

The Traveling Salesman Problem

Introduction

The Traveling Salesman Problem (TSP) is perhaps the most studied discrete optimization problem. Its popularity is due to the facts that TSP is easy to formulate, difficult to solve, and has a large number of applications. It appears that K. Menger [31] was the first researcher to consider the Traveling Salesman Problem (TSP). He observed that the problem can be solved by examining all permutations one by one. Realizing that the complete enumeration of all permutations was not possible for graphs with a large number of vertices, he looked at the most natural *nearest neighbor* strategy and pointed out that this *heuristic*, in general, does not produce the shortest route. (In fact, the nearest neighbor heuristic will generate the worst possible route for some problem instances of each size [17].) For interesting overviews of TSP history, see [20, 40].

Basic Definitions and Notation. In applications, both the symmetric and asymmetric versions of the TSP are important. In the *Symmetric TSP* (*STSP*), given a complete (undirected) graph K_n with weights on the edges, our aim is to find a hamiltonian cycle in K_n of minimum weight (the weight a cycle is the sum of the weights of its edges). In the *Asymmetric TSP* (*ATSP*), given a complete directed graph K_n^* with weights on the arcs, find a hamiltonian cycle in K_n^* of minimum weight. The *Euclidean TSP* is a special case of *STSP* in which the vertices are points in the Euclidean plane and the weight on each edge is the Euclidean distance between its endpoints. A hamiltonian cycle in K_n or K_n^* is often called a *tour*. Notice that *ATSP* has $(n - 1)!$ tours, but *STSP* has $(n - 1)!/2$ tours (changing the direction of

the tour in *STSP* does not change the tour). By *TSP* we refer to both *STSP* and *ATSP* simultaneously.

Throughout this entry, the set $[n] = \{1, 2, \dots, n\}$ denotes the vertices of K_n or K_n^* or any other n -vertex graph under consideration. The weight of an edge (arc) ij is denoted by w_{ij} or $w(i, j)$. We also call w_{ij} the *distance from i to j* and the *length of ij* . A *cycle factor* is a collection of vertex-disjoint cycles in K_n^* covering all vertices of K_n^* .

Computational Complexity. The hamiltonian cycle problem on an n -vertex graph G can be transformed into *STSP* by converting G to an edge-weighted K_n as follows: assign weight 0 to each edge of G ; and assign weight 1 to each edge in the complement of G . A similar transformation can be used for digraphs and *ATSP*. This implies that *TSP* is NP-hard, even if the triangle inequality holds. By replacing the weights 0 by 1 and the weights 1 by $1 + nr$ in this transformation, we obtain the following result:

Proposition 1. *For an arbitrary constant r , unless $P=NP$, there is no polynomial time algorithm that always produces a tour of total weight at most r times the optimal.*

It was proved in [12, 38] that even Euclidean *TSP* is NP-hard. Despite this result, there was a feeling among some researchers that the Euclidean *TSP* is somewhat simpler than the general *STSP*. More precisely, Proposition 1 does not hold for the Euclidean *TSP*. This was confirmed by Arora [1] in 1996, see Theorem 2. Mitchell [33] independently made a similar discovery a few months later (see [2]).

Theorem 2. For every $\epsilon > 0$, there is a polynomial time algorithm \mathcal{A}_ϵ that, for any instance of the Euclidean TSP, finds a tour at most $1 + \epsilon$ times longer than the optimal one.

As of this writing, the fastest algorithm \mathcal{A}_ϵ has time complexity $O(n \log n + n/\text{poly}(\epsilon))$ [42]. These \mathcal{A}_ϵ algorithms have been implemented, but, in their current form, they are not competitive with best TSP heuristics [2].

Arora's result can be generalized to d -dimensional Euclidean space for any constant d . However, the next theorem limits the scope of this generalization.

Theorem 3. [45] There exists a constant $r > 1$ such that, for the Euclidean TSP in $O(\log n)$ -dimensional Euclidean space, the problem of finding a tour that is at most r times longer than the optimal tour is NP-hard.

We finish this subsection with a result from [16] that indicates another limitation for 'approximation' ATSP algorithms. The *domination number* of an ATSP heuristic \mathcal{H} is the maximum $d(n)$ such that for each instance of ATSP on n vertices, \mathcal{H} produces a tour T which is not worse than at least $d(n)$ tours including T itself.

Theorem 4. Unless $P=NP$, there is no polynomial time ATSP heuristic of domination number at least $(n-1)! - \lfloor n - n^\alpha \rfloor!$ for any constant $\alpha < 1$.

Formulations

Perhaps, the simplest combinatorial formulation of ATSP is as follows: given an $n \times n$ -matrix $W = [w_{ij}]$ find a permutation π of $[n]$ that minimizes the sum

$$w_{\pi(n),\pi(1)} + \sum_{i=1}^{n-1} w_{\pi(i),\pi(i+1)}.$$

For STSP, we require that W is symmetric.

