1st International Air Transport and Operations Symposium, ATOS 2010, 14. - 16. April **2010**, TU Delft, NL, ISBN 978-1-60750-559-4

Asymmetric Traveling Salesman Problem - Near Optimal Real-time Solution

Peter H . Richter
O&S Consultancy, Berlin, 10369, Germany

We present an $\Theta(|S|\cdot|E_G|)$ deterministic construction heuristic for the **Asymmetric Traveling Salesman Problem** ATSP on digraphs G. The heuristic relies on the fast determination of an approximate bidirectional Steiner Tree with respect to the stopovers $S \subseteq V_G$. The algorithm is a robust and very fast general ATSP-solution method. It has an astonishing approximation $\varepsilon \approx 0.05$ and is therefore especially appropriate for online and real-time navigation applications with a high number of stopovers and where graph changes have to be considered just in time. It turns out that the proposed method is a new serious competitor for all existing ATSP heuristics, especially qualified for navigation apps of large problem sizes that depend on the tours' real-time calculation instantly executed on online graphs while observing turn restrictions.

Nomenclature

G = $[V_G, E_G]$= finite, connected directed graph (digraph). We use **n**= $|V_G|$ and **m**= $|E_G|$.

λ: $E_G \rightarrow \boldsymbol{R}_+$ = edge cost: e= (p, q)$\in E_G \Rightarrow \lambda(e)=\lambda((a,b))$= cost (length, time, .) traversing e from p to $q \in V_G$.

G' = "subgraph" $[V_{G'}, E_{G'}] \subseteq G$ with $V_{G'} \subseteq V_G$ and $E_{G'} \subseteq E_G \cap V_{G'}^2$ and cost $C(\mathbf{G'})=\sum_{e \in E_{G'}} \lambda(e)$.

$\bar{\lambda}$: $E_G \rightarrow \boldsymbol{R}_+$ = "bidirectional" edge cost: (a, b)$\in E_G \Rightarrow \bar{\lambda}((a,b))=\bar{\lambda}((b,a))=\lambda((a,b))+\lambda((b,a))$.

χ: $V_G \rightarrow \mathfrak{P}(E_G^2)$ denotes turn restrictions for crossings V_G if necessary. $\chi(q)=\{\ldots,((p, q), (q, r)), ..\} \Rightarrow$ "Coming from p to q passing edge (p, q) continuing the journey via edge (q, r) is not allowed, see [28].

S $\subseteq V_G$ is the set of stopovers that are to be visited via a closed cost minimal cycle in **G** considering χ.

$\mathbf{P}_G(\{x\},\{y\}) \subseteq \mathbf{G}$ denotes an optimal path from $x \in V_G$ to $y \in V_G$ with respect to λ obeying turn restrictions χ such that its cost results to $C(\mathbf{P}_G(\{x\},\{y\}))=\min_{\substack{\mathbf{P}\subseteq G \text{ observes } \chi \\ \mathbf{P} \text{ is path from x to y}}} \{C(\mathbf{P})\}$.

$\mathbf{P}_G(\{x\}, Y) = \bigcup_{y \in Y} \mathbf{P}_G(\{x\},\{y\}) \subseteq \mathbf{G}$ denotes an Optimal Path Graph OPG consisting of optimal paths from $x \in V_G$ to all $y \in Y \subseteq V_G$ obeying χ.

P $\in S^{|S|}$ denotes a permutation as an ordered set $S=\bigcup_{i=1}^{|S|}\{P[i]\}$ with P stored as **[**P[1], P[2], ... , P[|S|]**]**.

Ω(P) $\subseteq$ **G,** called layout of P, maps a valid permutation P into a sub-graph **G'=** Ω(P) $\subseteq$ **G** as follows: $\Omega(P)=\mathbf{P}_G(\{P[|S|]\},\{P[1]\}) \cup \bigcup_{i=1}^{|S|-1} \mathbf{P}_G(\{P[i]\},\{P[i+1]\})$. Ω(P) is composed of |S| opt. paths observing χ.

π: $E_G \rightarrow \boldsymbol{R}_+$ is called edge potential $\approx$ "path length", i.e. π (e=(x,y)) are the preliminary or final cost (dependant of the algorithm's proceeding) of a path from the start via the final edge e to node $y \in V_G$. $\pi(e=(x,y))=\infty$ (max. real number) is used if y is not reachable by any path $\mathbf{P}_G(\{x\},\{y\})$.

σ: $E_G \rightarrow E_G$ is a context-dependant edge predecessor or edge successor function determined by edge-queuing optimal path algorithms.

κ: $V_G \rightarrow E_G$ is called minimal edge function related to an existing optimal path $\mathbf{P}_G(\{x\},\{y\})$ determined by an edge-queuing algorithm as follows: $e^*=\kappa(y) \in E_G \Leftrightarrow \pi(e^*)=\min_{e \in (V_G \times \{y\}) \cap E_G} \{\pi(e)\}$. Sequence e*= κ(y), σ(e*), $\sigma(\sigma(e^*))$, $\sigma(\sigma(\sigma(e^*)))$, .., e'=(x, p) leads (against edge direction) from target $y \in Y$ to start edge e' incident to node x . Due to χ, crossovers of paths are possible while σ: $E_G \rightarrow E_G$ remains unique!

μ: $E_G \rightarrow \{0, 1\}$ is a layout marker that describes the final graph **G*** determined by **A-TSP**: $\mathbf{G}^*=[V_{G^*}, E_{G^*}] \subseteq \mathbf{G}$ with $E_{G^*}=\{e \in E_G: \mu(e)=1\}$, $V_{G^*}=\boldsymbol{Fd}(E_{G^*})=\boldsymbol{dom}(E_{G^*}) \cup \boldsymbol{rng}(E_{G^*})$.

ν: $\mathbf{S} \rightarrow \{1,2,.. |\mathbf{S}|\}$, $1 \leq \nu(x) \leq |S|$, $x \in S$, uniquely assigns a positive address number to stopovers S.

τ_E: $E_G \rightarrow \{0, 1\}$ and τ_v: $V_G \rightarrow \{0, 1\}$ are markers defining the edges of an approximate bidirectional Steiner tree $T(S)= [V_{T(S)}, E_{T(S)}]$ with $V_{T(S)}= \{v \in E_G: \tau_V(v)= 1\}$, $E_{T(S)}= \{e \in E_G: \tau_E(e)= 1\}$, $S \subseteq V_{T(S)}$. It holds: $(a,b) \in E_{T(S)} \Leftrightarrow (b,a) \in E_{T(S)}$ whereas both edges might have different cost $\lambda((a,b)) \neq \lambda((b,a))$.

mx $= |\mathbf{S}| \times |\mathbf{S}|$ is a cost matrix: $mx[\nu(p), \nu(q)]$ contains the cost the optimal path $\mathbf{P}_G(\{p\}, \{q\})$ observing χ.

$\overline{P}_G(\{x\},\{y\}) \subseteq \mathbf{G}$ is a bidirectional optimal path from x to $y \in V_G$ observing $\overline{\lambda}$ with

$$C(\overline{P}_G(\{x\},\{y\}))= C(\overline{P}^*) \text{ cost } \min_{\substack{\overline{P} \subseteq G, \overline{P} \text{ bidirectional} \\ \overline{P} \text{ connects } x \text{ with } y}} \{ C(\overline{P})= \sum_{e \in E_{\overline{P}}} \overline{\lambda}(e) \}.$$

$\overline{P}_G(\{x\}, Y)= \bigcup_{y \in Y} \overline{P}_G(\{x\},\{y\}) \subseteq \mathbf{G}$ is a bidirectional optimal tree from $x \in V_G$ to all $y \in Y \subseteq V_G$ as to $\overline{\lambda}$.

