# The Traveling Salesman Problem: An Analysis and Comparison of Metaheuristics and Algorithms

Mason Helmick

A Senior Thesis submitted in partial fulfillment
of the requirements for graduation
in the Honors Program
Liberty University
Spring 2022

Acceptance of Senior Honors Thesis

This Senior Honors Thesis is accepted in partial
fulfillment of the requirements for graduation from the
Honors Program of Liberty University.

_____
Timothy Van Voorhis, Ph.D.
Thesis Chair

_____
Robert Rich, M.S.
Committee Member

_____
David Schweitzer, Ph.D.
Assistant Honors Director

_____
Date

**Abstract**

One of the most investigated topics in operations research is the Traveling Salesman Problem (TSP) and the algorithms that can be used to solve it. Despite its relatively simple formulation, its computational difficulty keeps it and potential solution methods at the forefront of current research. This paper defines and analyzes numerous proposed solutions to the TSP in order to facilitate understanding of the problem. Additionally, the efficiencies of different heuristics are studied and compared to the aforementioned algorithms' accuracy, as a quick algorithm is often formulated at the expense of an exact solution.

# The Traveling Salesman Problem: An Analysis and Comparison of Metaheuristics and Algorithms

## Introduction

### The Traveling Salesman Problem

The traveling salesman problem (TSP) is a combinatorial optimization problem characterized as NP-hard. NP-hard defines a problem that is nondeterministic polynomial acceptable, meaning there is no single algorithm that will provide an exact solution in polynomial time, though the solution may be verifiable in polynomial time (Rego et. al, 2011). The basic structure of the TSP involves a set of nodes and edges, with the nodes often being referred to as cities and the edges connecting the nodes referred to as routes. The whole path taken in the TSP is called the tour, and subtours refer to subsets or chunks of the original tour. The problem, then, is minimizing the length of the tour that visits each city exactly one time, then returns to the city in which the tour began, which is stated mathematically as minimizing the following equation.

$$z = \sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}x_{ij}\,, d_{ij} = \infty \; for \; all \; i = j$$

where $d_{ij}$ is the distance between cities and $x_{ij}$ is defined as follows.

$$x_{ij} = \begin{cases} 1, \text{if city } j \text{ is reached from city } i \\ 0, \text{otherwise} \end{cases}$$

Lastly, the TSP is subject to the following constraints.

$$\sum_{j=1}^{n} x_{ij} = 1, i = 1, 2, \dots, n$$

$$\sum_{i=1}^{n} x_{ij} = 1, j = 1, 2, \dots, n$$

$$x_{ij} \in \{0,1\}$$

Lastly, it is required that the solution forms a roundtrip *n*-city tour. Without this final constraint regarding a roundtrip, the original formulation allows for subtour solutions in which all the cities are no longer connected but split into multiple subsets.

In addition, the edge connecting two cities may be directed, functioning as a one-way path. In applications of the TSP, directed edges can be the results of traffic, tolls, or other inhibitors that could affect the time or cost of a path. A set of undirected edges naturally has fewer solutions, as they can be approached from both sides with the same effectiveness (Zia et. al, 2018). As this is not always the case, the magnitude of the TSP can quickly inflate to the point that a predetermined algorithm would be far more efficient than simply guessing at a tour until a sufficient one is found. This hurdle introduces the need for metaheuristics, which attempt to solve the TSP quickly and precisely.

**Metaheuristics**

When an exact solution to a problem cannot be obtained because of time constraints or the complexity of the problem, a heuristic approach is often used to reach an approximated answer. Heuristics are algorithms that achieve these ends by reaching a well performing solution and proposing that to be the best available solution. When the iterations continue only until a local optimum is reached, the algorithm is called "greedy," as it immediately stopped at the first potential solution. Greedy algorithms are typically only chosen when a rapid algorithm is preferable to one of high accuracy. However, especially in the case of the TSP, the local

optimum from the starting state of the system is rarely the best solution. Oftentimes, a move to an inferior state is needed to move from one local optimum to another in search for a global optimum.

A metaheuristic is a high-level problem-independent algorithmic framework that provides a set of guidelines or strategies to develop heuristic optimization algorithms (Sörensen & Glover, 2013). With a local optimum no longer needed to terminate the algorithm, new criteria are needed to decide when the process should be stopped. According to Taha (2017) these reasons include: a certain number of iterations has been reached, a certain number of iterations has been reached since the last optimum was reached, the neighborhood of the current search point is empty or cannot lead to another viable search point, or the quality of the last solution reached meets the desired standards. It follows, then, that metaheuristics are differentiated from each other not only in how they move from feasible solution to feasible solution, but also in how they decide to cope when a local optimum is reached.

**Simpler Algorithms**

Two well-known categories of algorithms that are relatively competitive with each other are simulated annealing (SA) and tabu search (TS). The former takes a neighboring state of the system and, in an iteration, "a no-worse neighborhood solution is always accepted as the next move" (Taha, 2017, p. 452). This particular aspect of the algorithm is very simple, as intuition suggests if a move to a neighboring state of the system improves the condition, then of course it should be taken. If that, however, is not the case, then a move is made based on the current tour length, the potential next tour length, and the temperature of the system. The temperature of the

system decreases with every iteration, decreasing the likelihood of making another move. For the purpose of this paper, these will not be laid out formally for a basic solution.

