CEAS 2009 European Air and Space Conference (ISBN 1 85768 208 4) 26 -29 October 2009, Manchester, UK

Asymmetric Traveling Salesman and Multi-Stopover Problem with Turn Restrictions - Exact Real-Time Computation for Stopovers ≤ 10

PETER H. RICHTER
O & S Consultancy

The Asymmetric Traveling Salesman Problem ATSP as well as a the Asymmetric Multi-Stopover Problem AMSP (also Multi-Destination Problem), is intractable (***NP*** -hard). However, using the fastest optimal path algorithms combined with the fastest permutation method enables the real-time solution for problem sizes $|S| \leq 10$ (12 including start and target, AMSP) subject to the set S of stopovers that are to meet via an optimal cost cycle (ATSP) or an optimal cost path (AMSP). We give an algorithm directly working on directed graphs observing turn restrictions indispensable for today's civil and military navigation applications.

Introduction

The Asymmetric Traveling Salesman Problem ATSP and the Asymmetric Multi-Stopover Problem AMSP represent a large number of important scientific and engineering problems. Unlike the well known Symmetric Travelling Salesman Problem STSP (shortly TSP), ATSP's and AMSP's instances are usually defined with digraphs **G** whose edges E_G are not necessarily symmetric and also not bounded to the triangle inequality $d(x, z) \leq d(x, y) + d(y, z)$ as to a distance function $d: V_G^2 \rightarrow \boldsymbol{R}_+$. The ATSP and AMSP remain intractable as long as they are ***NP*** –hard, [10], i.e. we have to accept instances with small problem sizes while persisting on optimal real-time solutions [1]).

Today's new navigation applications require solutions for ATSP and AMSP that have to meet stronger requirements than ever before: They ..

– cannot accept heuristic solutions that are 1.5-fold worse than the exact result.
– demand solutions on directed graphs (digraphs for "asymmetrical" apps) in contrast to the overwhelming TSP-heuristics that are based on undirected graphs ("symmetrical" apps).
– require the optimal tour computation instantly executed on the latest graphs (online road maps) without further data preparation and control variables' calibration.
– adhere on the real time solutions for the problems just in time.
– accept for the overwhelming applications problem sizes $\leq$ 10.
– unconditionally demand solutions with the observance of turn restrictions, an indispensable demand of today's civil and military navigation applications!

Despite the problem's intractability, we deliberately want to get exact solutions meeting the demands above, knowing that we can only reach real-time performance for small problem sizes. Algorithm **A-TSP_opt** presented here fulfills the requirements for problem sizes ≤ 10..

As the performance analysis of **A-TSP_opt** on large digraphs shows, the solutions fulfill the requirements given above! Furthermore, the proposed efficient implementation of turn restrictions shows that their solution time delay is negligible.

[1]) "realtime" ≈ "occurring immediately"≈ nominal event time < 2 sec")

1.1 Related Work

We regard the exact ATSP and AMSP solution although the problems are "intractable" what means that the exponential solution time consumption typically compels to look for heuristics that approximately solve the problem in polynomial and, hopefully, acceptable time.

Here, we will (and we have to) except the exponential time dependence. But, as we show below, for small problem sizes we even get the algorithm's real-time ability due to very fast sub-algorithms that could efficiently be designed to observe turn restrictions.

Carpaneto et al. [2] presented a lowest-first, branch-and-bound algorithm for the ATSP based on the Assignment Problem relaxation and on a subtour elimination branching scheme. The effectiveness of the algorithm derives from reduction procedures and parametric solution of the relaxed problems associated with the nodes of the branch-decision tree. Large-size, uniformly, randomly generated instances of complete digraphs with up to 2000 vertices are solved on a DEC station 5000/240 computer in less than 3 minutes of CPU time. The authors gave in [3] a Fortran CDT implementation of an ATSP-algorithm based on the Assignment Problem relaxation and on a subtour elimination branching scheme.

Miller et al. [21] gave an algorithm for the exact ATSP – solution. Their results show that the algorithm performs remarkably well for some classes of problems, determining an optimal solution even for problems with large numbers of cities, yet for other classes, even small problems thwart determination of a provably optimal solution.