η: $V_G \rightarrow S$ related to a bidirectional optimal forest $\overline{F}_G(S) \subseteq \mathbf{G}$ denotes with η(y) the nearest root for $y \in V_G$ in $\overline{F}_G(S)$. η is implicitly defined via σ: $y \rightarrow \sigma(y) \rightarrow \sigma(\sigma(y)) \rightarrow \sigma(\sigma(\sigma(y))) \rightarrow \ldots \rightarrow \eta(y) \in S$, $\sigma(\eta(y))= 0$.

$\overline{F}_G(S)= \overline{P}_G(S, Y) = \bigcup_{y \in V_G \setminus S} \overline{P}_G(\{\eta(y)\},\{y\}) \subseteq \mathbf{G}$ denotes a bidirectional optimal forest of disjoint bidirectional optimal path trees rooting in $S \subseteq V_G$ such that all reachable $y \in Y \subseteq V_G$ are connected via a bidirectional optimal path to its nearest root $\eta(y) \in S$ (used by algorithm **A-ST1**) .

arithmetic operations +, -, ∪, ∩, / are used in the statements' short form. E.g. a= a + b shortened to a+=b.

I. Introduction

IN contrast to the Symmetric Traveling Salesman Problem STSP (or shortly TSP), the Asymmetric Travelling Salesman Problem ATSP is much more important for real world apps because it considers the determination of cost-minimal cycles connecting stopovers S in directed graphs (digraphs). Further, ATSP solutions may contain cycles, STSP solutions not. A special demand coming not only from traffic managers is the ultimate request to consider turn restrictions χ whose implementation has not been regarded in the literature till now.! Last but not least, a variety of apps cannot be restricted to the triangle inequality $d(x, z) \leq d(x, y) + d(y, z)$ that must be hold for many ATSP and STSP algorithms. Algorithm **A-TSP** presented below meets all these considerations. Of course, ATSP like STSP is intractable, i.e. ***NP*** -hard [11]. However, ATSP- codes cannot compete with the TSP's general purpose codes. This results not only from the double number of distances for ATSP but from the necessity to use application dependant digraphs that require in each case the heuristics' adaptation. The optimal real-time computation (i.e. nominal time ≤ 2 sec) that has to consider the complete enumeration is restricted to problem sizes |S| < 11 (S= set of points to be visited, 2,3 GHz CPU), Richter [30]. To enable the ATSP's real time computation for high problem sizes we present a construction heuristic based on the search of an approximate bidirectional Steiner tree that efficiently regards turn restrictions χ. **A-TSP** has an average approximation $\overline{\varepsilon} = (C_{appr} - C_{opt}) / C_{opt} = 0{,}05$ $(0{,}0 \leq \varepsilon \leq 0{,}16)$ with an empirical standard deviation $\delta \leq 0{,}038$. The worst solution of the performance analysis of **A-TSP** is 1.155 times over the optimum! Algorithm **A-TSP** proposed here enables real-time ability for large problem sizes, e.g. here on grid graphs G with $n= |V_G|= 5000$, $m= |E_G|= 19716$, $|S|= 350$, nominal time ≈ 2 sec) . It has a worst case nominal run time $\boldsymbol{O}(|S| \cdot m^2)$. However, the performance tests reveal a time dependence of $\boldsymbol{\theta}(|S| \cdot m)$. The algorithm correspondingly solves also the STSP.

A. Statement of Problem

Given G, S, λ, and χ. Find a method determining directed minimum cycles $H^* \subseteq \mathbf{G}$, $S \subseteq \mathbf{V_G}$, that even large problem sizes can near-optimally be solved in real-time as to $C(\mathbf{H}^*)= \min_{\substack{\text{cycle } H \subseteq G, S \subseteq V_H \\ H \text{ observes } \chi}} \{C(H)= \sum_{r \in E_H} \lambda(r)\}$.

B. Literature

A lot of research on heuristics on this problem has concentrated on the symmetric case (the STSP). Reinelt [25] and Johnson et al. [17] experimentally examined a wide variety of heuristics on reasonably instances, and many papers study individual heuristics in more detail. For the general not necessarily symmetric case, typically referred to as the "asymmetric TSP" ATSP, there are far fewer publications, and none that comprehensively cover the current best approaches. This is unfortunate, as a wide variety of ATSP applications arise in practice. Cirasella et al. 7] gave a comprehensive study of the ATSP. They pointed out that the few previous ATSP studies focused on covering multiple algorithms [26] [34] [35] [13] would have several drawbacks. First, the classes of test instances studied have not been well-motivated in comparison to those studied in the case of the symmetric TSP. For the latter problem the standard test-beds are instances from TSPLIB [24] and randomly generated two-dimensional point sets, and algorithmic performance on these instances seems to correlate well with behavior in practice. For the ATSP, TSPLIB offers fewer and smaller instances with less variety, and the most commonly studied instance classes are random distance matrices (asymmetric and symmetric) and other classes with no plausible connection to real applications of the ATSP.

Zhang et al. [35] gave a Depth-first branch-and-bound (DFBnB) algorithm. They compared DFBnB against the Kanellakis-Papadimitriou local search algorithm, the best known approximation algorithm, on the ATSP. The experimental results show that DFBnB significantly outperforms the local search on large ATSP, finding better solutions faster than the local search; and the quality of approximate solutions from a prematurely terminated DFBnB, called truncated DFBnB, is several times better than that from the local search.

Cirasella et al. [7], divide current ATSP heuristics into three classes: (1) classical tour construction heuristics such as Nearest Neighbor and the Greedy algorithm, (2) local search algorithms based on re-arranging segments of the tour, as exemplified by the Kanellakis-Papadimitriou algorithm [19], and (3) algorithms based on patching together the cycles in a minimum cycle cover (which can be computed as the solution to an Assignment Problem, i.e., by constructing a minimum weight perfect bipartite matching). Examples of this last class include the algorithms proposed in [20] [21] and the ***O***(log N) worse-case ratio "Repeated Assignment" algorithm of [10].

Glover et al. [13] describe two construction heuristics for the ATSP and a further one combining the other two. They conclude, that their combined heuristic clearly outperforms well-known ATSP construction methods and proves significantly more robust in obtaining high quality solutions over a wide range of problems.

Buriol et al. [4] introduce a new memetic algorithm for the ATSP based on a new local search engine and other features that contribute to its effectiveness. Computational experiments are conducted on all ATSP instances available in the TSPLIB, and on a set of larger asymmetric instances with known optimal solutions. The comparisons show that the results compare favorably with those obtained by several other algorithms recently proposed for the ATSP. Kanellakis et al. [19] presented a local search algorithm for the solution of the ATSP that attempts to mimic the Lin-Kernighan heuristic [22] for the STSP, subject to the constraint that it does not reverse any tour segments.

C. Research Question

Starting from an arbitrary bidirectional ***O***(m log n) approximate Steiner tree T(S), [27], we ask: Is there a pretty successful algorithm that determines a convenient ATSP -tour based on orbiting T(S)?

