Combinatorial Optimization - Traveling Salesman Problem (TSP)

Augusto Mathias Adams*, Caio Phillipe Mizerkowski†, Christian Piltz Araújo‡ and Vinicius Eduardo dos Reis§

*GRR20172143, augusto.adams@ufpr.br, †GRR20166403, caiomizerkowski@gmail.com,
‡GRR20172197, christian0294@yahoo.com.br, §GRR20175957, eduardo.reis02@gmail.com

***Abstract*—In this academic work is presented the performance of various solvers on four TSP instances: ch130, ch150, eil101, and lin318. We examined the results obtained from Simulated Annealing, Random Multistart Local Search, Tabu Search, and Memetic Algorithm for solving these TSP instances. Overall, we found that all the solvers were able to produce high-quality solutions for these TSP instances. Specifically, Tabu Search and Simulated Annealing showed good performance for some instances, while Memetic Algorithm performed well for all instances except lin318. However, the performance of each solver was dependent on the characteristics of the TSP instance being solved. This highlights the importance of carefull selection of the appropriate solver for a specific TSP instance. The results suggest that a combination of several solvers, such as the Memetic Algorithm, may be the most effective approach to solving TSP instances. In summary, the findings provide valuable insights into the performance of different solvers on TSP instances and can aid in selecting an appropriate solver for solving specific TSP problems.**

***Index Terms*—Optimization Methods, Traveling Salesman Problem, Combinatorial Optimization, Metaheuristics, Performance analysis.**

I. Introduction

Note: *The content of this section is an overview or general summary of class notes, recommended bibliography and many articles read throughout the semester on numerical and combinatorial optimization, heuristics and metaheuristics, so that this work is the epitome of all the topics covered during the course. It's more like a study guide than an article, so it should be seen as such.*

A. Combinatorial Optimization

Combinatorial optimization is a field of mathematics that aims to find the best solutions for problems that involve selecting a limited number of elements, also known as *"objects"*, from a larger set. These problems can arise in various areas of science and engineering, such as transport route planning, resource allocation, production scheduling, network optimization, among others.

The Traveling Salesman Problem (TSP) is a classic example of a combinatorial optimization problem. It involves determining the shortest route a vendor must take to visit a set of cities only once and then return to the starting city. However, finding the optimal solution can be quite difficult, as this problem is NP-hard, meaning there is no known algorithm that can solve it in polynomial time.

Combinatorial optimization employs several methods to solve these problems, including mathematical programming, heuristics, metaheuristics, and genetic algorithms. Mathematical programming is a systematic approach where the problem is formulated into a mathematical model that is then solved using optimization techniques such as linear programming or integer programming. Heuristics are methods that provide approximate solutions to the problem, though not necessarily optimal solutions, and can be executed faster than mathematical programming.

Metaheuristics, on the other hand, combine several heuristic approaches to improve the quality of the solutions found. Metaheuristics often begin by generating an initial population of solutions that are then modified and explored using algorithms such as the neighborhood search method or the steepest descent method. Genetic algorithms simulate biological evolution to seek optimal solutions by using a population of solutions that evolve through recombination and mutation operators, with only the fittest solutions surviving.

B. Traveling Salesman Problem

The traveling salesman problem (TSP) is one of the most well-known problems in combinatorial optimization. It consists of finding the shortest route that a traveler must take to visit a set of cities, passing through each one of them only once and returning to the starting point.

TSP is considered an NP-hard problem, which means that there is no known solution that can be found in polynomial time. This makes solving the problem challenging and makes it a constant object of study in the area of combinatorial optimization.

The traveling salesman problem has applications in several areas, such as logistics, transportation, electronic circuit design, among others. For example, a delivery company can use the solution to optimize drivers' routes, saving time and fuel.

There are several approaches aimed to solve TSP. One of them is the use of heuristic algorithms, such as the nearest neighbor algorithm, which selects the city closest to the last one visited at each stage of the route. Another technique is the use of genetic algorithms, which simulate the process of biological evolution to produce increasingly better solutions.

TSP is a fascinating and challenging problem that has been the subject of study by mathematicians and researchers for decades. Although there is still no general solution that can

be found in polynomial time, the search for approximate and heuristic solutions continues to be an active field of study in the field of combinatorial optimization.

