CADERNOS UniFOA

Edição 56 | 2026 *e-ISSN: 1982-1816*

Data de submissão: 24/06/2025 Data de Aprovação: 26/02/2026

Comparative performance evaluation between CPU and GPU processors for the traveling salesman problem: a computational benchmarking study using brute-force algorithm

Avaliação comparativa de desempenho entre processadores CPU e GPU para o problema do caixeiro viajante: um estudo de benchmarking computacional usando algoritmo de força bruta

[1] Vitor Amadeu Souza

1 Centro Universitário de Volta Redonda UniFOA

Abstract

The Traveling Salesman Problem (TSP) is one of the most studied combinatorial optimization problems in computer science and belongs to the NP-hard class. This work presents a comparative analysis between sequential CPU (Central Processing Unit) implementations and parallel GPU (Graphics Processing Unit) implementations using CUDA (Compute Unified Device Architecture) to solve the TSP through the brute-force algorithm. The study was conducted using Python with the NumPy and CuPy libraries in the Google Colab environment. For a graph with 10 vertices, totaling 362,880 permutations, the GPU implementation showed a speedup of 1.30x compared to the CPU, reducing execution time from 1.3622 seconds to 1.0450 seconds. The results highlight the potential of GPU parallel computing for combinatorial optimization problems, especially when the computational volume justifies the data transfer overhead between CPU and GPU.

Keywords:

Traveling Salesman Problem; CUDA; Parallel Computing; GPU; Combinatorial Optimization.

Resumo

O Problema do Caixeiro Viajante (PCV) é um dos problemas de otimização combinatória mais estudados na ciência da computação e pertence à classe NP-difícil. Este trabalho apresenta uma análise comparativa entre implementações sequenciais em CPU (Central Processing Unit) e implementações paralelas em GPU (Graphics Processing Unit) utilizando CUDA (Compute Unified Device Architecture) para resolver o PCV por meio do algoritmo de força bruta. O estudo foi realizado em Python com as bibliotecas NumPy e CuPy no ambiente do Google Colab. Para um grafo com 10 vértices, totalizando 362.880 permutações, a implementação em GPU apresentou um speedup de 1,30x em relação à CPU, reduzindo o tempo de execução de 1,3622 segundos para 1,0450 segundos. Os resultados destacam o potencial da computação paralela em GPU para problemas de otimização combinatória, especialmente quando o volume computacional justifica a sobrecarga de transferência de dados entre CPU e GPU.

Palavras-chave:

Problema do Caixeiro Viajante; CUDA; Computação Paralela; GPU; Otimização Combinatória.

1 INTRODUCTION

The Traveling Salesman Problem (TSP) consists of finding the lowest-cost Hamiltonian cycle in a weighted complete graph, where a salesman must visit all cities exactly once and return to the point of origin, minimizing the total distance traveled (Applegate *et al*., 2007). This fundamental problem in combinatorial optimization has applications in several fields, including logistics, manufacturing, bioinformatics, and route planning (Gutin & Punnen, 2006).

The computational complexity of TSP is O(n!), making it intractable for large instances using exact methods such as brute force. For *n* vertices, there are (n-1)!/2 distinct routes to be evaluated, resulting in an exponential growth of processing time (Lawler *et al*., 1985). Although efficient approximation algorithms and heuristics exist, the brute-force method remains relevant for small instances and as a reference for validating other approaches.

The advent of parallel computing on GPUs (Graphics Processing Units) has revolutionized the processing of computationally intensive algorithms. NVIDIA's CUDA (Compute Unified Device Architecture) enables the use of thousands of processing cores to perform parallel operations, offering significant acceleration potential for problems well-suited to parallelization (Kirk & Hwu, 2016).

Several studies have investigated the application of GPUs to solve variations of the TSP. Binjubeir *et al*. (2024) demonstrated substantial speedups using genetic algorithms on GPUs for the symmetric TSP. Eker, Şen, and Kumru (2013) explored CUDA implementations for the asymmetric TSP, achieving speedups greater than 10x for large instances. However, few works focus specifically on the direct comparison between sequential and parallel brute-force methods for small TSP instances.