II Solution Strategy and Algorithms

A. Solution Strategy A-TSP

Algorithm **A-TSP** is a construction method that starts with the determination of an approximate bidirectional Steiner tree T(S) connecting $S \subseteq V_G$. Round trips (clockwise or reverse) directly made on that tree are used to acquire several permutations $P \in S^{|S|}$ directly serving for the layout determination $P \rightarrow \Omega(P)$. **Fig.** 1 enables a first self-explanatory overview of the mode of action **A-TSP** proceeds. **Fig.** 2 gives a complete algorithmic description with special references to the used sub-algorithms. The numbers **1** … **4** correspond to the blocks in **Fig.** 2. We believe that the algorithms' description based on verbal description combined with graph description language and simple

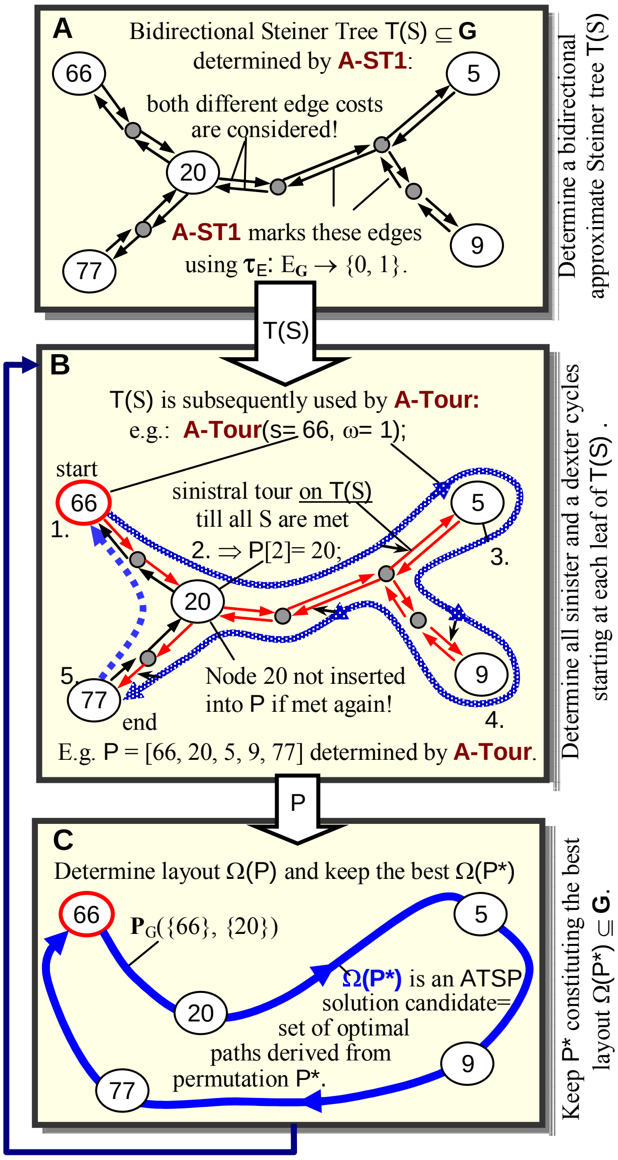

Fig. 1 Devolution of Algorithm **A-TSP**

algebraic notation is the best way to make them uniquely traceable for a successful and efficient implementation.

1 Build a unique function $\nu: S \to \{1,2, ..|S|\}$, e.g. directly derived from the input order. Build an edge-queuing optimal path graph OPG $\mathbf{P_G}(\{p\}, S)$ realized by one call to all $S \setminus \{p\}$ observing turn restrictions χ, [28] [29]. The corresponding path distances $\pi(\kappa(q))$ are stored in matrix **mx.**

2 **A-ST1, Fig.** 3, develops an approximate S-connecting bidirectional Steiner tree T(S) returning the edge- and vertex-marking τ_E and τ_V describing $T(S) \subseteq \mathbf{G}$.

3 Take all $s \in S$ as start node calling **A-Tour**(s, ω), **Fig.** 6, using each s twice (i.e. ω= 1 and ω= 2):
ω=1: orbiting clockwise T(S): P= [66,20,5,9,77, (66)],
ω=2: orbiting T(S) reverse: P= [66, 77, 9, 5, 20, (66)].
Permutation array P is set by **A-Tour**, see bolw.
Determine the cost **c** of the current round-trip Ω(P). Set P as P* if **c** underbids **c***. An anticlockwise tour is unnecessary ($d^* \leq 2$ => break cycle) if T(S) has the topology "snake" (d* in block **7** of algorithm **A-ST1**).

4 Edge label $\mu^*: E_G \to \{0,1\}$ is built by optimal path algorithm $\mathbf{P}_G()$ denoting the best approximate ATSP –tour $\mathbf{G}' = \Omega(P^*) \subseteq \mathbf{G}$ described by
$E_{G'} = \{e \in E_G: \mu^*(e)=1\}$; $V_{G'} = \boldsymbol{dom}(E_{G'}) \cup \boldsymbol{rng}(E_{G'})$.

Sub-graph $\Omega(P^*) \subseteq G$ is the best round-trip through S in **G** described by μ*. The executing optimal path algorithm $\mathbf{P}_G(\{P^*[i]\},\{P^*[i+1]\})$ efficiently observes turn restrictions, [28].

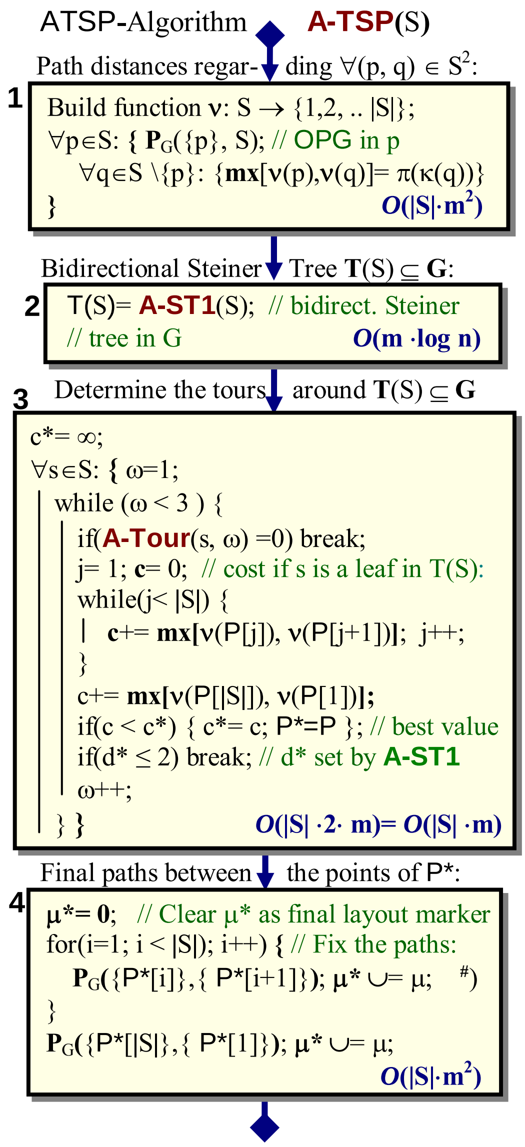

Fig. 2 ATSP Algorithm **A-TSP**

B. Steiner Tree Algorithm A-ST1

O(m log n) -Algorithm **A-ST1** develops a bidirectional Steiner tree T(S) , **Fig.** 3, from which we derive several permutations $P \in S^{|S|}$ in order to massively reduce the exponential time effort ***O***(|S|!), [27]. Each P serves as prescription for an ATSP –tour corresponding to **Fig.** 1.

Nomenclatura additionally necessary for A-ST1

π: $V_G \to \boldsymbol{R}_+$ is called vertex potential updated by **A-ST1** to determine the optimal path forest implicitly solved in Block 3 of **Fig.** 3. Finally, π(y)= $C(\bar{P}_G(\{\eta(y)\}, \{y\}))$ are the cost of the bidirectional connection from source η(y) to $y \in V_G$.

σ: $V_G \to V_G$ denotes a vertex-predecessor function updated by **A-ST1** determining the optimal path forest implicitly solved in Block 3 of **Fig.** 3. Finally, σ(y) is the predecessor of node y, i.e. edge (σ(x), x) $\in E_{\bar{F}_G(S)}$.

Q = first in – first out (FIFO) or last in – first out (LIFO) queue efficiently storing scan-eligible nodes $j \in V_G$ with potential π(j). To fetch an item q from Q we write q∈: Q. Inserting q is written by Q ∪= q.