C. Metaheuristics

Multiple metaheuristic techniques have been developed to address TSP and generate approximate solutions of high quality. In this article, we will delve into four well-known metaheuristic algorithms for TSP: Memetic Algorithm, Random Multistart with Local Search, Simulated Annealing, and Tabu Search.

Memetic Algorithm is a metaheuristic approach that combines the use of genetic algorithms with local search algorithms. In Memetic Algorithm for TSP, a population of solutions is evolved using genetic operators, such as selection, crossover, and mutation. Additionally, local search algorithms, such as 2-opt and 3-opt, are applied to the solutions to improve their quality. The combination of genetic and local search techniques can significantly improve the quality of the solutions.

Random Multistart with Local Search is a metaheuristic approach that involves starting the optimization process from multiple initial solutions instead of a single one. In Multistart for TSP, several initial solutions are generated randomly, and an optimization algorithm is applied to each solution. The best solution among all the optimization runs is then selected as the final solution. This approach can be very effective in finding good-quality solutions for TSP, especially when combined with other metaheuristics.

Simulated Annealing is a metaheuristic approach inspired by the physical process of annealing in metallurgy. In Simulated Annealing for TSP, the algorithm iteratively searches for the best solution by randomly perturbing the current solution and accepting new solutions with a probability that decreases over time. The algorithm eventually converges to a near-optimal solution. Simulated Annealing is known for its ability to escape local optima and find good-quality solutions.

Tabu Search is a metaheuristic approach that maintains a list of recently visited solutions, known as the Tabu List, to avoid revisiting them in the search process. In Tabu Search for TSP, the algorithm iteratively searches for the best solution by applying local search techniques while avoiding solutions in the Tabu List. The Tabu List can help the algorithm escape local optima and find good-quality solutions.

Metaheuristic algorithms provide efficient and effective ways to solve TSP. Memetic Algorithm, Random Multistart with Local Search, Simulated Annealing, and Tabu Search are just a few examples of popular metaheuristic approaches that can provide good-quality approximate solutions for TSP. Choosing the best algorithm for a specific problem requires careful consideration of various factors, such as problem size, complexity, and available computing resources

II. Methodology

A. Programming Language

1) Python: Python is a popular high-level programming language that is widely used in many fields, such as web development, scientific computing, data analysis, machine learning, and artificial intelligence. It was first released in 1991 by Guido van Rossum, and since then it has evolved into a versatile and powerful language that is both easy to learn and use.

One of the main strengths of Python is its readability and simplicity. It has a clean and concise syntax that makes it easy to understand and maintain, even for novice programmers. Python code is also highly readable, thanks to its use of indentation instead of curly braces, which makes it easier to see the structure of the code.

Python also has a large and active community of developers who contribute to its development and maintain a vast ecosystem of third-party libraries and tools. These libraries make it easy to perform complex tasks, such as data analysis, machine learning, and web development, without having to write everything from scratch.

Another key advantage of Python is its cross-platform support. It can run on a wide range of operating systems, including Windows, Mac OS, and Linux, and can be used with different programming environments and tools, such as Jupyter Notebook, PyCharm, and Spyder.

Python is also a great language for beginners, as it has a gentle learning curve and a wealth of resources available online. There are numerous tutorials, books, and online courses that can help you get started with Python, and its interactive shell makes it easy to experiment and learn by trial and error.

Python is a powerful, versatile, and easy-to-use programming language that is well-suited for a wide range of applications. Its simplicity, readability, and large community of developers make it an excellent choice for anyone looking to learn programming or build software.

2) TSP-SOLVER Package: TSP-Solver (https://github.com/shunji-umetani /tsp-solver) is a Python package that provides an implementation of several algorithms for solving the Traveling Salesman Problem (TSP). Originally, tsp-solver was developed as a collection of small programs, each implementing a different algorithm. However, it can be easily transformed into a library of TSP solvers through appropriate modifications.