The earliest (and very useful) integer programming formulation of ATSP is due to Dantzig, Fulkerson and Johnson [10]. Define

$n^2 - n$ zero-one variables x_{ij} by $x_{ij} = 1$, if the tour traverses arc ij and $x_{ij} = 0$, otherwise. Then ATSP can be expressed as:

$$\begin{aligned} \min z &= \sum_{i=1}^n \sum_{j=1}^n w_{ij} x_{ij} \\ \text{such that } \sum_{i=1}^n x_{ij} &= 1, j \in [n] \\ \sum_{j=1}^n x_{ij} &= 1, i \in [n] \\ \sum_{i \in S} \sum_{j \in S} x_{ij} &\leq |S| - 1 \text{ for all } |S| < n \\ x_{ij} &= 0 \text{ or } 1, i \neq j \in [n]. \end{aligned}$$

The first set of constraints ensures that a tour must come into vertex j exactly once, and the second set of constraints indicates that a tour must leave every vertex i exactly once. These two sets of constraints ensure that there are two arcs adjacent to each vertex, one in and one out. However, this does not prevent non-hamiltonian cycles. Instead of having one tour, the solution can consist of two or more vertex-disjoint cycles (called *sub-tours*), i.e., be a cycle factor with $t \geq 2$ cycles. The third set of constraints, called *sub-tour elimination constraints*, requires that no proper subset of vertices, S , can have a total of $|S|$ arcs.

For STSP, we can get the following similar formulation:

$$\min z = \sum_{1 \leq i < j \leq n} w_{ij} x_{ij} \quad (1)$$

$$\text{such that } \sum_{i=1}^n x_{ij} = 2, j \in [n] \quad (2)$$

$$\sum_{i \in S} \sum_{j \notin S} x_{ij} \geq 2 \text{ for all } 3 \leq |S| \leq n/2 \quad (3)$$

$$0 \leq x_{ij} \leq 1, i \neq j \in [n] \quad (4)$$

$$x_{ij} \text{ is integral for all } i \neq j \in [n] \quad (5)$$

While the Dantzig-Fulkerson-Johnson formulation of ATSP has an exponential number of sub-tour elimination constraints and, thus, of all constraints, there are other integer programming ATSP formulations that contain only a polynomial number of constraints.

One such example is the formulation of Miller, Tucker and Zemlin [32]. In this formulation, we use $(n-1)(n-2)$ additional constraints and $n-1$ additional variables. The following constraints replace the sub-tour elimination constraints in the Dantzig-Fulkerson-Johnson formulation:

$$(n-1)x_{ij} + u_i - u_j \leq (n-2) \\ \text{for all } i \neq j = 2, 3, \dots, n,$$

where u_i , $i = 2, 3, \dots, n$ are unrestricted real variables. If a solution is not a tour, it contains a cycle C without vertex 1. By adding the inequalities above corresponding to all arcs ij of C , we arrive at a contradiction.

Notice that the Dantzig-Fulkerson-Johnson formulation of ATSP is stronger than the Miller-Tucker-Zemlin formulation in the following sense: the optimal value of the linear relaxation of the former is larger than that of the latter [37]. As for the STSP, there are two other formulations that are as strong as the the Dantzig-Fulkerson-Johnson formulation for STSP, but only the latter have been used in computational practice. For more information on various formulations of TSP, see [40].

Applications

It appears that the most natural and well-studied application area of the TSP is machine scheduling. A simple scheduling application can be described as follows. Suppose there are n jobs $1, 2, \dots, n$ to be processed sequentially on a machine. Let w_{ij} be the set up cost required for processing job j immediately after job i . When all the jobs are processed, the machine is reset to its initial state at a cost of w_{j1} , where j is the last job processed. The aim of the *Sequencing Problem* is to find an order in which the jobs are to be processed so as to minimize the total setup cost. Observe that finding a permutation π of $[n]$ that minimizes $w_{\pi(n)\pi(1)} + \sum_{i=1}^{n-1} w_{\pi(i)\pi(i+1)}$ solves the problem. Thus, the Sequencing Problem is equivalent to ATSP.

Now consider a more interesting application introduced and studied by Gutin et al.

[15]. The *Seismic Vessel Problem (SVP)* is defined by a set of line segments (survey lines) on the plane, all of which need to be traversed exactly once. Some lines can be traversed in either direction, other have directional constraints imposed on them. The objective is to minimize the travel time between lines by choosing an optimal ordering of lines (and specifying in which direction each line has to be traversed). The function that defines the travel time between lines can be of arbitrary complexity and in general is defined as a matrix of ‘line change’ weights for all combinations of pairs of lines and traversing directions.

More formally, SVP can be stated as follows: We are given a weighted complete digraph K_n^* , whose vertices are partitioned into pairs P (representing survey lines). Each pair $\{u, v\} \in P$ is assigned a set F_{uv} such that $\emptyset \neq F_{uv} \subseteq \{uv, vu\}$. If $F_{uv} = \{uv\}$, then we must traverse the survey line corresponding to uv from u to v , and if $F_{uv} = \{uv, vu\}$, either traversing direction is possible. Let $F = \{uv : \{u, v\} \in P, uv \in F_{uv}\}$. Every arc in F is assigned weight zero (as we must traverse all survey lines and we assume that the time of traversal of a survey line in both directions is the same). We are required to find a minimum weight Hamilton cycle in K_n^* that traverses one arc from F_{uv} for every pair $\{u, v\} \in P$.

The *Stacker Crane Problem (SCP)* studied in [18, 21] is a special case of SVP. In SCP, F_{uv} consists of one arc for every pair $\{u, v\} \in P$. To see that SCP is equivalent to ATSP it suffices to contract all arcs of F .