An algorithm that employs TS is characterized by having memory, in that past moves from state to state are recorded. In each iteration, a set of edges is deleted and replaced to move toward a better solution. These moves are then added to a tabu list, constituting a collection of options that are off-limits. By restricting the next move made by the algorithm to choices that are unprecedented, loops are avoided and the algorithm is able to effectively terminate. Consequently, inferior moves that could lead to better overall solutions can be explored by the algorithm.

**Genetic Algorithms**

Genetic algorithms (GA) deviate from SA and TS in the sense that they are inspired by biology as opposed to another math-centric field. GA are based on evolutionary principles, investigating candidates and judging whether or not they are fit for survival. While intuition may point to the contrary, genetic algorithms have also proven to effectively solve the TSP within a reasonable amount of time. The algorithm first selects two system states from the population, called the "parents." Then, these states are simultaneously combined and adjusted to form the "children," and the feasible solutions with the least fit are replaced by children that are more optimized (Thamilselvan & Balasubramanie, 2009). It is clear to see how the repetition of such a process eventually reaches a state of approximate maximization or minimization, and similarly how it mimics the biological phenomenon of survival of the fittest.

In relation to the TSP, genetic algorithms start with two samples of the population, or two potential tours. Then, these tours generate two more tours for a total of four solutions. Then,

tour length is used to see which parent tours will be replaced by the children, provided that the parents are not both superior solutions. Given the broadness of the field of GA and their relative variance as compared to SA and TS, most of the analysis from this point forward will be focused on GA. Investigated algorithms will then be related back to how they can be used to solve the TSP.

## More Specific and Recent Search Methods

### Ant Colony Optimization

In the same way that GA were motivated by biological processes, other metaheuristics have been developed to mimic natural tendencies. Ant colony optimization (ACO), then, was inspired by the foraging and communication style of ants (Hanif, 2019). ACO falls under the category of swarm intelligence, which largely refers to artificial intelligence. Swarm intelligence begins with individual components that have no real knowledge or much intuition on their own, but as the agents interact with their environment and communicate with each other, the swarm as a whole begins to function efficiently (Selvi & Umarani, 2010). These patterns have also inspired algorithms derived from the behavior of bees, termites, and other insects following a comparable paradigm.

In the specific case of ants and the ACO, as ants find food outside of the nest, they naturally walk between the source and the nest, informing other ants of their discovery as they go. If an obstacle exists on the path to the food, or alternatively if one is placed there, the paths of the many ants will split. As the ants travel, they deposit pheromones along their trail that guide other ants toward the food and nest, so for a while multiple trails exist. However, the ants along the shorter pheromone trails reach the destination and the nest more frequently, so as more

members of the swarm travel the shorter path, it becomes more densely covered in pheromones. Hence, the shorter paths become more attractive and the less efficient route fades, leaving the ants with only the most efficient choices available.

When applied to the TSP, the nest, food source, and checkpoints along the way represent nodes, and the paths taken by the ants clearly take the place of edges. The ACO allows agents to start on paths that appear unfavorable, making it a stochastic process as it allows for deviation from local optima. For the purpose of the TSP, virtual ants completing tours are always in nodes as to be accounted for, though the probabilities associated with their choices are still represented by pheromones on the edges (Brezina & Čičková, 2011). Over time, the pheromones on less traveled edges evaporate and they become less considered in the search for an optimized tour in the TSP.

Mathematically, the updating process associated with pheromone values in relation to the TSP is

$$\tau_{ij} = (1 - \rho)\tau_{ij}$$

where $\tau_{ij}$ is the pheromone on the edge between node *i* and node *j*, and $\rho$ is the evaporation rate of the pheromone (Dorigo & Stützle, 2004). After this initial decrease in pheromone, the pheromones from ants that traveled along the edges are added:

$$\tau_{ij} = \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k}$$

The amount of pheromone deposited by ant *k* is the $\Delta\tau_{ij}^{k}$ term, which is broken down even further as

$$\Delta\tau_{ij}^{k} = \begin{cases} \frac{1}{C^{k}}, \text{if the arc } (i,j) \text{ belongs to the tour} \\ 0, \qquad\qquad \text{otherwise} \end{cases}$$

where $C^k$ is the length of the tour taken by the ant. With the value of the pheromone in hand, the probability that an ant chooses certain paths can be determined as follows.

$$p_{ij}^{k} = \frac{[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{l\epsilon N_i^k}[\tau_{il}]^{\alpha}[\eta_{il}]^{\beta}}$$

Here, $\eta_{ij}$ is equivalent to $1/d_{ij}$, or the reciprocal of the length of the edge, $\alpha$ is a value representative of the intensity of the pheromone trail, and $\beta$ is a value representative of the visibility of the pheromone trail (Divya, 2019). These foundations comprise the ant system (AS) algorithm, the first to be modeled after ACO.

Due to its late development and relative accuracy, the ACO has been developed into numerous other algorithms. The math for these is relatively simple, with nuances in some that are beyond the purpose of this paper.

**The Monarchy Metaheuristic**

Another algorithm that has surfaced very recently is the monarchy metaheuristic (MN). Instead of following a natural order such as insect behavior, the MN interestingly finds its roots in a form of government, with similarities to GA. Instead of parents, two initial solutions are chosen at random and one is named the king. The second solution chosen will then make a dynasty list of possible successors to the king solution (Ahmia & Aïder, 2019). The dynasty list will be a union of a child list, $L_1$, and a relative list, $L_2$.