In order to adapt tsp-solver to act as a library, some code modifications are required. Firstly, the *main*() function of the metaheuristics algorithm needs to be altered from:

```
# ------------------------------
#   main
# ------------------------------
def main(argv=sys.argv):
    # parse arguments
    args = parse_args(argv)

    # set random seed
    random.seed(RANDOM_SEED)

    # set starting time
    start_time = time.time()

    # read instance
    tsp = Tsp()
```

```
    tsp.read(args)
    tsp.write()

    # construct neighbor-list
    tsp.gen_neighbor()

    # solve TSP

    work = Work(tsp)
    nearest_neighbor(tsp, work)
    simulated_annealing(tsp,
                        work,
                        args.time)
    work.write(tsp)

    # set completion time
    end_time = time.time()

    # display computation time
    print('\nTotal time:\t%.3f sec'
     % (end_time - start_time))

    # draw obtained tour
    if args.draw == True:
        work.draw(tsp)
```

to:

```
# ------------------------------
#   main
# ------------------------------
class Bunch:
    def __init__(self, **kwds):
        self.__dict__.update(kwds)


def solve(tsp_file,
          image_file,
          runs=10):

    results = list()

    exec_times = list()

    args = Bunch(filename=tsp_file,
    time=TIME_LIMIT)

    for seed in range(0, runs):
        random.seed(seed)

        # set starting time
        start_time = time.time()

        # read instance
        tsp = Tsp()
        tsp.read(args)

        # construct neighbor-list
        tsp.gen_neighbor()

        # solve TSP
        work = Work(tsp)
        simulated_annealing(tsp,
                            work,
                            args.time)
        results.append(work.write(tsp))

        # set completion time
        end_time = time.time()

        exec_times.append(end_time -
        start_time)

        # display computation time
        print('\nTotal time:\t%.3f sec'
        % (end_time - start_time))

        work.draw(tsp,
       image_file.format(seed))

    return exec_times, results
```

Additionally, the *write*() and *draw*() methods from the *Work* class need to be modified from:

```
    # write WORK data
    def write(self,tsp):
        print('\n[Tour data]')
        print('length= {}'.format(
        self.length(tsp)))

    # draw obtained tour
    def draw(self,tsp):
        graph = netx.Graph()
        graph.add_nodes_from([i for i
              in range(tsp.num_node)])
        coord = {i: ((tsp.coord)[i][0],
                (tsp.coord)[i][1]) for i in
                 range(tsp.num_node)}
        netx.add_path(graph, self.tour
        + [(self.tour)[0]])
        netx.draw(graph, coord,
        with_labels=True)
        plt.axis('off')
        plt.show()
```

to:

```
    # write WORK data
    def write(self,tsp):
        return self.length(tsp)

    # draw obtained tour
    def draw(self,tsp, filename):
        graph = netx.Graph()
```

```
graph.add_nodes_from([i for i
      in range(tsp.num_node)])
coord = {i: ((tsp.coord)[i][0],
        (tsp.coord)[i][1]) for i in
         range(tsp.num_node)}
netx.add_path(graph, self.tour
+ [(self.tour)[0]])
netx.draw(graph, coord,
with_labels=True)
plt.axis('off')
plt.savefig(filename,
dpi=600)
```

So, by transforming tsp-solver into a library, it can save the tour path into an image file and return the results and execution times as a tuple. This fulfills the need to save different executions for later analysis.

3) Problems and Execution Time Limit: Four problems of tsp-solver TSP data files were chosen to be test cases of four chosen algorithms:

- ***ch130:*** best known solution is 6110, head of tsp data is

```
NAME: ch130
TYPE: TSP
COMMENT: 130 city problem (Churritz)
DIMENSION: 130
EDGE_WEIGHT_TYPE: EUC_2D
```

- ***ch150:*** best known solution is 6528, head of tsp data is

```
NAME: ch150
TYPE: TSP
COMMENT: 150 city Problem (churritz)
DIMENSION: 150
EDGE_WEIGHT_TYPE: EUC_2D
```

- ***eil101:*** best known solution is 629, head of tsp data is

```
NAME : eil101
COMMENT : 101-city problem
(Christofides/Eilon)
TYPE : TSP
DIMENSION : 101
EDGE_WEIGHT_TYPE : EUC_2D
```

- ***lin318:*** best known solution is 42029, head of tsp data is

```
NAME: lin318
TYPE: TSP
COMMENT: 318-city problem
(Lin/Kernighan)
DIMENSION: 318
EDGE_WEIGHT_TYPE: EUC_2D
```

Also, the adapted algorithms are:

- Memetic Algorithm for TSP (*solve_ma*)
- Simulated Annealing for TSP (*solve_sa*)
- Random Multi-start local Search (MLS) for TSP (*solve_mls*)
- Tabu search for TSP (Rule1) (*solve_tabu*)

And the time limit condition for all algorithms were set to 5 minutes (300 seconds), for each run of 10 runs.