Chekuri et al. [4] considered a problem related to the AMSP: Find a minimum cost path from s to $t \in V_G$ in digraph **G** of total length at most B that maximizes the number of distinct nodes visited by the path. Main results: An $\boldsymbol{O}(\log 2 \cdot c_{opt})$ approximation in digraphs G.

For ATSP heuristics we refer to Kumar et al. [19] (transforming ATSP instances in STSP instances, worst case performance ratio of 20/9 when the ratio d_{max}/d_{min} is smaller than 4/3.), Reinelt [23], Johnson et al. [18], Cirasella et al. [6] and multiple algorithms as in Repetto [24], Zhang et al. [31], [32], Glover et al. [12].

Richter [28] proposed a new fast constructive general application heuristic directly working in digraphs.

1.2 Notation

In order to be able to grasp the algorithm's detailed functioning we rely on the description based on the concepts of analysis and graph description language.

Graph G = [set of vertices (crossings)V_G, set of edges (streets)E_G], $E_G \subseteq V_G^2$ represents a net (e.g. road map, airlift net, ..). We use **n**= $|V_G|$ and **m**= $|E_G|$. **G** is finite, directed, and connected. We regard digraphs= "directed graphs" whose edges $(p, q) \in E_G$ and (q, p) may have different cost λ:

λ: $E_G \to \boldsymbol{R}_+$, denotes the edge cost: $e= (p, q) \in E_G \Rightarrow \lambda(p, q)$ are the cost (length, time, ..) for passing edge e from p to q.

Sub-graph of **G** is called a graph $\mathbf{H}= [V_H, E_H]$ if $V_H \subseteq V_G$ and $E_H \subseteq E_G \cap V_H^2$. For any $\mathbf{H} \subseteq \mathbf{G}$ we define its cost $C(\mathbf{H})= \sum_{e \in E_H} \lambda(e)$.

π: $E_G \to \boldsymbol{R}_+$ called "edge potential" is determined by the optimal path algorithms.
$\pi(e) < \infty \Leftrightarrow$ edge e belongs to a preliminary or final path (subgraph) from the destined start to the destined target. Otherwise, no path leads via edge e.

$\mathbf{s} \in V_G$ is start point for the AMSP.

$\mathbf{t} \in V_G$ is destination point (target) for the AMSP.

$\mathbf{S} \subseteq V_G$ is the set of sub-ordinate targets or "stopovers"
AMSP: stopovers to be visited via an optimal path from **s** to **t** considering turn restrictions.
ATSP : stopovers to be visited via a closed optimal round trip considering turn restrictions.
We use the writing p= **S**[i] to address that $p \in \mathbf{S}$ is stored at offset i, i.e. $\mathbf{S}= \bigcup_{i=1}^{|S|} \{S[i]\}$ using S as array.

χ: $V_G \to \mathfrak{P}(E_G^2)$ denotes turn restrictions attached to the crossings V_G. E.g. $\chi(q)= \{\ldots,((p, q), (q, r)), ..\} \Rightarrow$ "Coming from p to q passing edge (p, q) continuing the journey via edge (q, r) is not allowed. For the efficient implementation of turn restrictions we refer to [26].

∞ "infinite" $\cong$ maximum floating point number.

σ: $E_G \to E_G$ is calculated as follows:
A1a(x, Y) builds up an ***OPG*** from x to all targets $y \in Y$. Because **A1a** is an edge-queuing label-correcting method it holds for each target $y \in Y$ that all its incoming edges $e' = (p, y) \in V_G \times \{y\} \subseteq E_G$ get a potential $\pi(e')$.
But only the edge e with $\pi(e)= \min_{\forall e'=(p',y) \in E_G} \{\pi(e')\}$ reaching y is that edge whose predecessor sequence $\sigma(e), \sigma(\sigma(e)), \ldots$ leads back via a minimal cost path with cost $\pi(e)$. σ is an edge predecessor function uniquely assigning one predecessor edge $\sigma(e)$ to edge e backwards (opposite direction) to the start x.
A1b(X, y) builds up an ***ROPG*** from target y backwards to all start nodes $x \in X$.
Because **A1b** is an edge-queuing label-correcting method it holds for each target $x \in X$ that all its outgoing edges $e' = (x, p) \in \{x\} \times V_G \subseteq E_G$ get a potential $\pi(e')$.
But only the edge e with $\pi(e)= \min_{\forall e'=(x,p) \in E_G} \pi(e')$ leaving x is that edge whose successor sequence $e= (x,p), \sigma(e), \sigma(\sigma(e)), \ldots$ leads forward via a minimal cost path with cost $\pi(e)$ to target y (one target for all X). σ is an edge successor function that uniquely assigns one successor edge $\sigma(e)$ to edge $e \in E_G$ towards target y.
A2 (x, y) simultaneously builds up an ***ROPG*** and an ***OPG*** (i.e. a hybrid structure) resulting to an optimal path $\mathbf{P}_G(x, y)$ from x to y. $\sigma(e)$ is either predecessor or successor edge of edge e dependant of its ***OPG*** or ***ROPG*** membership.