The next king is determined using one of three principles that applies to real-world monarchy succession. The first of these is primogeniture, where the eldest child of the current

monarch inherits the throne. In the MN, primogeniture means that the next solution chosen as king will come from $L_1$. The second is seniority, in which the monarchy prefers the oldest generation available, so any available siblings of the king will rise to power. Thus, in the case of seniority, the next solution will be chosen from $L_2$. Third is tanistry, which is the odd case in which a monarch is chosen by election from the dynasty list. Naturally, the correlation to the MN is that the next solution will be the current best, feasible solution in the union of lists one and two. In addition to the normal three means of succession, Ahmia and Aïder propose a fourth method called intrusion. If the next solution is an intruder, that means it was chosen using a different algorithm, such as SA or TS.

The creation of $L_1$ is essentially a GA. First, an identical position in both parents is selected. Then, the cut points in the parent solutions remain in the same position for the children, with the subtours on either side keeping their original order and only switching places if the point cut from the parent is in a new position. $L_2$ is synthesized slightly more intuitively, with both a one-point neighborhood approach and a two-point neighborhood approach. In the former, one node is selected from the tour and is tested in all other possible positions. In the latter, one node is selected and switched with another node along the tour. As the length of realistic tours often render it difficult to make a complete list under these methods, the MN uses list length as a parameter.

If the next king solution offers no chance of improvement, the MN has two choices to move forward. The first consists of keeping the solution and diminishing the sizes of the lists. This is $MN_1$. The second goes back to the best solution so far and uses its dynasty list to try

again, known as $MN_2$ (Ahmia & Aïder, 2019). While not heavily researched, the MN offers a comparatively efficient and accurate solution to the TSP.

**Expansion on Genetic Algorithms**

When implementing an algorithm, a starting state needs to be initialized before actual iterations can begin. The beginning state of the system is often chosen at random. However, GA and many other techniques can implement the nearest-neighbor (NN) heuristic to begin somewhere a little more knowledge-based. "The algorithm starts a tour by selecting random city and adds the nearest unvisited city to the last city in the tour until all cities are visited" (Halim & Ismail, 2017, p.2). The NN is a clearly greedy approach, as it makes a series of quick decisions to reach a solution. The results of the NN, however, can vary greatly depending on which city it starts in, making it useful for not much more than the beginning of a more efficient metaheuristic.

Another option for finding a local optimum to start is a reversal heuristic (RH). In RH, the order of an open subtour is reversed in an attempt to improve the length of the tour as a whole. An open subtour is defined as a subtour that is missing at least one leg (Taha, 2017). By limiting the algorithm to open subtours, the entire tour does not get reversed and repeat the old solution. A chain of cities of any size may be reversed, though it is commonly limited to two or three at a time. The RH algorithm finishes once all possible reversals of the original tour are observed, and the best result is determined the local optimum.

Like in the MN, GA is overall based on a system of crossover and mutation using the characteristics of parents (Shim et. al, 2011). Before these operators are expounded on, it is important to note that, "unlike tabu and simulated annealing where a new search move can be

infeasible, infeasible parent tours may never lead to the creation of feasible child tours" (Taha, 2017, p. 455). The crossover is responsible for child creation and therefore happens as an interaction of the parents, with one parent being a particularly better solution than the other. First, the two crossover points are randomly selected. Once the crossover points are selected, two children are partially formed by swapping the nodes contained in the crossover points to the offspring. Then, the parent tours are rotated so the nodes maintain the same order but have shifted starting places. The nodes contained between the crossover points are subtracted from the shifted tours, and the difference is combined with the partially formed children to finish the iteration. After the crossover, mutation is allowed to take place to help urge the algorithm toward a better solution. Mutation does not occur in all children, and the probability of it occurring is rather low and assigned during the construction of the algorithm. The actual process of mutation consists of swapping two nodes in the child to create a completely new solution. Mutation is necessary to help the algorithm explore new states, as well as to keep it from staying converged on a local optimum (Larrañaga et. al, 1999).

## Other Implementations of Genetic Algorithms

Genetic algorithms are clearly somewhat sufficient on their own, otherwise they would not still be a topic of discussion in operations research today. Of course, as GA have gathered merit, ideas for the refinement of the algorithms have arisen. Instead of scrapping the method as a whole, changes have been made to the original algorithm idea to make it more robust and accurate. Changes to GA have largely included adjustments to the operators, crossover and mutation, in order to make tweaks to the original results obtained. On the other hand, other

methods of solving the TSP have been combined with GA to make hybrid solutions that reap the benefits of both components.

Hybrid GA (HGA) typically function by applying the standard operators to already discovered local optima (Katayama & Narihisa, 2001). For example, GA has been combined with SA and TS to yield better results than any of them could have produced on their own. The GA in this instance was also able to take advantage of the many parameters required for SA and TS. However, the algorithm initially ran into an error when switching from either metaheuristic to GA. The problem was eventually ameliorated by using the parameters from the other algorithms. When using SA, for example, the temperature is fixed for every generation and only changed during iterations of SA.

In HGA, the metaheuristic and GA essentially work independently. After the GA has completed crossover and mutation, the other metaheuristic being used performs a search for the next optimum (Jin et. al, 2012). Katayama and Narihisa (2001) suggest a model of HGA that uses SA instead of SA with TS. This strategy was able to simultaneously reduce the number of parameters and the demand of setting up the combined algorithm. The proposed pseudocode for the individual components of GA and SA are provided in Tables 1 and 2, respectively, with the resulting combination into HGA(SA) in Table 3.