R $\subseteq V_G^2$ is an equivalence relation called "Connected with". We use the following operations:

set $_R$(s) creates an equivalence class $[s]_R$ with $s \in V_G$ and $R = V_G^2$ (initially pure reflexivity). ***O***(n)

prove $_R$(x, y)? ⇔ (x, y)∈ R? proves if x and $y \in V_G$ are connected via the current Steiner tree T(S). ***O***(1)

unite $_R$(x, y) ⇔ $[x]_R \cup [y]_R$ unites two equivalence classes $[x]_R \subseteq V_G$ with $[y]_R \subseteq V_G$ to one larger class with the corresponding update of R. ***O***(n)

Ψ: $E_{T(S)} \times \{1, 2\} \to S^{|S|}$ with P= Ψ((u, v), i) is a permutation of S (sequence of stopovers) received when starting a tour on T(S) at edge $(u, v) \in E_{T(S)}$ observing ω. We define that each $s \in S$ is stored only once into P , i.e. when encountered for the first time.

H = priority queue for border edges $B = \{(a, b) \in E_G: \eta(a) \neq \eta(b)\}$ with cost c((a, b))= λ((a, b))+λ((b, a))+ π(a)+ π(b). B is affiliated to a bidirectional forest $\bar{F}_G$ (S). We use the following queue-operations:

insert $_H$(e, c(e)) inserts a border edge e∈ B into H observing the reorganizing of the queues internal order with respect to connection cost c(e); $O(\log |B|)$.

find_min $_H$() provides an edge e_{min} with $c(e_{min}) = \min_{e\in H}\{c(e)\}$, deletes e_{min} reorganizing H.

Fig. 3 describes algorithm **A-ST1**:

---

1 Initialization of the vertices V_G, the edges E_G, the vertices S. Fill the queue Q with set S.
2 Select a vertex q from Q and remove q from Q.
3 Take all incident neighbors q of p and try to improve their potential π(q) (currently $\pi(q) = \bar{P}_G(\{\eta(q)\},\{q\})$). Replace the potential π(q) if the path via p is better. For this case keep p as predecessor of q and keep η(p) as the new tree assignment η(q) for q. Put node q into the queue Q that contains all "scan-eligible" nodes.
4 If Q is not empty go to block 2. Otherwise, no further path improvement is possible.
5 The forest $F = \bar{F}_G(S) = \bigcup_{y\in V_G\setminus S} \bar{P}_G(\{\eta(y)\},\{y\})$ of |S| optimal path trees does exist with $E_F = \{(x, y)\in E_G: y\in V_G\setminus S \wedge x = \sigma(y)\}$ and $V_F = V_G$. Take all s∈S as disjoint sets (equivalence classes) as singletons $[s]_R$ of the binary relation $R \subseteq S^2$, necessary to initialize the further set operations ***prove***$_R$ and ***unite***$_R$ needed below. Take all $y\in V_G$ and add them to that set $[s]_R$ to which their roots η(s) already belong, i.e. ***unite***$_R$(η(y),y) corresponds to $R = R \cup \{(\eta(y), y)\}$ ≈ "y connected with root η(y)". Ending block 5, all vertices V_G have been uniquely assigned to equivalence classes, each of them corresponds uniquely to a tree in the forest $\bar{F}_G$.
6 Take all "border edges" $(x, y)\in E_G$ having **different** roots $\eta(x) \neq \eta(y)$. Take an edge (x, y) or (y, x), here with $x \geq y$, and determine the connection cost related to the chain

$$\bar{P}_G(\eta(x), x) \to (x) \underset{(y,\,x)}{\overset{(x,\,y)}{\rightleftarrows}} (y) \to \bar{P}_G(y, \eta(y)).$$

Calculate $c = \lambda((x,y)) + \lambda((y,x)) + \pi(x) + \pi(y)$ with $\pi(x) = C(\bar{P}_G(\{\eta(x)\},\{x\}))$ and π(y) correspondingly (cost determination π in Block **3**). ***insert*** edge e= (x, y) with cost c into queue H.
7 Select by ***find_min*** $_H$() all those |S|-1 minimal edges $e = (a, b)\in H$ (counted with z) that connect different sets $[a]_R \neq [b]_R$, checked with ***prove*** $_R$(a, b). Unite by ***unite*** the different sets $[a]_R$ and $[b]_R$. Mark the edges (with τ_E) and nodes (with τ_V) of the paths $\bar{P}_G(x, \eta(x))$ and $\bar{P}_G(y, \eta(y))$ including border edge (a, b) as "belonging to the current tree T(S)". End if |S|-1 minimal border edges from queue H had been selected. Steiner tree T(S) results to $T(S) = [V_{T(S)}, E_{T(S)}]$,

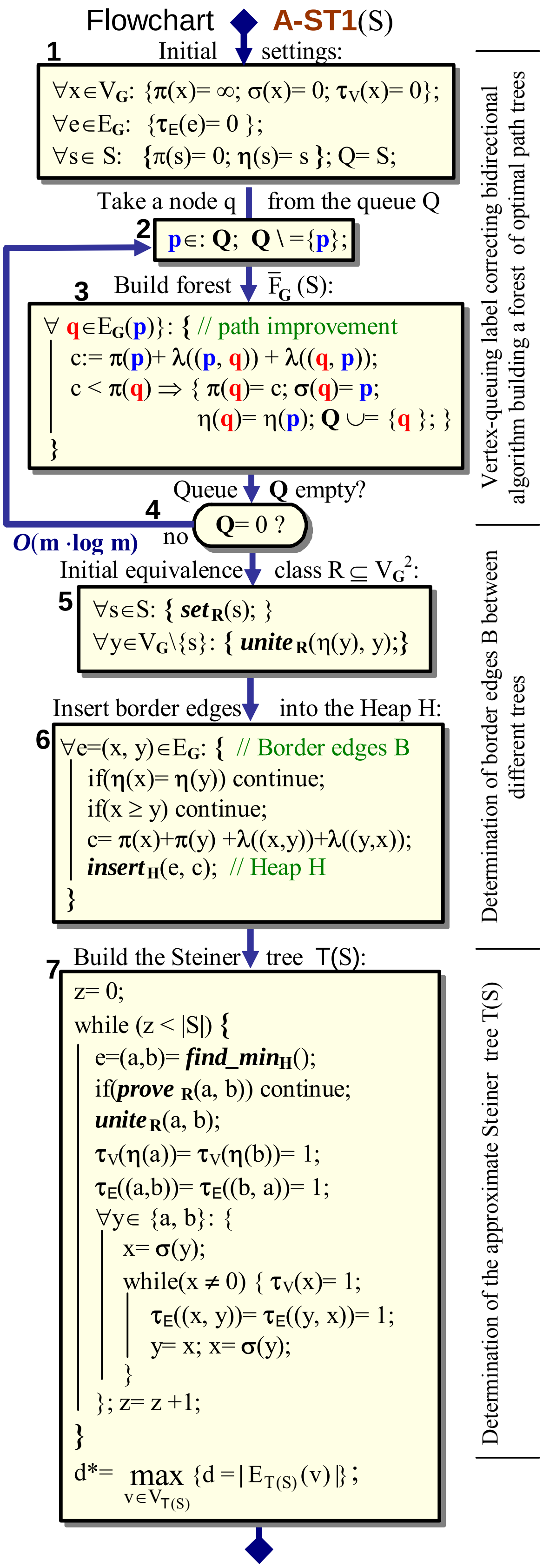

Fig. 3 Approx. Steiner Tree Algorithm **A-ST1**

$V_{T(S)}=\{x\in V_G: \tau_V(x)=1\}$, $E_{T(S)}=\{e\in E_G: \tau_E(e)=1\}$. d* represents the maximum out-degree comparing all vertices v $\in V_{T(S)}$. If $d^* \le 2$ then T(S) has the topology "snake". This knowledge is successfully used by sub-algorithm **A-TSP** above.

C. Algorithm A-Tour

A-Tour(s, ω) determines a permutation P as the sequence of stopovers encountered when start point s∈ S is a leaf and a tour on and around T(S) is executed (beginning with leave s) on the edges $E_{T(S)}$ clockwise (ω=1) or reverse (ω= 2). If start point s is not a leaf then **A-Tour** returns 0. Else, a permutation P is uniquely been filled with the sequence of stopovers encountered during the tour on T(S), **Fig.** 5 a).