$\mu \subseteq E_G$ denoted as "layout" describes that edges that constitute the result (sub-graph) **G*** (AMSP or ATSP). μ is cumulatively determined by **A2** or **A1a** using their latest σ: $e \in \mu \Leftrightarrow e \in E_{G^*} \subseteq E_G$, i.e.
G* has edges $E_{G^*} = \mu \subseteq E_G$ and
nodes $V_{G^*} = \{ \boldsymbol{dom}(E_{G^*}) \cup \boldsymbol{rng}(E_{G^*}) \}$.

ν= $\mathbf{S} \to \{1,2,.. |\mathbf{S}| \}$ uniquely assigns an offset for the array **S** the stopovers are stored. E.g. S written as array: **S**=[55, 63, 162, 302, 386] means $\nu[55]= 1$, $\nu[63]= 2$, $\nu[386]= 5 = |\mathbf{S}|$.

mx is an $|\mathbf{S}| \times |\mathbf{S}|$ cost matrix: **mx**[ν(p), ν(q)] is to contain the cost of an optimal path from $p \in V_G$ to $q \in V_G$ observing turn restrictions χ.

start is an 1-dimensional array to contain the cost **start**[ν(p)] of the optimal path from start **s** to the points $p \in \mathbf{S}$ observing turn restrictions.

target is an 1-dimensional array to contain the cost **target**[ν(p)] of the optimal paths from the points $p \in \mathbf{S}$ to the target **t** observing turn restrictions.

AP(&indi) with variable indi (initially 0) provides each call a new permutation over the set {1,2,.. |**S**|}, [29]. Once, the last of all the |S|! permutation has been delivered, indi is set by **AP** to 1.
Calling **P**= **AP**(&indi) means filling array **P**[1], **P**[2], ..**P**[|**S**|] with a new permutation not provided before. We regard the containing integers as offsets addressing array **S** as follows: [**S**[**P**[1]], **S**[**P**[2]], …**S**[**P**[|**S**|)]]] is the new proposed ordered set the stopovers are to run through graph **G**, $\mathbf{S}[\mathbf{P}[i]] \in V_G$.

Set operations $\cup$, $\cap$, / and arithmetic operations +, - are used in short form to reduce description size.
E.g. a= a + b; shortened to a+=b;
A= A $\cup$ B; shortened to A $\cup$= B;

Problem Definition

Given **G**, λ, $\{\mathbf{s}, \mathbf{t}\} \cup \mathbf{S} \subseteq V_G$.

ATSP: We look for a circular tour $\mathbf{H^*} \subseteq \mathbf{G}$ visiting **S** such that its cost C(**H***) are minimized:

$$\mathbf{c^*}= C(\mathbf{H^*})= \min_{\substack{\text{cycle } H \subseteq G, S \subseteq V_H \\ H \text{ observes } \chi}} \{C(H)\}. \qquad (2.1)$$

AMSP: We look for a path $H^* \subseteq \mathbf{G}$ from **s** to **t** visiting **S** such that its cost C(**H***)are minimized:

$$\mathbf{c^*}= C(\mathbf{H^*})=. \min_{\substack{\text{path } H \subseteq G \text{ from } s \text{ to } t \text{ observes } \chi, \\ S \subseteq V_H}} \{C(H)\}. \ (2.2)$$

Notice: Turn restrictions χ may cause the optimal structure to have cycles, see more in [26].