**Table 1**

*Pseudocode for Genetic Algorithm*

| | |
|---|---|
| 1 | Begin |
| 2 | Set $k = 0$ to denote the beginning generation |
| 3 | Set $P(k)$ to equal the number of desired initial feasible solutions |

| | |
|---|---|
| 4 | Evaluate the components in $P(k)$ |
| 5 | While the termination conditions have not yet been met |
| 6 | Do |
| 7 | $k = k + 1$ to improve the generation number |
| 8 | The new population $P(k)$ is set equal to a selection from $P(k + 1)$ |
| 9 | Perform crossovers on members of $P(k)$ |
| 10 | Perform mutation on members of $P(k)$ |
| 11 | Reevaluate the members of $P(k)$ |
| 12 | End |
| 13 | Return the solution |
| 14 | End |

SA uses a system of probability to escape the points of local optima (Taha, 2017). The probability used to decide whether or not the iterations should continue is largely dependent on a temperature value, with parameters decided prior to the execution of the algorithm. SA uses a neighborhood of solutions related to the initial feasible solution, drawing randomly from the determined neighborhood. When the new solution drawn is better than the previous, the temperature remains unchanged, and the iterations continue. However, if the drawn solution is not as optimized as the previous, the temperature is reduced and the probability of moving on is subsequentially determined (Albright & Kung, 2007). By undergoing this process, SA is able to move past local optima that greedy algorithms would stop at. In order to adjust the value of the temperature, a ratio is determined and factored together with the temperature. In an iterative

process, this process of reduction is made into a temperature schedule. The schedule is implemented by setting the temperature $T = T_i$, where $T_i$ is defined as

$$T_i = r_i T_{i-1}$$

Here, $i$ is representative of the iteration number and $r$ denotes the ratio that continually decreases the temperature. Applying this to the condition that defines SA, the chance of accepting the next solution is

$$P(\text{accept}) = \begin{cases} 1 & , \text{if } f(x) < f(x') \\ \exp\left(\dfrac{f(x') - f(x)}{T}\right) & , \text{if } f(x) > f(x') \end{cases}$$

where $f(x)$ is the old solution, $f(x')$ is the potential next solution found by the SA metaheuristic, and $T$ corresponds to its iteration on the temperature schedule. Clearly, a better solution will always be accepted. It is then easy to see in the probability function how inferior solutions become increasingly more likely to cause the process to terminate.

**Table 2**

*Pseudocode for Simulated Annealing*

| | |
|---|---|
| 1 | Begin |
| 2 | Set $\alpha$ and $T$ to sufficiently high values, and set $FT$ to a desired termination value |
| 3 | Start with $x$ equal to a feasible solution |
| 4 | Repeat |
| 5 | While the number of iterations has not reached the limit |
| 6 | Do |
| 7 | Choose an $x'$ in the neighborhood |
| 8 | If $f(x') < f(x)$ then |

| | |
|---|---|
| 9 | $x = x$’ |
| 10 | Else |
| 11 | $x = x$’ with the probability $\exp(-(f(x') - f(x))/T)$ |
| 12 | End |
| 13 | $T = T\alpha$ |
| 14 | Until $T \leq FT$ |
| 15 | Return $x$ |
| 16 | End |

In HGA(SA), the temperature associated with the SA aspect of the metaheuristic is instead decreased with every generation. This means that while the GA runs its course, the algorithm is operating with fixed temperature to avoid the aforementioned timing problem. Initializing the parameters for SA, then, will naturally be a part of HGA(SA), as well as setting certain variables for GA such as the desired number of children.

**Table 3**

*Pseudocode for Hybrid Genetic Algorithm with Simulated Annealing*

| | |
|---|---|
| 1 | Begin |
| 2 | Set $\alpha$, $T$, and $FT$ as in SA, set $k = 0$ as in GA to denote the first generations |
| 3 | Set desired number of offspring |
| 4 | Generate the population of initial feasible solutions $P(k)$ and evaluate its members |
| 5 | Repeat |
| 6 | $k = k + 1$ to improve the generation value |

| | |
|---|---|
| 7 | Set $P(k)$ to a selection from $P(k)$ |
| 8 | Alter structures in $P(k)$ by crossover (CSEX) |
| 9 | While not reaching the desired number of offspring |
| 10 | Do |
| 11 | Randomly choose an $x$' from the neighborhood |
| 12 | If $f(x') < f(x)$ then |
| 13 | $x = x'$ |
| 14 | Else |
| 15 | $x = x'$ with the probability $\exp(-(f(x') - f(x))/T)$ |
| 16 | End |
| 17 | End |
| 18 | Evaluate the structures in $P(k)$ |
| 19 | $T = T\alpha$ until $T \leq FT$ |
| 20 | Return the best solution |
| 21 | End |

