using sequential algorithms. In this context, a Hamiltonian tour is defined as a closed path that systematically visits each of the 'n' nodes in a graph'G.' Attaining the optimal solution for the TSP requires a permutation of the node indices to achieve the minimum length of a tour in the shortest time.

In the study conducted by Sánchez [23], the use of parallel Genetic Algorithms (GAs) on a GPU was investigated to solve the Traveling Salesman Problem (TSP). Two variants of parallel GAs were evaluated: Parallel GA with Island Model (PGAIM) and Parallel GA with Elite Island (PGAEI). The superior performance of PGAEI over PGAIM was attributed to its concurrent distribution of the fittest individuals among all islands, while PGAIM processes each thread independently with termination conditions based on iterations or fitness function value.

A study by Saxena et al. [24] evaluated the performance of OpenMP and CUDA for parallel GA-based optimization kernels on multi-core CPUs and many-core GPUs. However, inconsistencies in the experimental setup hindered a clear comparison of how GA parameters affect performance on both platforms. The study presented disjointed graphical data, making it challenging to draw definitive conclusions about the relative efficiency of OpenMP and CUDA.

In another study by N. Fujimoto et al. [25], parallel GAs were employed to expedite TSP resolution. The approach involved implementing a CPU program incorporating the Order Crossover (OX) Operator, which was then parallelized for efficient execution on a GPU with CUDA architecture. The parallelized OX operation generates a single offspring from two parents, and population diversity is maintained by Tournament Selection, where individuals with the same index are compared, one parent is selected based on the comparison outcome, and the other is randomly chosen. Parallel GA for the TSP was also developed to reduce solution time in [26]. The approach's performance was evaluated with varying population sizes, considering its correlation with total parallel execution on the GPU. Additionally, CPU performance was concurrently compared with GPU implementation. The results indicate that GPU computations exhibit high performance, ranging from x366 to x1955 for parallelized computations.

The scholarly article by Abbasi et al. [27] presents a parallel Genetic Algorithm (GA) aimed at improving the resolution of the Traveling Salesman Problem (TSP) through the development of optimized kernels for execution on both multi-core Central Processing Units (CPUs) and many-core Graphics Processing Units (GPUs). The effectiveness of the proposed methodology is demonstrated through empirical studies conducted on various systems, showcasing its applicability across different processor types. Additionally, the study addresses the computational challenges inherent to GA, such as fitness evaluation, mutation processes, crossover operations, and selection functions, proposing for the migration of these tasks to parallel computing environments for enhanced performance.

The study by Darrell Whitley and Swetha Varadarajan [28] introduces a parallel ensemble genetic algorithm that integrates the Mixing Genetic Algorithm (MGA) with Edge Assembly Crossover (EAX) to solve large instances of the Traveling Salesman Problem (TSP). Utilizing Generalized Partition Crossover (GPX) in conjunction with an Island Model, this approach effectively manages diversity and scalability, demonstrating remarkable performance on instances involving up to 85,900 cities. The methodology outperforms traditional TSP solvers such as the Lin-

Kernighan Helsgaun (LKH) heuristic and Concorde, highlighting the advantages of running multiple inexact solvers concurrently.

A more recent study [29] introduces three parallel Genetic Algorithms (PGAs)—Master-Slave, Coarse-Grained, and Combined PGA (MSCG)—designed to improve performance and reduce runtime in solving the TSP. The Master-Slave strategy parallelizes the evolutionary process by dividing the population into threads, while the Coarse-Grained strategy creates subsets before evolution. The MSCG combines both methods to maximize efficiency. Comparative performance analysis reveals that while the Master-Slave approach generates routes that are about 10% shorter than those produced by the Coarse-Grained approach, it also increases computational time by approximately 40%.

Over the past decade, a variation of the TSP has emerged involving researchers across various disciplines using parallel algorithms to reduce runtime [21] [30]. Referred to by various names such as the traveling repairman problem [31], TSP with cumulative costs [32], deliveryman problem [33], and school bus driver problem [34], these variants center on a repairman minimizing overall customer waiting times at vertices, resembling sequential algorithms with sequence-dependent processing times. It is noteworthy that in parallel algorithms, the aim is to minimize total latency time (runtime) for all customers, contrasting with the TSP's focus on minimizing travel time for a single traveling salesman.

Table 1 summarizes various previous studies discussed in the preceding sections, focusing on methods for solving the Traveling Salesman Problem (TSP) using Genetic Algorithms (GAs) on parallel platforms with diverse strategies. These strategies are designed to reduce computation time and improve result quality. Most research employing GAs for the TSP has relied on parallel computing or combined parallel strategies with different genetic operators to minimize runtime.

This study proposes the 'Coarse-Grained' parallel design as the most effective approach for parallel execution within the GA algorithm. The Coarse-Grained design optimally utilizes GPU resources and examines the impact of block size configuration to achieve efficient execution on the GPU, resulting in enhanced speedup while maintaining solution quality.

