

Travelling Salesman Problem: Local Search and Divide and Conquer working together

Andrej Kazakov, ak905@ecs

Supervisor: Dr Richard A. Watson, raw@ecs

Second Examiner: Prof Mark Zwolinski, mz@ecs

Abstract—A good solution to a Travelling Salesman Problem can be obtained in many different ways. Local search and Divide and Conquer are among the most popular approaches. This report gives an overview of some of the best algorithms from each of the categories.

A brand new algorithm is looked at next. This algorithm learns the structure of the problem through local search methods and then improves the local search method by recombining sub-tours using a the divide and conquer strategy. Although still in development, the new algorithm looks very promising.

I. INTRODUCTION

A. Optimisation problems

Any problem, where given a particular problem instance, one has to select the best solution from all possible solutions is called an *optimisation problem*. Such problems are often encountered in engineering projects.

For example, sorting a list of integers is a well known and relatively easy combinatorial problem, there are many algorithms able to solve in a feasible amount of time.

On the other hand, there are hard *combinatorial problems*, where the best known algorithms are not good enough. Here are some of the most well-known hard problems [1]:

- 1) Boolean satisfiability problem (SAT)
- 2) Travelling salesman problem
- 3) Graph colouring problem

More formally, problems are often classified as belonging to P or NP classes of problems. NP-complete [1] is a subclass of NP. Any other problem in NP can be reduced to any problem in NP-complete [2]. All problems listed above are NP-complete.

B. Travelling Salesman Problem

Travelling Salesman Problem (TSP) a combinatorial optimisation problem.

Given a set of cities C and distances between any two of them, return the shorted possible valid tour T .

Where $|C| \geq 3$, distances between cities are usually provided as a matrix $|C| \times |C|$. A tour T is valid iff it visits all cities exactly once [3] and returns to its starting point.

TSP can be specified over euclidean space [4], for example a real layout of cities in a region. Or the space can be arbitrary. Some algorithms work only over limited set of spaces, others don't rely on this information.

The size of TSP solution space grows very quickly with the problem size. If we have n cities, then the number of possible valid tours will be $n!$. If one uses brute force to solve the TSP, the algorithm will be $O(n!)$.

C. Uses of TSP

Apart from the obvious use of TSP as finding the shortest route between points in space, there also is [3]:

- 1) Gene mapping problem. Ordering of genes in chromosomes seems to be similar ordering of cities in a TSP [5]
- 2) Cryptanalysis and breaking of cryptographic systems [6].

Lenstra suggested a few ways of mapping problems similar to TSP in structure to TSP instances [7]

If a problem is similar to TSP in structure, means that it is also likely to be in the NP-complete class. By recognising this early one can save a lot of effort. By transforming a problem into a TSP instance, or any of the well-known NP-complete instances one can choose the best approximation algorithm for his needs.

II. TSP ALGORITHM OVERVIEW

A. TSP representations

Algorithms described in sections II-B, II-D rely on small improvements to the current solution – mutation [8]. A variety of mutation operators exist for TSP, they are described along with the appropriate algorithm classes.

A particular algorithm relies on a particular mutation operator, but the operator only works with a particular representation. While there are only a few representations, there many more operators and algorithms. This section gives an overview of some of the most well-known TSP representations.

1) *Why Does Representation Matter?*: The main goal of a mutation operator is given the current solution and the problem landscape, make a small change to the solution and see how it affects the solution. Most algorithms rely on the assumption that the new solution lies next to the old one in the solution space.

This section gives an overview of four most popular representations, the new algorithm described in Section III uses the Adjacency representation.

a) *Path representation*: One of the most natural ways to represent a TSP tour is a list of cities in the appropriate order. This is known as path representation [9]. Let's have a closer look at such a tour.

Given $C = \{1, 2, 3, 4\}$ ¹, let's take a tour [1, 4, 3, 2].

It is implied that 2 is connected to 1 to make a valid tour:

$$\dots \rightarrow 1 \rightarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow \dots$$

¹ C is a set of cities here and below

Making a small change (mutation) in this representation is not trivial. For example, let's change the second city we visit to 3. Then the tour becomes [1, 3, 3, 2].

Since the tour is invalid, it needs to be repaired [10]. In this case, replacing the third city by a 4 (the city we have omitted). New tour: [1, 3, 4, 2]

For the path representation the repair step is trivial, but the repaired tour can be quite different from the original tour. Such a mutation operator is disruptive and less effective.

The path representation is not the only one suffering from the disruptive effects on the repair mechanisms.

2) *Existing Representations:* The path representation, described in Section II-A1 is the most intuitive and therefore will be used to describe all other representations in this section.

a) *Ordinal representation:* First introduced by Grefenstette [11].

It is based on a list of relative offsets n , such that

$$1 < n < |C| - i + 1$$

where i is the order number of the element.

To reconstruct a path from ordinal representation one has to take the n -th element from the head of the ordered unvisited cities' list for each element of the array n . Table I has the details for ordinal representation.

<i>Example:</i>	
<i>Tour:</i>	[1, 3, 2, 1] $Unv = [1, 2, 3, 4]$
<i>Transformation:</i>	$i = 1, Tour[i] = 1, Unv[Tour[i]] = 1,$ $i = 2, Tour[i] = 3, Unv[Tour[i]] = 4,$ $i = 3, Tour[i] = 2, Unv[Tour[i]] = 3,$ $i = 4, Tour[i] = 1, Unv[Tour[i]] = 2,$
<i>Path:</i>	... $\rightarrow 1 \rightarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow \dots$
<i>Mutation:</i>	Change the value of an randomly selected positions in the array to another valid value
<i>Evaluation:</i>	No repair is needed after the mutation. Transforming to a path representation to assess the length of the new tour increases the running time without bringing significant decrease in the disruption rate [3]

TABLE I
ORDINAL REPRESENTATION EXAMPLE

b) *Random Ranks representation:* First suggested by Sywerda [12].

