12-20-2020

Comparing Greedy Constructive Heuristic Subtour Elimination Methods for the Traveling Salesman Problem

Petar Jackovich

Bruce A. Cox
Air Force Institute of Technology

Raymond R. Hill
Air Force Institute of Technology

Recommended Citation

Jackovich, P., Cox, B., & Hill, R. R. (2020). Comparing greedy constructive heuristic subtour elimination methods for the traveling salesman problem. Journal of Defense Analytics and Logistics, 4(2), 167–182. https://doi.org/10.1108/JDAL-09-2020-0018

Comparing greedy constructive heuristic subtour elimination methods for the traveling salesman problem

Petar Jackovich, Bruce Cox and Raymond R. Hill
Department of Operational Sciences, Air Force Institute of Technology, Wright-Patterson AFB, Ohio, USA

Received 27 September 2020
Revised 5 October 2020
Accepted 5 October 2020

Abstract

Purpose – This paper aims to define the class of fragment constructive heuristics used to compute feasible solutions for the traveling salesman problem (TSP) into edge-greedy and vertex-greedy subclasses. As these subclasses of heuristics can create subtours, two known methodologies for subtour elimination on symmetric instances are reviewed and are expanded to cover asymmetric problem instances. This paper introduces a third novel subtour elimination methodology, the greedy tracker (GT), and compares it to both known methodologies.

Design/methodology/approach – Computational results for all three subtour elimination methodologies are generated across 17 symmetric instances ranging in size from 29 vertices to 5,934 vertices, as well as 9 asymmetric instances ranging in size from 17 to 443 vertices.

Findings – The results demonstrate the GT is the fastest method for preventing subtours for instances below 400 vertices. Additionally, a distinction between fragment constructive heuristics and the subtour elimination methodology used to ensure the feasibility of resulting solutions enables the introduction of a new vertex-greedy fragment heuristic called ordered greedy.

Originality/value – This research has two main contributions: first, it introduces a novel subtour elimination methodology. Second, the research introduces the concept of ordered lists which remaps the TSP into a new space with promising initial computational results.

Keywords Traveling salesman problem, Constructive heuristic, Edge greedy, Multiple fragment heuristic, Subtour elimination, Vertex greedy, Heuristic, Fragment constructive heuristic, Exhaustive loop, Greedy tracker, Ordered greedy

Paper type Research paper

1. Introduction

Applegate *et al.* (2006) describe the traveling salesman problem (TSP) as:

> Given a set of cites along with the cost of travel between each pair of them, the traveling salesman problem, or TSP for short, is the problem of finding the cheapest way of visiting all the cities and returning to the starting point.

Journal of Defense Analytics and Logistics
Vol. 4 No. 2, 2020
pp. 167-182
Emerald Publishing Limited
2399-6439
DOI 10.1108/JDAL-09-2020-0018

It can also be mathematically defined as, given a complete graph $G = (V, E)$, cities are represented via the graph vertices $\nu \in V$, and edges $e \in E$ represent the paths between the cities where the edge weights are the distances between each city: What is the shortest tour that visits all vertices once and returns to the starting vertex? When traditional linear programming methods were applied to the TSP, intractability issues arose (Applegate *et al.*, 2006). It has since been shown that this is because the TSP falls into a class of known computationally "hard" problems called NP-hard (Rego *et al.*, 2011). No one has yet developed an efficient method for universally solving large instances of NP-hard problems to optimality (Khan *et al.*, 2012). The inclusion of the TSP in the set of NP-hard problems motivates the usage of other solving techniques such as heuristics.

One class of heuristics uses a greedy methodology, where the best immediate choice, according to a predefined parameter, is selected at each step of the method. An example of a greedy heuristic for the TSP is the multiple-fragment (MF) heuristic (Bentley, 1992), where the shortest available edge is iteratively added to form a tour. However, this greedy heuristic runs the risk of generating subtours, or disconnected tours of less than size N (where $N = |V|$), which prevent a single continuous tour from being formed. Some research has been completed to develop methodologies that avoid subtours when using the edge-greedy heuristic (Bentley, 1992; Wang *et al.*, 2018).

This paper makes the following contributions:

- First, this research further defines the class of fragment heuristics by clarifying vertex-greedy vs edge-greedy methods.
- Second, this research bifurcates the choice of greedy methodology from the subtour elimination method, thus highlighting the importance of subtour elimination methodologies within this class of fragment heuristics.
- Third, the research proposes a novel subtour elimination methodology, dubbed the greedy tracker (GT), for the edge-greedy heuristic and compares it to two known subtour elimination methodologies.
- Fourth, extensions to all three subtour elimination methodologies are introduced to allow them to handle asymmetric instances.
- Finally, motivated by the GT subtour elimination method, the research introduces the concept of ordered lists which remaps the TSP problem into a new space with promising initial computational results.

2. Literature review

2.1 Types of traveling salesman problem construction heuristics

Bentley's (1992) paper, in addition to many other contributions, established classifications for TSP constructive heuristics. Bentley breaks TSP construction heuristics into three general categories: heuristics that grow fragments, heuristics that grow tours and heuristics based on trees. The latter two classes each ensure viable TSP tours using internally consistent methodologies. For example, the class "heuristics that grow tours" contains heuristics that follow the insertion or addition expansion rules. These heuristics each start with a partial tour consisting of a single vertex which is then gradually grown by adding new vertices. This is accomplished by deleting an edge and adding two new edges, according to a specified rule-set, to connect the new vertex to the current tour, ensuring a valid TSP tour is constructed. However, the "heuristics that grow fragments" class does not have a consistent methodology to avoid premature partial circuits or subtours.