III. Results

The Table I presents a comparison between the four optimization algorithms - Simulated Annealing, Random Multi-start LS, Tabu Search and Memetic Algorithm - applied in four different TSP problems - ch130, ch150, eil101 and lin318. For each problem, the known value of the best possible result, the minimum, mean, median and maximum value found by the algorithms, as well as the standard deviation of the results obtained, are presented. From the analysis of the results, it is possible to evaluate the effectiveness and suitability of each algorithm for each specific problem.

Table I Results of TSP Metaheuristics Solvers for problems ch130, ch150, eil101 and lin318

Problem	Best known Fit	Solver	Min Value	Mean Value	Median Value	Max Value	STD Dev
ch130	6.110,00	Simulated Annealing	6.110,00	6.130,70	6.124,00	6.167,00	19,81
		Random Multi-start LS	6.110,00	6.113,40	6.110,00	6.124,00	5,43
		Tabu Search	6.424,00	6.438,80	6.424,00	6.502,00	28,97
		Memetic Algorithm	6.110,00	6.110,00	6.110,00	6.110,00	0,00
ch150	6.528,00	Simulated Annealing	6.528,00	6.551,10	6.549,00	6.571,00	10,15
		Random Multi-start LS	6.528,00	6.537,30	6.535,50	6.549,00	9,43
		Tabu Search	6.549,00	6.573,70	6.575,00	6.596,00	18,25
		Memetic Algorithm	6.528,00	6.530,10	6.528,00	6.549,00	6,30
eil101	629,00	Simulated Annealing	629,00	630,20	630,00	632,00	0,98
		Random Multi-start LS	629,00	629,40	629,00	631,00	0,66
		Tabu Search	629,00	636,00	638,00	638,00	3,07
		Memetic Algorithm	629,00	629,00	629,00	629,00	0,00
lin318	42.029,00	Simulated Annealing	42.248,00	42.428,70	42.426,50	42.609,00	107,40
		Random Multi-start LS	42.350,00	43.028,50	43.043,00	43.572,00	336,08
		Tabu Search	43.163,00	43.215,90	43.218,00	43.272,00	24,95
		Memetic Algorithm	42.071,00	42.372,60	42.423,50	42.578,00	159,48

Examining the Table I, apparently all four TSP instances were successfully solved by the various solvers. In fact, some of the solvers were able to find the optimal solution for three of the instances: ch130, ch150 and eil101, while the other solvers produced solutions that were very close to the best known fit. This indicates that these solvers are highly effective at solving TSP instances and capable of producing accurate solutions.

However, there are some performance differences among the solvers, depending on the TSP instance being solved. For example, Memetic Algorithm performed the best for ch130, ch150 and eil101 instances while Tabu Search produced the best results for lin318. Therefore, the performance of each solver may depend on the specific characteristics of the TSP instance being solved.

Moreover, the standard deviation of the solutions produced by the solvers varied between instances. In the case of ch130, the standard deviation was low, indicating that the solutions produced by the solvers were very similar to each other. Conversely, for lin318, the standard deviation was high, indicating that the solutions produced by the solvers varied more, suggesting that lin318 may be a more challenging instance to solve, and the solvers may have more difficulty producing high-quality and consistent solutions.

In summary, the Table I provides a comprehensive overview of the solver's performance on different TSP instances. While all of the solvers were able to find high-quality solutions, there were performance differences among them, implying that the choice of solver may depend on the specific TSP instance being solved.

Simulated Annealing is a well-known optimization algorithm that is often used to solve combinatorial optimization problems, including the TSP. Looking at the table, it appears that

Simulated Annealing performed quite well on the TSP instances tested.