Array of random values $n \leq N$, where $N \gg |C|$ and C is the set of cities. The order of the number in the tour identifies the city. Details are given in Table II.

<i>Example:</i>	$N = 100$
<i>Tour:</i>	[45, 15, 5, 6]
<i>Path:</i>	... $\rightarrow 4 \rightarrow 3 \rightarrow 1 \rightarrow 2 \rightarrow \dots$
<i>Mutation:</i>	Swap ranks between any two locations
<i>Evaluation:</i>	Slow and little loss of disruption compared to path representation [13].

TABLE II
RANDOM RANKS REPRESENTATION EXAMPLE

c) *Adjacency representation:* Adjacency representation was first introduced by Grefenstette [11].

A tour consists of edges between pairs of cities. One can implement the adjacency representation as an array of pairs [14] or use the array indexes as the first item in the pair [3].

Table III shows an example of the array index used as the first element in the pair.

<i>Example:</i>	array index as first item
<i>Tour:</i>	[3, 4, 2, 1] $1 \rightarrow 3, Tour[1] = 3,$ $3 \rightarrow 2, Tour[3] = 2,$ $2 \rightarrow 2, Tour[2] = 4,$ $4 \rightarrow 2, Tour[4] = 1$
<i>Transformation:</i>	
<i>Path:</i>	... $\rightarrow 1 \rightarrow 3 \rightarrow 2 \rightarrow 4 \rightarrow \dots$
<i>Mutation:</i>	Change a random index of the array to point to another city. Needs repair
<i>Evaluation:</i>	Although adjacency representation treats edges separately, in this implementation the Independence of the order of cities of an edge is replaced for efficiency of access.

TABLE III
ADJACENCY REPRESENTATION EXAMPLE, INDEX AS FIRST PAIR ITEM

Table IV shows an example an array of pairs

The main difference between two implementations is the strict position and hence direction in Table III and independent pairs in Table IV allow for undirected TSP Tour representation with flexible positions for pairs.

<i>Example:</i>	array of pairs
<i>Tour:</i>	[[1, 4], [2, 3], [1, 2], [4, 3]] $1 \rightarrow 4, Tour = [[2, 3], [1, 2], [4, 3]],$ $4 \rightarrow 3, Tour = [[2, 3], [1, 2]],$ $3 \rightarrow 2, Tour = [[1, 2]],$ $1 \rightarrow 2, Tour = [],$
<i>Transformation:</i>	
<i>Path:</i>	... $\rightarrow 1 \rightarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow \dots$
<i>Mutation:</i>	Replace a set of edges X with set of edges Y , where $ X = Y , Y \geq 2$
<i>Evaluation:</i>	Although there is no simple mutation operator and repair may be needed (depending on Y), the order of the items in the array or in the pair matters no more.

TABLE IV
ADJACENCY REPRESENTATION EXAMPLE, ARRAY OF PAIRS

d) *Summary:* The representations presented in this section are just a few most successful. There are others include: binary used by Holland [15], several attempts at matrix representation [16] [17] and [18], fixup permutations [13].

All of the research mentioned in this section attempts to simplify mutation through choosing a convenient representation. Operator comparison done by Chatterjee [13] and Larrañaga [3] suggest that it is much more efficient to use a simple representation, like the adjacent or path, and invent clever mutation operators and algorithms utilising them.

Sections II-B, II-B3 and II-D expand on the idea of simple representations by introducing the most efficient algorithms and operators.

B. Local Search Algorithms

TSP is a difficult problem. Its brute force complexity is $O(n!)$ to problem size n (see Section I-B). When brute force is not an acceptable, it is often considered to approximate the optimal solution. Local search algorithms are well suited for such approximation.

A Local search algorithm iteratively improves a given solution considering some local conditions. Often they would analyse the effect of small changes to the current solution discussed in Section II-A1.

Local search algorithms for TSP can be split into four main categories [19]:

1) *Creation heuristic*: algorithms create a tour choosing the best edges according to some heuristic. They stop after the tour is complete.

There are several ways to analyse the heuristics:

- Johnson looks at the worst case performance [19]. Two classes of heuristics can be identified:

- 1) *Fit for all TSP instances*. Since the heuristics are more general, the complexities are
 - $O(n^2)$ for Nearest neighbour [20], Nearest Insertion [21], Farthest Insertion [21] and Double MST [22]
 - $O(n^2 \log(n))$ for Greedy [23]
 - $O(n^3)$ for Christofides [24].
 Nearest neighbour is one of the most accurate, and it's expected tour length is only $\theta(\log(n))$ times optimal tour.
- 2) *Limited to TSP instances defined in a particular space, for example euclidean*. Since the heuristics rely on assumptions on the space the TSP is defined in, they perform better than the general ones:
 - $O(n \log(n))$ for Strip and Spacefilling Curve [25] and
 - $O(n)$ for Karp's Partitioning algorithm [26] and Litke's Recursive Clustering algorithm [27].

Accuracy of Strip and Spacelifting Curve is the best and it is at most 1.3 times optimal length.

- Frieze considers the worst accuracy of the heuristics [28] for these heuristics: greedy, cheapest insertion [29], interchange algorithms (e.g. 2-opt), repeated asymmetric heuristic [30] and Christofides heuristics.