2.2 Heuristics that grow fragments

In this section, we briefly introduce the heuristics described in the "Heuristics that Grow Fragments" section of Bentley's (1992) paper.

2.2.1 Nearest neighbor. The nearest neighbor (NN) heuristic was first applied to the TSP in a 1954 paper by Flood (1956) as the "next closest city method." The process was later refined by Dacey (1960) who coined its current name. The NN starts at an arbitrary city, and successively visits the closest unvisited city. Note that the NN heuristic maintains a single path fragment that originates at the predetermined starting city, and cannot be closed into a cycle until every vertex has been visited. Therefore the decision of "which edge to add" is limited to only those edges that leave the current tail of the fragment, yielding an algorithm run time of $O(N^2)$. Future work by Bentley (1992) allowed this heuristic to perform in $O(N \log N)$. This methodology allows NN to quickly create an initial tour which avoids subtours. However, NN is extremely sensitive to the choice of starting vertex especially in larger instances. Bentley also introduces a second variant of NN called the double-ended NN which allows the fragment to grow from both ends.

2.2.2 Multiple-fragment. The MF heuristic was first introduced by Papadimitriou and Steiglitz (1982) as a modification of a process first seen in a 1968 paper by Steiglitz and Weiner (Steiglitz, 1968). The heuristic is a more complex greedy TSP heuristic where all edges of the graph are sorted from shortest to longest. Edges are then added to the tour starting with the shortest edge as long as the addition of this edge will not make it impossible to complete a tour. Specifically, this means avoiding adding edges that make early cycles, and also avoiding creation of vertices of degree three. This process, as originally proposed, required $O(N^2 \log N)$ time. However, Bentley was able to speed up this process to $O(N \log N)$ (Bentley, 1992) in a paper introducing his MF version. This yields a similar run time to NN while maintaining a similar worst-case solution quality. MF's tour construction methodology causes the heuristic to only produce a single solution for each instance where NN can arrive at different solutions based on a different starting point. When compared to the average NN solution over all starting points, MF tends to outperform NN on an instance-to-instance basis (Bentley, 1992; Okano *et al.*, 1999).

3. Clarifying the "greedy heuristic"

At present, it is entirely unclear when referencing the "Greedy Heuristic" for the TSP if the heuristic under discussion is NN or MF. This is acknowledged in Aarts *et al.* (2003). Much of this confusion seems to draw from the poor naming conventions used with respect to how a greedy methodology works. Our first contribution is creating a framework for greedy heuristics which aligns with the framework established by Talbi (2009). Talbi notes that constructive heuristics involve two choices: First, determine a set of elements, $S_j = \{e_{1,j}, e_{2,j}, \ldots, e_{p,j}\}$, which comprise the neighborhood of the current solution (i.e. the set of possible choices at each iteration j). Second, define a methodology to choose an element $e_{i,j}$ from this set S_j. Thus, the framework of how this set S_j is defined is crucial to how the heuristic constructs the solution. In the case of TSP fragment heuristics, we propose a more comprehensive classification that splits this class into two greedy-type methodologies: vertex-greedy and edge-greedy heuristics based on how the set S_j is defined.

3.1 Vertex-greedy

In the vertex-greedy TSP fragment heuristic sub-class, the set of edges available at each iteration is defined by a given vertex. The key defining characteristic of this sub-class is that the set S_j is limited to the edges incident to a given vertex. Depending on the measure of merit being greedily optimized, this sub-class results in different heuristics. NN is one such example of this vertex-greedy sub-class, wherein the starting vertex is user defined. The elements of S_j are subsequently defined as the edges incident to the vertex at the head of the fragment. For NN the measure of merit driving the choice of element from Sj is the shortest edge.

3.2 Edge-greedy

The edge-greedy TSP fragment heuristic considers the set of all available edges at each iteration which will not make it impossible to complete a tour. The first criterion to narrow the scope of this set is to identify all edges that will not cause a vertex to have a degree of more than 2 when added. The second criterion, which is much more difficult, is to identify which edges will cause a tour of less than size N (i.e. a subtour) to form. An edge is then chosen from this reduced set based on the measure of merit being greedily optimized. As with the vertex-greedy sub-class, this sub-class can produce multiple different heuristics depending on the measure of merit.

Part of the confusion in the literature when referencing the "Greedy Heuristic" revolves around Bentley's MF being viewed as what we describe as *the* edge-greedy heuristic. However, we argue the true significance of Bentley's MF is not the edge-greedy methodology he proposes but rather the development of a subtour tracking and elimination methodology, which could be applied to any member of the edge-greedy class of TSP constructive heuristics.

4. Subtour tracking and elimination methodologies

In addition to Bentley's MF methodology (Bentley, 1992), a well-known, though to the best of our knowledge, previously undocumented methodology we dub exhaustive loop (EL) can also be used to track and eliminate subtours for edge-greedy TSP constructive heuristics. This paper introduces a third novel method we call the GT.

