Combinatorial and Engineering Optimization with Metaheuristic Algorithms

Tycho Bear
Department of Computer Science
Golisano College of Computing and Information Sciences
Rochester Institute of Technology
Rochester, NY 14623
tsb3342@rit.edu

***Abstract*—Many interesting combinatorial optimization problems are computationally intractable for exact algorithms. Also, some engineering problems are difficult to solve analytically. In these cases, metaheuristic algorithms may be used to find high-quality solutions in an efficient manner. This work applies metaheuristic algorithms to solve NP-hard combinatorial optimization problems, as well as engineering optimization problems. Simulated annealing (SA), genetic algorithms (GA), and particle swarm optimization (PSO) are implemented and evaluated. SA iteratively improves a given solution, occasionally accepting worse solutions to escape local optima. GA maintains a population of solutions that evolve through crossover and mutation operators, mimicking real populations in nature. PSO simulates "swarm intelligence" behavior to find good solutions. These algorithms are tested on benchmark instances of the traveling salesman problem (TSP), bin packing problem (BPP), and pressure vessel design problem. Results show that properly tuned configurations achieve solutions within 1-10% of optimal or best known results. For the pressure vessel problem, solutions consistent with the best in the literature are found. These results demonstrate the effectiveness of metaheuristics for finding high-quality solutions to optimization problems.**

I. Introduction

Combinatorial optimization is a branch of mathematical optimization that involves finding solutions from a finite set of discrete alternatives. Unlike continuous optimization, each solution must be discrete and countable. Metaheuristic algorithms are high-level frameworks that are designed to find good solutions to optimization problems, using a degree of randomness. While optimal solutions are not guaranteed, high-quality solutions can be found in relatively short amounts of time compared to exact algorithms.

Many interesting combinatorial optimization problems, such as the traveling salesman problem (TSP) and bin packing problem (BPP), are NP-hard [1] [2]. The TSP involves finding the shortest path through a given set of cities, where each city is visited exactly once. The bin packing problem deals with placing weighted objects into bins that have a weight limit, and finding the configuration that uses the smallest number of bins. Solutions to the TSP with a very small amount of cities may be found through exhaustive search, but this problem is generally intractable. Solutions to combinatorial optimization problems have many real-world applications. For example, a shortest path in the TSP may correspond to an optimal vehicle transportation or delivery route. There are even applications in fields such as astronomy, where astronomers would like to compute an efficient way to move the telescope between points of interest in the sky. Obtaining solutions to these NP-hard problems would be very beneficial in many areas.

For most combinatorial and engineering optimization problems, the true best solution is not needed. Usually, a very good solution will suffice. For example, with the TSP, there are many solutions within a narrow margin of the shortest path. In most applications, any one of these short paths will be useful. This is a scenario where metaheuristic algorithms can be highly effective. They can be used to efficiently find "good enough" solutions to optimization problems in an iterative manner, without the need for exhaustive search.

In this work, we apply several metaheuristic algorithms to various optimization problems. The first of these algorithms is simulated annealing (SA) [3]. It begins with an initial solution, and iteratively improves it through local modifications. In other words, at each iteration, a neighboring solution is generated, and then compared to the current solution. Better solutions are always accepted, and worse solutions may be accepted, based on an exponential probability. This allows the algorithm to escape local optima.

Genetic algorithms (GA), in contrast, maintain a population of solutions, as opposed to an individual solution [4]. In each iteration, or generation, new solutions are generated through crossover operations. For this, two parent solutions are selected based on some fitness metric, and combined to produce a child solution. Solutions may also undergo random mutation with a low probability to encourage diversity. Solution quality improves across generations, eventually converging on a very good solution to the problem.

Particle swarm optimization (PSO) is another population-based algorithm. In this algorithm, particle movement is carried out through vector perturbations based on attraction to other particles. This results in a "swarm intelligence" behavior where individual particles are drawn towards the best known solutions. Particle movement slows down over time, resulting in convergence around a strong solution.

Preliminary results indicate that metaheuristic algorithms perform quite well on these optimization problems. Appropriate combinations of parameters result in solutions within

3-10% of the optimal value on benchmark instances of the TSP. For the BPP, configurations near the theoretical lowest number of bins can often be found. For pressure vessel design, we find solutions consistent with the best in the literature. Overall, these results are very promising, and showcase the ability of metaheuristic algorithms to find good solutions to combinatorial optimization problems.

II. Background and Motivation

A. Overview of Terms

1) Combinatorial Optimization: This type of mathematical optimization focuses on finding the best solution from a finite set of discrete options. Each potential solution represents a specific configuration or arrangement, and the goal is to identify the configuration that optimizes a given criterion. Because these problems are discrete, traditional calculus-based optimization techniques do not work [5]. Additionally, many such problems are NP-hard, so efficient algorithms to solve them are not known to exist. Examples of combinatorial optimization problems include the traveling salesman problem and bin packing problem, which are explained later in this section.

2) Metaheuristic Algorithms: These are high-level frameworks that use guided randomness to find good solutions to optimization problems. Metaheuristic algorithms are also problem-independent, which means they can be applied to many problems without having problem-specific knowledge. Examples of these algorithms include simulated annealing or genetic algorithms, both explained in Section 3.

Metaheuristic algorithms also allow for navigating search spaces that are strongly multimodal and highly nonlinear. They start with initial candidate solutions, and iteratively improve them through various operations. For many optimization problems, such as the traveling salesman problem, any near-optimal solution will suffice. This makes metaheuristic algorithms a good choice for solving such problems, especially when solving them exactly is computationally infeasible.

3) Objective Function: This is used to evaluate how good candidate solutions are. In most cases, it returns a numerical score that corresponds to solution quality. Additionally, the implementation of the objective function will be different for each problem. For example, thinking again about the traveling salesman problem, the objective function may compute the tour distance of a given configuration. For the bin packing problem, this function could calculate the total number of bins used. The design of the objective function is extremely important for algorithm performance.