Consider SVP. In order to enforce the requirement that a Hamilton cycle has to traverse one arc in F_{uv} for each pair $\{u, v\} \in P$, we apply a transformation which results in a weighted complete undirected graph. Solving STSP on the transformed graph provides a solution to the original problem. The transformation replaces each pair $\{u, v\} \in P$ with a graph. We consider only the more interesting case when the line $\{u, v\}$ is undirected (that is, $|F_{uv}| = 2$). In this case,

the two vertices are replaced with the so-called diamond graph D_8 with $V(D_8) = \{N, W, E, S, a, b, c, d\}$ and $E(D_8) = \{Na, aW, Nb, bE, Wc, cS, Ed, dS, bc\}$. The diamond graph can be traversed in two possible ways, N - S and W - E (see Chapter 19 in [39]). These correspond to traversing the original pair of vertices, $\{u, v\}$ via arcs uv and vu , respectively. To make the weight of the tour consistent with the original graph:

- We set the weight of edges incident to W to be the same as the weight of the corresponding original arcs entering vertex v ;
- The weight of edges incident to E are taken to be the same as weight of arcs leaving u ;
- The weight of edges incident to N are taken to be the same as weight of arcs entering u ;
- The weight of edges incident to S are taken to be the same as weight of arcs leaving v ;
- Since arcs uv and vu have zero weight, all edges inside the diamond graph have their weight set to 0.
- The vertices a, b, c, d are not adjacent to any vertices outside their copy of D_8 .

For more TSP applications, see, e.g., [40].

Methods

The methods to solve TSP can be divided into two large classes: *exact algorithms* that solve the problem or its special cases to optimality and the algorithms that normally provide non-optimal tours. The members of the second class are called TSP *heuristics* or TSP *approximation algorithms* (the latter is often used if there is some kind of approximation guarantee). Exact algorithms are used when we want to obtain an optimal tour. This may not be possible as exact algorithms may well require several hours or days of running time

even for instances of moderate size (for example, the authors of [11] found out that no state-of-the-art exact algorithm could solve some ATSP instances with 316 vertices within the limit of 10^4 sec.). When running time is limited or the data of the instance is not exact, one can use TSP heuristics. For discussion of TSP software implementing both exact algorithms and heuristics, see [30] and the site www.or.deis.unibo.it/research.html.

Exact Algorithms. The *brute-force* method of explicitly examining all possible TSP tours is impractical even for moderately sized problem instances because there are $\frac{(n-1)!}{2}$ different tours in K_n and $(n-1)!$ different tours in K_n^* . The well-known dynamic programming algorithm of Hell and Karp [19] reduces the running time to $O(n^2 2^n)$ only, but this time complexity is still far too large to solve even TSP instances of moderate size. On the other hand, branch-and-bound, branch-and-cut and other branching algorithms are proved to be quite efficient in practice; branch-and-bound algorithms will be discussed in this subsection.

While every STSP instance can be considered as an ATSP instance and, thus, solved using ATSP algorithms, normally STSP-specialized algorithms are used for STSP instances as such algorithms are often more efficient than their ATSP counterparts (partially because they exploit a more special structure of STSP and partially because STSP algorithms have received significantly more attention than their ATSP counterparts). Moreover, in many cases ATSP instances are transformed into STSP instances and subsequently solved using STSP algorithms. (This situation may change in the future when advanced ATSP solvers will have been developed.)

In this subsection, we will consider two ATSP-to-STSP transformations and basic ideas behind STSP branch-and-bound algorithms. We will not consider special polynomial-time solvable cases of TSP; instead we refer the reader to [5, 26] which are excellent surveys on the topic.

The following are well-known ATSP-to-

STSP transformations:

The 2-node transformation: Replace every vertex i of K_n^* by a pair i^-, i^+ of vertices to form K_{2n} . The weights of edges of K_{2n} are defined as follows: all weights are equal to $+\infty$ apart from $w(i^-, i^+) = 0$ and $w(i^+, j^-) = w(i, j) + M$ for all $i \neq j \in [n]$, where $w(i, j)$ is the weight of arc ij in K_n^* and M is a sufficiently large constant. The transformation value nM has to be subtracted from the STSP optimal weight to obtain the ATSP optimal weight. The transformation was introduced by Jonker and Volgenant [24].

The 3-node transformation: Replace every vertex i of K_n^* by a triple i^-, i^0, i^+ of vertices to form K_{3n} . The weights of edges of K_{3n} are defined as follows: all weights are equal to $+\infty$ apart from $w(i^-, i^0) = w(i^0, i^+) = 0$ and $w(i^+, j^-) = w(i, j)$ for all $i \neq j \in [n]$, where $w(i, j)$ is the weight of arc ij in K_n^* . The transformation was introduced by Karp [28].

Each transformation has its pros and cons, see [11, 21].

Now we consider basic ideas behind TSP branch-and-bound and branch-and-cut algorithms using the Dantzig-Fulkerson-Johnson formulation of STSP. The formulation allows us to treat STSP as an integer programming problem. If we drop (5), we will get a linear programming problem whose solution will give us a lower bound to STSP. The linear program is called the *linear relaxation* of STSP. A branch-and-bound algorithm for STSP could be as follows.

Step 1. A list L of problems to solve is initialized by including into it the linear program discussed above. This problem is called the *root problem*.