4.1 Symmetric versus asymmetric instances

TSP instances can be either symmetric or asymmetric; similarly subtour tracking methodologies can be either directional or non-directional. Non-directional subtour tracking methodologies construct a tour with no regard to the direction of travel for each edge while ensuring no vertex has a degree of more than 2. This methodology can only be used with symmetric TSP instances. This allows some computational advantages as only $N^*(N+1)/2$ edges need to be initially sorted. Directional subtour tracking methodologies can be used on either symmetric or asymmetric instances when the direction of edge travel is either of importance to the final solution and/or takes different costs to travel based on direction. In a directional scenario, all edges of each direction $N^2 - N$, are sorted from shortest to longest and rather than tracking the total degree of each vertex, each vertex can only be entered and left once, ensuring a continuous direction throughout the tour.

4.2 Multiple-fragment

The non-directional variant of MF is well documented throughout TSP literature (Bentley, 1992; Wang *et al.*, 2018; Okano *et al.*, 1999). However, a formalization of the directional variant, required for asymmetric instances, is missing from this literature. This process appears to have been used in a 1999 paper by Glover *et al.* (2001). However, neither pseudocode nor an explicit description of altering the methodology for directional instances is available. The primary alterations are to consider all edges of each direction and splitting the degree array into “To” and “From” arrays. The process then continues as described by Bentley (1992), where after each edge is added, the tails of the associated fragment are updated to ensure no subtours are formed. Pseudocode for this modified methodology is included as Algorithm 1.

Algorithm 1 Multi-Fragment (Directional) Pseudocode

```
1: Sort edges (i, j) : Shortest to Longest
2: while Number of visited vertices < N − 1 do
3:   if next shortest edge (i, j) is not in tour and no edges leave i or j
     then
4:     add edge (i, j) to tour
5:     if no other edges in tour enter or leave vertices i or j then
6:       Set vertex j as the tail of vertex i
7:       Set vertex i as the head of vertex j
8:     else if Another edge enters vertex i and no edges leave vertex j
       then
9:       Set the vertex on the end of the fragment entering vertex i as
         head of vertex j
10:      Set vertex j as the tail of the vertex on the end of the fragment
         entering vertex i
11:    else if No edge enters vertex i and an edge leave vertex j then
12:      Set the vertex at the end of the fragment leaving vertex j as
         the tail of vertex i
13:      Set vertex i as the head of the vertex at the end of the fragment
         leaving vertex j
14:    else if Another edge enters vertex i and an edge leave vertex j
       then
15:      Edge (i, j) is connecting two fragments. (i.e. Set the vertex
         at the end of the   fragment entering vertex i as the head of
         the end of the fragment leaving vertex j, and   vice versa.)
16:    end if
17:  end if
18:  Increment number of visited vertices
19: end while
```

4.3 Exhaustive loop

The EL is not well documented in academic literature, often simply referenced as “the standard way.” To the best of our knowledge, no formal coverage of this method currently exists in literature. EL cycles through every edge in the fragment containing the most recently added edge. Once an edge e_{ij} is added to the partial tour, vertex i is identified as the “start vertex” and vertex j will be set as “current vertex.” A trace along the current partial tour then begins. At each step of the trace, the “current vertex,” vertex j, is checked to see if it is connected to another vertex k via edge e_{jk} in the partial tour. If it is, then vertex k becomes the new “current vertex.” If the trace returns back to the “start vertex” in under N steps, where $N = –V–$ (the number of vertices in the instance), then the added edge e_{ij} has created a subtour and is an illegal edge. If no edge leaves the “current vertex,” the addition of edge e_{ij} is valid and the current portion of the tour is still a fragment. Each time an edge is added, a count is incremented and the process continues until N −1 edges have been added upon which the last two endpoints are connected.

The EL can also be modified to handle a directional methodology by splitting the "Degree" array into a "To" and "From" array, and rather than checking if current vertex has a degree of 2, check if the current vertex has a value of 1 in "From" array.

4.4 Greedy tracker

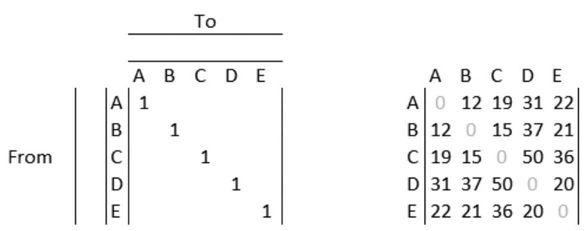
We introduce a novel way to track the progress of the edge-greedy construction heuristic, and ensure subtours are not created. This new method is called the "greedy tracker." The GT tracks a vertices' connection with other vertices when constructing a TSP tour. While we define both non-directional and directional GT variants, it is conceptually easier to visualize the GT using its directional variant on a symmetric instance and then generalize the process for use on asymmetric instances or to the non-directional variant. Thus, the following introduction to the GT uses the directional variant on a symmetric matrix and is accomplished using the following structures:

- X = binary n by n matrix of x_{ij};
- F = binary n by 1 array of f_i;
- T = binary n by 1 array of t_i;
- $x_{ij} = 0$ if edge from i to j is eligible, greater than 0 if not eligible;
- f_i = binary for whether vertex i has been left; and
- t_i = binary for whether vertex i has been entered.

These structures track each move to prevent subtours. Given our prior example, the initial condition of GT, and associated structures, can be seen in Figure 1.

The X (identity), F (from) and T (to) structures can be seen above on the left and, for ease of reference, the associated distance matrix from the TSP instance can be seen on the right. The 1s loaded on the diagonal of the X matrix (where $i = j$) signal ineligible moves. Note that the diagonal on the distance matrix has been colored grey, and set to zero, to correspondingly show these ineligible edges. The distance matrix indicates that the shortest edge is either from A to B or vice versa, thus edge A to B is selected. The X, F and T matrices are updated with 1s to indicate this move, as shown in Figure 2.