4) Exploration and Exploitation: These two concepts represent the tradeoff that all metaheuristic algorithms must balance. Exploration means to generate diverse solutions so as to explore the search space on a global scale [5]. This encourages the algorithm to look at different areas in the search space and avoid becoming trapped in local optima. On the other hand, exploitation means to focus the search in a local region, and essentially hone in on a smaller area. In short, it means to examine promising regions of the search space.

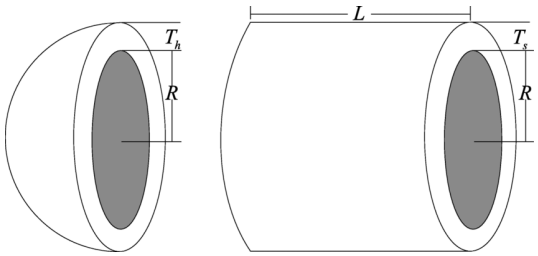


Fig. 1: A visual representation of the design variables in the pressure vessel problem. The goal is to find the combination of these variables that minimizes the total cost of building the vessel, while also following constraints. Image credit: [6].

Balancing this tradeoff is critical for algorithm performance. Too much exploration may prevent convergence on a good solution, or make the algorithm run for an excessively long time. Too much exploitation may prevent the algorithm from fully exploring the search space, leading to premature convergence to local optima.

B. Overview of Problems

1) Traveling Salesman Problem: The TSP is a classic NP-hard combinatorial optimization problem. Given a set of cities and the distances between each pair of cities, the objective is to find the shortest route visiting each city exactly once, and also returning to the starting city [3]. This problem models a salesperson who wants to visit all cities on the route in the most efficient manner possible. The difficulty in this problem arises from the factorial growth in the number of possible tours, as the number of cities increases.

2) Bin Packing Problem: The BPP is another NP-hard optimization problem with many practical applications. Given a set of items, each with a specific weight, and a collection of bins with a fixed weight capacity, the goal is to pack all of the items into the minimum number of bins, without exceeding any bin's weight limit [2]. Some real-world applications include loading cargo containers, or filling trucks before starting a delivery route. The challenge lies in finding the arrangement that makes the most efficient use of all the bins, and wastes as little space as possible. Like in the TSP, the size of the solution space grows exponentially as the number of items increases.

3) Pressure Vessel Design Problem: This is not a combinatorial optimization problem, but instead an engineering optimization benchmark where metaheuristic algorithms also shine. This problem involves designing a cylindrical pressure vessel capped at both ends by hemispherical heads. The objective is to minimize the total cost of the material, forming, and welding, while also meeting pressure and volume requirements. In most cases ([7], [8], [9]), there are four design variables: thickness of the head (d_1), thickness of the cylindrical shell (d_2), the inner radius (r), and the length of the cylindrical section (L). Additionally, the variables d_1 and

d_2 must be integer multiples of 0.0625 inches, while r and L are continuous. Formally, this problem may be written as:

Minimize:

$$f(\boldsymbol{x}) = 0.6224 d_1 r L + 1.7781 d_2 r^2 + 3.1661 d_1^2 L + 19.84 d_1^2 r \tag{1}$$

subject to these constraints:

$$g_1(\boldsymbol{x}) = -d_1 + 0.0193 r \leq 0$$

$$g_2(\boldsymbol{x}) = -d_2 + 0.00954 r \leq 0$$

$$g_3(\boldsymbol{x}) = -\pi r^2 L - \frac{4\pi}{3} r^3 + 1296000 \leq 0$$

$$g_4(\boldsymbol{x}) = L - 240 \leq 0.$$

Additionally, the values are bounded as follows:

$$0.0625 \leq d_1, d_2 \leq 99 \times 0.0625$$

$$10.0 \leq r, \qquad L \leq 200.0.$$

A visual depiction of this problem may be found in Figure 1.

C. Motivation and Our Approach

The three problems examined here have many real-world applications in different fields. For example, solutions to the TSP may correspond to optimized transportation or delivery routes. Solutions to the bin packing problem could help load vehicles with weight capacity constraints, or even help with resource allocation in cloud computing [10]. Pressure vessels are quite common in everyday life, and optimization of their design has several applications in various engineering fields, such as aerospace components [11].

For all of these problems, finding solutions via exhaustive search requires lots of computational effort, and is generally infeasible. In this work, we aim to find high-quality, near-optimal solutions to these problems that are still useful in practice. For example, a delivery route within a narrow margin of the shortest path is still quite useful. These solutions will be found with the three different metaheuristic algorithms mentioned previously, which are described in further detail in the next section.

All of these problems are formulated as minimization problems. However, it is worth noting that they can be converted to maximization problems by multiplying the objective function by -1. Each algorithm will be tested on each problem, and the results are displayed in Section 4.

III. Overview of Algorithms

This section details the individual metaheuristic algorithms used to solve the various optimization problems. In this work, three algorithms are used: simulated annealing, a genetic algorithm, and particle swarm optimization.

A. Simulated Annealing

Simulated annealing is a metaheuristic algorithm inspired by the cooling process in metallurgy, where metals are slowly cooled to a minimum energy state [12]. Starting with a random initial solution and high temperature, the algorithm iteratively generates new solutions by making random modifications to the current solution. That is, it generates random neighboring solutions in the search space. If the new solution is better than the current solution, then it is always accepted, and becomes the current solution. If the new solution is worse, it may still be accepted, based on a probability p, given by

$$p = \exp\left(\frac{-\Delta f}{T}\right),$$

where Δf represents the change in objective function value, and T is the current temperature [5]. This probabilistic acceptance allows the algorithm to escape local optima while the temperature is still high.