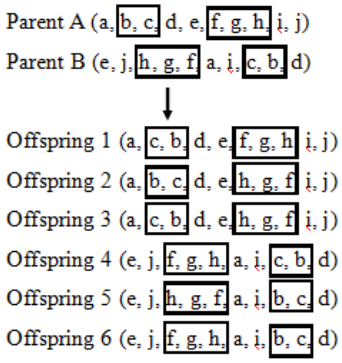
In the HGA(SA) pseudocode, CSEX stands for complete subtour exchange crossover. In CSEX, identical subtours are identified on each of the permutations that become the parents in the GA. Given that the edges between cities are undirected, a subtour with the same cities as another subtour in reverse order would still be counted as identical. However, if three or more cities that neighbor each other in both parents are not displayed in the same order or reverse order, CSEX will not select them as identical subtours. As a result of this distinction, the

efficiency of the crossover was reduced from $O(n^5)$ to $O(n)$ (Katayama & Narihisa, 2001). The resulting offspring will rearrange the chosen subtours in as many ways as possible, including switching their positions from where they were in both parents. Compared to the normal method of crossover, CSEX does not tend toward extremely different solutions from the current ones being evaluated. The children, therefore, are not too far removed from the parent tours, allowing for a focused path in the direction of optimality (Freisleben & Merz, 1996). In doing so, the local search algorithm is also able to execute effectively.

To clarify the appeal of GA for both solutions on their own and in conjunction with other methods, a visual representation of CSEX as a method of crossover is provided in Figure 1.

**Figure 1**

*Complete Subtour Exchange Crossover*




The relationship between each child and the parents is apparent, with the rearrangements being denoted by boxes. In a GA, crossovers direct the heuristic toward an optimal solution before committing to a new iteration, making it appealing as it more quickly eliminates unlikely moves. Similarly, applying crossover methods to other heuristics helps narrow the size of the search space before each new iteration begins investigating new possibilities. The MN offers the same appeal as it also follows very closely from GA.

The other nuances in the HGA(SA) that are inherited from its smaller components are more easily recognized. For example, the if condition reliant on temperature appears the same as it did for SA. In fact, only two integral additions were really made, one being the specification of CSEX over a normal crossover operator. The other was setting the parameter for minimum children yielded by the GA portion of the algorithm.

## Drawing Connections to the Traveling Salesman Problem

### Simpler Algorithms

Mathematically, the application of SA algorithms to the TSP does not really venture far from its basic formulation. First, a starting tour is chosen. This can be done with a nearest-neighbor algorithm or with a random selection process. After the selection of an initial tour, a subtour reversal is conducted. In a subtour reversal, typically two legs, or routes between cities, are removed and replaced by new legs (Wang et. al, 2009). The temperature schedule then reduces the temperature as it did in HGA(SA). The neighborhood at the iteration then includes every tour, including infeasible results, that can be formed from subtour reversals on the starting state of the system. If a shorter, feasible tour becomes available, it becomes the subject of the next iteration. If no new tours offer an improvement to the current tour, Taha (2017) offers a simpler condition reminiscent of the basic SA algorithm that can be applied to determine the probability of continuing the process.

$$R < \exp\left(\frac{L_{\text{cur}} - L_{\text{new}}}{T}\right)$$

Here, *R* is a randomly generated number between zero and one, and the *L* terms specify the metaheuristic results as tour lengths. Once the temperature has been reduced to the point that *R* is greater than the expression on the right side of the equation, the algorithm terminates.

TS is another algorithm often combined with more complex solution methods to reach a streamlined result. To begin a basic TS algorithm, a starting state of the system is determined using one of the same methods as SA. Then, subtour reversal can again be used to determine the neighborhood of feasible and infeasible moves from the current solution. When a local solution has been found that improves the state of the system, the algorithm makes a short hop to the new solution (Thamilselvan & Balasubramanie, 2009). The deleted legs from the subtour reversal are then added to the tabu list. A subtour reversal has a set number $\tau$ of iterations that it stays on the tabu list, referred to as the tenure period. A typical termination condition for TS is reaching a set number of successful iterations.

**Monarchy Metaheuristic**

Recall that the MN resembles a variation of GA, with solutions being referred to as titles of nobility. To decide on a beginning tour, a nearest-neighbor algorithm is employed. Next, a crossover operator called ordered crossover is used to determine the first part of the dynasty list, which corresponds to primogeniture. Ordered crossover favors one of the parent solutions, with the favored parent maintaining its order while both parents contribute a subtour cut from the same position in each tour. Typically, the preferred parent will be the king as it represents the current optimal solution. The cut city or subtour is then spliced into the favored parent in the same location as the cut, with cities outside of the cut switching with added cities to avoid duplicates when necessary. To create the list that correlates to the seniority approach of the MN, the same principles are applied as described in the fundamental algorithm, with one and two-point neighborhoods. Additionally, subtour reversal can be used, which in this case is referred to as a two-opt neighborhood (Mladenović et. al, 2012). Below is the pseudocode for the MN.