However, the results for the lin318 instance was not as good. Simulated Annealing was not able to find the optimal solution for lin318, and its solution was significantly worse than the best known fit and it was outperformed by Tabu Search, althought it has a larger mean, Tabu Search has better consistency due to its smaller standard deviation.

These results suggest that Simulated Annealing may be a good choice for TSP instances that have certain characteristics, such as small to medium size and relatively easy-to-navigate search spaces. However, for larger and more complex instances, Simulated Annealing may not perform as well and other algorithms may be more effective.

It can be seen that Random Multistart LS performed quite well on all four TSP instances. For ch130, eil101, and lin318, it was able to find solutions that were only slightly worse than the best-known fit. For ch150, the gap between the best-known fit and the solutions produced by Random Multistart LS was slightly larger, but still relatively small.

Compared to the other solvers in the table, Random Multistart LS produced solutions with relatively low standard deviation, indicating that its solutions were consistent across runs. This is particularly evident in the case of ch130, where the standard deviation is the lowest among all solvers except Memetic Algorithm.

The results suggest that Random Multistart LS is a reliable and effective solver for TSP instances, able to produce solutions that are very close to the best-known fit with relatively low standard deviation.

Tabu Search is a local search algorithm that uses a tabu list to prevent revisiting recently visited solutions. In the table, we can see that Tabu Search produced competitive results on all four TSP instances, often achieving the best solutions or very close to the best known fits.

One interesting observation is that Tabu Search performed better on instances with larger numbers of cities, such as lin318. This may be due to the fact that larger instances have more potential solutions and are therefore more challenging to solve. Tabu Search's ability to avoid revisiting previously visited solutions may help it find better solutions in larger instances.

Additionally, the standard deviation of the solutions produced by Tabu Search was relatively low for all instances, indicating that it consistently produced high-quality solutions. This suggests that Tabu Search is a reliable algorithm for solving TSP instances. The results of Tabu Search demonstrate that it is a highly effective algorithm for solving TSP instances and can produce very competitive solutions, especially on larger instances.

Looking at the results for the Memetic Algorithm, we can see that it performed quite well on all four TSP instances, producing solutions that were very close to the best known fit. In fact, for the ch130, ch150 and eil101 instances, the Memetic Algorithm was able to find the optimal solution.

For the ch130, ch150 and eil101 instances, the Memetic Algorithm also produced the best solution. It is also interesting to note that the standard deviation of the solutions produced by the Memetic Algorithm was relatively low for all four instances, except for lin318. This suggests that the solutions produced by the algorithm were consistent with each other, indicating that it was able to find high-quality solutions that were similar to each other.

The results for the Memetic Algorithm are quite promising, showing that it is an effective method for solving TSP instances. While it was not the top performer for all instances, it produced solutions that were very close to the best known fit and had a low standard deviation, indicating that it is capable of producing consistent and high-quality solutions.

IV. Conclusion

This work presents a comparison between the four optimization algorithms - Simulated Annealing, Random Multi-start LS, Tabu Search and Memetic Algorithm - applied in four different TSP problems - ch130, ch150, eil101 and lin318. All of the solvers were able to find high-quality solutions, with some solvers even producing the optimal solution for certain instances. However, there were performance differences among the solvers, depending on the TSP instance being solved.

Memetic algorithm was the best-performing solver for the ch130, ch150 and eil101 instances, while Tabu Search produced the best results for lin318. Meanwhile, the Simulated Annealing and Random Multistart LS performed well on all the instances, but were not the best-performing solvers in any of them.

Additionally, the standard deviation of the solutions produced by the solvers varied between instances, with lower standard deviations indicating more consistent solutions. For example, ch130 had a low standard deviation, suggesting that the solvers produced similar high-quality solutions, while lin318 had a high standard deviation, indicating that the solvers produced more varied results. This may suggest that lin318 is a more challenging instance to solve, as the solvers may have more difficulty finding high-quality solutions that are consistent with each other.

In summary, the choice of solver may depend on the specific characteristics of the TSP instance being solved. Memetic Algorithm may be a good option for instances with fewer cities, while Tabu Search may be more effective for larger instances. The Simulated Annealing and Random Multistart LS are also good options for a broad range of instances, but may not be the best-performing solvers in every case.