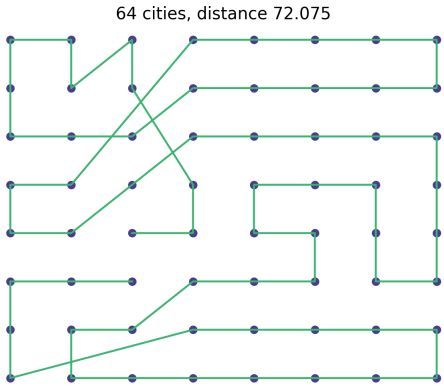
With every iteration of the algorithm, the temperature T decreases according to some cooling schedule. In the literature, geometric cooling schedules are commonly used ([3], [13]), and we use one here. Essentially, at each iteration, the current temperature is multiplied by a constant less than 1, but very close to 1, such as 0.99 or 0.999. As the temperature decreases, so does the probability of accepting worse solutions. This allows the algorithm to explore the search space early on, and then converge to a higher-quality solution as time passes, thus balancing exploration and exploitation.

It should be noted that, as with most algorithms, parameter tuning is vital for the performance of this algorithm. Cooling too quickly may trap the algorithm in local optima, and cooling too slowly can lead to excessive computation time without much improvement in solution quality.

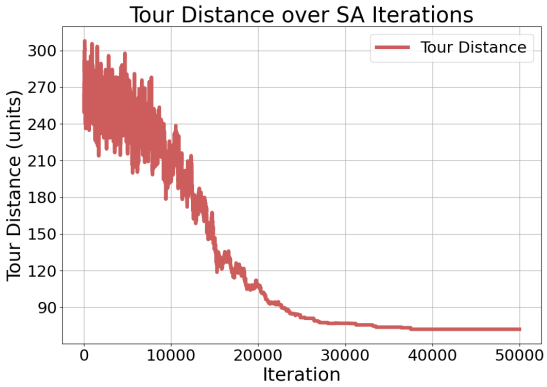
B. Genetic Algorithm

Genetic algorithms are population-based metaheuristic algorithms inspired by natural selection [14]. Unlike SA, which works with a single solution, GAs have a population of solutions, where individuals evolve over multiple generations. Population size can range from a few dozen to a few hundred [15]. Each generation creates a new population through three main operations: elitism, crossover with selection, and mutation [5].

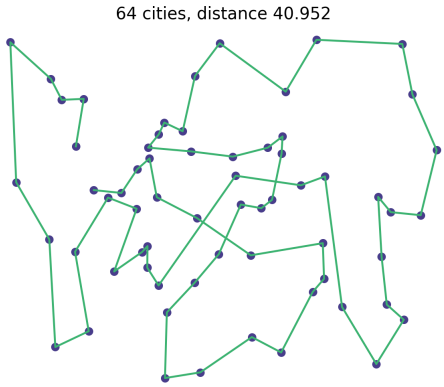
Elitism makes sure the best solutions in a given generation survive to the next generation. This is done by directly copying over a small percentage (5-10%) of the most fit individuals, without modification. Most of the new population ($\sim$75%) is generated with crossover operations. Here, two parent solutions are selected via tournament selection. Tournament selection randomly chooses a small handful of individuals (3-5), and returns the most fit individual. After this, the selected parent solutions are combined through crossover to produce a child solution that inherits characteristics from both parents. Next comes mutation, where each individual has a small probability (5-10%) of being changed randomly. This helps to ensure diversity in the population.



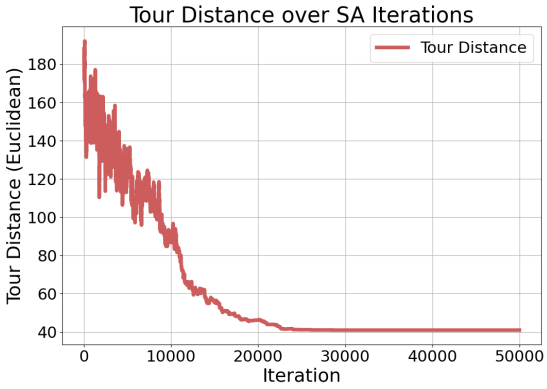
(a) The final tour obtained after running simulated annealing on a square grid of 64 cities. The shortest path is 64 units long.



(b) Tour distance over consecutive iterations of simulated annealing on the grid from Figure 2a.



(c) The final tour obtained after running simulated annealing on a random arrangement of 64 cities.



(d) Tour distance over consecutive iterations of simulated annealing on the cities shown in Figure 2c.

Fig. 2: Simulated annealing results on both grid and random instances of the TSP. Each row shows the final tour and its convergence plot.

Finally, the remaining population slots are filled with randomly generated solutions. As this whole process is repeated over many generations, the population should converge on a good solution to the problem being solved, assuming parameters are tuned appropriately.

C. Particle Swarm Optimization

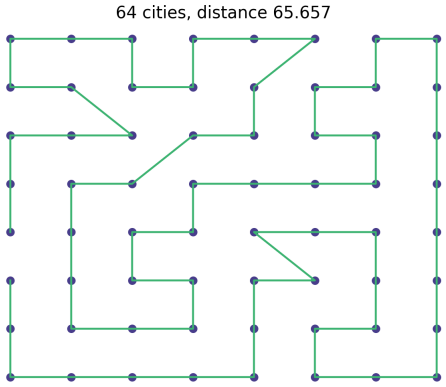
Particle swarm optimization is another population-based algorithm inspired by swarm intelligence behaviors [16]. PSO is similar to a genetic algorithm in that it maintains a population of solutions (as "particles"), but the operations it carries out are different. Each particle moves based on attraction to two positions: the current global best in the swarm, and the best location the particle has seen so far. Additionally, random vectors influence the direction and distance of movement [5].

One drawback of the standard PSO is that it may converge very quickly. This can lead to particles getting trapped in local optima before adequately exploring the search space. To address this, we modify the standard algorithm by adding mutation after each iteration, much like how it's done in the genetic algorithm. When additional randomness is introduced, particles are able to escape local optima, and solution diversity is increased. This helps increase exploration and results in better solutions.