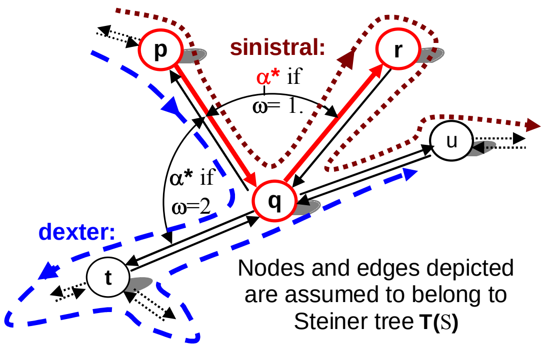

Fig. 4 **A-Tour** starting with (p, q)∈ $E_{T(S)}$

Nomenclatura additionally necessary for A-Tour

δ_V: $V_{T(S)} \rightarrow \{0, 1\}$: $\delta_V(x)=1$ indicates that node x∈ $V_{T(S)}$ has been "already scanned" during the specific trip regarding s and ω around T(S).

δ_E: $E_{T(S)} \rightarrow \{0, 1\}$ denotes with $\delta_E(e)=1$ that e∈ $E_{T(S)}$ has been marked as "already scanned" during the specific trip regarding parameters s and ω around T(S).

degout$_{T(S)}$: $V_{T(S)} \rightarrow \mathbb{N}$ yields with ***degout***$_{T(S)}$(x) the out-degree of node x∈$V_{T(S)}$, i.e. the number of edges leaving x.

angle(p, q, r, ω) provides the angle between the current edge (p, q) and the next edge (q, r) regarding to ω= 1 (sinistral) or ω= 2 (dexter), **Fig.** 4.

improve(p, q, r, ω, α*) is a function that returns **true** if the angle spanned by the nodes p, q, and r is (a) smaller than α* as to ω= 1 ≈ sinistral around T(S) ⇔ (b) larger than α*, as to ω= 2 ≈ dexter around T(S).

L = {x∈ S: ***degout***$_{T(S)}$(x)= 1} is the set of leaves of the Steiner tree T(S). See **Fig.** 5 L= {9, 13, 66, 77}.

Lemma 1
The determination of all round-trips on T(S) can be restricted to those tours that start at the points $S \subseteq V_{T(S)}$.

Proof: We regard a tour Ψ((u, v), ω) on the edges $E_{T(S)}$ starting at an arbitrary edge (u, v)∈$E_{T(S)}$ with u∈$V_{T(S)}$\S and assume the first encountered stopover is u'∈S. I.e. **A-Tour** makes the first entry $P_2[1]$= u' ∈S. Now we take that edge (u', v')∈$E_{T(S)}$ that is used by Ψ((u, v),ω) to proceed the tour passing u'∈S. A second tour Ψ((u', v'), ω) makes its first entry $P_2[1]$= u', i.e. $P_2[1]= P_1[1]$. Because parameter ω remains unchanged it follows the equal proceeding of both tours, i.e. $\Omega(P_1)= \Omega(P_2)$ completing $P_1= P_2$. ■

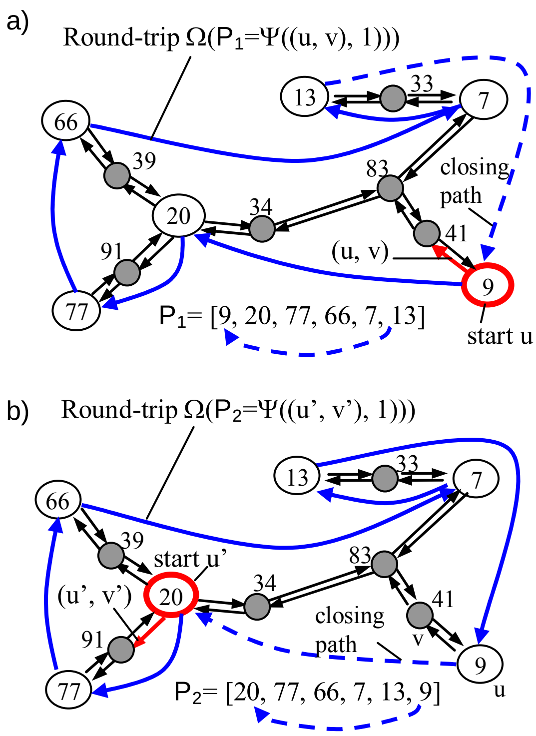

Fig. 5 Elucidating Lemma 2

Example, see **Fig.** 5:
$P_1= \Psi((u, v), \omega) = \Psi((83, 34), 1)= P_2= \Psi((u', v'), \omega) = \Psi((20, 91), 1)=[20,77,66,7,13,9]$.

Conclusion: Considering **Lemma 1** for the implementation of **A-TSP** brings about an essential reduction of solution time due to the restriction to the start points = stopovers S.

Lemma 2 For any ATSP-cycle $\Omega_1= \Omega(P_1=\Psi((u, v), \omega))$ with leaf u∈ L ⊆ S there is at least one cycle $\Omega_2= \Omega(P_2=\Psi((u', v'), \omega))$ with (u, v) ≠ (u', v') such that $\Omega_1 = \Omega_2$

Proof Let us regard permutation $P_1= \Psi((u, v), \omega))$ connected with a tour directly executed on the bidirectional Steiner tree T(S) starting with edge (u, v)∈ $E_{T(S)}$ at leaf u∈ L ⊆ S. We correspondingly regard now $P_2=\Psi(($ u'=

P_1[2], v'), ω) as the tour starting with the second stopover $P_1[2]= P_2[1]$= u' on the first tour (same ω). Let (u', x) ∈ $E_{T(S)}$ that edge that is passed by the first tour after crossing u'. We take v'= x to define Ψ((u', v'), ω) we look for. Since $P_1[2]= P_2[1]$ and u= $P_1[1]$ is a leaf of T(S) no further stopover can be situated between u and u'. That means that u is the last stopover of Ψ((u', v'), ω). It follows that P_1= [u, u', u"…] and P_2= [u', u", …, u] provide the same ATSP cycles $\Omega(P_1)= \Omega(P_2)$. That means that a round trip on T(S) is not necessary because there is another round trip that makes this one's determination superfluous. ■

Example **Fig.** 5: P_1= Ψ((9, 41), 1)= [9,20,77,66,7,13], **Fig.** 5 a), P_2= Ψ((20, 91), 1)= [20,77,66,7,13,9], **Fig.** 5 b). It follows with **Lemma 2**: $\Omega(P_1)= \Omega(P_2)$. I.e. that Ψ((20, 91), 1) is not necessary to be determined by **A-Tour**.

Remark: **Lemma 2** is based on the fact that u is a leaf (u∈ L⊆ S). If u is not a leaf of T(S) the tour P_2= Ψ((u', v'), ω) cannot be completed such that u is the last stopover of P_1. E.g. **Fig.** 5b): $\Omega(P_1$=Ψ((20, 91), 2)) = [20, 77, 9, 7, 13, 66] ≠ $\Omega(P_2$=Ψ((77, 91), 2)) = [77, 20, 9, 7, 13, 66] because node 20∈ S is not a leaf in $E_{T(S)}$.

Conclusion for designing algorithm **A-Tour**

Regarding a round trip directly executed by **A-Tour** on Steiner tree T(S) for the sake of permutation generation it is now clear that only the following nodes in $\bar{S} \subseteq V_{T(S)}$ should be start nodes on T(S) ⊆ G:

$\bar{S}$ = S \ {u'∈S: u' has a predecessor (foregoing last stopover) u∈L ⊆ S}.