Repeated asymmetric heuristic, which is an extension of the Christofides heuristic appear to have the least worst case accuracy:

$$R_n \leq \log_2(n), \text{ where } R_n = \frac{|TourDevised|}{|OptimalTour|}$$

Johnson [19] has shown that heuristic TSP algorithms are fast, especially if they are allowed to make spacial assumptions. On the other hand, Frieze [28] has noted the accuracy of such algorithms is not as good.

In summary, heuristic TSP algorithms provide a quick way to crudely estimate the solution.

2) *Local optimisation*: algorithms utilise the notion of neighbours and try to improve the current tour by picking the best neighbours to connect together.

The main difference between such algorithms is the amount of steps they look ahead.

So for example 2-opt [31] and 3-opt [32] and k-opt [33] look two, three and k steps ahead respectively. 2-opt and 3-opt are sometimes considered special cases of k-opt, which was introduced by Lin and Kernighan in their algorithm described in Section II-B3.

Considering as many steps ahead as possible is not always the best policy as it considerably increases the search space and hence the algorithm complexity.

Local optimisation is usually implemented as a hill climber. The algorithm starts with a randomly generated solution and iteratively improve it [34].

3) *Lin-Kernighan*: Lin-Kernighan algorithm, defined by Lin and Kernighan [35], is a hillclimber using k-opt mutation operator. Lin and Kernighan chose not to use a clever representation to avoid the disruptive repair step. Instead Lin-Kernighan uses adjacency representation (see Section II-A2) and a k-opt [33] mutation operator.

a) *k-opt*: K-opt is a complex mutation operator. It combines the mutation, the heuristic of selecting the next step and the repair mechanism. Let's consider 2-opt - a special case of k-opt.

Algorithm 1 shows a general outline of 2-opt. $Neighbours(x)$ is a set of cities close to x , but not connected to it. The choice of t_1, t_2 and t_3 is arbitrary, it can be exhaustive or random. Whereas there can only be one t_4 , or T' will be invalid.

Algorithm 1 2-opt

Require: C - set of cities

Require: E - set of edges

Require: T - current tour

Require: $t_1 \in C$

1: choose $t_2 \in C$, st $\exists x_1 \in T \wedge x_1 = [t_1, t_2]$

2: choose $t_3 \in Neighbours(t_2)$,

st $y_1 \notin T \wedge y_1 = [t_2, t_3]$

3: choose $t_4 \in C$,

st $x_2 \in T \wedge x_2 = [t_3, t_4]$

$\wedge y_2 \notin T \wedge y_2 = [t_1, t_4]$

4: $T' = \{x_1, x_2\} + \{y_1, y_2\}$

5: **if** $Length(T') < Length(T)$ **then**

6: $T = T'$

7: **end if**

k-opt is much more complex than 2-opt. Instead of using x_2 to close the gap, one would use t_4 as a new t_1 in the next iteration if this one was beneficial. Figure 1 shows an example of k-opt one step after 2-opt.

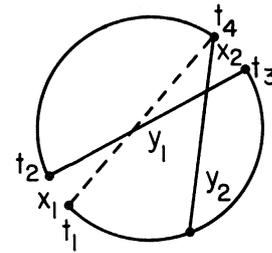


Fig. 1. k-opt example [35]

b) *Lin-Kernighan implementations*: According to the study done by Helsgaun [36], the original Lin-Kernighan would solve:

- 50 city TSP with almost 100% certainty,
- 100 cities – with 20%-30% certainty.

The accuracy can be improved by restarting the algorithm with a randomly solution several times.

Helsgaun also suggests a number of improvements [36], which improve the performance and accuracy of the original algorithm.

Lin-Kernighan is considered one of the best local search algorithms for TSP.

4) *Simulated annealing*: Simulated annealing(SA) is an extension of local search algorithms. Simulated annealing also uses a heuristic to improve the current tour. It uses a temperature factor T , which decreases with each next iteration.

Before choosing a city the SA algorithm takes a random number i , if $i < T$, it makes a random choice, otherwise it uses a heuristic [37].

Simulated annealing loses some of the performance of local search algorithms in favour of increase in accuracy [19].

C. Genetic algorithms

Genetic algorithms [38] use the theory of natural selection [39] to evolve solutions to specific problems.

Classic genetic algorithms, like the one described by Braun [40] contain a set of solutions (population) and evolve it utilising mutation and crossover.

The idea of a mutation operator has been described along with the representations in Section II-A1.

Crossover takes two good solutions and uses them to make a third one. Since the solutions are continuously improved by natural selection it allows the population to find an optimal solution faster.

When applying genetic algorithms to TSP one can use several strategies and representations. A review by Larrañaga [3] included 19 crossover and 9 mutation operators for a selection of representations.

Jong has shown that Genetic algorithms outperform Simulated annealing with Neural Networks on several NP-complete problems [41].

Merz describes a selection of algorithms for TSP, which use local search methods to obtain local optima and the uses genetic algorithms to obtain a global optima [34]. The main gain for such algorithms is the combination of the speed from local search methods and the diversity of solution space exploration from genetic algorithms.

1) *Edge Recombination*: Edge Recombination [14] is one of the crossover operators used by the Genetic algorithms. Tang has produced some promising results for Edge Recombination [42]: it achieves little disruption from the original tours and is efficient to compute.