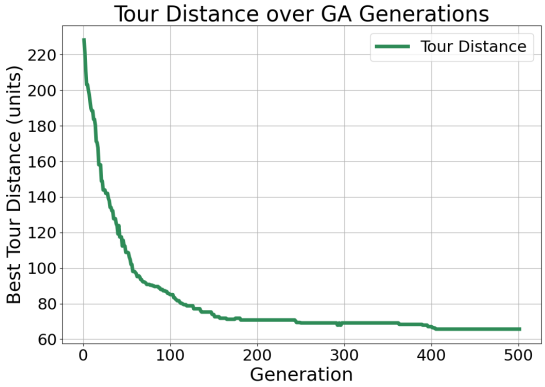
D. Overview of Operations

Each of these algorithms carries out one or more operations to generate new candidate solutions. Such operations include neighbor generation in simulated annealing, crossover in genetic algorithms, and velocity-based movement in particle swarm optimization. Implementation of these operations will be different for each problem being solved.

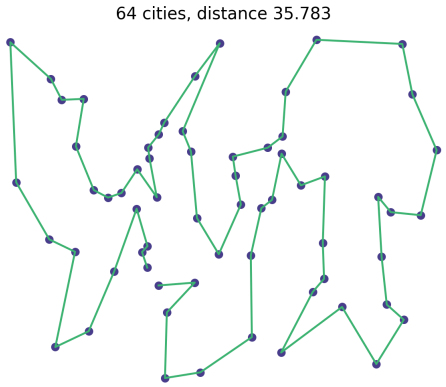
For the TSP, neighbor generation is done by swapping two cities' positions in the tour. The crossover method used is "ordered crossover," where a random part of the tour from one parent is chosen, and remaining cities are filled from the second parent. Mutation can involve swapping two cities' positions, similar to neighbor generation in simulated annealing. It may also include reversing or scrambling a random part of the tour. PSO uses the concept of swap sequences, as described in [17].



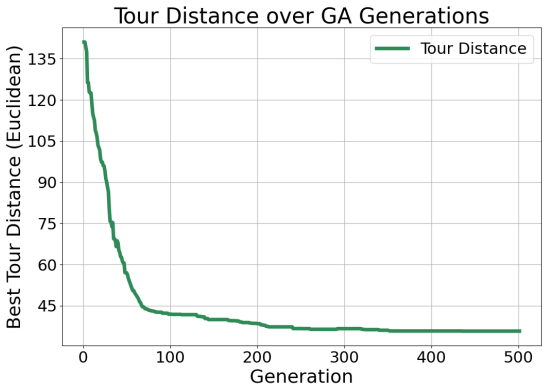
(a) The final tour obtained after running the genetic algorithm on a square grid of 64 cities. The algorithm gets very close to the shortest path of 64 units.



(b) Tour distance of the best individual in the population over each generation. The genetic algorithm is being run on the grid of cities in Figure 3a.



(c) The final tour obtained after running the genetic algorithm on the same random arrangement of cities as in Figure 2c.



(d) Tour distance of the best individual in the population over each generation for the cities in Figure 3c.

Fig. 3: Genetic algorithm results on both grid and random instances of the TSP. Each row shows the best tour in the final population, as well as the algorithm's convergence plot.

For bin packing, neighbor generation is also done by swapping the positions of two items. Crossover is done by randomly selecting half of the items from the first parent, and filling the remaining items from the second parent. Mutation is done in the same way as in the TSP. For PSO, a similar version of swap sequences is also used.

For the pressure vessel problem, SA uses vector perturbation to generate neighbors. The intensity is controlled through parameters. Crossover in the GA uses a combination of averaging and rounding for discrete values, and random perturbations for continuous values. PSO uses the normal velocity behavior as in [5], but discrete values are rounded after particle movement.

E. Objective Functions

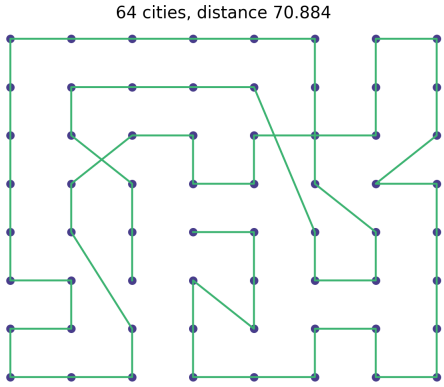
The fitness of the solutions for each problem are also evaluated differently. For the TSP, fitness is calculated by summing the Euclidean distance from each individual city to the next, and then from the ending city back to the starting city. For bin packing, the first-fit packing method is used on the internal list of items. Each item is checked against the existing bins, and is placed in the first one where it fits. If it doesn't fit anywhere, a new bin is created. A different ordering of the items may result in a different number of bins used. For pressure vessel design, the total cost is calculated by Equation (1).

IV. Evaluation

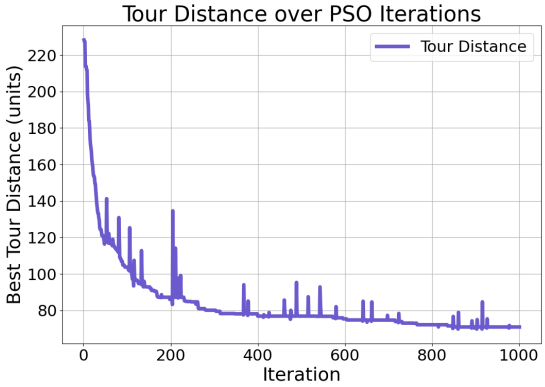
1) Experimental Setup: All algorithms were run on a 64-bit Windows 10 computer with an AMD Ryzen 9 3900X 3.79 GHz 12-core processor and 32 GB memory. Code was run from inside JetBrains' PyCharm IDE.