Next, the column of the X matrix associated with the new edge is processed. Every row where a 1 appears is combined with the "From" row of the created edge. Figure 3 illustrates this operation. As seen in Figure 3, as row 2 has a 1 in the same column as our new edge, the two rows were combined so that any 1s that were in row 1 are now also in row 2. Note that for the example we only show values of 1 so as to not detract from their



		To				
		A	B	C	D	E
From	A	1				
	B		1			
	C			1		
	D				1	
	E					1

	A	B	C	D	E
A	0	12	19	31	22
B	12	0	15	37	21
C	19	15	0	50	36
D	31	37	50	0	20
E	22	21	36	20	0

Figure 1.
GT example – initialization

Note: X matrix, to and from arrays are on left, and distance matrix is on the right for ease of reference. "1s" on diagonal of X matrix indicate illegal edges purpose of referring to an ineligible move. However, in the code, the values in each row are added and values of greater than 1 appear.

For ease of reference, in this example, ineligible values in the distance matrix are turned grey. As can be seen in Figure 4, distances that correspond with a 1 in the X matrix are ineligible moves. Note that any row or column that has a 1 in the T or F array is also marked as an ineligible move. This information is used in the first step of the next iteration where the shortest available edge is identified.

As seen in Figure 5, the shortest available edge is B–C and once again the X matrix, from and to arrays are updated with 1s to indicate the move.

From		A	B	C	D	E
To			**1**			
1	A	1	**1**			
	B		1			
	C			1		
	D				1	
	E					1

	A	B	C	D	E
A	0	$\boxed{12}$	19	31	22
B	12	0	15	37	21
C	19	15	0	50	36
D	31	37	50	0	20
E	22	21	36	20	0

Note: Bold 1s in the to and from array and in X matrix represent the addition of edge A–B to tour, also seen with framed 12 in distance matrix

Figure 2. GT example – iteration 1, step A

From		A	B	C	D	E	
To			**1**				
1	A	1	**1**				
	B	**1**	1				= R1 + R2
	C			1			
	D				1		
	E					1	

	A	B	C	D	E
A	0	$\boxed{12}$	19	31	22
B	12	0	15	37	21
C	19	15	0	50	36
D	31	37	50	0	20
E	22	21	36	20	0

Note: Addition of edge A–B propagates additional 1s in X matrix to capture illegal edges

Figure 3. GT example – iteration 1, step B

From		A	B	C	D	E
To			1			
1	A	1	1			
	B	1	1			
	C			1		
	D				1	
	E					1

	A	B	C	D	E
A	0	12	19	31	22
B	12	0	15	37	21
C	19	15	0	50	36
D	31	37	50	0	20
E	22	21	36	20	0

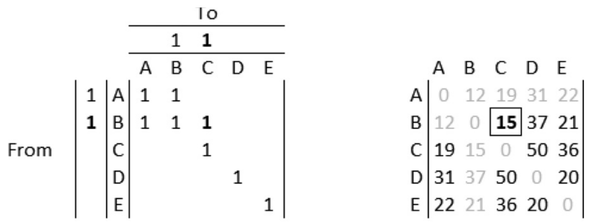
Note: The illegal edges can be shown for ease of reference as greyed out numbers in distance matrix

Figure 4. GT example – iteration 1, step B

The column of X associated with the “To” vertex of the new edge is processed and every row where a 1 appears is combined with the “From” row of the created edge which can be seen in Figure 6.

All the distances that correspond with a 1 in the X matrix are marked as ineligible moves in the distance matrix, as well as any distances associated with a 1 in the T and F arrays. The resulting step can be seen in Figure 7.

The grey numbers in the distance matrix indicates that adding edge A–C is no longer possible because vertex C already has an edge entering it. This process prevents the formation of the subtour. The process shown above continues until all vertices have been visited which creates a Hamiltonian path. The final connection to complete the tour is made using the to and from arrays as each has an index that is still empty.

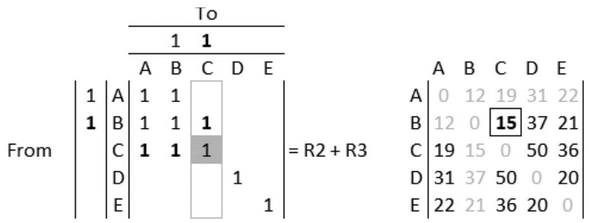


		To				
			1	**1**		
		A	B	C	D	E
1	A	1	1			
1	B	1	1	**1**		
From	C			1		
	D				1	
	E					1

	A	B	C	D	E
A	0	12	19	31	22
B	12	0	$\boxed{15}$	37	21
C	19	15	0	50	36
D	31	37	50	0	20
E	22	21	36	20	0

Figure 5.
GT example – iteration 2, step A

Note: Edge B–C is added to tour; the to array, from array and X matrix are updated

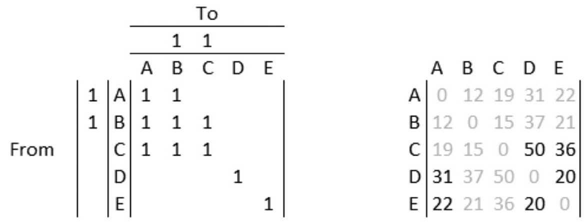


		To					
			1	**1**			
		A	B	C	D	E	
1	A	1	1				
1	B	1	1	**1**			
From	C	**1**	**1**	1			= R2 + R3
	D				1		
	E					1	