Step 2. If $L = \emptyset$, then the best known feasible solution (tour) is optimal. Otherwise, choose a problem P and delete it from the list.

Step 3. (a) Solve the linear relaxation of P . If the solution is integral, return to Step 2 after eventually updating the best known integral solution and the best known solution value.

(b) If the value of the objective function exceeds that of the best known feasible solution, return to Step 2.

(c) Otherwise, using some linear inequality, partition the current problem into two new problems which are added to L . The union of the feasible (integral) solutions to each of these two problems contains all the feasible solutions of the problem that has been partitioned. This is commonly done by choosing a variable with a current fractional value \bar{x}_{ij} and imposing $x_{ij} \geq 1$ in one problem and $x_{ij} \leq 0$ in the other. Return to Step 2.

Due to computer memory limitations, the branch-and-bound algorithm is appropriate for an STSP formulation with a polynomial number of constraints, but this is not the case for the Dantzig-Fulkerson-Johnson formulation of STSP. Thus, we need to use a method that allows to store only a small number of constraints at any given moment of time. One such method is *row generation*. Using row generation, we replace Step 1 of the above algorithm by the following: we initially solve the problem consisting of (1), (2) and (4) obtaining a solution \bar{x} . Now we try to find a set $S \subset [n]$ such that $\sum_{i \in S} \sum_{j \notin S} \bar{x}_{ij} < 2$. To do that we can use an efficient algorithm for computing a minimum cut in a weighted undirected graph applied to K_n with weight function $\bar{x} : E(K_n) \rightarrow \mathbb{R}$ (see, e.g., [7, 25]). If a desired set S is found, we add the constraint to the current linear program and solve it to find a new vector \bar{x} , and continue as above. If no desired set S has been found, the problem is solved.

Similarly, one can solve other problems from the list L . In practice, we need to apply a minimum cut algorithm very few times, see,

e.g., [36]. The solution found in Step 1 usually provides a good lower bound called the *Held-Karp bound* in the literature.

STSP computational practice indicates that while branch-and-bound algorithms are fairly efficient in solving STSP, branch-and-cut algorithms are normally much more efficient. For an excellent overview of STSP branch-and-cut algorithms, see [36].

TSP Heuristics. TSP heuristics can be roughly partitioned into two classes: construction heuristics, and improvement heuristics. Both classes and their performances in computational experiments are discussed below. More comprehensive overviews of TSP heuristics can be found in [14], [21] and [22]. Notice that [21] and [22] discuss families of instances on which many TSP heuristics have been tested. Many new heuristics are now tested using these families. This allows one to compare new heuristics with many known ones without too much effort.

We finalize the Heuristic subsection by a brief discussion of approximation analysis of TSP heuristics.

Construction Heuristics. *Construction heuristics* build a tour from scratch and stop when one is produced. The simplest and most obvious construction heuristic is *nearest neighbor (NN)*: the tour starts at any vertex x of the complete directed or undirected graph; we repeat the following loop until all vertices have been included in the tour: add to the tour a vertex (among vertices not yet in the tour) closest to the vertex last added to the tour. The *greedy algorithm* is based on the observation that a vertex-disjoint collection of paths in K_n^* (K_n) can be extended to a tour in K_n^* (K_n). In the ATSP greedy algorithm, we order all arcs $a_1, a_2, \dots, a_{n(n-1)}$ such that $w(a_i) \leq w(a_{i+1})$ for each $i = 1, 2, \dots, n(n-1) - 1$, set $C := \emptyset$ and, in the i 'th iteration, we check whether the arcs of C and a_i form a vertex-disjoint collection of paths or a tour, and if it is so, we add a_i to C .

Computational experiments in [21] indi-

cate that, in fact, on most real-world-like problem instances of ATSP, NN performs better than the greedy algorithm; the greedy algorithm fails completely on one family of instances, where the average greedy-tour is more than 2000 % above the optimum. Computational experiments for STSP in [22] show that both the greedy algorithm and NN perform relatively well on Euclidean instances and perform poorly for general STSP. The greedy algorithm appears to perform better than NN for STSP.

Vertex insertion (VI) is another type of TSP construction heuristic. For ATSP, the insertion algorithm begins with a cycle of length 2, and in each iteration, inserts a new vertex into the cycle. For STSP, the algorithm begins with a cycle of length 3. We describe only the ATSP vertex insertion, but the STSP algorithm is similar. Let C be a cycle in K_n^* , and let v be a vertex not on C . For any arc ab on cycle C , the *insertion of vertex v at arc ab* is the operation of replacing arc ab with the arcs av and vb . The resulting cycle is denoted $C(a, v, b)$. Observe that the difference between the weights of $C(a, v, b)$ and C equals $w(a, v) + w(v, b) - w(a, b)$. The VI algorithm always inserts a vertex v at arc ab of C for which $w(a, v) + w(v, b) - w(a, b)$ minimum.

Random vertex insertion (RVI), *nearest vertex insertion (NVI)*, and *farthest vertex insertion (FVI)*, which are defined below, are three different versions of algorithm VI. Each one of them is determined by how it chooses vertex v to be inserted into the current cycle C . Given a vertex v and a cycle C in K_n^* , $d(v, C)$ denotes the distance from v to C , that is, $d(v, C) = \min\{w(v, x) : x \in V(C)\}$. The algorithm RVI chooses vertex v randomly. The algorithm NVI chooses vertex v so that its distance to cycle C is a minimum. That is, $d(v, C) = \min\{d(u, C) : u \notin V(C)\}$. The algorithm FVI chooses vertex v so that its distance to cycle C is a maximum. That is, $d(v, C) = \max\{d(u, C) : u \notin V(C)\}$.