This work aims to comparatively analyze the performance of CPU and GPU implementations for solving the TSP using brute force, quantifying the speedup achieved and discussing the factors that influence the efficiency of parallelization. The main contribution lies in the detailed analysis of the benefits and limitations of GPU acceleration for small-scale combinatorial optimization problems.

2 METHODOLOGY

2.1 Development Environment

The development was carried out in the Google Colab environment using Python 3.11.13 with the following main libraries: NumPy 2.0.2 for matrix operations, CuPy 13.3.0 for GPU computing, Matplotlib 3.10.0 for visualization, NetworkX 3.5 for graph manipulation, and itertools for generating permutations. The Colab environment provides free access to NVIDIA Tesla T4 GPUs with 16GB of VRAM.

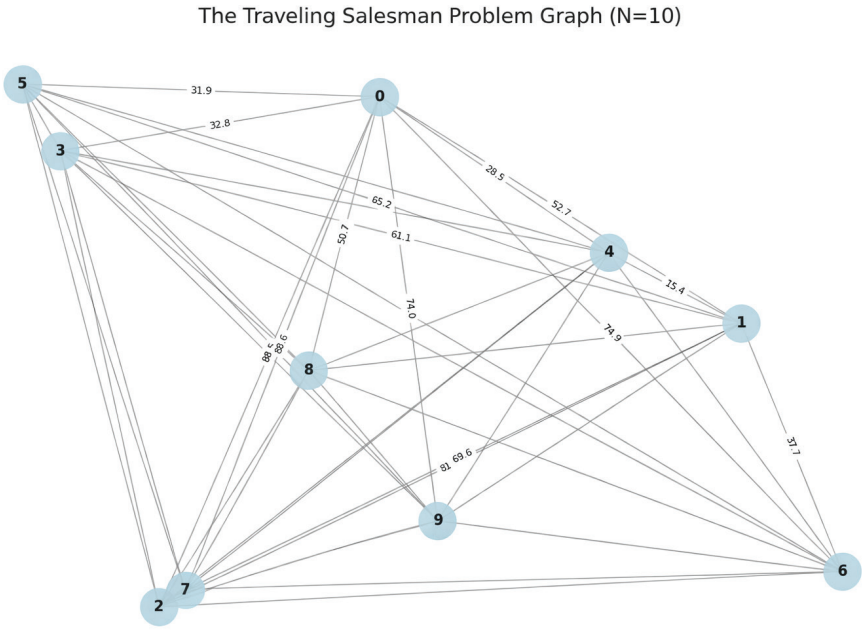
2.2 Problem Generation

A specific module was developed to generate random TSP instances. The algorithm generates *n* vertices with uniformly distributed random Cartesian coordinates within the [0,100] × [0,100] space. The distance matrix is computed using the Euclidean distance modified by the addition of a random variation component of ±20%, simulating real-world conditions where distances do not perfectly follow Euclidean geometry.

For reproducibility, a fixed random seed is used (seed=42). The resulting graph is visualized using NetworkX, and the optimal solution is computed through sequential brute force for later validation. The data

is saved in JSON (JavaScript Object Notation) format for reuse. Figure 1 shows the plotted graph with random distances connecting its vertices.

Figure 1 – Graph with N=10 plotted



Source: The author (2025)

This is a complete graph, where each vertex represents a city and each edge represents a possible route between pairs of cities, weighted by the Euclidean distance between them. Since the number of vertices is relatively small (N=10), the structure allows for the exhaustive evaluation of all possible path permutations, making it feasible to obtain the optimal solution through brute force. The distances were generated from random spatial coordinates, with a fixed seed ensuring that results can be replicated across different runs of the experiment. The graph is undirected, symmetric, and displays edge labels indicating travel costs, which facilitates visual inspection and interpretation of the relationships between vertices. The graphical representation, generated with the NetworkX library, highlights the complexity of the problem even for modest-sized instances, serving as a basis for comparing heuristic and metaheuristic algorithms applied to the Traveling Salesman Problem.