A. Traveling Salesman Problem Results

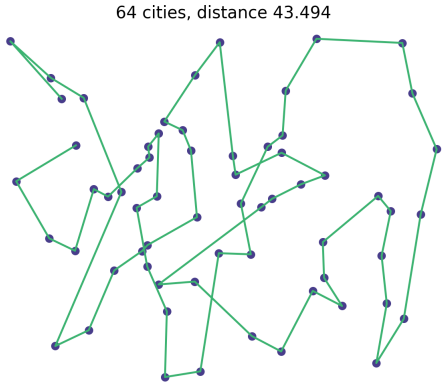
1) Simulated Annealing: First, the results from simulated annealing are examined. Figure 2a shows the final grid obtained after running simulated annealing on a benchmark instance of the traveling salesman problem. The cities are arranged in a square 8×8 grid, where each city is 1 unit apart. Because of this, we know the optimal tour distance is 64, and thus can quantitatively evaluate the algorithm's performance.



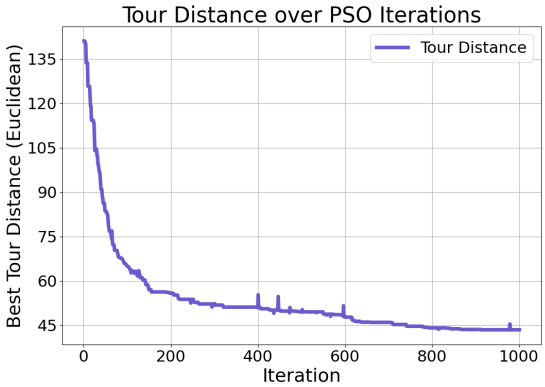
(a) The final tour obtained after running particle swarm optimization on a square grid of 64 cities.



(b) Tour distance of the best individual in the population over each iteration of PSO. The algorithm is being run on the cities in Figure 4a.



(c) The final tour obtained after running particle swarm optimization on the same random arrangement of cities as in Figure 2c.



(d) Tour distance of the best individual in the population over each iteration of PSO. The algorithm is being run on the cities in Figure 4c.

Fig. 4: Particle swarm optimization results on both grid and random instances of the TSP. Each row shows the best tour in the final population, as well as the algorithm's convergence plot.

The algorithm found a tour of distance 72.075 units, which falls within 12.7% of the true optimum. The algorithm was run for 50,000 iterations with an initial temperature of 25 and a cooling rate of 0.9998, and took 22.1 seconds to run.

Figure 2b displays the current tour distance stored by the simulated annealing algorithm over consecutive iterations. The tour distance varies at first, due to the high temperature, but stabilizes as the algorithm progresses.

Figures 2c and 2d tell a similar story. This time, the cities are positioned randomly within the latitude and longitude coordinate ranges of Washington state. For this configuration, the optimal tour distance is not known, but the visual representation of the solution can be used to roughly judge its quality. The algorithm found a tour distance of 40.952 units. However, as we will see later, this is not the optimum. Like before, the algorithm was run for 50,000 iterations, this time with an initial temperature of 5 and a cooling rate of 0.9998, and took 22.0 seconds to run.

Because the initial temperature was set lower this time, the algorithm converges slightly faster. This can be seen by the shape of the plot in Figure 2d, and how it immediately moves downward instead of horizontally.

2) Genetic Algorithm: Next, we examine the results from running the genetic algorithm on this problem. Figure 3a shows the best individual in the population after running the genetic algorithm on the benchmark grid. Figure 3b plots the tour distance of the best individual in the population at each generation. As we can see, this solution looks much better than the one shown in Figure 2a. The genetic algorithm found a tour of distance 65.657 units, which is within 2.6% of the global optimum of 64 units. The algorithm was run for 500 generations, with a population size of 600, elitism percentage of 0.06, crossover percentage of 0.75, mutation rate of 0.1, and tournament size of 4. The algorithm took 304.7 seconds to run, which is significantly slower than simulated annealing, as can be seen in Table II.

Figures 3c and 3d show the results from applying a genetic algorithm to the same random arrangement of cities from Figure 2c. This time, the algorithm finds a tour distance of 35.783, which is much better than the result obtained by

simulated annealing. The algorithm was run with the same configuration as before, except with an elitism percentage of 0.05 instead of 0.06. The algorithm took 305.5 seconds to run. These results indicate that genetic algorithms are able to find better solutions to the TSP than simulated annealing, at the cost of more computational effort, which is consistent with previous work [13].

3) Particle Swarm Optimization: Finally, we examine the results from running the modified particle swarm optimization algorithm on both instances of the TSP. Much like with the genetic algorithm, Figure 4a shows the solution of the best particle in the population after optimizing the benchmark grid. Figure 4b displays the tour distance of the best particle at each iteration. PSO found a slightly better result than SA on this problem, with a solution distance of 70.844, but took 719.0 seconds to run, which is much slower than SA or GA. The algorithm was run for 1000 iterations with a population size of 500, α of 0.4, β of 0.4, and a mutation rate of 0.10.

Similarly, Figures 4c and 4d show the results for the random city configuration. In this instance, PSO is outperformed by both SA and GA, finding a solution with distance 43.494. The algorithm was run with the same settings as before, except with α and β set to 0.2, and took 777.9 seconds to complete. This indicates that while PSO may find competitive solutions to this problem, it does so at a much higher computational cost.

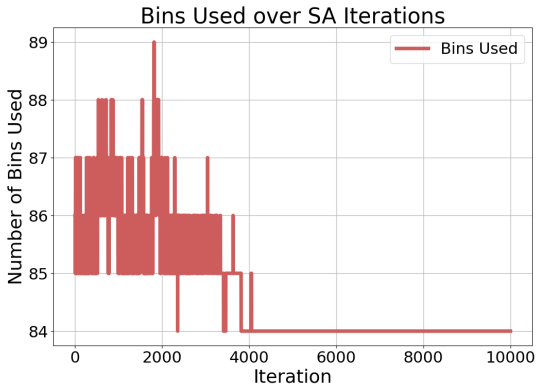
B. Bin Packing Problem Results

Next, we show the results for the bin packing problem. For this problem, there were 200 items, each with a weight between 1 and 50. Item weights were generated using a beta distribution with $\alpha = 2$ and $\beta = 3$, which slightly favors smaller weights. The capacity of each bin was 50. First are the results found by simulated annealing.