ATSP is ***NP*** –hard, [10]. AMSP is ***NP*** –hard too.

Proof:

(1) AMSP belongs to the class ***NP*** because any instance of AMSP is polynomial time verifiable (so-called "non-deterministically solvable").

(2) We regard the special case in equation (2.2), that **s** = **t**, what means that this problem instance belongs to ATSP. Thus, ATSP $\propto$ AMSP ≈ "If ATSP and AMSP belong to ***NP***, ATSP is ***NP***-hard, and ATSP $\propto$ AMSP, then AMSP is ***NP***-hard". [2]) ■

That means for both problems, that the exact solution exponentially depends on the problem size |**S**|.

Exact Algorithm A-TSP_opt Observing Turn Restrictions

The exact ATSP and AMSP algorithm **A-TSP_opt** relies on the following *Edge Queuing* (EQ) Optimal Path Algorithms that consider turn restrictions:

A1a (label correcting method) solves the "One-to-All" problem. **A1a** (x, Y) builds an ***Optimal Path Graph OPG*** containing the paths $\{\mathbf{P}_G(x, y)\}_{y\in Y}$ from node x to all nodes $y \in Y \subseteq V_G$

A1b (label correcting method) solves the "All-to-One" problem. **A1b** (X, y) builds a ***Reverse Optimal Path Graph ROPG*** containing the paths $\{\mathbf{P}_G(x, y)\}_{x\in X}$ from all node $x \in X$ to the only target y.

A2 (label setting method) solves the "One-to-One" problem. The result consists of a partial ***OPG*** and a partial ***ROPG*** containing the optimal path $\mathbf{P}_G(\mathbf{s}, \mathbf{t})$ from start s to target t in **G.**

Description of **A-TSP_opt**, Fig. 1

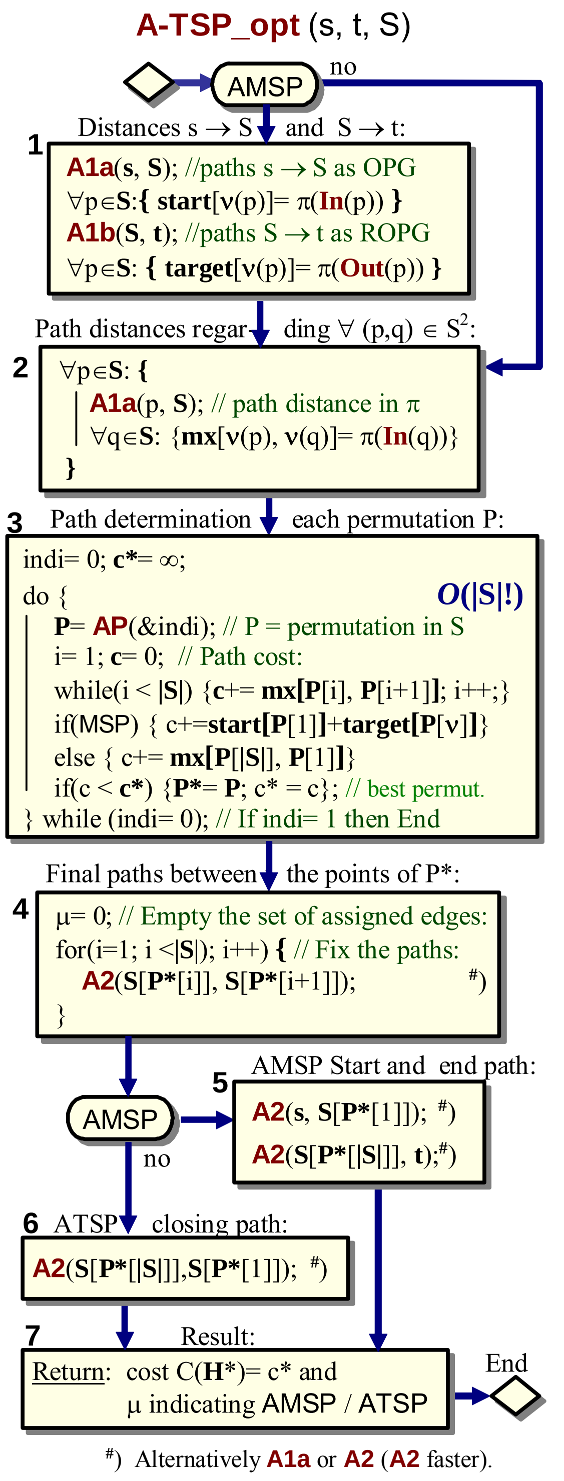
Block 1: In the case of AMSP:
Build (with one call of **A1a**) all ν optimal paths $\mathbf{P}_G(\mathbf{s}, \mathbf{S})$ storing their cost in the array **start** and (with one call of **A1b**) all ν optimal paths $\mathbf{P}_G(\mathbf{S}, \mathbf{t})$ storing their cost in the array **target**.