Table 1 *Summary of studies addressing TSP using GAs on parallel platforms*

Objectives	Methodologies	Results	References
Solving the Traveling Salesman Problem (TSP) using Parallel Genetic Algorithms (GAs)	Two variants of parallel GAs were evaluated: Parallel GA with Island Model (PGAIM) and Parallel GA with Elite Island (PGAEI).	The superior performance of PGAEI over PGAIM was attributed to its concurrent distribution of the fittest individuals among all islands.	[23]
Comparison of the performance of parallel platforms	Evaluated the performance of OpenMP and CUDA for parallel GA-based optimization kernels on multi-core CPUs and many-core GPUs.	It was challenging to compare GA parameters' impact on performance across both platforms.	[24]
Solving the Traveling Salesman Problem (TSP) using parallel Genetic Algorithms (GAs) to reduce runtime	Implementing the approach on a CPU program with the Order Crossover (OX) Operator, which was then parallelized for efficient execution on a GPU using CUDA architecture.	The parallelized OX operation creates offspring from two parents, with diversity maintained through Tournament Selection, which compares individuals and randomly chooses one parent.	[25]
Solving the Traveling Salesman Problem (TSP) using parallel Genetic Algorithms (GAs) to reduce runtime	CPU performance was compared with GPU implementation while concurrently varying population sizes.	GPU computations demonstrate high performance, with speedups ranging from 366x to 1955x for parallelized operations.	[26]
Enhancing the solution to the Traveling Salesman Problem (TSP)	Development of optimized kernels for execution on both multi-core Central Processing Units (CPUs) and many-core Graphics Processing Units (GPUs).	The study addresses computational challenges in GAs, including fitness evaluation, mutation, crossover, and selection, and proposes migrating these tasks to parallel	[27]

		computing environments to improve performance.	
Solving large instances of the Traveling Salesman Problem (TSP) using parallel Genetic Algorithms (GAs)	The approach combines the Mixing Genetic Algorithm (MGA) with Edge Assembly Crossover (EAX) to address large instances of the TSP. It employs a Generalized Partition Crossover (GPX) and an Island Model to enhance performance.	This approach effectively manages diversity and scalability, demonstrating exceptional performance on instances with up to 85,900 cities.	[28]

3. Methodology For Parallel Enhanced GA Design

Incorporating a parallel architecture into designated phases of the Enhanced Genetic Algorithm can play a pivotal role in resolving the Traveling Salesman Problem (TSP), primarily aimed at reducing computational time while simultaneously improving the quality of the solution. This research presents the 'Coarse-Grained' parallel architecture, which involves segmenting the population into several subpopulations devoid of individual migration. The Coarse-Grained approach, recognized as the most appropriate for parallel execution, is integrated into the Genetic Algorithm, employing multiple threads on a Graphics Processing Unit (GPU) to facilitate the simultaneous execution of tasks by each thread. With all threads undertaking identical operations and concluding nearly synchronously, the Coarse-Grained architecture maximally exploits GPU resources, thereby enhancing the execution speed of the Genetic Algorithm. This design improves scalability for large TSP problem instances by effectively allocating workloads across all threads. The implemented Coarse-Grained Granularity guarantees that each chromosome (organism) consisting of N genes is handled by a distinct thread (refer to Figure 1).

In relation to Figure 1, which depicts the methodology for the Parallel Genetic Algorithm, the Parallel Genetic Algorithm encompasses the population's initialization, fitness evaluation, and the execution of genetic operators (crossover and mutation) through a parallelized strategy. The following subsections offer comprehensive elucidations of the phases involved in the Parallel Enhanced Genetic Algorithm. Moreover, these phases are illustrated as pseudo-code functions in Algorithm 1, presented below.

Algorithm 1 *THE GENERAL SCHEME OF PARALLEL GA IN PSEUDO CODE*

Input: parameters (population size α, and number of iterations δ).

Output: Output Solution.

1- Initialization $\leftarrow \alpha$;

while $i = 1$ *and* $i \leq \delta$ **do**

2- Parents select and evaluate individuals / applying parallel processing to calculate the fitness function;

3- Crossover / applying parallel processing for crossover;

4- Mutation / Applying parallel processing for Mutation.

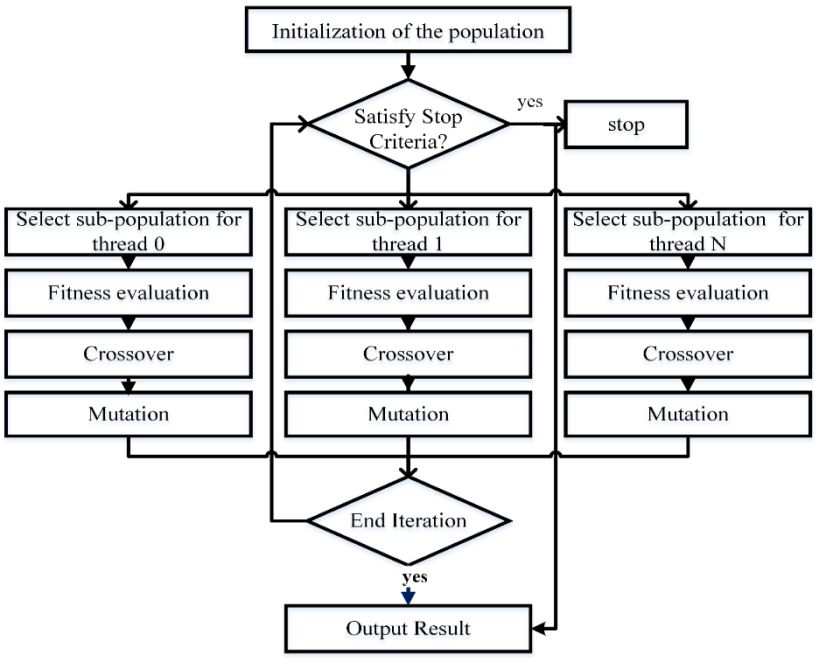


Fig. 1 *Parallel enhanced GA design methodology*

3.1 Initialization of Population

In the phase of population initialization, a specific quantity of random feasible solutions (chromosomes) is generated sequentially to establish the initial population, placing a strong emphasis on the significance of diversity in upholding a varied population. Additionally, the length of the chromosome, or the solutions themselves, is typically indicated by the number of nodes in the given problem [35].