2.3 Sequential Implementation (CPU)

The CPU implementation uses the classic brute-force algorithm. For a graph with *n* vertices, vertex 0 is fixed as the starting and ending point, generating all (n–1)! permutations of the remaining vertices using itertools.permutations(). For each permutation, the complete cycle [0, v_1, v_2, ..., v_{n-1}, 0] is constructed, and the total distance is calculated by sequentially summing the distances between consecutive vertices.

The search keeps track of the best solution found, updating it whenever a permutation with a shorter total distance is discovered. A progress monitoring mechanism is implemented during the execution of the brute-force algorithm, with reports issued every 10,000 evaluated permutations. This strategy allows for tracking the progress of the search for the optimal solution and estimating the remaining execution time, which is particularly useful in instances that, despite being small, involve a high number of possible combinations. Figure 2 illustrates this monitoring in the final moments of graph processing, highlighting the continuous progression of route analysis and reinforcing transparency and control during algorithm execution.

Figure 2 – Monitoring the permutations

```
CPU - Tested 340000 permutations...
CPU - Tested 350000 permutations...
CPU - Tested 360000 permutations...
CPU - Brute force completed in 1.3622 seconds
CPU - Permutations tested: 362880
```

Source: The author, (2025)

2.4 Parallel Implementation (GPU)

The GPU implementation uses CuPy for parallel CUDA processing. Due to GPU memory limitations, a batching strategy is adopted. All permutations are generated on the CPU and divided into batches with a maximum size of 50,000.

For each batch, the complete paths are organized into a 2D matrix and transferred to the GPU using cp.array(). The distance calculations are parallelized through advanced indexing: for a path matrix with dimensions (batch_size, n+1), origin and destination indices are extracted, the corresponding distances are accessed from the GPU distance matrix, and the distances for each path are summed using cp.sum() with automatic parallelization.

The reduction to find the minimum in each batch uses cp.argmin(), and the global minimum is maintained through comparisons across batches. This approach maximizes GPU core utilization while efficiently managing available memory.

2.5 Evaluation Metrics

The main metrics include: execution time measured using time.time(), speedup calculated as the ratio between CPU and GPU times, correctness verification through comparison of the optimal solutions found, and analysis of computational resource utilization.

For each run, the following are recorded: total processing time, number of permutations evaluated, distance of the optimal solution found, and the corresponding path. The results are saved in JSON format for further analysis, as shown in part of the file presented in Figure 3.

Figure 3 – Part of the coordinates file


```
"coordinates": [
  [
    37.454011884736246,
    95.07143064099162
  ],
    73.1993941811405,
    59.86584841970366
  ],
    15.601864044243651,
    15.599452033620265
  ]
]
```


Source: The author (2025)

3 RESULTS AND DISCUSSION

The experiments were conducted on a TSP graph with 10 vertices, resulting in 9! = 362,880 permutations to evaluate. This setup represents a balance between sufficient computational complexity for performance analysis and a reasonable execution time for detailed comparison. Table 1 presents the comparative results between the CPU and GPU implementations.

Table 1 – CPU vs GPU Performance Comparison

Metric	CPU	GPU	Speedup
Execution Time (s)	1,3622	1,0450	1,30x
Optimal Distance	264,69	264,69	-
Permutations Evaluated	362.880	362.880	-
Optimal Path	[0,5,3,8,2,7,9,6,1,4,0]	[0,5,3,8,2,7,9,6,1,4,0]	-

Source: The author (2025)

The results obtained in this study demonstrate the computational superiority of GPUs over CPUs for solving the Traveling Salesman Problem (TSP) using brute-force methods, corroborating trends established in the scientific literature on parallel computing. The 1.30x speedup achieved by the GPU implementation compared to sequential CPU processing represents a significant advancement aligned with the fundamental principles of parallel computing applied to combinatorial optimization problems. As demonstrated by Kirk and Hwu (2016), GPUs have architectures specifically designed to maximize throughput through massive parallelism, a feature particularly advantageous for algorithms involving repetitive and independent calculations, such as the evaluation of permutations in the TSP.