Block 2: Use **A1a** to determine the cost of all paths $\{\mathbf{P}_G(p, q)\}_{(p,q)\in S\times S}$ to be stored into the matrix **mx**.

Block 3: Calculate for each possible permutation **P** the connection cost c and add the following cost:
AMSP: the path cost from **s** to the first point of **P** and the path cost from the last point of **P** to **t**.
ATSP: the path cost from the last point of **P** to the first point of **P** (closing the round trip).
Keep the best permutation **P*** and the total connection cost **c***.

Block 4: To get the final layout μ:
Empty μ and recalculate the connection paths for the best permutation **P*** using **A2** or **A1a**.

Block 5: Complete the final layout μ for the AMSP:
Recalculate with **A2** or **A1a** the path from **s** to the first point of **P*** and the path from the last point of **P*** to **t** .


A-TSP_opt (s, t, S)
AMSP
no
Distances s → S and S → t:
1
Path distances regarding ∀ (p,q) ∈ S²:
2
3 Path determination each permutation P:
Final paths between the points of P*:
4
AMSP Start and end path:
5
AMSP
no
6 ATSP closing path:
7 Result:
End

```
1:
A1a(s, S); //paths s → S as OPG
∀p∈S:{ start[ν(p)]= π(In(p)) }
A1b(S, t); //paths S → t as ROPG
∀p∈S: { target[ν(p)]= π(Out(p)) }

2:
∀p∈S: {
    A1a(p, S); // path distance in π
    ∀q∈S: {mx[ν(p), ν(q)]= π(In(q))}
}

3:
indi= 0; c*= ∞;
do {                                                  O(|S|!)
    P= AP(&indi); // P = permutation in S
    i= 1; c= 0;  // Path cost:
    while(i < |S|) {c+= mx[P[i], P[i+1]]; i++;}
    if(MSP) { c+=start[P[1]]+target[P[ν]]}
    else { c+= mx[P[|S|], P[1]]}
    if(c < c*) {P*= P; c* = c}; // best permut.
} while (indi= 0); // If indi= 1 then End

4:
μ= 0; // Empty the set of assigned edges:
for(i=1; i <|S|); i++) { // Fix the paths:
    A2(S[P*[i]], S[P*[i+1]]);                      #)
}

5:
A2(s, S[P*[1]]); #)
A2(S[P*[|S|]], t);#)

6:
A2(S[P*[|S|]],S[P*[1]]); #)

7:
Return:  cost C(H*)= c* and
         μ indicating AMSP / ATSP
```

#) Alternatively **A1a** or **A2** (**A2** faster).

Fig. 1 Algorithm **A-TSP_opt** exactly solving ATSP and AMSP observing turn restrictions.

Block 6: Complete the final layout μ for the ATSP :
Recalculate with **A2** or **A1a** the path from the last point of **P*** to the first point of **P*** (closing cycle) .

Block 7: Return total cost c* and edge marking μ. The optimal result is the subgraph $H^* \subseteq \mathbf{G}$ with $E_{H^*} = \mu \subseteq E_G$; $V_{H^*} = \{\mathit{dom}(\mu) \cup \mathit{rng}(\mu)\} \subseteq V_G$.

[2]) $\Pi_1 \propto \Pi_2$ ≈ There exists a polynomial transformation from problem Π_1 to Π_2.