**Table 4**

*Pseudocode for the Monarchy Metaheuristic*

| | |
|---|---|
| 1 | Construct the first king, $K_1$, and initialize a list to contain all kings, $LK$ |
| 2 | Set size for $L_1$ and $L_2$ and generate lists corresponding to first king |
| 3 | Set $\delta = 0$ and set a max number of iterations |
| 4 | Choose a method for king selection |
| 5 | If primogeniture, use $L_1$, else use $L_2$ for seniority, or the union of $L_1$ and $L_2$ for tanistry |
| 6 | Else use a different metaheuristic for intrusion |
| 7 | If $\delta < 0$ |
| 8 | Option 1: add the king to LK and diminish the size of $L_1$ and $L_2$ accordingly |
| 9 | Option 2: return to the best king so far and use its dynasty list |
| 10 | Else if $\delta \geq 0$ |
| 11 | Add new king solution to $LK$ and increase the size of $L_1$ and $L_2$ according to $\delta$ |
| 12 | Terminate when max number of iterations is reached |
| 13 | Return the best king solution received |

The pseudocode for the MN demonstrates the two different versions it can undertake as explained before, $MN_1$ and $MN_2$, with respect to a conditioned variable. When an inferior move is made, the algorithm has the choice to keep the solution as the most recent on the king's list. If the most recent solution is to be kept, then the sizes of the components of the dynasty list, $L_1$ and $L_2$, need to be decreased accordingly. In the pseudocode, the factor that determines how much the lists will be reduced by is $\delta$. The variable $\delta$ denotes the percentage of improvement of the

new solution. The other option is to abandon the new solution and return to the best king solution so far. In either case, if the solution is improved upon and the new king is accepted, then $\delta$ is used to appropriately increase the length of lists $L_1$ and $L_2$. MN terminates after a preset number of iterations is reached.

**Ant Colony Optimization**

Now that the terms associated with ACO have been established in relation to the TSP, it is important to note how they fit into the flow of the algorithm. There seems to be no generally accepted principle for beginning the metaheuristic, or at least one supported by intuition. Instead, ants are typically put in a randomly selected starting city, and then from that node they move from city to city probabilistically until a tour is completed (Dorigo & Stützle, 2004). However, since no pheromone has been deposited on the initial tour, each city is equally likely, so the tours are completely random.

In order to determine the probability associated with an amount of pheromone, the three basic parameter variables for ACO are initialized: the evaporation rate $\rho$, the density of the pheromone $\alpha$, and the visibility of the pheromone $\beta$. Using the distance of edges as before, $d_{ij}$, another variation of the equation for the length of a tour in the TSP can be formulated to exhibit the path toward optimality.

$$f(\pi) = \sum_{i=1}^{n-1} d_{\pi(i)\pi(i+1)} + d_{\pi(n)\pi(1)}$$

A good solution to the TSP minimizes $f(\pi)$, which would align intuitively with minimizing the individual routes between cities. In this equation, $\pi$ is indicative of the permutation of possible number labels of the cities in the tour (Dorigo & Stützle, 2004). The variable $n$ simply denotes

the total number of cities. As is the case with many other metaheuristics, the process is terminated when a maximum number of iterations is reached.

Although many variations of ant systems (AS), the first and most fundamental application of ACO, have been proposed and proven effective, few of them offer minor, simple changes to the original algorithm. One such add-on that can alter the course of the ACO metaheuristic in use is the max-min ant system (MMAS) (Kang et. al, 2020). In this specification of ACO, only the optimal route in an iteration is permitted to receive a pheromone deposit. Naturally, the equations for pheromone updates are adjusted slightly.

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \Delta\tau_{ij}^{\text{best}}$$

$$\Delta\tau_{ij}^{\text{best}} = 1/f(s^{\text{best}})$$

All variables inherit their meaning from before, with "best" clearly identifying association with the best tour length in an iteration. The denominator in the second equation, $f(s^{\text{best}})$, is the explicit optimal solution. With these changes, new conditions are needed to avoid entrapment at local optima. These conditions consist of new maximums and minimums also being set for the pheromone deposits on the edges, as to avoid unnecessary evaporation of pheromones or excess deposit of pheromones.

***An Example of ACO***

To facilitate understanding of the probabilistic nature of ACO, an example will be stepped through by hand using the most basic form of ACO, Ant System. Before executing an ACO heuristic, some of the parameters need to be set. For the $\rho$ constant, previous research has shown that a value of 0.5 has proven to be effective (Dorigo & Stützle, 2004). The same research also suggests that an $\alpha$ value of 1 and a $\beta$ value between 2 and 5 yields acceptable

results for the TSP. For the purposes of demonstration, the chosen value for $\beta$ will be 3. To initialize the value for a pheromone along an edge, a relatively large, constant value is assigned to all edges. This method of initialization is typically effective in avoiding such low values of pheromone that early tours constructed by ants significantly affect the solution.

Consider ant *k* that has arrived at an edge between cities *i* and *j* with a weight $d_{ij}$ of 10. Assume that the initial value of pheromone $\tau_{ij}$ on the edge is 1. Suppose there are 2 cities that the ant may choose from, and the weight along the second possible edge is 15. The probability that ant *k* chooses the edge at all is

$$p_{ij}^{k} = \frac{[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{l \epsilon N_i^k}[\tau_{il}]^{\alpha}[\eta_{il}]^{\beta}} = \frac{1 \cdot 0.1^3}{(1 \cdot 0.1^3) + (1 \cdot 0.0667^3)} = \frac{.001}{.0013} = 0.77$$

Naturally, the probability for the other edge works out to 0.23, so the ant does not get trapped in the same city. Once the tour $T^k$ of ant *k* has been completed, the pheromone values are updated. First, evaporation of pheromone takes place, and in the first iteration, the evaporation will be constant across all edges because the initial amount of pheromone was also constant. Using the evaporation constant, the new amount of pheromone on an edge will be:

$$\tau_{ij} = (1 - \rho)\tau_{ij} = 0.5 \cdot 1 = 0.5$$

After evaporation takes place, pheromones are deposited on edges traveled on by the ant. For each ant, the increase in pheromone is equal to the inverse of the tour length taken by the ant. For example, for a tour of length 100, the amount of pheromone added to an edge by one ant would be 0.01. After the pheromone values are updated, a new iteration begins, and iterations are repeated until a max number of iterations is reached or until an acceptable tour length is achieved.