	A	B	C	D	E
A	0	12	19	31	22
B	12	0	$\boxed{15}$	37	21
C	19	15	0	50	36
D	31	37	50	0	20
E	22	21	36	20	0

Figure 6.
GT example – iteration 2, step B

Note: Additional 1s are propagated through X matrix to prevent usage of illegal edges



		To				
			1	1		
		A	B	C	D	E
1	A	1	1			
1	B	1	1	1		
From	C	1	1	1		
	D				1	
	E					1

	A	B	C	D	E
A	0	12	19	31	22
B	12	0	15	37	21
C	19	15	0	50	36
D	31	37	50	0	20
E	22	21	36	20	0

Figure 7.
GT example – iteration 2, step B

Note: For ease of reference, illegal edges in distance matrix are greyed out

Real-world implementation of GT requires adjustments to achieve slower computational growth by reducing the total number of operations that occur within each iteration. This is achieved by removing the addition of values with respect to vertices that have been exited and entered. This decreases the dimensionality of the GT as the tour is constructed and is possible because once a vertex has been entered (respectively, exited) no more edges may enter (respectively, exit) that vertex. Therefore, it is unnecessary to track what edges could produce a subtour by entering (respectively, exiting) that vertex. Consider for example the same five-by-five instance. After completing the row additions after adding edge A–B, column B can be deleted. Figure 8 shows the resulting GT and distance matrix.

This process can also be applied to rows to further reduce problem dimensionality. When working with a non-directional instance, a column and row would be deleted after the vertex had a degree of 2.

These row and column deletions are a key feature. Fundamentally, as a new edge is added to the tour, the GT is identifying which edge(s) would form a subtour for the fragment including that new edge. Bentleys MF on the other hand tracks the tails of every fragment and prevents the head and tail of a fragment from connecting. There appears to be some advantage in GTs approach as only one number is added to the X matrix at each step. However, the per step computational burden to find that one number may be higher as GT must search for and identify the shortest eligible edge, whereas while MF references more vertices (tails), there is no searching necessary.

5. Results and analysis

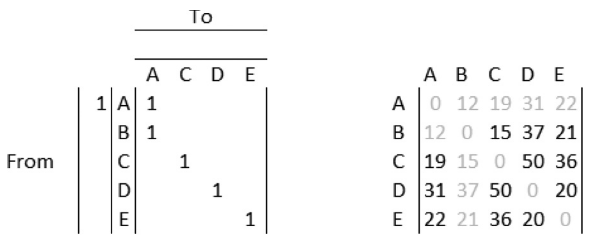
To compare the run times for the MF, EL and GT subtour elimination methods, all three methods were run across 17 symmetric and 9 asymmetric TSP instances.

5.1 Traveling salesman problem instances

TSP instances are available in an online library, TSPLIB, maintained by Ruprecht-Karls-Universitat Heidelberg located in Baden-Wurttemberg, Germany (TSPLIB, 2018). For the purposes of this research, testing was performed on the instances seen in Table 1, where the alpha prefix is an identifier and the numerical suffix indicates the instance size (in number of vertices).

5.2 Testing

Initial tests verified that each subtour method (MF, EL and GT) resulted in the same tour for all TSP instances. These tests were conducted with both directional and non-directional versions of codes on symmetric TSP instances. In addition, the directional code versions were run on the asymmetric TSP instances.



			To			
			A	C	D	E
From	1	A	1			
		B	1			
		C		1		
		D			1	
		E				1

	A	B	C	D	E
A	0	12	19	31	22
B	12	0	15	37	21
C	19	15	0	50	36
D	31	37	50	0	20
E	22	21	36	20	0

Note: Delete column B as vertex B has been entered

Figure 8. GT revised iteration 1, step B

Table 1.
TSP instances used in current testing

TSP instances	
Symmetric	Asymmetric
bays29	br17
gr48	ry48p
gr51	ft53
berlin52	ft70
pr76	kro124p
kroa100	rgb323
gr120	rgb358
gr130	rgb403
gr195	rgb443
ts225	
pma343	
pcb442	
dsj1000	
pr1002	
pr2392	
fnl4461	
rl5934	

Once testing verified each subtour elimination method produced identical greedy tours, remaining testing focused on computational run-time comparisons. Each subtour elimination method was placed in the same edge-greedy heuristic shell so that testing would fairly compare the speed of the three subtour tracking and elimination methodologies. Bentley (1992) and Wang *et al.* (2018) each used advanced computer techniques (i.e. *k-d* trees) and additional data structures to speed up the process of finding the next shortest available edge. However, as these do not affect the speed of the subtour tracking and elimination methodologies, they were not used.

Speed tests were conducted using the R package "microbenchmark." A total of 100 iterations of each method were run per instance to create summary statistics on 17 different symmetric TSP instances and 9 asymmetric instances. Both symmetric and asymmetric instances were tested to determine if symmetry effected run time.

5.3 Symmetric instance results

Table 2 provides summary statistics for the 17 symmetric instances. The mean run time is provided on first line, with associated standard deviation below.

When looking at the directional variants, the GT is the fastest method on small instances followed by EL then MF. Once instance size is 442 or larger, MF and GT are undifferentiated. However, MF does begin to consistently produce lower means then the other methods. This is probably because of its linear growth in operation count as instance size grows. GT does not necessarily possess this characteristic but the dimensionality reduction of GT may prove a similar role.