One-to-All Path Algorithm A1a Observing Turn Restrictions

A1a (x, Y), Fig. 2, determines with one call all optimal paths $\{\mathbf{P}_G(x, y)$: $y \in Y$, $\mathbf{P}_G(x, y)$ observes $\chi\}$ building an Optimal Path Graph (***OPG***) from x to all points $V_G/\{x\}$. **A1a** cannot be finished if all $\{y \in Y \subseteq V_G$: y reachable from x$\}$ have been reached because it is an LC-algorithm. Only if no edge potential can be improved the algorithm ends. It returns as result the cost of the structure (sub-graph), π and layout marker μ. Even in the case of cycles (due to χ), the edge predecessor function $\sigma: E_G \rightarrow \boldsymbol{R}_+$ remains unique. That is the advantage of this edge-queuing technique compared to usual vertex queuing algorithms using a vertex predecessor function σ': $V_G \rightarrow \boldsymbol{R}_+$ that cannot be unique in the case of cycles (due to χ)!

Notation

For remaining notations see chapter 1.2.

$\mathbf{x} \in V_G$ is start point for the current path determination.

$\mathbf{Y} \subseteq \mathbf{V_G}$ are the targets for the current path determination.

π: $E_G \rightarrow \boldsymbol{R}_+$ is called edge potential or edge cost
$\pi(e) < \infty$ means that edge e belongs to a preliminary or final path from the start **x** to one and only one $\mathbf{y} \in Y$ (there is "shorter" path from x to y).

$\mathbf{Q} \subseteq E_G$ is a queue established as LIFO queue (Last In, First Out). It contains all edges $e \in E_G$ that came into the queue due to the improvement of their potential. L is being set as new predecessor edge of edge e: $\sigma(e)$= L. Edge e gets the new potential and is will be set as a newly improved edge into Q.
If Q is empty, there are no scan-eligible edges available and the algorithm ends.

$\mathbf{L} \in E_G$ is an edge taken from Q whose neighbor edges are to scan for possible path improvements.

In(y) provides the edge $e \in E_G$ of minimum potential arriving $y \in V_G$ after the execution of **A1a**(x, Y), start node $x \in V_G$ and targets $Y \subseteq V_G$, corresponding to:

$$\left| \; e = \mathbf{In}(y) \text{ with } \pi(e) = \min_{\forall e'=(p',y) \in E_G} \{\pi(e)\}. \right.$$

Description A1a, Fig. 2

Block 1: Set for all edges $e \in E_G$ their potential $\pi(e)$ to infinite (∞= max. float number), their predecessors $\sigma(e)$ to e. Clear priority queue Q. Set the potential of all edges leaving start x to a cost that consist of their length. Put these start edges into the queue Q.

Block 2: If Q is empty, there is no edge whose potential can be improved by another predecessor edge.

Block 3: Take an edge L= (i, p) directed from i to p from Q observing the queue's corresponding data management.

Block 4: Regard all edges e leaving p so far its sequence is allowed by χ: If a new potential (on trial) $\pi(L)+\lambda(e)$ is smaller than the old one $\pi(e)$ a shorter current path from start x via edge L leads to edge e. Thus, L is the new predecessor of edge e, i.e. $\sigma(e)$= L; $\pi(e)$= c. Since the potential $\pi(e)$ of e has been improved, edge e has to be pushed into queue Q.