The vertex insertion heuristics described above perform quite well for Euclidean TSP

(see [22]). Computational experiments with RVI for ATSP in [13] show that RVI is good only for instances close to Euclidean.

The following heuristic was initially suggested, in a different form, for the Vehicle Routing Problem by Clark and Wright [9]. In the *savings heuristic*, we choose one vertex, say, n and compute new weights $w'(i, j) = w(i, j) - w(i, n) - w(n, j)$ for all $i \neq j \in [n - 1]$. Then the greedy algorithm is applied for the new weights in $K_n - n$ ($K_n^* - n$) until all vertices (but n) are included in a path. Then n is added to the path to form a tour. The savings heuristic showed very good results for STSP in the computational experiments discussed in [22], in which the heuristic clearly outperformed the greedy algorithm, NN, RVI, NVI, FVI and a large number other heuristics. (The saving heuristic has not been tested for ATSP in [21].)

The only heuristic, some versions of which could successfully compete with the savings heuristic in the experiments in [22], was the well-known Christofides heuristic [8]. The Christofides heuristic is designed only for STSP and proceeds as follows: First we find a minimum weight spanning tree T in K_n . Let X be the vertices of odd degree in T . It is well-known that $|X|$ is even and, thus, the subgraph G of K_n induced by X has a perfect matching. We compute a minimum weight perfect matching M in G . The edges of T and M form an Euler graph H as all vertex degrees are even. We find an Euler trail R of H and ‘short-cut’ it, i.e., delete all repetitions of the same vertex in R . As a result, we obtain a tour. The way of short-cutting is very important for getting good quality tours [22].

According to [21] the best ATSP construction heuristics are based on finding a minimum weight cycle factor (a vertex-disjoint collection of cycles covering all vertices of K_n^*) and merging the cycles (the process often called *patching* in the literature) to obtain a tour. The operation of patching of two cycles C and Z deletes an arc in each of the cycles and adds an arc from C to Z and an arc from Z to C

such that we obtain a cycle containing all vertices of C and Z . Often patching of cycles $C = i_1 i_2 \dots i_s i_1$ and $Z = j_1 j_2 \dots j_t j_1$ is done *optimally*, i.e., we delete arcs $i_p i_{p+1}$ and $j_q j_{q+1}$ such that the cycle

$$i_1 i_2 \dots i_p j_{q+1} j_{q+2} \dots j_t j_1 \dots j_q i_{p+1} i_{p+2} \dots i_s i_1$$

is of minimum possible weight.

The following simple yet very successful patching heuristic was introduced by Karp and Steele [29]. In the Karp-Steele heuristic, we always choose a pair of cycles (in the current cycle factor) with maximum number of vertices and patch them optimally. The Karp-Steele heuristic performs not so good when the minimum weight cycle factor has many cycles with just two vertices. In such cases, another patching heuristic, *contract-or-patch (COP)* gives better results [21]. COP partitions the cycles of the cycle factor into short and long cycles (a short cycle has at most t vertices for some fixed t). COP deletes the heaviest arc from each short cycle and contracts each such path using the operation of path-contraction defined shortly. COP finds a minimum cost cycle factor in the new complete digraph and continues as above until the current cycle factor has no short cycles. In the last case, COP applies the Karp-Steele heuristic, computes a tour and ‘extends’ it to a tour in K_n^* in the obvious way. For a directed path $P = x_1 x_2 \dots x_p$ in K_n^* , the operation of *path-contraction* (see [3] for the case of general weighted digraphs) consists of replacing all vertices of P in K_n^* with a single new vertex v and assigning weights in the new digraph K_{n-p+1}^* as follows: the weight between vertices not including v is the same as in K_n^* , the weight $w(v, u)$ in K_{n-p+1}^* equals $w(x_p, u)$ in K_n^* and the weight $w(u, v)$ in K_{n-p+1}^* equals $w(u, x_1)$ in K_n^* for each $u \in V(K_n^*) \setminus V(P)$. The contract-or-patch heuristic was introduced by Glover et al. [13].

Improvement Heuristics. *Improvement heuristics* start from a tour normally obtained using a construction heuristic and iteratively improve it by changing some parts of it at each iteration. Improvement heuristics are typically

much faster than the exact algorithms, yet often produce solutions very close to the optimal one.

It appears that currently the best improvement heuristics are based on local search, on genetic algorithm approach, or on a mixture of the two, which is often called *memetic algorithms*. The most developed TSP improvement algorithms are local search algorithms that use *edge exchange*, in which a tour is improved by replacing k its edges with k edges not in the solution. For STSP, the *2-opt* algorithm starts from an initial tour T and tries to improve T by replacing two of its non-adjacent edges with two other edges to form another tour. Once an improvement is obtained, it becomes the new T . The procedure is repeated as long as an improvement is possible (or a time limit is exceeded). For $k \geq 3$, the *k-opt* algorithm is the same as *2-opt* except that k edges are replaced at each iteration.