Interestingly, all three non-directional variants run slower than their directional counterparts up to around the 100 vertex instances. As with the directional variants, GT is the fastest method till instance size 225, after which MF becomes the fastest.

5.4 Asymmetric instance results

The directional variants of each subtour elimination codes were also run on asymmetric TSP instances to compare runtimes to determine if any trends changed. The mean run times and associated standard deviations are in Table 3.

Milliseconds		Directional			Non-directional		
Instance	Stat	EL	MF	GT	EL	MF	GT
bays29	Mean	23.2	52.0	22.4	29.7	44.2	24.4
	Std dev.	0.7	0.8	1.0	0.9	0.8	0.7
gr48	Mean	25.4	56.1	25.0	30.5	46.8	25.8
	Std dev.	0.8	0.8	0.8	0.7	1.1	0.8
eil51	Mean	26.7	53.2	24.4	32.3	50.4	28.1
	Std dev.	1.5	0.8	0.8	0.7	1.3	1.1
berlin52	Mean	26.6	53.1	24.1	31.4	47.9	26.3
	Std dev.	1.2	0.8	0.8	0.7	1.4	0.8
pr76	Mean	30.8	56.9	28.5	32.8	46.9	29.9
	Std dev.	1.3	0.8	1.3	0.6	0.6	1.3
kroa100	Mean	36.8	60.8	34.5	36.5	50.4	30.8
	Std dev.	1.6	1.2	0.6	1.1	1.3	0.8
gr120	Mean	41.9	66.5	43.1	40.5	52.2	34.9
	Std dev.	0.9	1.9	1.9	1.4	1.0	1.1
ch130	Mean	48.9	70.4	43.7	43.3	52.9	35.7
	Std dev.	1.5	1.2	1.8	1.7	1.2	0.8
rat195	Mean	76.8	98.6	73.6	54.5	65.0	50.3
	Std dev.	1.6	2.2	2.7	1.1	0.7	1.2
ts225	Mean	97.8	113.9	95.4	61.8	68.1	57.3
	Std dev.	2.2	3.1	3.5	1.7	0.8	2.9
pma343	Mean	193.7	193.9	177.0	121.2	111.9	107.7
	Std dev.	3.4	3.4	3.5	3.0	2.2	4.5
pcb442	Mean	363.7	312.8	317.2	177.0	166.1	180.5
	Std dev.	4.4	4.4	4.1	2.3	2.1	4.7
dsj1000	Mean	1.667(s)	1.341(s)	1.440(s)	906.9	660.1	750.2
	Std dev.	3.7	4.6	3.5	7.8	5.6	8.1
pr1002	Mean	1.595(s)	1.308(s)	1.374(s)	863.3	637.3	761.8
	Std dev.	5.4	5.0	4.7	3.8	4.0	17.1
pr2392	Mean	9.268(s)	7.214(s)	7.611(s)	4.900(s)	3.405(s)	3.796(s)
	Std dev.	20.4	24.2	22.9	15.6	9.8	20.6
fnl4461	Mean	27.416(s)	23.096(s)	24.198(s)	16.752(s)	11.268(s)	12.286(s)
	Std dev.	42.3	48.2	53.1	49.0	40.9	39.6
rl5934	Mean	72.697(s)	51.579(s)	54.703(s)	41.937(s)	22.301(s)	25.171(s)
	Std dev.	0.042(s)	5.789(s)	5.368(s)	4.113(s)	0.742(s)	2.616(s)

Note: For each instance, a mean of 100 runs is provided on first line and standard deviation on second line

Table 2. Greedy subtour methodology run times (symmetric instances)

For the asymmetrical instances, we note that GT is the fastest method through instance size 124. MF became the fastest methodology when the instance size reached 358 vertices, but was tied by the GT at an instance size of 403 vertices. This leads us to believe that some of the subtour methodologies may have computational advantages for specific TSP instances dependent on how the tour is constructed. For example, specific instances may have vertices spaced in such a fashion that for a majority of the tour construction, the tracking methodology is maintaining a small number of large fragments. It is possible that one of the methodologies is computationally more efficient for these instances and may be computationally slower in TSP instances where vertices force an edge-greedy tour construction that results in many small fragments to be tracked. Future analysis of instance geometry and resulting run times should be conducted to test whether this hypothesis holds any merit. With the exception of the variation seen for the 403 vertex instance, prior overall trends from the symmetric instances remain, where GT is

Table 3.
Greedy subtour methodology run times (asymmetric instances)

Milliseconds		Directional		
Instance	Stat	EL	MF	GT
br17	Mean	22.2	52.5	21.1
	Std dev.	0.7	1.0	0.8
ry48p	Mean	24.5	53.9	23.2
	Std dev.	0.8	1.4	0.8
ft53	Mean	25.6	54.0	24.7
	Std dev.	0.9	1.4	0.8
ft70	Mean	28.5	56.5	27.4
	Std dev.	0.8	1.4	0.8
kro124p	Mean	35.9	64.4	34.6
	Std dev.	1.4	1.9	0.7
rgb323	Mean	230.3	195.5	199.1
	Std dev.	5.8	5.2	6.5
rgb358	Mean	242.7	219.7	231.3
	Std dev.	2.8	2.8	2.5
rgb403	Mean	311.9	275.2	274.5
	Std dev.	3.7	3.5	3.8
rgb443	Mean	366.5	315.3	330.5
	Std dev.	2.9	2.5	2.6

Note: For each instance, a mean of 100 runs is provided on first line and standard deviation on second line

competitive for small- to medium-sized asymmetric instances, but MF is fastest for larger instances.