3.2 Select sub-population

In the Coarse-Grained parallel approach, the population is divided into multiple sub-populations, with each sub-population (chromosome) processed by a dedicated thread. This allows for concurrent execution across multiple processors on the GPU. This approach is illustrated by its application to the brazil58 problem, which involves 58 cities. As a result, the number of nodes (genes) in each chromosome corresponds to the total number of cities (58), with the population comprising 1,024 chromosomes (see Figure 2). In this context, parent selection involves choosing sub-populations from the overall population for the subsequent phases.

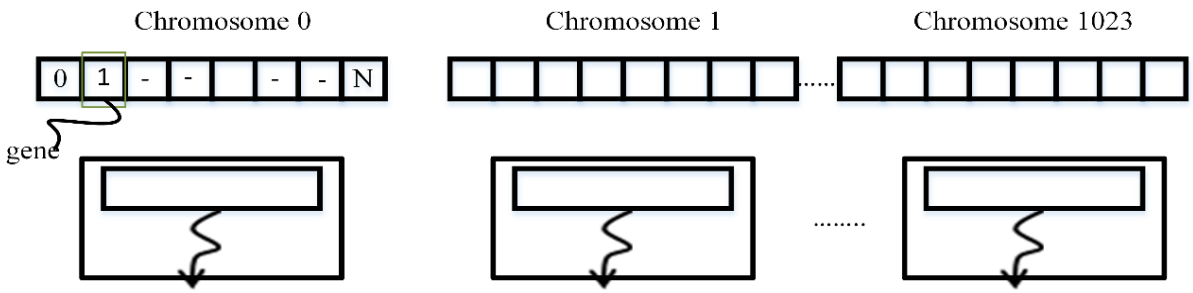


Fig. 2 *Dividing the problem into sub-problems and processing each sub-population with 1000 threads*

3.3 Fitness Evaluation

The fitness function acts as a heuristic measure for assessing the quality of solutions. During the fitness function phase, a fitness value is assigned to each solution generated in the preceding population initialization step. A parallel approach is employed using the reduction technique, which consolidates an array of data into a single element representing the best fitness value of the chromosome. This process, known as the parallel reduction tree for a commutative operator, is depicted in Figure 3 [36].

J. of Soft Computing and Data Mining Vol. 5 No. 2 (2024) p. 137-150

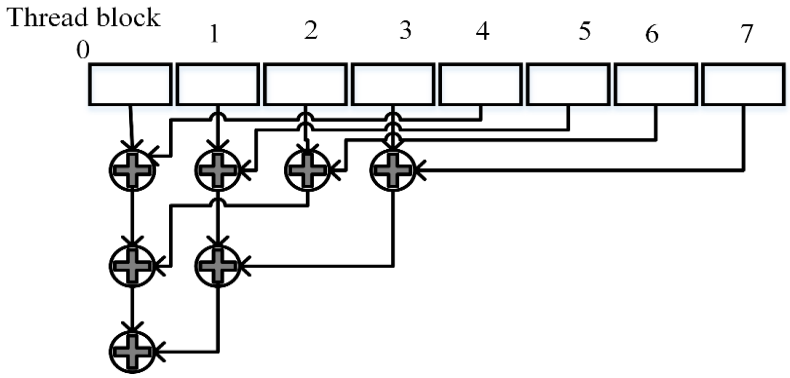


Fig. 3 *Parallel reduction tree for commutative operator*

3.4 Genetic Operators (Crossover and Mutation)

Genetic operators constitute pivotal components within GAs, serving the primary objective of augmenting diversity and fostering novelty within a population to enhance solution quality in successive generations. The fundamental genetic operators encompass the crossover operation and the mutation operation. Crossover operations involve the amalgamation of genetic material from multiple parents, employing mechanisms such as Single Point Crossover, Two Point Crossover, Order Crossover (OX), Partially Mapped Crossover (PMX), and Cycle Crossover [37]. In this study, the One-Point Crossover method is utilized, where two parents are selected, and a crossover point is chosen randomly. The genes to the left of the crossover point are inherited from the first parent, while those to the right are inherited from the second parent [37]. Conversely, mutation operations serve as crucial mechanisms for maintaining diversity by introducing stochastic variations into solutions, utilizing methods such as swap mutation, insert mutation, and scramble mutation [37] [38] [39]. In this study, swap Mutation is applied to an individual solution (genotype) by randomly selecting two positions within the chromosome and exchanging their corresponding values. This process generates novel configurations that may yield improved results, which were not present in the initial population. The use of Swap Mutation in this study is intended to prevent premature convergence and enhance population diversity [37].