The best local search algorithms use a variable k -Opt search called the *Lin-Kernighan local search*, where at each iteration the actual value of k varies depending on which value of k gives the best improvement, for details see, e.g., [43]. Although the Lin-Kernighan local search can be applied only to STSP, ATSP can be transformed into STSP (see above). However, there is an approach, the *ejection chain methods*, which include the Lin-Kernighan search, that are applicable to ATSP. Recently, Rego et al. [44] developed a new ejection chain method, the doubly-rooted Stem-and-Cycle method that can be directly applied to ATSP. Computational experiments in [44] clearly demonstrated high efficiency of the new method. One interesting aspect of the method indicated in [44] is the fact that the method allows one to construct tours, in polynomial time, that are better than an exponential number of other tours.

The main problem with any kind of local search is that no further improvement is possible once we have found a local optimum. To get around this problem, one can restart the local search from another tour and repeat this

many times. In the end, the best of all found tours gives us a solution. In practice, two ways to obtain restarting tours have been used. In the first (called *iterated local search*), a restarting tour is produced by a construction heuristic as before. In the second (*chained local search*), a kind of perturbation is applied to the current or previous local optimum to obtain a restarting tour. It seems Baum [6] was the first to introduce chained local search; this method proved to be significantly better than the iterated local search for large instances of STSP (see, e.g., Johnson and McGeoch [22]).

Genetic algorithms operate with a large number of tours at any given time. They produce the initial *population* of tours and consecutively several other populations such that the best tour in the previous population is not worse than the best tour in the current population. *Genetic operators* that change tours include mutations (a *mutation* makes small changes to a single tour) and crossovers. A *crossover* selects two tours and produces a new tour from them. It appears that the currently most efficient crossovers are variations of the *edge assembly crossover (EAX)* introduced by Nagata and Kobayashi [35]. In EAX, we identify a set A of edges from the first tour and a set B of edges from the second tour such that $A \cup B$ forms a collection of alternating cycles (i.e., cycles in which edges alternate between the first and second tours) and replace all edges from A by the edges of B resulting in a cycle factor. Then an operation of patching is applied to the cycle factor. Recently, Nagata [34] reported on very impressive results for large instances of STSP achieved by a genetic algorithm using a new version of EAX and no local search.

Worst Case Analysis of Heuristics. While computational experiments are important in the evaluation of heuristics, they cannot cover all possible families of instances of TSP and, in particular, they normally do not cover the most difficult instances. Moreover, certain applications may produce families of instances that are much harder than those normally used

in computational experiments. For example, such instances can arise when the Generalized TSP is transformed into TSP. Thus, theoretical analysis of the worst possible cases is also important in evaluating and comparing TSP heuristics. One way to analyze worst cases of heuristics is Domination Analysis, see its entry in this book.

We provide only a brief overview of the second approach to the worst case analysis of heuristics, Approximation Analysis. For the STSP with *triangle inequality* (i.e., $w_{ij} + w_{jk} \geq w_{ik}$ for all vertices i, j, k), the best known approximation is $3/2$ provided by the Christofides algorithm discussed earlier. The performance guarantee $3/2$ means that a tour produced by the heuristic has weight which is at most 50% larger than that of an optimal tour. For the Euclidean TSP, we can obtain much better approximation as we saw earlier. For ATSP with triangle inequality, no algorithm with constant approximation guarantee is known. The best approximation ratio so far was obtained by Kaplan et al. [27]: $0.841 \cdot \log n$. Recently, Blaeser et al. [4] obtained a constant approximation guarantee when a *strengthen triangle inequality* holds: for some $\gamma \in [1/2, 1)$ we have $\gamma \cdot (w_{ij} + w_{jk}) \geq w_{ik}$ for all vertices i, j, k . The authors of [4] proved that their algorithm always produces a tour at most $\frac{1+\gamma}{2-\gamma-\gamma^3}$ times longer than an optimal one.

We saw above that, if no triangle inequality is imposed, there is no polynomial-time TSP algorithm with constant approximation guarantee (unless $P=NP$). We can overcome the inapproximability, by using another measure of performance guarantee. One such measure was defined by Zemel [46] who provided some mathematical arguments to show that his measure is better, in some sense, than the traditional performance (approximation) ratio. Let \mathcal{A} be a heuristic for TSP and I a problem instance. Then $w_{\min}(I)$, $w_{\max}(I)$, $w_{\mathcal{A}}(I)$ denote the weights, respectively, of an optimal tour, a heaviest tour, and a tour produced by \mathcal{A} for instance I . The *Zemel mea-*

sure of \mathcal{A} , denoted $\rho_z(\mathcal{A})$, is the supremum of $(w_{\mathcal{A}}(I) - w_{\min}(I)) / (w_{\max}(I) - w_{\min}(I))$, taken over all TSP instances I for which $w_{\max}(I) \neq w_{\min}(I)$. The following theorem was proved by Hassin and Khuller [18].