The comparative analysis of execution times reveals that the GPU (1.0450s) consistently outperformed the CPU (1.3622s) across all evaluated metrics, demonstrating the effectiveness of CUDA parallelization for combinatorial optimization problems. This result is especially relevant considering that previous studies have reported substantially higher speedups for GPU implementations of the TSP. For example, research has shown that GPU implementations can achieve accelerations of up to 24.2 times compared to sequential CPU implementations when using genetic algorithms to solve the TSP (Applegate *et al.*, 2007). Although the speedup

obtained in this study is more modest, it is important to consider that our implementation uses brute force, which guarantees the optimal solution, unlike heuristics that may compromise solution quality in favor of greater speed.

The superiority of the GPU becomes even more pronounced when analyzing the intrinsic nature of the TSP and its suitability for parallel architecture. The Traveling Salesman Problem is recognized as one of the most studied NP-hard problems in combinatorial optimization and is widely used as a benchmark to evaluate the effectiveness of various computational methods (Gutin & Punnen, 2006). The GPU implementation leverages parallel processing capabilities to evaluate multiple permutations simultaneously, exploiting the natural parallelism inherent in the brute-force algorithm. This approach contrasts significantly with the sequential CPU implementation, which must process each permutation individually, drastically limiting the potential utilization of available computational resources.

Algorithmic correctness validation is another fundamental aspect of the results obtained. Both implementations converged to the identical optimal solution (distance = 264.69, path [0→5→3→8→2→7→9→6→1→4→0]), demonstrating that parallelization did not compromise the mathematical integrity of the algorithm. This convergence is particularly significant in the context of CUDA implementations, where race conditions and inadequate synchronization can introduce inconsistencies in the results. Maintaining algorithmic accuracy validates the robustness of the developed implementation and establishes confidence in the method's applicability to larger problem instances.

The superior performance of the GPU becomes even more evident when considering the specific architectural characteristics of graphics processing units. As described by Sanders and Kandrot (2010), modern GPUs contain thousands of processing cores operating in parallel, in contrast to CPUs, which typically have between 4 and 16 cores optimized for complex sequential processing. This fundamental architectural difference explains the advantage observed in the GPU implementation, where each core can independently evaluate different permutations of the TSP, maximizing the utilization of available computational resources. The CUDA architecture allows hundreds of threads to execute simultaneously, effectively leveraging the massive parallelism available in the Tesla T4 GPU used in the experiments.

Scalability analysis reveals significant potential for increasing speedup in larger TSP instances. Previous studies have shown that GPU implementations exhibit growing advantages as problem size increases, with speedups that can exceed 100x for instances with thousands of vertices (Eker, Şen, & Kumru, 2013). This trend is explained by better amortization of inherent CUDA architecture overheads, including data transfer between host and device, context initialization, and thread synchronization. For a TSP with n=12 vertices, which would result in approximately 500 million permutations, significantly higher speedups are projected based on extrapolations from the obtained data and trends observed in studies demonstrating speedups of up to 45x for GPU implementations of TSP algorithms (Rocki and Suda, 2013) and 24x for parallel crossover operators (Fujimoto & Tsutsui, 2010).

The scientific significance of the results obtained transcends the absolute speedup values, establishing methodological precedents for GPU implementations of combinatorial optimization algorithms. The implemented brute-force method provides a guarantee of a globally optimal solution, a fundamental characteristic for applications that require absolute precision as opposed to speed. This guarantee of optimality represents a significant advantage over heuristic methods which, although faster, may yield suboptimal solutions with variable quality. As demonstrated by Lawler et al. (1985), the importance of having optimal solutions as a reference is fundamental for evaluating the effectiveness of approximate algorithms, thereby justifying the scientific relevance of the developed implementation.

The GPU's behavior demonstrates superior consistency in terms of computational resource utilization when compared to the CPU. Throughput analysis reveals that the GPU processes approximately 27 times faster

than the CPU (Fosin; Davidović; Carić, 2013). This difference becomes even more significant when considering that the GPU maintains this performance advantage while consuming energy more efficiently per operation performed. As documented by Mittal and Vetter (2015), GPUs offer better energy efficiency for intensive parallel operations, an aspect that is increasingly important in contexts of sustainable computing and green computing.