E.g., in **Fig.** 5 the following Ψ-related tours on T(S) are not necessary to be determined for finding further solution candidates:

Ψ((20,91),1), Ψ((20,39),1), Ψ((20,34),1), // clockw.
Ψ((20,91),2), Ψ((20,34),2), // reverse
Ψ((7,83),1), Ψ((7,33),1), // clockw.
Ψ((7,83),2), Ψ((7,33),2). // reverse

We decided to confine the permutation search to start. points {s∈ S: ***degout***$_{T(S)}$(s) = 1} = set of leaves of S.

Algorithm **A-Tour**, **Fig.** 6, determines with each call two round-trips Ψ((u, v), 1) and Ψ((u, v), 2) and stores the best of them. Each time a node s∈ S is encountered during the trip for the very first time it is plotted into the next entry in P. **A-Tour** is controlled by parameters s∈ S and ω∈ {1, 2}: s∈ S ⊆ $E_{T(S)}$ is the start node in T(S) at which the tour around T(S) has to begin for this call.

1 Returns 0, if the node s∈ S is not a leaf in the Steiner Tree T(S).
2 Empty all labels δ_E and labels δ_V as "not being scanned on T(S)". Note, $E_{T(S)}$(s) is the set of all direct neighbors reachable from s. Since ***degout***$_{T(S)}$(s) = 1, only one edge (s, y)∈ $E_{T(S)}$ leaves s pointing uniquely to only one node y∈$V_{T(S)}$. Mark s and edge (s, y) as "scanned" and set s∈ S as the first entry for the permutation P, initialize count= 1 (used as end condition ≈ number of stopovers found).
3 If y∈ S was scanned then node y cannot be a stopover not yet been encountered before. Go to **7**.
4 If y is not a stopover go to 7.
5 Node y is a stopover: Set y into the next entry of the permutation P and mark it "scanned".
6 Prove whether all stopovers have been collected. If so, return 1 (completion).
7 Regard all direct neighbors z∈$E_{T(S)}$(y) and the corresponding edges e= (y, z)∈ $E_{T(S)}$. If y is a leaf in

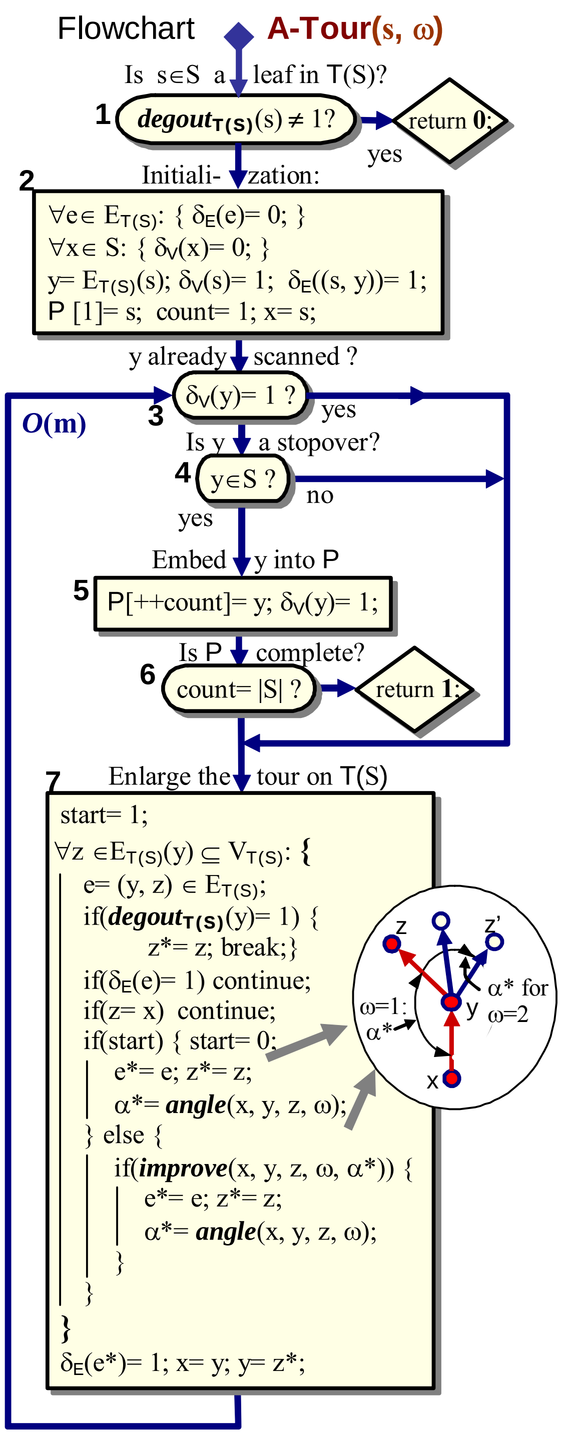

Fig. 6 Algorithm **A-Tour**

T(S) then break; If edge e has already been scanned then continue ($\delta_E(e)= 1$). If edge (y, z) points back to x (because z= x) continue. If start =1 take (e*, z*, α*) as temporary best successor information. Function ***angle()*** indirectly determines (relatively to edge (x,y)) that edge (leaving y, i.e. edge (y,z)) that is successor edge dependant on ω=1 or 2: α* represents the current minimum angle (ω= 1) or the current maximum angle (ω= 2) spanned by x – y – z. If start=2, then update (function ***improve***) the stored best values e*, z*, α* so far a further edge(s) leaves y on the Steiner Tree T(S) providing a "better" (angle) successor. If all edges of the bunch on y have been treated the edge e*= (y, z*) is that edge on which **A-Tour** has to proceed. Mark this edge as "scanned" ($\delta_V(e^*)= 1$) and set the presupposition (x=y; y= z*) to proceed with vertex y as before: Go to **3**.

III Performance Analysis

Testing near real-world applications we have constructed a random graph generator capable to produce large two-dimensional digraphs generally having edge cost $\lambda((x, y)) \neq \lambda((y, x))$ being randomly generated from a user-dependant closed interval. Turn restrictions at crossings might be set during the simulation to prove the algorithm's accuracy (layout change). **Table 1**, related to **Fig. 7** and **Fig. 8**, compares algorithm **A-TSP** and exact algorithm **A-TSP_opt**, Richter [30]), on a 2.4 GHz duo-core AMILO Xi 2528 PC. The analysis is based on a random grid graph G_1 having n= 5000 nodes and m= 19716 edges with edge cost randomly selected from the interval [0, 40]. First, we randomly generated sets of stopovers S with different but small cardinality $8 \leq |S| \leq 13$ to enable a comparing with optimal solutions. For each of these sets we randomly generated 8 different clusters of stopovers S and recorded the mean run-time and mean $\bar{\varepsilon}$ approximation. Corresponding to **Fig.** 2 algorithm **A-TSP** runs with $O(|S| \cdot m^2)$. However, **Fig. 7** and **Fig. 8** approve a nearly linear dependence: **Table 2** likewise reveals a linear dependence on the graphs' edge number m. This is attributed to the established optimal path algorithm (block **1** and **4** of **Fig.** 2). There, $P_G(..)$ is an edge-queuing label correcting (LC) path algorithm whose run-time is always better than forecasted by $O(m^2)$. **Fig. 8** reveals the **A-TSP** time performance: Solution and the directly included representation of the corresponding connection structure (colored emphasized within the nets) fulfill the requirements given above! The proposed implementation of turn restrictions shows that their influence on solution time is absolutely negligible.