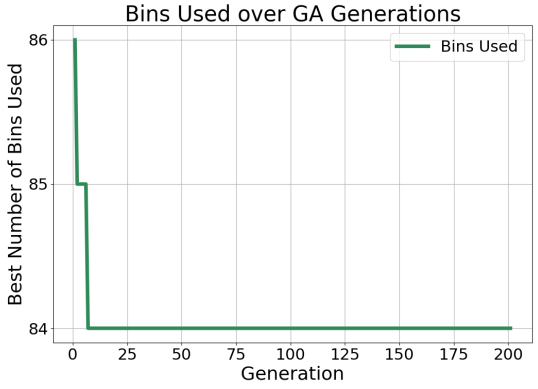
1) Simulated Annealing: The algorithm was run for 10,000 iterations, with an initial temperature of 10 and a cooling rate of 0.999, and it took 10.5 seconds to complete. A configuration using 84 total bins to store all of the items was found, and the solution fitness plot may be seen in Figure 5a.

Due to the number of bins used, it is not possible to show a visualization of every single one. Instead, Figure 6 shows a subset containing the first 15 bins. Each color block is an individual item, and block size corresponds to item weight. Items are packed with alternating shades of dark and light for better visibility. Blank space indicates the amount of capacity available in a given bin after all items are packed. Most bins achieve upwards of 95% space efficiency.

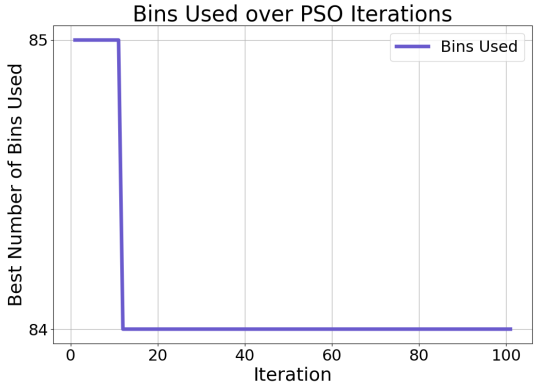
This problem cannot be benchmarked exactly like the TSP, however, it is possible to use the bin capacity and the sum of the item weights to calculate the theoretical minimum number of bins needed to pack every item. It is important to note that this minimum may be impossible to achieve in practice, based on the distribution of item weights. In this case, the theoretical minimum number of bins is 82, and the 84 bins used in the final solution found by the algorithm is within 2.5% of this value.



(a) Number of bins used over each iteration of simulated annealing. A subset of the bins in the final configuration is shown in Figure 6. The final configuration uses 84 bins in total.



(b) Number of bins used by the best configuration in the population over each generation of the genetic algorithm. The configuration using 84 total bins is found quickly.



(c) Number of bins used by the best configuration in the population over each iteration of particle swarm optimization.

Fig. 5: Convergence plots for all three algorithms on the bin packing problem. Simulated annealing is shown in red, the genetic algorithm in green, and particle swarm optimization in purple. All three algorithms find a configuration using 84 bins to store the items.

Fig. 6: A visualization of how the first 15 bins are packed after running simulated annealing. Alternating colors correspond to individual items, and blank space represents available capacity.

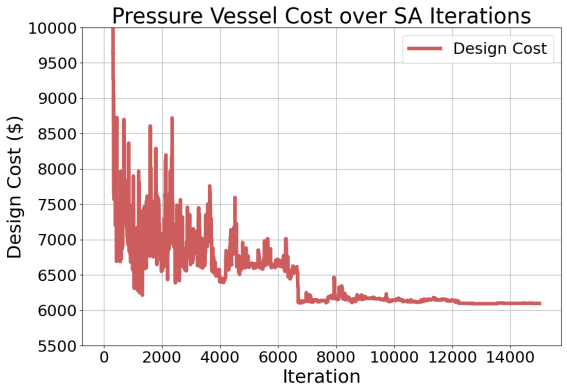
2) Genetic Algorithm: Here, the results from running the genetic algorithm on this problem are discussed. The number of items and range of weights is the same as before. The genetic algorithm was run for 200 generations, with a population size of 150, elitism percentage of 0.05, crossover percentage of 0.75, mutation rate of 0.1, and tournament size of 4. The algorithm took 33.4 seconds to run, which, as before, is much slower than simulated annealing.

A visualization of the first 15 bins in the solution would look very similar to Figure 6. The solution uses 84 bins total, which matches the result obtained by simulated annealing. Figure 5b shows the number of bins used by the best individual solution in the population over each generation. Unlike simulated annealing, the genetic algorithm very quickly converges on an optimum.

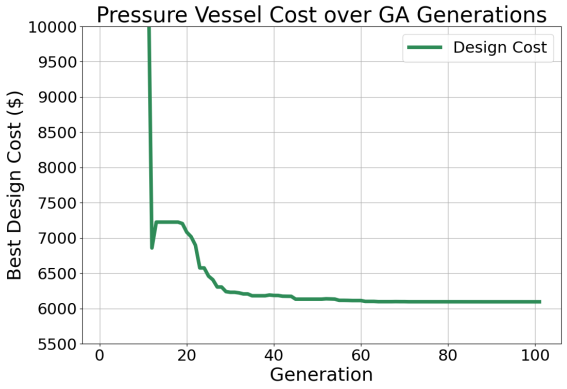
3) Particle Swarm Optimization: The results for PSO on the bin packing problem are very similar to GA's results. The algorithm was run for 100 iterations, with a population size of 100, α and β of 0.4, and no mutation at all. Similarly, PSO found a solution that uses 84 bins, and the algorithm's convergence plot may be seen in Figure 5c. However, the algorithm only took 13.7 seconds to run, which is several times faster than GA. This may be due in part to the different parameters, with a lower population size in particular improving performance.