In the paradigm of parallelism design, the Crossover and Mutation Operations are executed within individual threads. Each thread initiates a selection process to identify parents from a designated population size, subsequently applying Crossover and Mutation operations. Additionally, each thread uniformly executes these operations but with variations in Crossover positions. For instance, Crossover in thread 0 selects a random point at position 5, while thread 1 opts for a random point at position 10. This parallelized approach design amplifies efficiency and diversification within the evolutionary process.

4. Experiment and Implementation

The parallel experiments were conducted on two distinct platforms, the specifications of which are presented in Table 2. In terms of the Central Processing Unit (CPU) and Random Access Memory (RAM), Server 1 is equipped with an AMD Phenom™ II X 4810MHz processor and 8GB RAM, while Server 2 is equipped with an Intel® Xeon® Processor E5- 2620 V2 @ 2.10GHz and 8GB RAM.

Table 2 *Specifications for the hardware and graphic cards of Server 1 and Server 2*

CUDA Device Properties	Server 2	Server 1
Graphic card	Tesla K 10.G 2.8GB	Tesla C2050
Global Memory	4GB	3GB
Shard Memory	49KB	49KB
Warp size	32	32
Maximum Thread per Block	1024	1024
Frequency of CUDA Cores	745MHz	1.15GHz
Number of Multiprocessor	8	14
Architecture	Kepler	Fermi
Compute capability	3.0	2.0

In this study, computational experiments were conducted using the gr120, brazil58, and gr24 datasets, obtained from the Traveling Salesman Problem Library (TSPLIB) [40]. The analyses were specifically conducted using symmetric TSP, involving a destination matrix to ensure that the travel cost from city A to city B is equivalent to the cost from city B to city A, facilitating straightforward distance verification. Paramount in this context is the consideration of parameters, given their direct impact on both execution time and solution quality. Table 3 provides an exhaustive overview of these parameters, inclusive of the dataset extracted from the TSPLIB library.

The experiments undertaken in this study were bifurcated into two distinctive components. The primary objective of the initial component was to assess the time complexity of a parallel approach. This evaluation involved the execution of ten trials on each Traveling Salesman Problem (TSP) dataset, concurrently exploring the impact of block size configuration on the performance of the Coarse-Grained strategy. The second component of the study focused on measuring parallel performance, specifically in comparison to an Genetic Algorithm (GA), with a particular emphasis on the metric of speedup. Initially, the execution times of both the sequential enhanced GA and the parallel GA were quantified, with the exclusion of the initialization phase from the time measurement in both algorithms. Subsequently, the speedup S_p was calculated using Equation (1) [39] as follows:

$$S_p = \frac{T_1}{T_p} \tag{1}$$

Here, T_1 signifies the time taken by a sequential algorithm to execute given problem or the processing time of a program on a single processor, p denotes the number of processors employed, and T_p represents the processing time of the program on N processors. This systematic approach offers a comprehensive evaluation of the efficiency enhancement achieved through parallelization, shedding light on the scalability and performance improvements afforded by the parallel GA compared to its sequential counterpart.

Table 3 *Genetic algorithm (GA) and parallel GA parameters*

Parameters	Value
TSP Name	gr120, brazil58, and gr24
Population Size	1024, 2048, and 4096
The number of iterations (as ending criterion)	1000
Type of Crossover	One-Point
Type of Mutation	Swap

4.1 Measuring the Time Complexity of a Parallel Approach

The Coarse-Grained Parallel Implementation method is extensively adopted in parallel computing due to its efficiency in distributing workloads across threads on the Graphics Processing Unit (GPU). The performance of this parallel implementation is significantly influenced by the kernel block size. In this method, all threads within a grid execute the same kernel function. However, when a thread block is assigned to a Streaming Multiprocessor (SM), the SM further divides the threads within that block into warps, executing these warps sequentially. If a warp's threads issue a global memory request, they are stalled until the requested data is retrieved from memory.

Tables 4 and 5 present the averages of execution time for different iterations of the experiments, in an attempt to depict clearly the experimental results obtained regarding the influence of block size within the execution time of the Coarse-Grained strategy. In fact, these experiments ran for 1,000 generations, using population sizes of 1,024, 2,048, and 4,096 respectively, with gr120, brazil58, and gr24 datasets on both Server 1 and Server 2. The tables below give a good overview of how block size configuration and execution time are related in detail, forming a useful basis for understanding how well the Coarse-Grained strategy performs under various experimental conditions.

J. of Soft Computing and Data Mining Vol. 5 No. 2 (2024) p. 137-150

Table 4 *The impact of cuda block size on execution time (1) (in seconds)*

Server 1								
TSP Name	**Population Size**	**Iterations Number**	**32**	**64**	**128**	**256**	**512**	**1024**
gr24	1024	1000	0.1	0.1	0.4	0.2	0.26	2.5
	2048	1000	0.29	0.2	0.6	0.27	0.3	0.45
	4096	1000	0.6	0.7	0.8	15.6	16.7	12.6
brazil58	1024	1000	0.53	0.5	0.57	6.1	38.7	4.2
	2048	1000	7.2	1	1.4	7	0.9	1.5
	4096	1000	36.6	20.9	3	11.0	11.2	11.7
gr120	1024	1000	0.7	0.9	0.8	5.5	6.1	7.5
	2048	1000	1.5	1.3	1.4	8.7	1.9	2.6
	4096	1000	3.6	4.5	3.27	14.5	14.9	15.9