IV Conclusion and Prospects

Construction Algorithm **A-TSP** is a very fast deterministic near-optimal ATSP heuristic being directly executed in directed graphs that do not need to consider the triangle inequality $d(u, w) < d(u, v) + d(v, w)$. The algorithm bases upon the search for a cost minimal cycle around an approximate bidirectional Steiner with respect to the set of stopovers S. **A-TSP** has an average approximation $\bar{\varepsilon} = (C_{appr} - C_{opt}) / C_{opt} = 0{,}05$ $(0{,}0 \leq \varepsilon \leq 0{,}16)$ with an empirical standard deviation $\delta \leq 0{,}038$. The performance tests' worst solution of **A-TSP** is 1.16 times over the optimum! It reveals a time dependence proportional to nearly |S| ·m. Therefore, **A-TSP** enables real-time ability for large problem sizes, e.g. here on grid graphs G with n= $|V_G|$= 5000, m= $|E_G|$= 19716, |S|= **350** ⇒ nominal time ≈ **2 sec**; |S|= **1440** ⇒ nominal time ≈ **10 sec**. The algorithm correspondingly solves also the Symmetric TSP (STSP). **A-TSP** doesn't need any preparation nor parameter adjustment. It turns out that **A-TSP** is a new serious competitor for all existing ATSP heuristics, especially qualified for real world apps and navigation tools that depend on the tours' real-time calculation instantly executed on the latest online digraphs while observing turn restrictions! Further investigations have to consider (a) comparisons with TSPLIB instances, [24], (b) theoretical cost bound related to the optimal solution.

|S|	t_opt / ms	A-TSP / ms	$\bar{\varepsilon}$	δ
8	79	86	0,0317	0,0365
9	135	93	0,0169	0,0187
10	660	95	0,0715	0,0385
11	6.001	98	0,0553	0,0381
12	73.430	103	0,0712	0,0534
13	1.032.552	109	0,0542	0,0421

Table 1 ε-approximation of algorithm **A-TSP** measured on a random grid graph having n= 5000 nodes, m= 19716 edges. It denotes:

- t_opt nominal time needed by **A-TSP$_{opt}$**;
- t_appr nominal time needed by **A-TSP**;
- $\bar{\varepsilon}$ average approximation $(c_{appr}-c_{opt}) / c_{opt}$ (mean of eight clusters);
- δ empirical stand. deviation $\sqrt{\frac{1}{n-1}\sum_{k=1}^{n}(\epsilon_k - \bar{\varepsilon})^2}$;

n= |V(G)|	m= |E(G)|	|S|	nominal time / ms
20.000	79.434	500	11.172
40.000	159.200	500	24.332
60.000	239.020	500	36.961
80.000	318.868	500	49.848
100.000	398.734	500	64.156

Table 2 Runtime of **A-TSP** on digraphs up to 100.000 nodes with constant |S|= 500

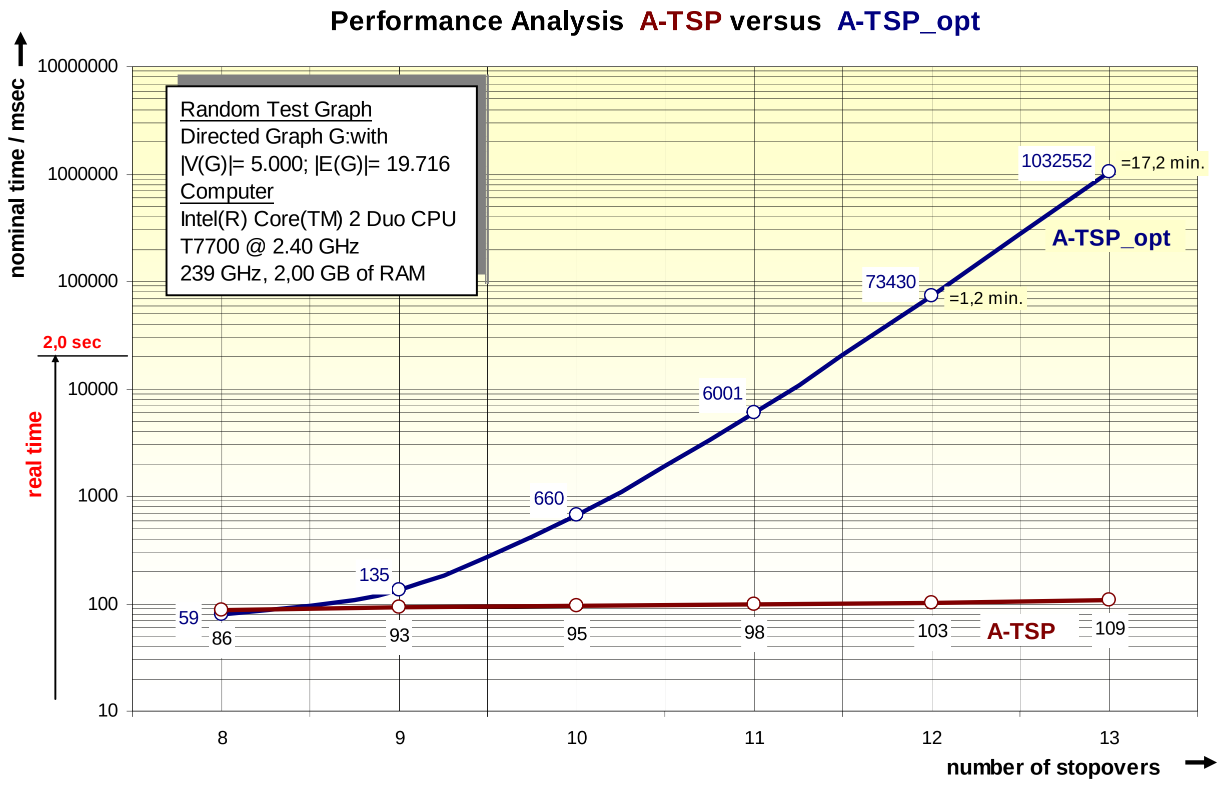

Fig. 7 **Nominal runtime A-TSP ⇔ A-TSP_opt (real-time ability only for** ≤ 11 stopovers).

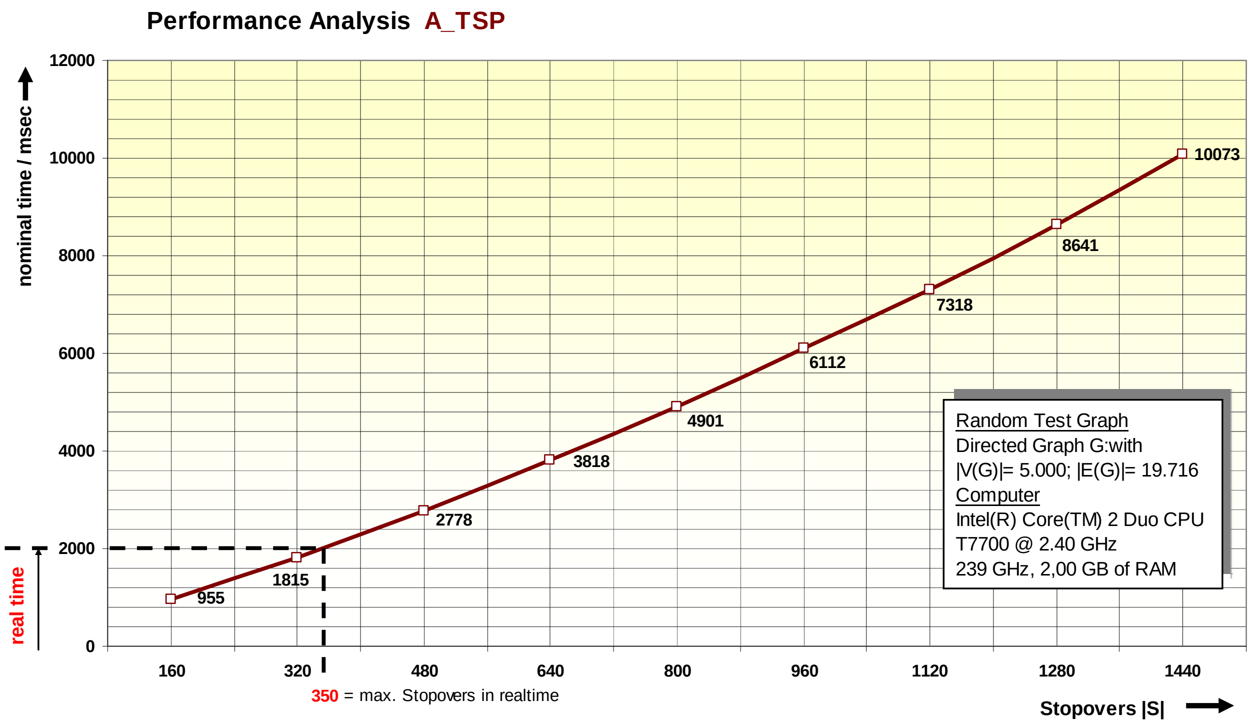

Fig. 8 Nominal runtime of the ATSP- algorithm **A-TSP** corresponding to Fig. 1 and Fig. 2