C. Pressure Vessel Design Results

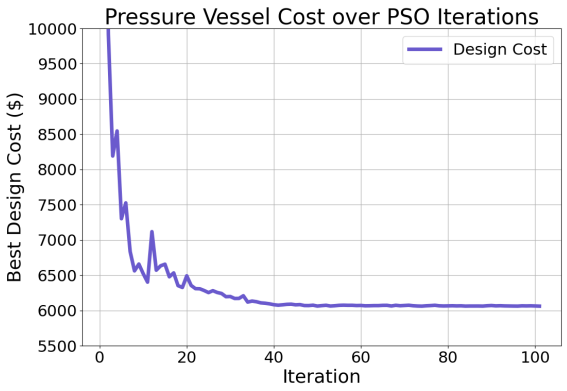
1) Simulated Annealing: The simulated annealing algorithm was run for 15,000 iterations, with an initial temperature of 400 and a cooling rate of 0.9997. A thickness step size of 2×0.0625, radius step size of 1, and length step size of 2 were used, and the algorithm took 1.3 seconds to complete. The final cost found was \$6096.407, which is within 1% of the best known solution [8]. The fitness plot may be found in Figure 7a. The cost is initially extremely high due to the way constraints are handled. The design variables d_1, d_2, r, and L in the final solution are as follows:



(a) Design cost over repeated iterations of simulated annealing. The final cost is \$6096.407.



(b) Design cost of the best solution in the population over each generation of the genetic algorithm. The final cost is \$6094.163.



(c) Design cost of the best solution in the population over each iteration of particle swarm optimization. The final cost is \$6060.638.

Fig. 7: Convergence plots for all three algorithms on the pressure vessel design problem. As in Figure 7, SA is shown in red, GA in green, and PSO in purple. Each algorithm finds a very good solution, but our modified PSO finds the best solution.

TABLE I: Performance of optimization algorithms across each problem. Each cell shows the solution found by the algorithm. For TSP instances, this is measured in distance units, and for bin packing, the number of bins is shown. Lower values are better in all cases.

Algorithm	TSP (Grid)	TSP (Random)	BPP	PVD
SA	72.075	40.952	84	$6096.407
GA	**65.657**	**35.783**	84	$6094.163
PSO	70.884	43.494	84	**$6060.638**

TABLE II: Runtime of each algorithm across each problem instance, measured in seconds. Simulated annealing is by far the fastest of the three algorithms. The others are likely slower because they are population-based. Lower values are better in all cases.

Algorithm	TSP (Grid)	TSP (Random)	BPP	PVD
SA	**22.1s**	**22.0s**	**10.5s**	**1.3s**
GA	304.7s	305.5s	33.4s	1.6s
PSO	719.0s	777.9s	13.7s	2.1s

$$\boldsymbol{x}_* \approx (0.875, 0.4375, 45.3337, 140.4895).$$

2) Genetic Algorithm: We now examine the results from running the genetic algorithm on this problem. This time, the algorithm was run for 100 generations, with a population size of 300, elitism percentage of 0.05, crossover percentage of 0.75, mutation rate of 0.15, and tournament size of 4. Additionally, the algorithm used a thickness step size of 2×0.0625, radius step size of 2, and length step size of 6. The algorithm took 1.6 seconds to complete.

The final cost was very similar to that found by simulated annealing, at $6094.163. The fitness plot can be seen in Figure 7b. The design variables in the final solution are very similar to those found by simulated annealing, and they are shown here:

$$\boldsymbol{x}_* \approx (0.875, 0.4375, 45.3050, 140.5784).$$

3) Particle Swarm Optimization: Finally, we show the results for particle swarm optimization. This time, the algorithm came within $1 of the best known solution in the literature [8], at $6060.638. The algorithm was run for 100 iterations, with a population size of 300, α of 1.3, β of 1.2, and inertia weight of 0.7. Because this is a modified version of the algorithm using mutation, it also used a thickness step size of 2×0.0625, radius step size of 1.5, length step size of 3, and mutation rate of 0.15.

The fitness plot is shown in Figure 7c. The design variables are as follows:

$$\boldsymbol{x}_* \approx (0.8125, 0.4375, 42.0981, 176.6789).$$

These are quite similar to the values found by Yang [9]. Overall, particle swarm optimization appears to perform the best on this problem.

4) Summary: A compilation of the results obtained from running each algorithm on each problem may be found in Table I. Running times are displayed in Table II. It can be seen that the GA performs the best on the TSP, and all three algorithms perform similarly on the pressure vessel problem. However, in terms of runtime, simulated annealing is an order of magnitude faster than the other algorithms, with only a minor decrease in performance.

D. Notes on Parameters

The performance of each algorithm highly depends on its configuration of parameters. Some examples of parameters include the cooling rate in simulated annealing, or the percentage of the population generated via crossover in a genetic algorithm. Tuning all of these parameters is critical for algorithm performance. For example, too much randomness in simulated annealing from an excessively high geometric cooling schedule may slow down convergence. Additionally, generating too much of the next generation's population through crossover in a genetic algorithm can reduce solution diversity and lead to premature convergence.

For each algorithm and problem combination, we test several different parameter configurations and select the one that gives the best results. In general, the values described by Yang, as well as in previous work ([5], [3], [7]) are used as a starting point, and then adjusted as needed. In the cases where no such starting values existed, we used trial and error. This step is necessary for achieving strong performance with metaheuristic algorithms on different problems.

V. Related Work

Simulated annealing and genetic algorithms have both been used to obtain very good solutions to the TSP. Bookstaber provides benchmark results for simulated annealing on the TSP [3], and Zhan et al. apply simulated annealing with a modified temperature schedule to obtain very good results [18]. For genetic algorithms, Braun shows their capacity to solve larger instances of the TSP in just a few minutes [15]. Several surveys have also investigated genetic algorithms' design and performance for this problem, including crossover operators, mutation techniques, and solution representations [4] [19].