**Hybrid Genetic Algorithms**

Application of HGA to the TSP begins with the initialization of any parameters required by the GA used, followed by any additional parameters that would apply to the other search method. Furthermore, parameters may need to be conditioned on each other or other variables to avoid any timing errors as were experienced in early formations of HGA(SA). To begin a search for an optimal tour, a starting point is then chosen. The starting point is often randomized or found using another greedy algorithm to begin close to a local optimum.

The TSP will be handled by a hybrid algorithm by alternating between methods, with the local search metaheuristic typically being used for the purpose of providing unusually fit tour lengths for crossover and mutation. The solutions yielded are referred to as unusual as they are almost always more optimized than the next parent tours that would have been chosen by a GA acting alone. In some cases, the local search method will have to repeat itself multiple times to generate the parent tours required by the GA, as one iteration is not always enough to reach the standards of the individual metaheuristic being used (Deng et. al, 2021). Otherwise, extending HGA to the TSP is mostly a substitution of vocabulary into the included pseudocode. Once the index of the generation is initialized, parent tours are chosen. Then, crossover is performed on the parents to produce children. As HGA deal with parameters beyond the normal scope of generic crossovers, specialized techniques are employed to carefully handle subtours while constructing new tours. Mutation is performed subsequently, and until a termination condition specified by the HGA is reached, the algorithm will switch back to the local search technique operating in tandem with GA.

## A Comparison of Solution Methods to the Traveling Salesman Problem

The existence of multiple proposed solution algorithms to the TSP is not a testament to the overall failure of previous heuristics, but it is an evident attempt to refine a search toward optimality. Considering how robust the TSP is in that it can be applied to a plethora of different scenarios, it is natural that so much research has been conducted regarding its behavior and minimization. Older algorithms tend to operate on a basis of intuition, and though some of them are outdated in efficiency, understanding them is vital to understanding more modern techniques. It is then interesting to see not only how recent algorithms compare to each other in terms of accuracy and operation time, but also how much improvement these metaheuristics have offered to the foundation of operations research and the TSP.

When running larger TSP tests, Halim and Ismail (2017) found TS and ACO took particularly longer than other algorithms, taking as long as six times the amount of runtime as the fastest algorithm in some iterations. While this was to be expected of TS, the fact that ACO was developed so recently makes it seem that ACO would lend itself to a quicker solution. However, this result also likely stems from the fact that a basic AS metaheuristic was used in testing. Also, it is prudent to note that speed does not equate to accuracy. In fact, the greedy algorithm of nearest-neighbor was run alongside the other metaheuristics and it naturally yielded exemplary performance time, but often simultaneously poor results as far as tour length. From these results, it can be gathered that TS performs best at low to moderate numbers of cities, and for ACO to continue to compete with other higher-level algorithms that use GA or similar evolutionary elements, it needs to undergo alterations to its procedures. Otherwise, efficiency has not presented a significant concern among the other algorithms.

Accuracy showed a little more spread than the efficiency of the algorithms, though there interestingly was not one single method that consistently outperformed the others. At lower population numbers, the simpler algorithms competed well with newer algorithms, with SA and TS sometimes producing the best tour lengths, though never by a large margin. However, as the size of the tour increased older algorithms became clearly inferior by extravagant proportions. The MN often performed comparably with HGA as far as the ending result for optimal tour length. However, $MN_2$ consistently outperformed $MN_1$, leading to the assumption that it was superior especially at large population sizes (Ahmia, & Aïder, 2019). On that note, algorithms that followed the ACO procedure typically yielded a tour length that was just slightly greater than the MN, though it still managed to produce a better solution in some instances. Though basic GA did not seem to be the best option available, it was proven to typically perform with less than a 15% margin of error from the true solution (Halim & Ismail, 2017).

## Conclusion

Of the metaheuristics proposed to the basic TSP, no single process offered an ultimately streamlined path to a solution. In practical applications, more rudimentary algorithms such as TS and SA yield results that suggest that their usage should be limited to combinations with other methods as opposed to operating on their own. ACO metaheuristics are still on the rise, though they certainly show promise in their ability to eventually hone in on a minimized tour in the TSP. Other algorithms under seemingly completely new pretenses are also being developed, such as the MN, though it does show strong roots in GA. One other consideration that could reveal the superiority of an algorithm is its application to more variations of the TSP, such as the multiple traveling salesman problem (MTSP). Regardless, the constant addition of more metaheuristics

speaks to the hidden complexity of the TSP. Perhaps one of the most promising routes of algorithm development is in the direction of hybrids. By combining advanced algorithms with each other as opposed to with simple ones, the TSP can likely be moved towards greater minimization in tour length.