Theorem 5. *There is a polynomial-time heuristic \mathcal{A} for ATSP with $\rho_z(\mathcal{A}) \leq \frac{1}{2}$, and one for STSP with $\rho_z(\mathcal{A}) \leq \frac{1}{3}$.*

See also: **Approximation Algorithms for Combinatorial Optimization, Domination Analysis in Combinatorial Optimization, Heuristic and Metaheuristic Algorithms for the Traveling Salesman Problem.**

References

- [1] ARORA, S.: 'Polynomial-time approximation schemes for Euclidean TSP and other geometric problems', *JACM* **45** (1998), 753-782.
- [2] ARORA, S.: 'Approximation algorithms for geometric TSP', in G. Gutin and A.P. Punnen (eds.): *The Traveling Salesman Problem and its Variations*, Kluwer, Dordrecht, 2002.
- [3] BANG-JENSEN, J., AND GUTIN, G.: *Diagraphs: Theory, Algorithms and Applications*, Springer, 2000.
- [4] BLAESER, M., MANTHEY, B., AND SGALL, J.: 'An Improved Approximation Algorithm for the Asymmetric TSP with Strengthened Triangle Inequality', *Journal of Discrete Algorithms* **4** (2006), 623-632.
- [5] BURKARD, R.E., DEINEKO, V.G., VAN DAL, R., VAN DER VEEN, J.A.A., AND WOEGINGER, G.J.: 'Well-solvable special cases of the traveling salesman problem: a survey' *SIAM Review* **40** (1998), 496-546.

- [6] BAUM, E.B.: ‘Iterated descent: A better algorithm for local search in combinatorial optimization problems’, 1986, unpublished manuscript.
- [7] CHEKURI, C., GOLDBERG, A., KARGER, D., LEVINE, M., AND STEIN, C.: ‘Experimental Study of Minimum Cut Algorithms’, in *Proc. 8th ACM-SIAM Symp. Discrete Algorithms (SODA’97)*, ACM/SIAM, 1997, 324–333.
- [8] CHRISTOFIDES, N.: ‘Worst-case analysis of a new heuristic for the traveling salesman problem’, *Technical Report CS-93-13*, Carnegie Mellon University, 1976.
- [9] CLARKE, G. AND WRIGHT, J.W.: ‘Scheduling of vehicles from a central depot to a number of delivery points’, *Oper. Res.* **12** (1964), 568–581.
- [10] DANTZIG, G.B., FULKERSON, D.R., AND JOHNSON, S.M.: ‘Solution of large scale traveling salesman problem’, *Oper. Res.* **2** (1954), 393–410.
- [11] FISCHETTI, M., LODI, A., AND TOTH, P.: ‘Exact Methods for the Asymmetric Traveling Salesman Problem’, in G. Gutin and A.P. Punnen (eds.): *The Traveling Salesman Problem and its Variations*, Kluwer, 2002.
- [12] GAREY, M.R., GRAHAM, R.L., AND JOHNSON, D.S.: ‘Some NP-complete geometric problems’, in *Proc. 8th ACM Symp. Theory Comput.* (1976), 10–22.
- [13] GLOVER, F., GUTIN, G., YEO, A., AND ZVEROVICH, A.: ‘Construction heuristics for the asymmetric TSP’, *Eur. J. Oper. Res.* **129** (2001), 555–568.
- [14] GOLDEN, B.L., AND STEWART, W.R.: Empirical Analysis of Heuristics. In *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization* (E. Lawler, J. K. Lenstra, A.H.G. Rinnooy Kan and D.B. Shmoys, eds.), Wiley, 1985.
- [15] GUTIN, G., JAKUBOWICZ, H., RONEN, S., AND ZVEROVITCH, A.: ‘Seismic vessel problem’, *Communications in DQM* **8** (2005), 13–20.
- [16] GUTIN, G., KOLLER, A., AND YEO, A.: ‘Note on Upper Bounds for TSP Domination Number’, *Algorithmic Oper. Res.* **1** (2006), 52–54.
- [17] GUTIN, G., YEO, A., AND ZVEROVICH, A.: ‘Traveling salesman should not be greedy: domination analysis of greedy-type heuristics for the TSP’, *Discrete Appl. Math.* **117** (2002), 81–86.
- [18] HASSIN, R., AND KHULLER, S.: ‘ z -Approximations’, *J. Algorithms* **41** (2001), 429–442.
- [19] HELD, M. AND KARP, R.M.: ‘A dynamic programming approach to sequencing problems’, *J. Soc. Ind. Appl. Math.* **10** (1962), 196–210.
- [20] HOFFMAN, A.J., AND WOLFE, P.: ‘History’, in E. L. Lawler, J. K. Lenstra, A.H.G. Rinnooy Kan and D.B. Shmoys (eds.): *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, Wiley, 1985.
- [21] JOHNSON, D.S., GUTIN, G., MCGEOCH, L.A., YEO, A., ZHANG, W., AND ZVEROVITCH, A.: ‘Experimental Analysis of Heuristics for ATSP’, in G. Gutin and A.P. Punnen (eds.): *The Traveling Salesman Problem and its Variations*, Kluwer, Dordrecht, 2002.
- [22] JOHNSON, D.S., AND MCGEOCH, L.A.: ‘Experimental Analysis of Heuristics for STSP’, In *The Traveling Salesman Problem and its Variations* (G. Gutin and A. P. Punnen, eds.), Kluwer, 2002.