6. Ordered greedy heuristic

The separation of the subtour tracking methodology from the underlying greedy heuristic methodology is not just pedantic. This realization allows researchers to use subtour elimination methodologies to develop new greedy-fragment heuristics to build viable TSP tours. This research concludes with a new constructive heuristic called the ordered-greedy (OG) heuristic. The OG heuristic is a vertex-greedy heuristic that takes as input a complete ordered list of vertices. Starting at the top of the list, each vertex is considered in turn and the available set of choices S_j at each step is limited to the feasible edges originating at that vertex. What differentiates the OG from NN, another vertex-greedy heuristic, is that multiple fragments may exist during the tour construction of OG.

The motivation of the OG heuristic is to apply a more structured approach in which vertices are given priority in connecting to their NNs. Vertices higher in the list have maximum flexibility with minimal concern for vertex degree or subtours and thus typically choose better edges than vertices later in the list which experience significantly fewer degrees of freedom in their legal edge choices. The quality of the solution found is thus heavily dependent on the order of the list.

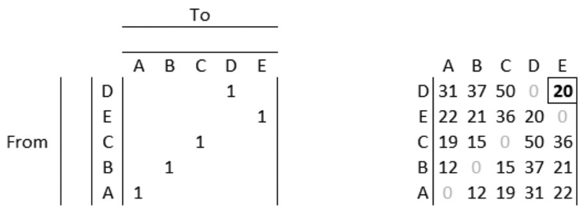
To introduce the methodology of the OG heuristic, consider the following example. In this example, an ordered list of D, E, C, B and A has been, through some unspecified fashion, predetermined. This ordering of the vertices list is reflected, for ease of reference, in the X matrix and to and from arrays on the left side, and in the distance matrix on the right side of Figure 9 whose rows are now sorted according to this list order. The constructive heuristic now makes greedy decisions starting at the top of this list and working down. The first greedy decision is made with respect to vertex D.

The greediest or shortest edge, from vertex D is edge D–E as indicated above. This edge and its associated vertex are tracked via the GT. The next step is made with respect to vertex E. This is not because of vertex E being the head of the previous edge added, but rather because it is the second vertex in the provided ordered list: D, E, C, B and A. Looking at the row in the distance matrix associated with vertex E along with the GT output that captures ineligible moves (as seen in Figure 10), the shortest legal edge available is edge E–B.

This process continues row by row until the final row is reached which is where the to and from arrays are scanned to find the final legal edge as seen in Figure 11.

After adding edge A–D, the resulting tour becomes A–D–E–B–C–A, which is also the optimal tour for this TSP instance.

The OG heuristic is in-and-of itself of marginal interest when viewed as another NN style vertex greedy construction heuristic. It is more intriguing when viewed as a general function which takes as input an ordered list and outputs a tour. This OG function is neither one-to-



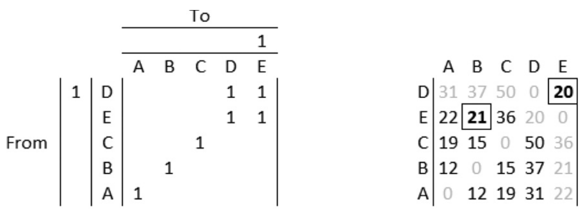
To

From		A	B	C	D	E
	D				1	
	E					1
From	C			1		
	B		1			
	A	1				

	A	B	C	D	E
D	31	37	50	0	**20**
E	22	21	36	20	0
C	19	15	0	50	36
B	12	0	15	37	21
A	0	12	19	31	22

Note: OG picks greediest legal edge incident to vertex D

Figure 9. OG example – iteration 1



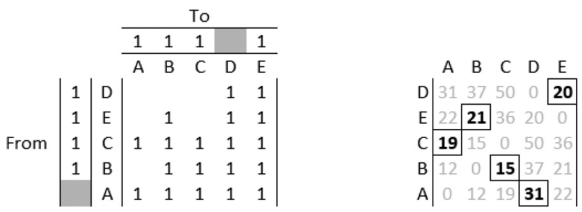
To

		A	B	C	D	E
						1
1	D				1	1
	E				1	1
From	C			1		
	B		1			
	A	1				

	A	B	C	D	E
D	31	37	50	0	**20**
E	22	**21**	36	20	0
C	19	15	0	50	36
B	12	0	15	37	21
A	0	12	19	31	22

Note: OG picks greediest legal edge incident to vertex E

Figure 10. OG example – iteration 2



To

		A	B	C	D	E
		1	1	1	▨	1
1	D				1	1
1	E		1		1	1
1	C	1	1	1	1	1
1	B		1	1	1	1
▨	A	1	1	1	1	1

From

	A	B	C	D	E
D	31	37	50	0	**20**
E	22	**21**	36	20	0
C	**19**	15	0	50	36
B	12	0	**15**	37	21
A	0	12	19	**31**	22

Note: X matrix on the left and reordered distance matrix on the right

Figure 11. OG step 3

one, nor onto. In other words, while each ordered list uniquely maps to a tour, multiple ordered lists may map to the same tour. Thus the range of this mapping is a proper subset of the set of all tours. This naturally leads to a question of quality. The size of the ordered list space and the size of the tour space are identical (for the symmetric Euclidean TSP both are $\frac{(N-1)!}{2}$), hence the questions:

Q1. Is exploring the ordered list space, in some fashion, worth it?