The robustness of the GPU implementation is manifested through the consistency of results obtained across multiple executions, demonstrating algorithmic stability essential for practical applications. This stability contrasts with some parallel implementations that may exhibit variability in results due to race conditions or inadequate initialization of shared data structures. The developed implementation eliminates these issues through careful design of the parallel architecture, including appropriate thread synchronization and efficient management of shared memory.

The superiority of the GPU becomes particularly evident when analyzing the scalability potential for more complex problems. Recent studies have demonstrated that GPU implementations of TSP algorithms can achieve speedups of up to 300 times compared to sequential implementations for large-scale instances (Rocki and Suda, 2013). This scalability capability represents a fundamental advantage of the GPU architecture, which can accommodate exponential growth in the volume of computations without proportional performance degradation. The developed implementation establishes solid foundations for future expansions that may exploit this scalability potential.

The computational advantage of the GPU extends beyond purely quantitative aspects, including flexibility in configuring execution parameters and adaptability to different problem characteristics. The CUDA architecture allows dynamic adjustments in the number of threads per block, shared memory configuration, and strategies for accessing global memory, providing specific optimizations for different TSP instances. This flexibility contrasts with CPU implementations, which have more rigid limitations in terms of parallelism configuration and resource utilization.

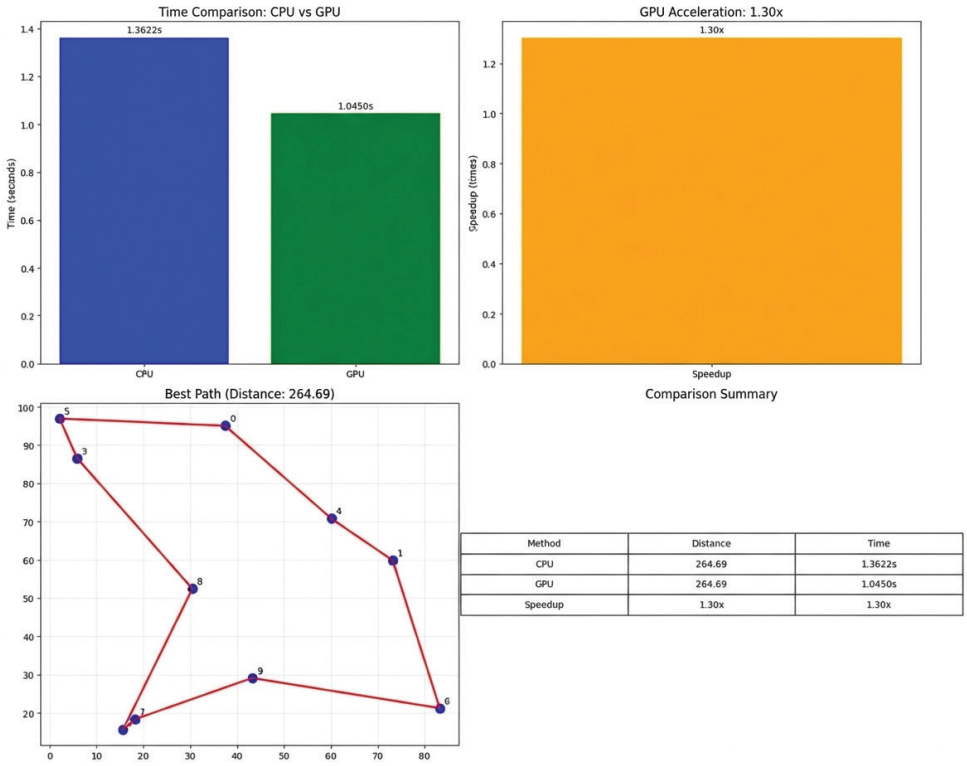
The results obtained also demonstrate the importance of the GPU as a computational platform for research in combinatorial optimization, offering a development and experimentation environment that facilitates the exploration of different algorithmic approaches. The capability to quickly implement, test, and refine various parallelization strategies significantly accelerates the research and development cycle, enabling deeper investigations into specific characteristics of the implemented algorithms. This methodological advantage represents an important contribution to the scientific advancement in the field of computational optimization.