- [23] JOHNSON, D.S., AND PAPADIMITRIOU, C.H.: ‘Performance guarantees for heuristics’, In *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization* (E. L. Lawler, J. K. Lenstra, A.H.G. Rinnooy Kan and D.B. Shmoys, eds.), Wiley, 1985.
- [24] JONKER, R. AND VOLGENANT, T.: TRANSFORMING ASYMMETRIC INTO SYMMETRIC TRAVELING SALESMAN PROBLEMS, *Oper. Res. Lett.* **2** (1983), 161–163.
- [25] JÜNGER, M., RINALDI, G., AND THIENEL, S.: ‘Practical Performance of efficient minimum Cut Algorithms’, *Algorithmica* **26** (2000), 172–195.
- [26] KABADI, S.N.: ‘Polynomially solvable cases of the TSP’, In *The Traveling Salesman Problem and its Variations* (G. Gutin and A. P. Punnen, eds.), Kluwer, 2002.
- [27] KAPLAN, H., LEWENSTEIN, M., SHAFRIR, N., AND SVIRIDENKO, M.: ‘Approximation Algorithms for the Asymmetric TSP by Decomposing Regular Multigraphs’, *JOURNAL OF THE ACM* **52** (2005), 602–626.
- [28] Karp, R.M.: ‘Reducibility among combinatorial problems’, in R.E. Miller and J.W. Thatcher (eds.): *Complexity of Computer Computations*, Plenum Press, New York, 1972, 85–103.
- [29] : KARP, R.M., AND STEELE, J.M. ‘Probabilistic analysis of heuristics’, in E. L. Lawler, J. K. Lenstra, A.H.G. Rinnooy Kan and D.B. Shmoys (eds.): *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, Wiley, 1985.
- [30] LODI, A., AND PUNNEN, A.P.: ‘TSP Software’, in G. Gutin and A.P. Punnen (eds.): *The Traveling Salesman Problem and its Variations*, Kluwer, Dordrecht, 2002.
- [31] MENGER, K.: ‘Das botenproblem’, *Ergebnisse Eines Mathematischen Kolloquiums* **2** (1932), 11–12.
- [32] MILLER, C.E. AND TUCKER, A.W. AND ZEMLIN, R.A.: ‘Integer Programming formulations and traveling salesman problems’, *J. Assoc. Comput. Mach.* **7** (1960), 326–329.
- [33] MITCHELL, J.C.B.: ‘Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial time approximation scheme for geometric TSP, k -MST, and related problem’, *SIAM J. Comput.* **28** (1999), 1298-1309.
- [34] NAGATA, Y.: ‘New EAX crossover for large TSP instances’, *Lecture Notes in Computer Science* **4193** (2006), 372–381.
- [35] NAGATA, Y., AND KOBAYASHI, S.: ‘Edge Assembly Crossover: A High-power Genetic Algorithm for the Traveling Salesman Problem’, in *Proc. of the 7th Int. Conference on Genetic Algorithms*, 1997, 450–457.
- [36] NADDEF, D.: ‘Polyhedral Theory and Branch-and-Cut Algorithms for the Symmetric TSP’, In *The Traveling Salesman Problem and its Variations* (G. Gutin and A. P. Punnen, eds.), Kluwer, 2002.
- [37] PADBERG, M.W. AND SUNG, T.Y.: ‘An analytical comparison of different formulations of the traveling salesman problem’ *Math. Program. Ser. A* **52** (1991), 315–357.
- [38] PAPADIMITRIOU, C.H.: ‘The Euclidean traveling salesman problem is NP-complete’, *Theoret. Comput. Sci.* **4** (1977), 237-244.
- [39] PAPADIMITRIOU, C.H., AND STEIGLITZ, K.: *Combinatorial Optimization*, Prentice-Hall, 1982.

- [40] PUNNEN, A.P.: ‘The Traveling Salesman Problem: Applications, Formulations and Variations’, in G. Gutin and A.P. Punnen (eds.): *The Traveling Salesman Problem and its Variations*, Kluwer, Dordrecht, 2002.
- [41] PUNNEN, A.P., MARGOT, F., AND KABADI, S.: ‘TSP heuristics: domination analysis and complexity’, *Algorithmica*
- [42] RAO, S., AND SMITH, W.: ‘Approximating geometric graphs via “spanners” and “banyans”’, *Proc. 30th Ann. ACM Symp. Theory Comput.* (1998), 540-550.
- [43] REGO, C., AND GLOVER, F.: ‘Local Search and Metaheuristics’, in G. Gutin and A.P. Punnen (eds.): *The Traveling Salesman Problem and its Variations*, Kluwer, Dordrecht, 2002.
- [44] REGO, C., GLOVER, F., GAMBOA, D., AND OSTERMAN, C.: ‘A Doubly-Rooted Stem-and-Cycle Ejection Chain Algorithm for Asymmetric Traveling Salesman Problems’, submitted.
- [45] TREVISAN, L.: ‘When Hamming meets Euclid: the approximability of geometric TSP and MST’, in *Proc. 29th ACM Symp. Theory Comput.* (1997), 21–39.
- [46] ZEMEL, E.: ‘Measuring the quality of approximate solutions to zero-one programming problems’, *Math. Oper. Res.* **6** (1981), 319–332.

Gregory Gutin

Department of Computer Science

Royal Holloway, University of London

Egham, Surrey TW20 0EX, UK

E-mail Address: gutin@cs.rhul.ac.uk

MSC2000: 90B06, 90B35, 90C06, 90C10, 90C27, 90C39, 90C57, 90C59, 90C60, 90C90

Key words and phrases: traveling salesman problem, asymmetric traveling salesman problem, symmetric traveling salesman problem.