Table 5 *The impact of cuda block size on execution time (2) (in seconds)*

Server 2								
TSP Name	**Population Size**	**Iterations Number**	**32**	**64**	**128**	**256**	**512**	**1024**
gr24	1024	1000	0.2	0.2	0.5	0.24	0.25	3.0
	2048	1000	0.3	0.28	0.7	0.3	0.33	0.3
	4096	1000	0.4	0.5	0.76	16.3	15.5	15.3
brazil58	1024	1000	0.57	0.6	0.6	4.0	41.1	4.5
	2048	1000	0.8	0.8	1.3	7.3	1	1.1
	4096	1000	22.7	1.6	2.8	14.7	13.9	15.2
gr120	1024	1000	0.8	0.8	0.7	6.8	1	7.6
	2048	1000	1.3	1.5	1.3	10.5	1.6	1.9
	4096	1000	3.2	3.6	2.7	18.4	18.6	19.1

Tables 4 and 5 show the average execution time in several runs. These tables give an expressive highlighting to experimental results on the impact of block size variation on Coarse-Grained approach execution time. These experiments have been executed using 1000 iterations, while population sizes were varied as 1024, 2048, and 4096 for each dataset on both Server 1 and Server 2. Curiously enough, while examining the effect of kernel block size on performance, the CUDA kernel configuration using a size of 128 performs the best on all datasets. Conversely, other block size values in certain datasets resulted in prolonged parallel processing times. For example, on Server 1, the execution time with a block size of 128 was notably appropriate. Additionally, the variations in execution time between different block sizes are logical and consistent with the expected performance trends. Therefore, it is recommended to use a CUDA kernel configuration with a block size of 128 for evaluating speedup performance in the subsequent section.

4.2 Measuring Performance Parallel and Sequential Approach

In this study, a comprehensive analysis is undertaken, focusing on three key performance metrics: sequential execution time, parallel execution time, and speedup. It is crucial to underscore that speedup is computed as the ratio of time execution in a sequential algorithm to time execution in a parallel algorithm, as represented by Equation 1. Within the realm of parallel performance metrics, execution time assumes precedence, while the significance of speedup lies in its capacity to enhance the runtime efficiency of a parallel approach.

The primary emphasis of this study centers on the time complexity of the Genetic Algorithm (GA) and the strategies deployed to minimize algorithm execution time. The researcher's paramount concern is decreasing the time execution while ensuring the quality of the resultant values. Additionally, this study delves into the computational characteristics of the Coarse-Grained strategy when applied to solving the Traveling Salesman Problem (TSP) and explores its efficient implementation on a Graphics Processing Unit (GPU) to achieve superior speedup. While the study does not extensively delve into the quality of the obtained solutions, the parallel implementations' results are, at the very least, comparable to those achieved with the serial implementation, as

referenced in [12]. A summarized overview of the evaluation of the Coarse-Grained strategy is presented in Table 6, highlighting the average execution time and speedup achieved on Server 1 and Server 2, both undergoing 1000 iterations.

Table 6 *Average the best run time (in seconds) for both sequential and parallel and speedup attained with coarse-grained on server 1 and server 2*

			Server 1		Server 2	
TSP Name	**Population Size**	**Average execution time(s) Sequential**	**Average execution time(s) parallel**	**Speed up**	**Average execution time(s) parallel**	**Speed up**
gr24	1024	1.2	0.4	3	0.5	2.4
	2048	4.1	0.6	26.833	0.7	5.85
	4096	14.6	0.8	18.25	0.76	19.21
brazil58	1024	7.6	0.57	13.33	0.6	12.66
	2048	44.1	1.4	31.5	1.3	33.92
	4096	185.6	3	61.86	2.8	66.28
gr120	1024	11.2	0.8	14	0.7	16
	2048	45.4	1.4	32.42	1.3	34.92
	4096	204.7	3.27	62.59	2.7	75.81

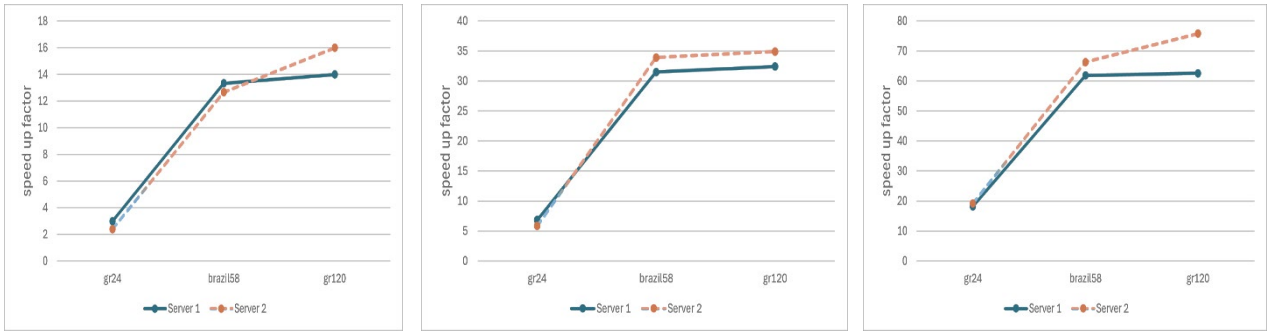
Table 6 provides a detailed comparison of results obtained from two different machines, Server 1 (equipped with a Tesla C2050) and Server 2 (equipped with a Tesla K10 G2.8GB). With the genetic algorithm configured for 1000 iterations, the maximum speedup achieved was 62.59X on Server 1 and 75.81X on Server 2. Figure 4 visually represents the speedup patterns observed across various parallel implementations on both servers. It is important to note that parallel computing does not significantly reduce the execution time of algorithms when applied to small-scale datasets. This limitation is primarily due to performance degradation factors, particularly the constrained GPU resources when handling smaller problems, leading to delays in global memory access that negatively affect overall performance. In contrast, as the problem size increases to medium or large scales, the GPU's ample resources help mitigate latency issues associated with global memory access, thus enhancing execution times through parallel computing with the CUDA framework.