Solutions to the bin packing problem have also been found with metaheuristic algorithms. For simulated annealing approaches, Fleszar has shown good results using various heuristics and hybrid algorithms [2]. Additionally, Falkenauer introduced a genetic algorithm specifically designed for grouping problems such as bin packing, with efficient crossover and mutation operators [20].

The pressure vessel design problem has also been solved with metaheuristic algorithms. Coello used a genetic algorithm to find solutions on par with those in the literature at the time [7]. Cagnina applied a constrained particle swarm optimizer, which found higher quality solutions than had previously

been found [8]. More recently, Yang and Deb investigated a recently developed eagle strategy algorithm, and benchmarked its performance on this problem [9].

VI. Conclusion

This work demonstrates the effectiveness of metaheuristic algorithms for solving combinatorial and engineering optimization problems. Simulated annealing, a genetic algorithm, and a modified version of particle swarm optimization all found high-quality solutions to the traveling salesman problem, bin packing problem, and pressure vessel design problem. In most cases, solutions found were within 10% of the optimal or best known solutions. The quality of the solutions found here are consistent with those found in the literature. This reinforces the idea that metaheuristics can produce good solutions to NP-hard combinatorial optimization problems and engineering problems.

A. Future Work

Future work may include exploring simulated annealing with a list-based temperature schedule as in [18]. Additionally, a two-stage eagle strategy approach as evaluated by Yang may also yield good solutions with a fraction of the computational cost [9]. For bin packing, alternative heuristics such as minimal bin slack could be explored [2]. Finally, other forms of engineering problems should also be examined.

B. Implementation

The algorithms and problems in this work were implemented in Python 3.10.0. All code may be found in [21].

References

[1] C. H. Papadimitriou and K. Steiglitz, "Some complexity results for the traveling salesman problem," in *Proceedings of the eighth annual ACM symposium on theory of computing*, 1976, pp. 1–9.

[2] K. Fleszar and K. S. Hindi, "New heuristics for one-dimensional bin-packing," *Computers Operations Research*, vol. 29, no. 7, pp. 821–839, 2002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0305054800000824

[3] D. Bookstaber, "Simulated annealing for traveling salesman problem," *SAREPORT. nb*, 1997.

[4] P. Larranaga, C. M. H. Kuijpers, R. H. Murga, I. Inza, and S. Dizdarevic, "Genetic algorithms for the travelling salesman problem: A review of representations and operators," *Artificial intelligence review*, vol. 13, pp. 129–170, 1999.

[5] X.-S. Yang, "Nature-inspired optimization algorithms," in *Nature-Inspired Optimization Algorithms*. Oxford: Elsevier, 2014, p. i. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780124167438000166

[6] X. Zhao, Y. Fang, L. Liu, J. Li, and M. Xu, "An improved moth-flame optimization algorithm with orthogonal opposition-based learning and modified position updating mechanism of moths for global optimization problems," *Applied Intelligence*, vol. 50, 12 2020.

[7] C. A. Coello Coello, "Use of a self-adaptive penalty approach for engineering optimization problems," *Computers in Industry*, vol. 41, no. 2, pp. 113–127, 2000. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0166361599000469

[8] L. C. Cagnina, S. C. Esquivel, and C. A. C. Coello, "Solving engineering optimization problems with the simple constrained particle swarm optimizer," *Informatica*, vol. 32, no. 3, 2008.

[9] X.-S. Yang and S. Deb, "Two-stage eagle strategy with differential evolution," 2012. [Online]. Available: https://arxiv.org/abs/1203.6586

[10] B. Li and B. Wu, "Online robust bin packing for resource allocation in cloud computing," in *2023 26th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, 2023, pp. 1063–1068.

[11] E. Solangi, T. M. B. Albarody, S. Al-Challabi, J. A. Khan, and S. Ali, "Design and failure analysis of a vacuum pressure vessel for aerospace applications using finite element analysis (fea)," *Engineering, Technology amp; Applied Science Research*, vol. 14, no. 6, p. 17888–17893, Dec. 2024. [Online]. Available: https://etasr.com/index.php/ETASR/article/view/7673

[12] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.220.4598.671

[13] K. Otubamowo, T. Egunjobi, and A. Adewole, "A comparative study of simulated annealing and genetic algorithm for solving the travelling salesman problem," 2012.

[14] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. The MIT Press, 04 1992. [Online]. Available: https://doi.org/10.7551/mitpress/1090.001.0001

[15] H. Braun, "On solving travelling salesman problems by genetic algorithms," in *International Conference on Parallel Problem Solving from Nature*. Springer, 1990, pp. 129–133.

[16] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95 - International Conference on Neural Networks*, vol. 4, 1995, pp. 1942–1948 vol.4.

[17] K.-P. Wang, L. Huang, C.-G. Zhou, and W. Pang, "Particle swarm optimization for traveling salesman problem," in *Proceedings of the 2003 International Conference on Machine Learning and Cybernetics (IEEE Cat. No.03EX693)*, vol. 3, 2003, pp. 1583–1585 Vol.3.

[18] S.-h. Zhan, J. Lin, Z.-j. Zhang, and Y.-w. Zhong, "List-based simulated annealing algorithm for traveling salesman problem," *Computational intelligence and neuroscience*, vol. 2016, no. 1, p. 1712630, 2016.

[19] J.-Y. Potvin, "Genetic algorithms for the traveling salesman problem," *Annals of operations Research*, vol. 63, pp. 337–370, 1996.

[20] E. Falkenauer, A. Delchambre *et al.*, "A genetic algorithm for bin packing and line balancing." in *ICRA*. Citeseer, 1992, pp. 1186–1192.

[21] [Online]. Available: https://github.com/tycho-bear/capstone-project