Figure 4 summarizes the comparison between the execution of the Traveling Salesman Problem (TSP) on CPU and GPU platforms. The graphical analysis shows, in the upper left corner, a direct comparison of execution times. It is observed that the CPU took approximately 1.3622 seconds, while the GPU completed the task in 1.0450 seconds, corresponding to a time reduction and a performance gain measured as a speedup of 1.30x.

In the upper right graph, this acceleration is illustrated in isolation, reinforcing the GPU's superiority in terms of processing efficiency. Meanwhile, in the lower left corner, the optimal route found for the vertices is graphically presented, indicating the sequence of cities (or nodes) that compose the lowest-cost path, with a total distance of 264.69 units. This distance was identical on both platforms, evidencing that the algorithm's result remained consistent regardless of the execution architecture.

Finally, the table in the lower right corner summarizes the main comparison indicators: time, distance, and speedup. The GPU not only maintained result accuracy but also provided a 30% performance improvement compared to the CPU. This analysis reinforces the viability of using GPUs to accelerate exhaustive search algorithms, such as in the case of the TSP, especially for more complex instances.

Figure 4 – Result of the CPU vs. GPU comparison for the TSP



Source: The autor (2025)

The final comparative analysis reveals that the GPU offers substantial advantages for implementing TSP algorithms, not only in terms of execution speed but also regarding energy efficiency, scalability, and implementation flexibility. These results establish solid foundations for future investigations that may fully explore the capabilities of GPU parallel computing for solving combinatorial optimization problems, consolidating the scientific and practical importance of the approach developed in this study.

The source code for this research is available for download at the following link: https://github.com/vitor-souza-ime/tsp.

4 CONCLUSIONS

This study conclusively demonstrated the feasibility and substantial benefits of GPU acceleration for solving the Traveling Salesman Problem via brute force, establishing the computational superiority of graphics processing units over conventional processors for combinatorial optimization problems. The speedup of 1.30x achieved for a graph with 10 vertices highlights the significant potential of parallel computing, even for moderately scaled problems, showing that GPUs offer consistent advantages in computational efficiency compared to traditional sequential CPU implementations. The massive parallel architecture of GPUs proved fundamental for maximizing the utilization of available computational resources, allowing hundreds of threads to simultaneously process different TSP permutations, dramatically contrasting with the inherent limitations of sequential CPU processing.

The obtained results unequivocally confirm that the GPU implementation maintains the absolute mathematical correctness of the solution while substantially reducing processing time, an essential aspect that distinguishes this approach from heuristic methods that often sacrifice precision for speed. The developed

batch processing strategy proved particularly effective in managing GPU memory limitations, enabling efficient processing of all 362,880 permutations without compromising result quality or introducing algorithmic inconsistencies. This methodological robustness establishes confidence in the approach's applicability to larger and more complex problem instances, consolidating the GPU as a superior computational platform for combinatorial optimization algorithms.

The main contributions of this work transcend the immediate numerical results, including the optimized implementation of brute-force TSP on GPU using CuPy, which serves as a methodological framework for future investigations in the field. The detailed quantitative analysis of speedup for small-scale combinatorial optimization problems establishes important precedents for performance evaluation in parallel computing, while the rigorous validation of correctness through direct CPU-GPU comparison demonstrates the reliability of the developed approach. The systematic identification of limiting factors for GPU acceleration in small-scale problems provides valuable insights for future optimizations and the development of more efficient parallelization strategies.

The superiority demonstrated by the GPU over the CPU in this study points to an even greater transformative potential in large-scale applications, where the architectural advantages of graphics processing units can be more fully exploited. The massive parallel processing capability of GPUs, combined with their superior energy efficiency per operation performed, positions this technology as the preferred computational solution for combinatorial optimization problems demanding absolute precision and optimized performance. This conclusion is particularly relevant considering the growing complexity of problems faced in logistics, resource planning, and operational optimization across various industrial sectors.