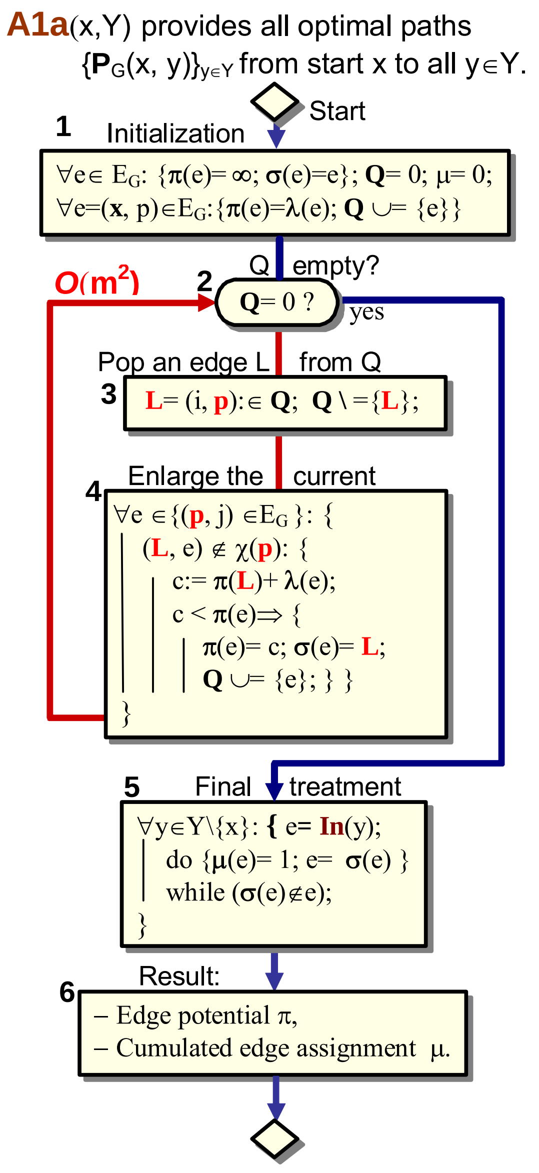


Fig. 2 Edge queuing label correcting algorithm **A1a** solving One-to-All considering turn restrictions. Effort= $O(m^2)$.

Block 5: Build the layout μ derived from σ: Regard all optimal paths from x to Y starting from the destinations' minimum potential edge ($y \in Y \Rightarrow e = \mathbf{In}(y)$) . Notice, several edges e with different potential $\pi(e)$ usually arrive some $y \in Y$. Only tracing back the path from y using edge **In**(y) ensures an optimal path backwards to start x! μ is cumulatively enlarged with each call of **A1a** and used by the superior application **A-TSP_opt**. Notice: Since Block 4 of **A-TSP_opt** contains the Zero-Setting of μ, the genuine edge marker accumulation begins with **A2** called in Block 4 of **A-TSP_opt** .

Block 6: Return π and the newly cumulated μ.

All-to-One Path Algorithm A1b Observing Turn Restrictions

A1b (X, y), Fig. 3, determines with <u>one call</u> all optimal paths $\{\mathbf{P}_G(x, y)\}_{x\in X}$ building an Reverse Optimal Path Graph (***ROPG***) from the target y backwards (i.e. reverse edge direction) to all points $\{x\in X \subseteq V_G$: y reachable from x$\}$. It returns as result π and μ. For the rest we refer to the introduction of **A1a**.

<u>Notation</u>

For remaining notations see chapter 1.2.

For Q and L we refer to 4.1.

$\mathbf{X} \subseteq V_G$ are the start points.

$\mathbf{y} \in V_G$ is the target.

σ: $E_G \to E_G$ is successor function that uniquely assigns successor edge $\sigma(e)$ to edge e. If $\sigma(e)$ has been set to a non-negative value, edge tupelo(e, $\sigma(e)$) is part of a preliminary or final path from some $\mathbf{x}\in X$ to target **y**.

π: $E_G \to \boldsymbol{R}_+$ is called edge potential or edge cost $\pi(e)<\infty$ means that edge e belongs to a preliminary or final path from some start $\mathbf{x}\in X$ to **y**.

Out(x) provides the edge $e\in E_G$ of minimum potential leaving some $x\in X$ after the execution of **A1b**(X, y) with target $y\in V_G$ corresponding to

$$e= \mathbf{Out}(x) \text{ with } \pi(e)= \min_{\forall e'=(x,p)\in E_G} \{\pi(e')\}.$$

Description A1b, Fig. 3

Block 1: Set the potential $\pi(e)$ of all edges E_G to infinite (max. float number) and their successors $\sigma(e)$ to e. Clear priority queue Q. Set the potential of all edges e= (p, y)$\in E_G$ arriving target y to their cost $\pi(e)$. Put all these edges arriving y as start edges into Q.

Block 2: If Q is empty, no further edge is available for a potential improvement.

Block 3: Take an edge L= (p, i)) directed from p to i from Q observing the queue's corresponding data management.