## References

Ahmia, I., & Aïder, M. (2019). A novel metaheuristic optimization algorithm: The monarchy metaheuristic. *Turkish Journal of Electrical Engineering & Computer Sciences*, *27*(1), 362–376. https://doi.org/10.3906/elk-1804-56

Albright, B., & Kung, S. (2007). An introduction to simulated annealing. *The College Mathematics Journal, 38*(1), 37-42. http://ezproxy.liberty.edu/login?qurl=https%3A%2F%2Fwww.proquest.com%2Fscholarly-journals%2Fintroduction-simulated-annealing%2Fdocview%2F203902358%2Fse-2%3Faccountid%3D12085

Brezina, I., Jr., & Čičková, Z. (2011). Solving the travelling salesman problem using the ant colony optimization. *Management Information Systems, 6*(4), 10-14. http://www.ef.uns.ac.rs/mis/

Deng, Y., Xiong, J., & Wang, Q. (2021). A hybrid cellular genetic algorithm for the traveling salesman problem. *Mathematical Problems in Engineering, 2021* http://dx.doi.org/10.1155/2021/6697598

Divya, M. (2019). Solving travelling salesman problem using ant systems: a programmer's approach. *International Journal of Applied and Computational Mathematics, 5*(4), 1-12. http://dx.doi.org/10.1007/s40819-019-0662-7

Dorigo, M., & Stützle, T. (2004). *Ant Colony Optimization*. MIT.

Freisleben, B., & Merz, P. (1996). New genetic local search operators for the traveling salesman problem. *Parallel Problem Solving from Nature, 4*, 890–899. https://doi.org/10.1007/3-540-61723-x_1052

Halim, A. H., & Ismail, I. (2017). Combinatorial optimization: Comparison of heuristic algorithms in travelling salesman problem. *Archives of Computational Methods in Engineering*, 1-14. http://dx.doi.org/10.1007/s11831-017-9247-y

Hanif, S. (2019). Travelling salesmen problem solution with ant-colony optimization tabu search algorithm. *I-Manager's Journal on Software Engineering, 14*(2), 1-9. http://dx.doi.org/10.26634/jse.14.2.17019

Jin, F., Shan, R., Zhang, Y., & Wang, H. (2012). Hybrid genetic algorithm based on an effective local search technique. *The Institution of Engineering & Technology*. http://dx.doi.org/10.1049/cp.2012.1181

Kang, Y., Xiaoming, Y., Liu, S., & Pan, H. (2020). A novel ant colony optimization based on game for traveling salesman problem. *Applied Intelligence, 50*(12), 4529-4542. http://dx.doi.org/10.1007/s10489-020-01799-w

Katayama, K., & Narihisa, H. (2001). An efficient hybrid genetic algorithm for the traveling salesman problem. *Electronics & Communications in Japan, Part 3: Fundamental Electronic Science, 84*(2), 76–83. https://doi.org/10.1002/1520-6440(200102)84:2<76::AID-ECJC9>3.0.CO;2-O

Larrañaga, P., Kuijpers, C.M.H., Murga, R. H., Inza, I., & Dizdarevic, S. (1999). Genetic algorithms for the travelling salesman problem: A review of representations and

operators. *The Artificial Intelligence Review, 13*(2), 129-170. http://dx.doi.org/10.1023/A:1006529012972

Mladenović, N., Urošević, D., Hanafi, S., & Ilić, A. (2012). A general variable neighborhood search for the one-commodity pickup-and-delivery travelling salesman problem. *European Journal of Operational Research, 220*(1), 270–285. https://doi.org/10.1016/j.ejor.2012.01.036

Rego, C., Gamboa, D., Glover, F., & Osterman, C. (2011). Traveling salesman problem heuristics: Leading methods, implementations and latest advances. *European Journal of Operational Research, 211*(3), 427. http://ezproxy.liberty.edu/login?qurl=https%3A%2F%2Fwww.proquest.com%2Fscholarly-journals%2Ftraveling-salesman-problem-heuristics-leading%2Fdocview%2F856439769%2Fse-2%3Faccountid%3D12085

Selvi, V., & Umarani, D. (2010). Comparative analysis of ant colony and particle swarm optimization techniques. *International Journal of Computer Applications, 5*, 1-6. https://doi.org/10.5120/908-1286

Shim, V., Tan, K., Chia, J., & Chong, J. (2011). Evolutionary algorithms for solving multi-objective travelling salesman problem. *Flexible Services & Manufacturing Journal, 23*(2), 207–241. https://doi.org/10.1007/s10696-011-9099-y

Sörensen, K., & Glover, F. (2013). Metaheuristics. In S.I. Gass and M. Fu (eds) *Encyclopedia of Operations Research and Management Science* (pp. 960–970). Springer.

Taha, H. A. (2017). *Operations Research: An Introduction* (10th ed.). Pearson.

Thamilselvan, R., & Balasubramanie, P. (2009). A genetic algorithm with a tabu search (gta) for traveling salesman problem. *International Journal of Recent Trends in Engineering, 1*(1), 607-610. http://ezproxy.liberty.edu/login?qurl=https%3A%2F%2Fwww.proquest.com%2Fscholarly-journals%2Fgenetic-algorithm-with-tabu-search-gta-traveling%2Fdocview%2F613351349%2Fse-2%3Faccountid%3D12085

Wang, Z., Geng, X., & Shao, Z. (2009). An effective simulated annealing algorithm for solving the traveling salesman problem. *Journal of Computational and Theoretical Nanoscience, 6*(7), 1680-1686. http://dx.doi.org/10.1166/jctn.2009.1230

Zia, M., Cakir, Z., & Dursun, Z. S. (2018). Spatial transformation of equality - generalized travelling salesman problem to travelling salesman problem. *ISPRS International Journal of Geo-Information, 7*(3), 115. http://dx.doi.org/10.3390/ijgi7030115