Algorithm 2 Edge Recombination

Require: T_1, T_2 - initial tours

Require: $UC = C$ - set of unvisited cities

```

1:  $EM = MakeAnEdgeMap(T_1, T_2)$ 
2:  $T' = \{\}$ 
3: choose  $city$ ,
4: while  $|UC| > 0$  do
5:    $T' = T' + city$ 
6:    $RemoveOccurrences(EM, city)$ 
7:    $UC = UC - city$ 
8:    $city = MinCity(EM, city)$ 
9:   if  $city = 0$  then
10:    random  $city$  from  $UC$ 
11:   end if
12: end while

```

Algorithm 2 shows the general structure of Edge Recombination. EM is mapping each city to the cities it has been connected to in T_1 and T_2 [43].

$MinCity(EM, city)$ takes selects a city from EM , which is connected to $city$ and itself has the least connections.

In step 3 one can choose a city randomly or using the criteria from $MinCity(EM, city)$.

Step 10 is reached if the flow is broken and needs to be repaired. Whitley [43] recommends the repair to be random. But the author has discovered, that if one were to use the $MinCity(EM, city)$ criteria again, it would not break sub-tours.

As one can see from Algorithm 2, Edge Recombination is very good at taking a partial tour and a complete one and making a new tour from the tour.

D. Divide and Conquer

Most Local Search and Genetic algorithms try to solve TSP problems without inferring anything about the structure of the problem.

Combinatorial problems are difficult because they have multiple internal dependencies, a complex structure. Divide and Conquer algorithms aim to reduce the complexity by reducing the problem into sub problems and then assembling them.

Valenzuela suggests an EDAC algorithm, which divides the TSP space into smaller areas and solves each of them separately [44] as can be seen on Figure 2. The smaller solutions are combined into larger ones until the global solution is reached. The algorithm assumes an euclidean space.

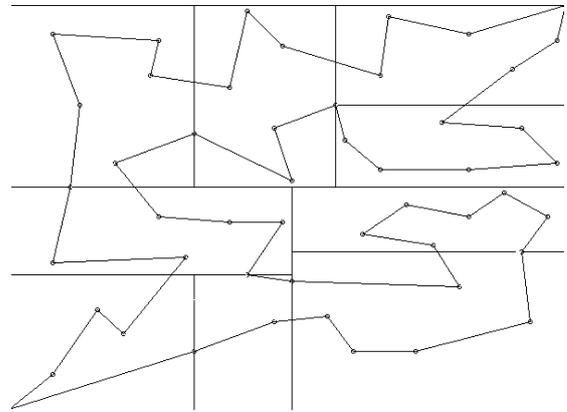


Fig. 2. EDAC example [44]

There are no other Divide and Conquer algorithms for TSP, however Iclanzan, while working on HIFF [45], has suggested a Building Block Hillclimbing method [46].

The algorithm splits the problem according to the results of a hillclimber, described in Section II-B2. Variables that tend to do well together are kept together as a *building block*. Each next iteration works with a combination of the building blocks. The size of the blocks grows until the entire solution is a single block.

Iclanzan divides the problem from the bottom up, whereas Valenzuela starts from the top and moves downwards. Although the results cannot be compared as Iclanzan works on HIFF rather than TSP problems, it is clear that Building Blocks hillclimber does not need to make assumptions about the problem, whereas the top to bottom approach does.

E. Previous work on Building Block Hillclimber for TSP

Hayes has performed an initial investigation [47] on applying the ideas of Building Block Hillclimbing (BBH) described in Section II-D. Although the algorithm is not finished, it has shown very promising results already.

The algorithm utilises the idea of Building Block Hillclimbing (BBH) to study the structure of the problem. A modification of the edge recombination, described in Section II-C1, is used to recombine the blocks.

The improvements to this algorithm by the author are presented in Section III.

F. Summary

In this section the Travelling Salesman Problem has been introduced along with some of its less traditional uses. TSP representation have been discussed. Many of them were trying to avoid creating invalid tours through specialised data structures. Experience has shown this method ineffective.

Algorithmic approaches to solving TSP have been presented in order of increasing complexity.

Local search algorithms were discussed first as the simplest, local optimisation algorithms appeared to be most effective. Lin-Kernighan has been discussed in detail along with 2-opt and k-opt operators.

Simulated annealing, building on top of local search algorithms, but avoiding local optima through a temperature measure has been discussed next.

Followed by Genetic algorithms. Genetic algorithms have led the development of many of the representations and operators. They have the potential to outperform simulated annealing.

As a next step, genetic algorithms have taken local search methods on board and presented incredible results. Edge Recombination has been discussed in more detail as one of the crossover operators from the Genetic algorithms.

The next step to improve the algorithms is to employ a divide and conquer strategy. And finally, a particular attempt at applying such a strategy to TSP.

The next section is going to describe the work done by the author to improve on this attempt.

III. IMPROVEMENTS ON THE BBH FOR TSP

The BBH TSP algorithm described in Section II-E has not been completed, it has potential, but it does not produce optimal tours reliably even on test problems.

On the other hand, Iclanzan [46] has shown that BBH method is very effective on optimisation problems. Edge recombination gives us confidence in the validity of the produced tours, so there is confidence that the BBH approach can work on a combinatorial problem, like TSP.

A. Algorithm

This section shows the main bits of the new algorithm as implemented by the author.

Algorithm 3 BBH for TSP outline

Require: C - set of cities
Require: P - TSP problem
Require: n_{tours} - number of tours

- 1: $Tours = \{\}$
- 2: **for all** $i < n_{tours}$ **do**
- 3: $T = RandomTour(P)$
- 4: $T = LocalOptimisation(T, P)$
- 5: $Tours = Tours + T$
- 6: **end for**
- 7: **while** making progress **do**
- 8: $M = DeviseDependancyMatrix(Tours)$
- 9: $Tours = DeviseNewTours(P, M)$
- 10: **end while**

a) *Structure Overview:* Algorithm 3 provides an outline of the algorithm.