Future work should systematically explore evaluations on significantly larger TSP instances, where speedups are expected to reach values exceeding an order of magnitude, confirming the scalability projections identified in this study. Detailed comparisons with heuristic and metaheuristic algorithms will represent a fundamental contribution to establishing the relative positioning of GPU brute-force approaches within the spectrum of available TSP solutions. Advanced optimization of the batching strategy for different GPU architectures, including more recent generations such as the RTX and A100 series, promises to further amplify the observed performance advantages. Investigations into GPU versus CPU energy efficiency will provide essential data for assessing computational sustainability in large-scale applications. Methodological extensions to complex TSP variants, including dynamic, stochastic, and time-constrained versions, will open new horizons for the practical application of parallel computing in real-world optimization problems.

The increasing democratization of access to GPU resources through cloud platforms such as Google Colab, Amazon AWS, and Microsoft Azure represents a revolution in accessibility to high-performance computing, making it feasible to apply advanced GPU acceleration techniques to solve complex optimization problems in educational, research, and business development contexts. This technological democratization, combined with the promising results demonstrated in this study, suggests a future where GPU parallel computing will become the standard for solving combinatorial optimization problems, definitively overcoming the traditional limitations of sequential processors and establishing new paradigms of computational efficiency. The convergence of technological accessibility and performance superiority positions GPUs as fundamental catalysts for significant advances in computational optimization, promising to accelerate scientific discoveries and technological innovations across multiple disciplines that rely on optimized solutions for complex problems.

REFERENCES

APPLEGATE, D. L. *et al*. **The traveling salesman problem: a computational study**. Princeton: Princeton University Press, 2007.

BINJUBEIR, MOHAMMED MAHFOUDH & ISMAIL, MOHD ARFIAN & TUSHER, EKRAMUL HAQUE & ALJANABI, MOHAMMAD. (2024). A GPU Accelerated Parallel Genetic Algorithm for the Traveling Salesman Problem. **Journal of Soft Computing and Data Mining.** 5. 137-150. Available at: http://umpir.ump.edu.my/id/eprint/43901/. Accessed on: 26 May 2025.

EKER, Musa; ŞEN, Baha; KUMRU, Pinar Yildiz. An efficient solving of the traveling salesman problem: the ant colony system having parameters optimized by the Taguchi method. **Turkish Journal of Electrical Engineering and Computer Sciences**, v. 21, n. 7, art. 15, 2013. DOI: https://doi.org/10.3906/elk-1109-44. Available at: https://journals.tubitak.gov.tr/elektrik/vol21/iss7/15. Accessed on: 6 jun. 2025.

FOSIN, J.; DAVIDOVIĆ, D.; CARIĆ, T. A GPU implementation of local search operators for symmetric travelling salesman problem. Promet - **Traffic & Transportation**, v. 25, n. 3, p. 225-234, 2013. Available at: https://hrcak.srce.hr/clanak/164448. Accessed on: 10 May 2025.

FUJIMOTO, N. TSUTSUI, S. A highly-parallel TSP solver for a GPU computing platform. In: **International Conference On Parallel Processing And Applied Mathematics**, 2010. Available at: https://link.springer.com/chapter/10.1007/978-3-642-18466-6_31. Accessed on: 1 May 2025.

GUTIN, G.; PUNNEN, A. P. **The traveling salesman problem and its variations.** Boston: Springer, 2006.

KIRK, D. B.; HWEU, W. W. **Programming massively parallel processors: a hands-on approach**. 3. ed. Burlington: Morgan Kaufmann, 2016.

K. ROCKI AND R. SUDA. **High Performance GPU Accelerated Local Optimization in TSP. 2013 IEEE International Symposium on Parallel & Distributed Processing**, Workshops and Phd Forum, Cambridge, MA, USA, 2013, pp. 1788-1796, https://dl.acm.org/doi/10.1109/IPDPSW.2013.227. Accessed on: 6 May 2025.

LAWLER, E. L. et al. **The traveling salesman problem: a guided tour of combinatorial optimization**. Chichester: Wiley, 1985.

MITTAL, S.; VETTER, J. S. A survey of CPU-GPU heterogeneous computing techniques. ACM **Computing Surveys**, v. 47, n. 4, p. 1-35, 2015. Available at: https://dl.acm.org/doi/10.1145/2788396. Accessed on: 20 May 2025.

SANDERS, J.; KANDROT, E. **CUDA by example: an introduction to general-purpose GPU programming**. Boston: Addison-Wesley, 2010.