Q2. Do the tours generated by the OG heuristic for random ordered lists represent a sufficient improvement over randomly generated tours?

To answer these questions, we took five symmetric TSP instances and two asymmetric instances. From each instance, we randomly selected nine nodes. We exhaustively generated all string permutations for these nine nodes and examined these strings both as tours and as input to the OG function. The results of this analysis can be seen in Table 4. As expected, the mean and even maximum tour lengths decreased as a result of the OG function. The most striking, and most promising, outcome however was the number of optimal length tours the OG function created. This result strongly suggests that searching through the ordered list space may prove beneficial. However, significant additional testing is required to determine if exploration of the ordered list space is sufficiently desirable to overcome the, albeit minimal, cost of calling the OG function.

Table 4. Exhaustive search results for tours vs output of OG function on nine random vertices drawn from five symmetric and two asymmetric instances

Symmetric		Tour	OG
eil51	Mean	241.3	174.7
	Max	316	239
	Number opt	18	18,180
gr120	Mean	2476.0	1981.1
	Max	3,267	2,677
	Number opt	18	3,156
rat195	Mean	309.0	193.3
	Max	410	310
	Number opt	198	13,586
ts225	Mean	15,000.00	9153.5
	Max	20,000	14,000
	Number opt	1,152	177,218
pma343	Mean	93.5	70.4
	Max	113	97
	Number opt	18	10,160
Asymmetric		Tour	OG
br17	Mean	163.3	82.1
	Max	295	166
	Number opt	288	34,316
ft70	Mean	6488.0	5328.1
	Max	7,613	6,378
	Number opt	9	9,504

Note: Mean tour length, maximum tour length and number of occurrence of optimal tour length are provided

7. Conclusion

As an NP-hard combinatorial optimization problem, the TSP is often solved via heuristic methodologies. One of the biggest considerations when constructing solutions is avoiding subtours, or a loop of interconnected vertices that prevents a single continuous tour among all cities within the instance. This paper introduced a novel subtour elimination methodology for the edge-greedy heuristic that is compared to two known subtour elimination methodologies. Computational results were generated across multiple TSP instances for each method.

When using an edge-greedy type heuristic, additional steps must be taken to ensure that subtours are avoided and resulting tour is a valid TSP solution. This paper recognized two accepted edge-greedy subtour elimination methodologies, the EL and Bentley's MF, and compared them to our GT. The comparison used both directional and non-directional variants of each code on 14 symmetric TSP instances and the directional variants on 9 asymmetric instances.

The results of the comparison between each of these edge-greedy subtour elimination methodologies showed that the GT was the fastest tracking methodology for small- to medium-sized instances. However, Bentley's MF maintains the computational advantage for larger instances.

However, these results also indicated that given a more efficient coding implementation of methodology used for the X matrix, the GT could become the preferred methodology for all instance sizes. For future research, the GT should be modified to handle a new row/column generation and delete technique to minimize the computational time used in the searching portions of the GT.

References

Aarts, E., Aarts, E.H. and Lenstra, J.K. (2003), *Local Search in Combinatorial Optimization*, Princeton University Press.

Applegate, D.L., Bixby, R.E., Chvatal, V. and Cook, W.J. (2006), *The Traveling Salesman Problem: A Computational Study*, Princeton University Press.

Bentley, J.J. (1992), "Fast algorithms for geometric traveling salesman problems", *ORSA Journal on Computing*, Vol. 4 No. 4, pp. 387-411.

Dacey, M.F. (1960), "Letter to the editor-selection of an initial solution for the traveling-salesman problem", *Operations Research*, Vol. 8 No. 1, pp. 133-134.

Flood, M.M. (1956), "The traveling-salesman problem", *Operations Research*, Vol. 4 No. 1, pp. 61-75.

Glover, F., Gutin, G., Yeo, A. and Zverovich, A. (2001), "Construction heuristics for the asymmetric TSP", *European Journal of Operational Research*, Vol. 129 No. 3, pp. 555-568.

Khan, A.A., Khan, M.U. and Iqbal, M. (2012), "Multilevel graph partitioning scheme to solve traveling salesman problem", *2012 Ninth International Conference on Information Technology-New Generations, IEEE*, pp. 458-463.

Okano, H., Misono, S. and Iwano, K. (1999), "New TSP construction heuristics and their relationships to the 2-opt", *Journal of Heuristics*, Vol. 5 No. 1, pp. 71-88.

Papadimitriou, C.H. and Steiglitz, K. (1982), *Combinatorial Optimization*, Prentice Hall, Englewood Cliffs Vol. 24.

Rego, C., Gamboa, D., Glover, F. and Osterman, C. (2011), "Traveling salesman problem heuristics: leading methods, implementations and latest advances", *European Journal of Operational Research*, Vol. 211 No. 3, pp. 427-441.

Steiglitz, K. (1968), "Some improved algorithms for computer solution of the traveling salesman problem", *Proceedings of the 6th Annual Allerton Conference on Communication, Control, and*

Computing, 1968, Department of Electrical Engineering and the Coordinated Science Laboratory.

Talbi, E.-G. (2009), *Metaheuristics: From Design to Implementation*, John Wiley and Sons Vol. 74.

TSPLIB (2018), available at: www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/ (accessed 15 August 2018).

Wang, S., Rao, W. and Hong, Y. (2018), "A distance matrix based algorithm for solving the traveling salesman problem", *Operational Research*, pp. 1-38.

Corresponding author
Bruce Cox can be contacted at: bruceacox1@gmail.com