Steps 1 to 6 initialise the algorithm by creating a list of locally optimal tours.

Step 7 begins the search. The concept of making progress can be as simple as decrease in the length of the shortest tour or some assertions on the structure of M .

M is a matrix containing the building blocks. M is $N_{edges} \times N_{edges}$. $M_{i,j}$ is the probability that edges i and j occur together.

Step 8 analyses the given tours and identifies which edges go well together. Such relationships are stored in M and are the building blocks. Algorithm 4 describes this function.

Step 9 builds new tours from the building blocks in M .

Each iteration M is completely replaced to remove possibly suboptimal relationships from the previous iteration.

b) *Retrieving building blocks:* Algorithm 4 shows how the building blocks are recognised.

Algorithm 4 $DeviseDependancyMatrix(Tours)$

Require: E - set of all possible edges

- 1: **for all** $i \in E$ **do**
- 2: **for all** $j \in E$ **do**
- 3: $M_{i,j} = \frac{\sum_{T \in Tours} Cooccurrence(i,j,T)}{|Tours|}$
- 4: **end for**
- 5: **end for**

Equation (1) shows the co-occurrence function, which is the means to recognise if any two edges should go together.

$$Cooccurrence(i, j, T) = \begin{cases} Penalty & \text{if } (i \in T \wedge j \notin T) \\ & \vee (i \notin T \wedge j \in T), \\ 0 & \text{if } i \notin T \wedge j \notin T, \\ 1 & \text{if } i \in T \wedge j \in T. \end{cases} \quad (1)$$

c) *Creating new tours:* Algorithm 5 shows how the building blocks are applied to create new tours.

Step 9 $ModEdgeRecombination(Edges, T)$ is the modified edge recombination algorithm, which is guaranteed to include all edges from $Edges$ and use T to fill in the gaps.

Step 8 uses notation $|Edges|$ for number of edges and step 13 uses $Length(T')$ for length of tour such edges form.

Algorithm 5 *DeviseNewTours*(P, M)

Require: M contains the building blocks from old $Tours$
Require: E - set of all possible edges
Require: P - TSP problem
Require: n_{tours} - number of tours
Require: $n_{iterations}$ - number of iterations allowed

```

1:  $Tours = \{\}$ 
2:  $Edges = \{\}$ 
3: for  $t$  to  $n_{tours}$  do
4:    $T = RandomTour(P)$ 
5:    $T = LocalOptimisation(T, P)$ 
6:   for  $it$  to  $n_{iterations}$  do
7:      $Edges = ApplyM(P, M)$ 
8:     if  $|Edges| < N_{cities}$  then
9:        $T' = ModEdgeRecombination(Edges, T)$ 
10:    else
11:       $T' = Edges$ 
12:    end if
13:    if  $Length(T') < Length(T)$  then
14:       $T = T'$ 
15:    end if
16:  end for
17:   $Tours = Tours + T$ 
18: end for

```

d) *Combining building blocks into a tour:* Algorithm 6 describes the creation of an incomplete tour from M .

Step 1 *FitnessProportionate*(M) select an edge from M proportionate to its fitness. It is assumed that if edge is unseen, i.e. $M_{e,e} = 0$ where $e \in E$, it is not considered.

Step 7 *Edges.wouldMakeTourInvalid?*(e) is false if e would invalidate $Edges$ if added.

Step 10 *CondProb*($M, Edges, j$) is approximation of the conditional probability $P(j|Edges \wedge E) \approx \prod_{e \in Edges} \frac{M_{e,j}}{M_{e,e}}$

Algorithm 6 *ApplyM*(P, M)

Require: M contains the building blocks from old $Tours$
Require: P - TSP problem

```

1:  $i = FitnessProportionate(M)$ 
2:  $Edges = i$ 
3:  $unselectedEdges_0 = \forall j \in E | M_{j,j} \neq 0 \wedge j \neq i$ 
4: while  $|unselectedEdges_i| < |unselectedEdges_{i-1}|$  do
5:    $i = i + 1$ 
6:   for all  $e \in unselectedEdges_i$  do
7:     if Edges.wouldMakeTourInvalid?( $e$ ) then
8:        $unselectedEdges = unselectedEdges - e$ 
9:     else
10:    if  $rand() < CondProb(M, Edges, e)$  then
11:       $unselectedEdges = unselectedEdges - e$ 
12:       $Edges = Edges + e$ 
13:    end if
14:  end for
15: end while

```

B. Analytics

Hayes looked at the algorithm as a whole and had limited time, so he could not determine why BBH algorithm did

not work on all TSP instances. The key to improving this algorithm is breaking it up and making sure the components behave well. This investigation has been split into three steps:

- 1) Values of M that should generate optimal tours
- 2) Creating optimal tours from such optimal M
- 3) Creating optimal M from locally optimal tours

Because of time restrictions, only the first two questions were looked into and are discussed later in this section.

C. Values of M

What values of M should generate an optimal tour?

1) *Co-occurrence function:* The *Cooccurrence*(i, j, T) function defined in equation (1) has a Penalty variable. Hayes used a constant -1 there. -1 penalty is quite strict and forces the algorithm to omit most of the information in the $Tours$. An examination of M structure as the Penalty gets closer to 0 showed that the amount of information retained increases with $Penalty \rightarrow 0$ as can be seen from Figure 3. The thicker the line on the figure the stronger the relationship between the dotted edge and that edge.