References

[1]Ascheuer N., Fischetti M., Grötschel M.: Solving the Asymmetric Traveling Salesman Problem with Time Windows by Branch-and-Cut, *ZIB Berlin, Preprint SC 99-31* (Aug 1999), ISSN 0933-7811

[2]Ascheuer N., Fischetti M., Grötschel M.: A polyhedral study of the asymmetric traveling salesman problem with time windows, , *ZIB Berlin, Preprint SC 97-11* (Feb 1997), ISSN 0933-7911

[3]Blom M., Krumke S., de Pape W. , Stougie L.: The Online-TSP Against Fair Adversaries, ZIB Berlin, *ZIB-Report 00-09 (Mar 2000), ISSN 1438-0064*

[4]Buriol L., Franga P. M., and Moscato P.: A new memetic algorithm for the asymmetric traveling salesman problem. Submitted for publication, 2001.

[5]Carpaneto G., Dell'Amico M., and Toth P.. Exact solution of large-scale asymmetric traveling salesman problems. *ACM Transactions on Mathematical Software*, 21:394-409, 1995.

[6]Carpaneto G. and Toth P.. Some new branching and bounding criteria for the asymmetric traveling salesman problem. *Management Science*, 26:736-743, 1980.

[7]Cirasella J., Johnson D.S., McGeoch L.A., and Zhang W.: The asymmetric traveling salesman problem: Algorithms, instance generators, and tests. In A.L. Buchsbaum and J. Snoeyink, editors, *Algorithm Engineering and Experimentation*, Third International Workshop, ALENEX 2001, Lecture Notes in Computer Science 2153, pages 32-59. Springer 2001.

[8]Dantzig G., Fulkerson R., Johnson S.. 1954. Solution of a large-scale travelling salesman problem. *Op. Res. 2* 393-410.

[9]Floyd Robert W.: Algorithm 97 Shortest Path, *Communication of the ACM 5*, 1962, page 345

[10]Frieze A., Galbiati G., and Maffioli F.. On the worst-case performance of some algorithms for the asymmetric traveling salesman problem. *Networks*, 12:23-39, 1982

[11]Garey M.R., Johnson D.S.: *Computers and Intractability*, ISBN 0-7167-1044-7, W.G ‘. Freeman, New York 1979

[12]Gilsinn I., Witzgall C.: A Performance Comparison of Labelling Algorithms for Calculating Shortest Path Trees, *NBS Technical Note 772*, U.S. Department of Commerce, 1973

[13]Glover F., Gutin G., A.Yeo, and A.Zverovich: Construction heuristics and domination analysis for the asymmetric TSP. *European J. Oper. Res.,* 129:555-568, 2001.

[14]Gutin G. and Zverovich A.. Evaluation of the Contractor-Patch Heuristic for the Asymmetric TSP. (to be published).

[15]Gutin G., Punnen Abraham P., [Ed.]: *The Traveling Salesman Problem and Its Variations*, 2007, Springer-Verlag New York Inc. 0-387-44459-9, ISBN-13: 9780387444598

[16]Johnson D. S., McGeoch L. A.: and Rothberg E. E.. Asymptotic experimental analysis for the Held-Karp traveling salesman bound. In Proc. *7th Ann. ACM-SIAM Symp. on Discrete Algorithm*s, pages 341–350. Society for Industrial and Applied Mathematics, Philadelphia, 1996.

[17]Johnson D. S. and McGeoch L. A.: The traveling salesman problem: A case study in local optimization. In *E. H. L. Aarts and J. K. Lenstra, editors, Local Search in Combinatorial Optimization*, pages 215–310. John Wiley and Sons, Ltd., Chichester, 1997.

[18]Herrmann D.: *Algorithmen Arbeitsbuch*, Addison-Wesley, 1992, ISBN 3-89319-481-9

[19]Kanellakis P. C. and Papadimitriou C. H.: Local search for the asymmetric traveling salesman problem. *Oper. Res.,* 28(5):1066-1099, 1980.

[20]Karp R. M.: A patching algorithm for the nonsymmetric *TSP. SIAM J. Comput.,* 8(4):561–573, 1979.

[21]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, editors, *The Traveling Salesman Problem*, pages 181–205. John Wiley and Sons, Chichester, 1985

[22]Lin S. and Kernighan B.W.: An effective heuristic algorithm for the traveling salesman problem. *Operations Research*, 21:972-980, 1973

[23]Miller D.L. and Pekny J.F.: Exact solution of large asymmetric traveling salesman problems. *Science*, 251:754-761, 1991.

[24]G.Reinelt G.: 1991. TSPLIB - a traveling salesman library. *ORSA J. Computing 3* 376-384.

[25]G. Reinelt G.: *The Traveling Salesman: Computational Solutions of TSP Applications*. LNCS 840. Springer-Verlag, Berlin, 1994.

[26]Repetto B.W.: Upper and Lower Bounding Procedures for the Asymmetric Traveling Salesman Problem. *PhD thesis, Graduate School of Industrial Administration*, Carnegie-Mellon University, 1994.

[27]Richter P.: Present Approximate Algorithms for Steiner's Problem in Graphs - Classification and Two New Fast Approaches, *Systems Science* (PL ISSN 0137-1223, Technical University of Wroclaw); Volume 17 - Number 2 1991 2-28.

[28]Richter P.: Optimal Path Determination Observing Turn Restrictions, 1 st CEAS European Air and Space Conference, ID CEAS-2007-707, Berlin, 10.-13.9. 2007, *ISSN 0700-4083* (DGLR Bonn, Germany)

[29]Richter P.: Efficient Double Root Optimal Path Determination, 1 st CEAS European Air and Space Conference, ID CEAS-2007-706, Berlin, 10.-13.9. 2007, *ISSN 0700-4083* (DGLR Bonn, Germany)

[30]Richter P.: Asymmetric TSP and MSP with Turn Restrictions - Exact Real-Time Computation for Stopovers < 11, CEAS 2009, *in Proc. of the CEAS 2009, ISBN 1857682130,* Manchester, UK, 26-29 Oct 2009

[31]Tamaki G.: Alternating cycle contribution: a tour-merging strategy for the traveling salesman problem. *Max-Planck Institute Research Report MPI-I-2003-1-007*. Saarbrücken, Germany.

[32]Van Der Veen Jack A. A.: Solvable cases of the traveling salesman problem with various objective functions, Ruksuniversiteit Groningen, 1992

[33]Vella D., 2001. Using DP-Constraints to Obtain Improved TSP Solutions. *M.S. Thesis,* University of Ottawa, Canada.

[34]Zhang W.: Truncated branch-and-bound: A case study on the asymmetric TSP. In *Proc. of AAAI1993 Spring Symposium on AI and NP-Hard Problems,* pages 160-166, Stanford, CA, 1993.

[35]Zhang W.: Depth-first branch-and-bound versus local search: A case study. In *Proc. 17th National Conf. on Artificial Intelligence (AAAI-2000),* pages 930-935, Austin, TX, 2000.