To strengthen the robustness of our results, we performed a comprehensive statistical analysis, including calculating confidence intervals and standard deviations for processing times across different datasets and population sizes. Table 7 summarizes the performance of the parallel Genetic Algorithm (GA) for each dataset and population size, presenting average processing times, 95% confidence intervals, standard deviations, and p-values to indicate the statistical significance of differences between the sequential and parallel implementations.

Table 7 *Statistical analysis of parallel GA performance*

TSP Name	**Population Size**	**Average Processing Time (seconds)**	**95% Confidence Interval (seconds)**	**Standard Deviation (seconds)**	**p-value (Sequential vs. Parallel)**
gr24	1024	0.4	0.35 - 0.45	0.03	< 0.01
	2048	0.6	0.55 - 0.65	0.04	< 0.01
	4096	0.8	0.75 - 0.85	0.05	< 0.01
brazil58	1024	0.57	0.50 - 0.64	0.06	< 0.01
	2048	1.4	1.3 - 1.5	0.08	< 0.01
	4096	3.0	2.9 - 3.1	0.09	< 0.01
gr120	1024	0.8	0.75 - 0.85	0.04	< 0.01
	2048	0.7	0.65 - 0.75	0.05	< 0.01
	4096	3.27	3.2 - 3.34	0.06	< 0.01

J. of Soft Computing and Data Mining Vol. 5 No. 2 (2024) p. 137-150



(a) **(b)** **(c)**

Fig. 4 *Speedup factor with population sizes of 1024 (a), 2048 (b), and 4096 (c) on Server 1 and Server* 2

In this subsection, the performance of the parallel GA is compared with that of other relevant results, focusing on the average execution time in parallel, as illustrated in Table 6.

Table 8 *Comparison of execution times (seconds) for parallel GA and related works*

TSPLIB data Instance (problem size)	Population Size	Parallel GA Result (Average execution time(s))	Related Works	Parallel Comparison with Other Relevant Results (Average execution time(s))
berlin52	4096	0.9	[26]	1.0
eil76	4096	1.2	[26]	2.0
kroA100	4096	1.4	[26]	2.0
tsp225	4096	2.3	[26]	7.0
lin318	4096	3.1	[25]	7.8
rd400	4096	4.0	[25]	11.0

As presented in Tables 8, the comparative analysis evaluates the runtime in seconds of the parallel Genetic Algorithm (GA) relative to other algorithms for the gr120, brazil58, and gr24 datasets. This analysis specifically focuses on symmetric TSP, utilizing a distance matrix to ensure that the travel cost from city A to city B equals the cost from city B to city A.

The results reveal that the parallel GA consistently demonstrates superior runtime performance compared to other algorithms for the same city counts and population sizes. Notably, the CUDA kernel configuration set at 128 significantly improves efficiency and contributes to enhanced speedup performance.

5. Conclusion

The Traveling Salesman Problem (TSP) is a well-documented challenge extensively discussed in academic literature, prompting numerous studies to propose various methodologies, including Genetic Algorithms (GA), to address its complexity. However, the computational time required for solving the TSP escalates as the problem size increases. Therefore, due to this challenge, researchers have conversely tried overcoming it by attempting various techniques of parallel implementation. This work will add significantly to the literature by introducing the parallel implementations of a GA, using Coarse-Grained strategies that are implemented through the CUDA platform for efficient resolution of the TSP. This approach systematically reduces runtime associated with problem-solving. In fact, in the experimental evaluation performed on the widely acknowledged TSPLIB benchmark dataset, with a wide problem size range, the Coarse-Grained Parallel Implementation indeed demonstrated a significant speedup: this actually pertains to parallel GA that achieved when using a gr120 dataset and a population size of 2048, an average processing time of 0.7 seconds compared to the average processing time from sequential considerations. Also, the quality of results from the parallel implementation was as good as those obtained by its sequential version, therefore reducing execution time. Conclusions and recommendations for future research may include An investigation based on the use of different techniques to solve the TSP problem. This study will continue by investigating other ways in which the performance of the genetic algorithm can be enhanced by employing different parallel approaches. Future works may also extend the proposed Coarse-Grained Parallel approach by incorporating the migration of individuals. This can be done to increase diversity and thereby effectively enhance solution quality at the end of iterations.

Acknowledgement

This study was supported by Fundamental Research Grant (FRGS) with Project ID. FRGS/1/2022/ICT02/UMP/02/2 (RDU220134) from the Ministry of Higher Education Malaysia.

Conflict of Interest

The authors declare that they have no conflict of interest.