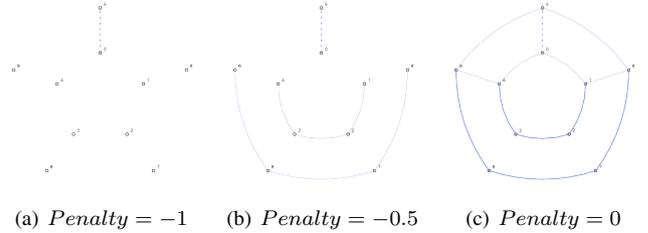


Fig. 3. M for a vertical edge (dotted line)

A range of experiments was conducted with $Penalty \in [0, -1]$. It was shown that the problem lies elsewhere and the change in $Penalty$ has only a minor effect on accuracy. It was decided to use $Penalty = 0$ in further experiments as it contains the most information.

2) *Test Problem:* Hayes has describes a Circle TSP problem well suited to investigate this algorithm. The problem is non-euclidean and consists of two closed circles of cities as shown on Figure 4. The problem is defined by distance to next city on a circle and distance between circles.

If the diagonals accurately model the 3D object there is little neutrality and the problem can be easily solved by most hill climbers. If one sets all diagonals constant, but longer than vertical or horizontal distances the problem becomes much more difficult.

The optimal solutions depend on the parameters, but if one uses 3 for next city, 4 between circles and 5 for diagonals, then the optimal solution is as shown on Figure 4.

If one creates the M matrix from a single optimal tour, an optimal tour is achieved as expected. On the other hand, it is far more likely to have the M matrix consisting of several locally optimal tours. To mimic this behaviour a set of optimal tours is created by rotating an the gap $[0, 9]$.

D. Creating optimal tours

Given an optimal M , how should the algorithm be changed to produce an optimal tour?

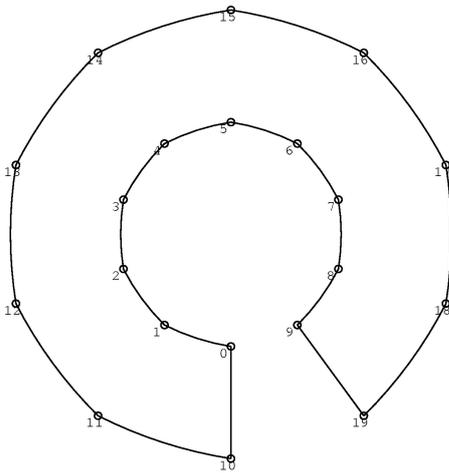


Fig. 4. Circles Problem Optimal Solution, flat projection

1) *Starting edge*: Step 1 of algorithm 6 chooses the first edge to form new tours. This edge is guaranteed to be in the solution. Hayes suggests using a random edge, but there are $C_2^n = \frac{n!}{2!(n-2)!}$ edges to choose from, where $n = |C|$ - number of cities. Whereas M has information at most on $n_{tours} * |C|$ edges.

As $C_2^n \gg |Tours| * |C|$, for the usual case of $|Tours| \ll |C|$, the creation of new tours appears to be ineffective most of the time, one should choose the first edge by a fitness proportionate method [48].

2) *Conditional Probability*: When creating tours from M in Algorithm 6, step 10

Hayes recommends using the $M_{i,j}$ as the probability of adding this edge. Then the entire tour depends only on the first edge.

What one really would like if for the new edges to depend on all added edges. Then the most logical move is to use the conditional probability $P(j|Edges \wedge E)$ approximated by the $CondProb(M, Edges, j)$ function.

This has decreased the average tour length by 71% from 1079,77 to 315,874 on a 100 city problem with optimal distance of 302. Still only 10 out of 500 tours were optimal.

The probability of edges forming the circles is high as they occur often. On the other hand, edges connecting the circles are rare and therefore their probability is very low. For example, $CondProb(M, Edges, j) = 4.36 \times 10^{-13}$, where $|C| = 20$, $|Edges| + 2 = |C|$ only the vertical edges are missing, j - one of the missing vertical edges.

The vast majority of tours would have just the intra-circle edges missing.

E. Future work

1) *Creating optimal M.*: The part of the investigation is still to be done. *How to create the optimal M from locally optimal tours.*

2) *Stopping criteria*: Step 7 of algorithm 3 does not define a formal stopping criteria. When implementing it should be noted that it is possible to have a local improvement, for example if we follow the $d(i) = shortestPath_{i-1} - shortestPath_i$, where i is the current iteration: $d(n) = 0, d(n+1) > 0, d(n+2) = 0, d(n+3) = 0, \dots$. A look-ahead strategy might be appropriate.

3) *Conditional Probability*: Although results show that the conditional probability is a great improvement, it is still incomplete. If we look at the incomplete tours produced it is obvious that the only edges we have information about and are still legal are the two missing edges. Finding a way for the algorithm to use include them at the tour creation stage would improve the algorithm greatly.

4) *Co-occurrence function*: It is also possible use another co-occurrence function. Mills suggests using a measure of surprise [49]. Let $Occurrence(A) = a$ and $Occurrence(B) = b$, then $surprise = 0$ if $Occurrence(A \wedge B) = a \times b$. And if $Occurrence(A \wedge B) > a \times b$, the edges go well together, so $surprise > 0$. The opposite is also true.

5) *Test Cases*: The algorithm is still under development. It has only been tested with a special tailored problem type described in Section III-C2. It is unknown how it will perform on well-established hard TSP problems, for example those defined in TSPLIB [50].

IV. CONCLUSION

This report has presented the best approaches to solving a TSP in order of increasing complexity: local search methods from greedy heuristics to lin-kernighan and simulated annealing, genetic algorithms, divide and conquer. The research as of what heuristics under what conditions work well on TSP is still ongoing.

Improvements to one of the algorithms still in development were looked at in more detail next. The algorithm is still incomplete, but its current shortcomings have been thoroughly analysed and it has been substantially improved.

To summarise, there are many ways to solve a TSP. New algorithms are still being created. They differ by performance, accuracy and complexity. One has to choose the right balance for each application.

REFERENCES

- [1] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)*. W. H. Freeman, January 1979. [Online]. Available: <http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20&path=ASIN/0716710455>
- [2] C. Papadimitriou and M. Yannakakis, "Optimization, approximation, and complexity classes," in *STOC '88: Proceedings of the twentieth annual ACM symposium on Theory of computing*. New York, NY, USA: ACM, 1988, pp. 229–234.
- [3] P. Larrañaga, 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, no. 2, pp. 129–170, April 1999. [Online]. Available: <http://dx.doi.org/10.1023/A:1006529012972>
- [4] J. L. Kelley, *General Topology*. New York: Springer, 1975.
- [5] J. Gunnels, P. Cull, and J. L. Holloway, "Genetic algorithms and simulated annealing for gene mapping," Corvallis, OR, USA, Tech. Rep., 1994.
- [6] R. Spillman, M. Janssen, B. Nelson, and M. Kepner, "Use of a genetic algorithm in the cryptanalysis of simple substitution ciphers," *Cryptologia*, vol. XVII, no. 1, pp. 31–44, 1993.
- [7] J. K. Lenstra and R. A. H. G. Kan, "Some simple applications of the travelling salesman problem," *Operational Research Quarterly (1970-1977)*, vol. 26, no. 4, pp. 717–733, 1975. [Online]. Available: <http://dx.doi.org/10.2307/3008306>
- [8] M. Mitchell, *An introduction to genetic algorithms*. Cambridge, MA, USA: MIT Press, 1996.
- [9] L. J. Schmitt and M. M. Amini, "Performance characteristics of alternative genetic algorithmic approaches to the traveling salesman problem using path representation: An empirical study," *European Journal of Operational Research*, vol. 108, no. 3, pp. 551 – 570, 1998. [Online]. Available: <http://www.sciencedirect.com/science/article/B6VCT-3TMR6M6-S/2/f2f66f2987383d564aac0fbc5f0ad82c>