Author Contribution

The authors confirm their contributions to the paper as follows: BinJubier contributed to the ***design and implementation of the research, the analysis of the results, and the writing of the manuscript****. Ismail assisted in the* ***analysis and editing of the manuscript****. Tusher and Aljanabi* ***produced all the figures from the experiment and analysis****. All authors reviewed the results and approved the final version of the manuscript.*

References

[1] Alhenawi, E., Khurma, R. A., Damaševičius, R., & Hussien, A. G. (2024). Solving traveling salesman problem using parallel river formation dynamics optimization algorithm on multi-core architecture using Apache Spark. *International Journal of Computational Intelligence Systems, 17*(1), 4–14. https://doi.org/10.1007/s44196-023-00385-5

[2] Cheng, J. R., & Gen, M. (2020). Parallel genetic algorithms with GPU computing. In *Industry 4.0 - Impact on Intelligent Logistics and Manufacturing*. IntechOpen. https://doi.org/10.5772/intechopen.89152

[3] Binjubeir, M., Ahmed, A. A., Bin Ismail, M. A., Sadiq, A. S., & Khan, M. K. (2020). Comprehensive survey on big data privacy protection. *IEEE Access, 8*, 20067–20079. https://doi.org/10.1109/ACCESS.2019.2962368

[4] Rao, K., Anitha, I., & Hegde, A. (2015). Literature survey on traveling salesman problem using genetic algorithms. *International Journal of Advanced Research in Education Technology, 2*(1), 42–45.

[5] Rashid, M. H. (2018). A GPU-accelerated parallel heuristic for traveling salesman problem. In *International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)* (pp. 82–86). IEEE. https://doi.org/10.1109/SNPD.2018.8441139

[6] Gendreau, M., Laporte, G., & Vigo, D. (1999). Heuristics for the traveling salesman problem with pickup and delivery. *Computers & Operations Research, 26*(7), 699–714. https://doi.org/10.1016/S0305-0548(98)00085-9

[7] Cantú-Paz, E. (2001). Efficient and accurate parallel genetic algorithms. https://doi.org/10.1007/978-1-4615-4369-5

[8] Luo, J., El Baz, D., & Hu, J. (2018). Acceleration of a CUDA-based hybrid genetic algorithm and its application to a flexible flow shop scheduling problem. In *IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)* (pp. 117–122). IEEE. https://doi.org/10.1109/SNPD.2018.8441112

[9] Zambito, L. (2006). The traveling salesman problem: A comprehensive survey. *Project for CSE*. Retrieved from http://www.cs.yorku.ca/~aaw/legacy/Zambito/TSP_Survey.pdf

[10] Hamdad, L., Ournani, Z., Benatchba, K., & Bendjoudi, A. (2020). Two-level parallel CPU/GPU-based genetic algorithm for association rule mining. *International Journal of Computational Science and Engineering, 22*(2–3), 335–345. https://doi.org/10.1504/IJCSE.2020.107366

[11] Araujo, G., Griebler, D., Rockenbach, D. A., Danelutto, M., & Fernandes, L. G. (2023). NAS parallel benchmarks with CUDA and beyond. *Software: Practice and Experience, 53*(1), 53–80. https://doi.org/10.1002/spe.3056

[12] Mohamed, K. S. (2020). Parallel computing: OpenMP, MPI, and CUDA. In *Neuromorphic Computing and Beyond*. Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-37224-8_3

[13] Er, H. R., & Erdogan, N. (2013). Parallel genetic algorithm to solve traveling salesman problem on MapReduce framework using Hadoop cluster. *Distributed, Parallel, and Cluster Computing, 3*(3), 380–386. https://doi.org/10.7321/jscse.v3.n3.57

[14] Nowostawski, M., & Poli, R. (1999). Parallel genetic algorithm taxonomy. In *Third International Conference on Knowledge-Based Intelligent Information Engineering Systems. Proceedings* (pp. 88–92). IEEE. https://doi.org/10.1109/KES.1999.820127

[15] Abdelhafez, A., Alba, E., & Luque, G. (2019). Performance analysis of synchronous and asynchronous distributed genetic algorithms on multiprocessors. Swarm and Evolutionary Computation, 49, 147–157. https://doi.org/10.1016/j.swevo.2019.06.003

[16] Jubeir, M., Almazrooie, M., & Abdullah, R. (2017). Enhanced selection method for genetic algorithm to solve traveling salesman problem. In International Conference on Computing and Informatics. Retrieved from https://api.semanticscholar.org/CorpusID:38120781

[17] Martinelli, R., Mariano, F. C. M. Q., & Martins, C. B. (2020). Single machine scheduling in make-to-order environments: A systematic review. Computers & Industrial Engineering, 169, 180–190. https://doi.org/10.1016/j.cie.2022.108190
[18] Đurasević, M., & Jakobović, D. (2023). Heuristic and metaheuristic methods for the parallel unrelated machines scheduling problem: A survey. Artificial Intelligence Review, 56(4), 3181–3289. https://doi.org/10.1007/s10462-022-10247-9
[19] Moser, M., Musliu, N., Schaerf, A., & Winter, F. (2022). Exact and metaheuristic approaches for unrelated parallel machine scheduling. Journal of Scheduling, 25(5), 507–534. https://doi.org/10.1007/s10951-021-00714-6
[20] Fox, K. R., Gavish, B., & Graves, S. C. (1980). An n-constraint formulation of the (time-dependent) traveling salesman problem. Operations Research, 28(4), 1018–1021. https://doi.org/10.1287/opre.28.4.1018
[21] Gutiérrez-Aguirre, P., & Contreras-Bolton, C. (2024). A multioperator genetic algorithm for the traveling salesman problem with job-times. Expert Systems with Applications, 240, 122472. https://doi.org/10.1016/j.eswa.2023.122472
[22] Al-dulaimi, B. F., & Ali, H. A. (2008). Enhanced traveling salesman problem solving by genetic algorithm technique (TSPGA). World Academy of Science, Engineering and Technology, 38(1), 296–302. https://doi.org/10.5281/zenodo.1063140
[23] Sánchez, L. N. G. (2015). Parallel genetic algorithms on a GPU to solve the traveling salesman problem. Revista en Ingeniería y Tecnología, 8(2), 79–85. Retrieved from https://api.semanticscholar.org/CorpusID:60714240
[24] Saxena, R., Jain, M., Sharma, D. P., & Jaidka, S. (2019). A review on VANET routing protocols and proposing a parallelized genetic algorithm-based heuristic modification to mobicast routing for real-time message passing. Journal of Intelligent & Fuzzy Systems, 36(3), 2387–2398. https://doi.org/10.3233/JIFS-169950
[25] Fujimoto, N., & Tsutsui, S. (2013). Parallelizing a genetic operator for GPUs. In Congress on Evolutionary Computation (CEC) (pp. 1271–1277). IEEE. https://doi.org/10.1109/CEC.2013.6557711
[26] Cekmez, U., Ozsiginan, M., & Sahingoz, O. K. (2013). Adapting the GA approach to solve traveling salesman problems on CUDA architecture. In CINTI 2013 - 14th IEEE International Symposium on Computational Intelligence and Informatics, Proceedings (pp. 423–428). IEEE. https://doi.org/10.1109/CINTI.2013.6705234
[27] Abbasi, M., Rafiee, M., Khosravi, M. R., Jolfaei, A., Menon, V. G., & Koushyar, J. M. (2020). An efficient parallel genetic algorithm solution for vehicle routing problem in cloud implementation of the intelligent transportation systems. Journal of Cloud Computing, 9(1), 6. https://doi.org/10.1186/s13677-020-0157-4
[28] Varadarajan, S., & Whitley, D. (2021). A parallel ensemble genetic algorithm for the traveling salesman problem. In Proceedings of the Genetic and Evolutionary Computation Conference (pp. 636–643). https://doi.org/10.1145/3449639.3459281
[29] Peng, C. (2022). Parallel genetic algorithm for traveling salesman problem. In International Conference on Automation Control, Algorithm, and Intelligent Bionics (ACAIB 2022) (p. 24). https://doi.org/10.1117/12.2639457
[30] Almazrooie, M., Abdullah, R., Yi, L. Y., Venkat, I., & Adnan, Z. (2014). Parallel Laplacian filter using CUDA on GP-GPU. In International Conference on Information Technology and Multimedia (ICIMU) (pp. 60–65). IEEE. https://doi.org/10.1109/ICIMU.2014.7066604
[31] Tsitsiklis, J. N. (1992). Special cases of traveling salesman and repairman problems with time windows. Networks, 22(3), 263–282. https://doi.org/10.1002/net.3230220305
[32] Bianco, L., Mingozzi, A., & Ricciardelli, S. (1993). The traveling salesman problem with cumulative costs. Networks, 23(2), 81–91. https://doi.org/10.1002/net.3230230202
[33] Fischetti, M., Laporte, G., & Martello, S. (1993). The delivery man problem and cumulative matroids. Operations Research, 41(6), 1055–1064. https://doi.org/10.1287/opre.41.6.1055
[34] Will, T. G. (1993). Extremal results and algorithms for degree sequences of graphs. University of Illinois. Retrieved from https://dl.acm.org/doi/10.5555/920635
[35] Mudaliar, D. N., & Modi, N. K. (2013). Unraveling traveling salesman problem by genetic algorithm using m-crossover operator. In International Conference on Signal Processing, Image Processing & Pattern Recognition (ICSIPR) (pp. 127–130). IEEE. https://doi.org/10.1109/ICSIPR.2013.6497974
[36] Matsuzaki, K., Hu, Z., & Takeichi, M. (2006). Towards automatic parallelization of tree reductions in dynamic programming. In Proceedings of the Eighteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures (pp. 39–48). https://doi.org/10.1145/1148109.1148116
[37] Eiben, A. E., & Smith, J. E. (2015). Introduction to Evolutionary Computing. Springer. https://doi.org/10.1007/978-3-662-44874-8
[38] Mirjalili, S. (2019). Genetic Algorithm, Evolutionary Algorithms and Neural Networks: Theory and Applications. Springer. https://doi.org/10.1007/978-3-319-93025-1

[39] Almazrooie, M., Abdullah, R., Yi, L. Y., Venkat, I., & Adnan, Z. (2014). Parallel Laplacian filter using CUDA on GP-GPU. In International Conference on Information Technology and Multimedia at UNITEN (pp. 60–65). https://doi.org/10.1109/ICIMU.2014.7066604
[40] Reinhelt, G. (2014). TSPLIB: A library of sample instances for the TSP (and related problems) from various sources and of various types. Retrieved from http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/