- [10] J. Grefenstette, "Optimization of control parameters for genetic algorithms," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 16, no. 1, pp. 122–128, Jan. 1986.
- [11] J. J. Grefenstette, R. Gopal, B. J. Rosmaita, and D. V. Gucht, "Genetic algorithms for the traveling salesman problem," in *Proceedings of the 1st International Conference on Genetic Algorithms*. Hillsdale, NJ, USA: L. Erlbaum Associates Inc., 1985, pp. 160–168.
- [12] G. Sywerda, "Uniform crossover in genetic algorithms," in *Proceedings of the third international conference on Genetic algorithms*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1989, pp. 2–9.
- [13] S. Chatterjee, C. Carrera, and L. A. Lynch, "Genetic algorithms and traveling salesman problems," *European Journal of Operational Research*, vol. 93, no. 3, pp. 490 – 510, 1996. [Online]. Available: <http://www.sciencedirect.com/science/article/B6VCT-3VV42PR-P/2/6b67dfac4284fb5543d79fbd356d056>
- [14] T. Bui and B. Moon, "A new genetic approach for the traveling salesman problem," Jun 1994, pp. 7–12 vol.1.
- [15] J. H. Holland, *Adaptation in natural and artificial systems*. Cambridge, MA, USA: MIT Press, 1992.
- [16] Fox and McMahon, *Foundations of Genetic Algorithms: Genetic operators for sequencing problems*, 1st ed. Morgan Kaufmann, July 1991. [Online]. Available: <http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20&path=ASIN/1558601708>
- [17] D. Seniw, "A genetic algorithm for the traveling salesman problem," *MSc Thesis, University of North Carolina at Charlotte*, 1991.
- [18] A. Homaifar, S. Guan, and G. E. Liepins, "A new approach on the traveling salesman problem by genetic algorithms," in *Proceedings of the 5th International Conference on Genetic Algorithms*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993, pp. 460–466.
- [19] D. S. Johnson, "Local optimization and the traveling salesman problem," in *Proceedings of the seventeenth international colloquium on Automata, languages and programming*. New York, NY, USA: Springer-Verlag New York, Inc., 1990, pp. 446–461.
- [20] M. Bellmore and G. L. Nemhauser, "The traveling salesman problem: A survey," *Operations Research*, vol. 16, no. 3, pp. 538–558, 1968. [Online]. Available: <http://www.jstor.org/stable/168581>
- [21] D. J. Rosenkrantz, R. E. Stearns, P. M. Lewis, and II, "An analysis of several heuristics for the traveling salesman problem," *SIAM Journal on Computing*, vol. 6, no. 3, pp. 563–581, 1977. [Online]. Available: <http://link.aip.org/link/?SMJ/6/563/1>
- [22] S. Arora, "Nearly linear time approximation schemes for euclidean tsp and other geometric problems," in *FOCS '97: Proceedings of the 38th Annual Symposium on Foundations of Computer Science*. Washington, DC, USA: IEEE Computer Society, 1997, p. 554.
- [23] J. Edmonds, "Matroids and the greedy algorithm," *Math Program*, vol. 1, pp. 127–136, 1971.
- [24] N. Christofides, "Worst-case analysis of a new heuristic for the traveling salesman problem," GSIA, Carnegie Mellon University, Tech. Rep., 1976.
- [25] L. K. Platzman and J. J. Bartholdi, III, "Spacefilling curves and the planar travelling salesman problem," *J. ACM*, vol. 36, no. 4, pp. 719–737, 1989.
- [26] R. M. Karp, "Probabilistic analysis of partitioning algorithms for the traveling-salesman problem in the plane," *Mathematics of Operations Research*, vol. 2, no. 3, pp. 209–224, August 1977.
- [27] J. D. Litke, "An improved solution to the traveling salesman problem with thousands of nodes," *Commun. ACM*, vol. 27, no. 12, pp. 1227–1236, 1984.
- [28] A. M. Frieze, G. Galbiati, and F. Maffioli, "On the worst-case performance of some algorithms for the asymmetric traveling salesman problem," *Networks*, vol. 12, no. 1, pp. 23–39, 1982.
- [29] T. A. J. Nicholson, "A Sequential Method for Discrete Optimization Problems and its Application to the Assignment, Travelling Salesman, and Three Machine Scheduling Problems," *IMA J Appl Math*, vol. 3, no. 4, pp. 362–375, 1967. [Online]. Available: <http://imamat.oxfordjournals.org/cgi/content/abstract/3/4/362>
- [30] A. M. Frieze, "An extension of christofides' heuristic to the k-person traveling salesman problem," *Discrete Appl. Math.*, 1983.
- [31] J. L. Bentley, "Experiments on traveling salesman heuristics," in *SODA '90: Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1990, pp. 91–99.
- [32] G. Sierksma, "Hamiltonicity and the 3-opt procedure for the traveling salesman problem," *Applicaciones Mathematicae*, vol. 22, no. 3, pp. 351–358, 1994.
- [33] S. LIN, "Computer solution of the traveling salesman problem," *Bell System Tech. J.*, vol. 44, pp. 2245–2269, 1965.
- [34] P. Merz and B. Freisleben, "Genetic local search for the tsp: New results," in *In Proceedings of the 1997 IEEE International Conference on Evolutionary Computation*. IEEE Press, 1997, pp. 159–164.
- [35] S. Lin and B. W. Kernighan, "An effective heuristic algorithm for the traveling-salesman problem," *Operations Research*, vol. 21, no. 2, pp. 498–516, 1973. [Online]. Available: <http://www.jstor.org/stable/169020>
- [36] K. Helsgaun, "An effective implementation of the lin-kernighan traveling salesman heuristic," *European Journal of Operational Research*, vol. 126, no. 1, pp. 106 – 130, 2000. [Online]. Available: <http://www.sciencedirect.com/science/article/B6VCT-40TY2N6-8/2/1ec4c8829b13c4bd01b6e6a2edfb2c80>
- [37] D. Mitra, F. Romeo, and A. Sangiovanni-Vincentelli, "Convergence and finite-time behavior of simulated annealing," vol. 24, Dec. 1985, pp. 761–767.
- [38] Z. Michalewicz, *Genetic algorithms + data structures = evolution programs (2nd, extended ed.)*. New York, NY, USA: Springer-Verlag New York, Inc., 1994.
- [39] C. Darwin, *The origin of species by means of natural selection*, 6th ed. John Murray, London, 1875.
- [40] H. Braun, "On solving travelling salesman problems by genetic algorithms," in *PPSN I: Proceedings of the 1st Workshop on Parallel Problem Solving from Nature*. London, UK: Springer-Verlag, 1991, pp. 129–133.
- [41] K. A. De Jong and W. M. Spears, "Using genetic algorithms to solve np-complete problems," in *Proceedings of the third international conference on Genetic algorithms*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1989, pp. 124–132.
- [42] A. Y.-C. Tang and K.-S. Leung, "A modified edge recombination operator for the travelling salesman problem," in *PPSN III: Proceedings of the International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature*. London, UK: Springer-Verlag, 1994, pp. 180–188.
- [43] D. Whitley, T. Starkweather, and D. Shaner, "The traveling salesman and sequence scheduling: Quality solutions using genetic edge recombination," in *In Handbook of Genetic Algorithms*, 1991, pp. 350–372.
- [44] C. L. Valenzuela, "Evolutionary divide and conquer (i): novel genetic approach to the tsp," *Evolutionary Computation*, vol. 1, pp. 313–333, 1993.
- [45] R. A. Watson, G. Hornby, and J. B. Pollack, "Modeling building-block interdependency," in *PPSN V: Proceedings of the 5th International Conference on Parallel Problem Solving from Nature*. London, UK: Springer-Verlag, 1998, pp. 97–108.
- [46] D. Iclanzan and D. Dumitrescu, "Overcoming hierarchical difficulty by hill-climbing the building block structure," in *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*. New York, NY, USA: ACM, 2007, pp. 1256–1263.
- [47] N. Hayes, "Applying a compositional evolutionary algorithm to the travelling salesman problem," Tech. Rep., 2008.
- [48] D. Thierens and D. E. Goldberg, "Convergence models of genetic algorithm selection schemes," in *PPSN III: Proceedings of the International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature*. London, UK: Springer-Verlag, 1994, pp. 119–129.
- [49] R. Mills and R. A. Watson, "Adaptive units of selection can evolve complexes that are provably unevolvable under fixed units of selection (abstract)," p. 785, August 2008. [Online]. Available: <http://eprints.ecs.soton.ac.uk/15998/>
- [50] G. Reinelt, "TSPLIB–A Traveling Salesman Problem Library," *INFORMS JOURNAL ON COMPUTING*, vol. 3, no. 4, pp. 376–384, 1991. [Online]. Available: <http://jocjournal.informs.org/cgi/content/abstract/3/4/376>