Block 4: Regard all edges e= (j, p) incoming to p so far the edge sequence (e, L) is allowed by χ: If a new potential $\pi(L)+\lambda(e)$ is smaller than the old one $\pi(e)$ a shorter current path from j to target y leads via L. Thus, L is set as new successor edge of edge e: $\sigma(e)$= L. e gets the new potential and is set as a newly improved edge into Q.

Block 5: Build the layout μ derived from σ: Regard all optimal paths from X to y starting from the starts' minimum potential edge ($x\in X \Rightarrow$ e= **Out**(x)) . Notice, several edges e with different potential $\pi(e)$ usually leave some $x\in X$. Only tracing the path from x using edge **Out**(x) ensures an optimal path to the target y!

Block 6: Return π and the newly cumulated μ.

A1b(X, y) provides the paths $\{\mathbf{P}_G(x, y)\}_{x\in X}$ from all $x\in X$ to target y.

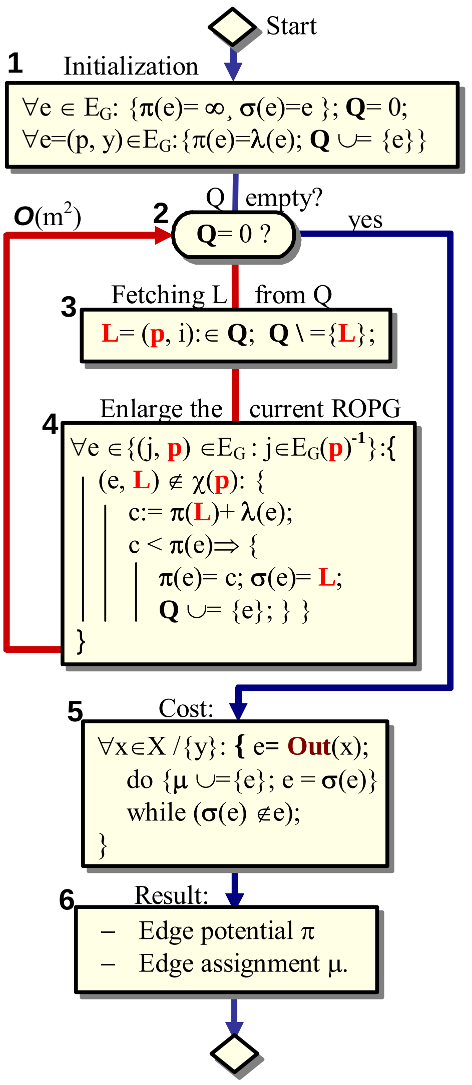


Fig. 3 Edge queuing label correcting algorithm **A1b** solving All-to-One observing turn restrictions. Effort= $O(m^2)$.

Double Root One-to-One Path Algorithm A2 Observing Turn Restrictions

Double-Rooted Edge-Queuing Label Setting Algorithm **A2**(x, y), see [26], determines an optimal path $\mathbf{P}_G(x, y)$ in a 2-wave-manner: It simultaneously develops

– an optimal path graph Ω= $[V_\Omega, E_\Omega]$ spreading forward around the passed start x and

– an optimal path graph $\bar{\Omega} = [V_{\bar{\Omega}}, E_{\bar{\Omega}}]$ spreading backwards (reverse edge direction) around the target y

till the both graphs meet (simplified).

Used in **A-TSP_opt**, **A2** has been introduced as rival for **A1a** because the performance comparison revealed a superiority over **A1a**, i.e. a time improvement up to 30% if $2 \le |\mathbf{S}| \le 9$ (see chapter 0).

Notation

x$\in$ V_G is the **start point** to find an optimal path $\mathbf{P}_G(x, y) \subseteq \mathbf{G}$ from x to y.

y$\in$ $\mathbf{V_G}$ is the **target point** to find an optimal path $\mathbf{P}_G(x, y) \subseteq \mathbf{G}$ from x to y.

$\Omega= [V_\Omega, E_\Omega] \subseteq \mathbf{G}$ is called "Optimal Path Graph" (***OPG***) that is developed by **A2** starting in the root $x \in V_G$. Initially, $\Omega= [0, 0]$.

$\bar{\Omega} = [V_{\bar{\Omega}}, E_{\bar{\Omega}}] \subseteq \mathbf{G}$ is called "Reverse Optimal Path Graph" (***ROPG***) that is developed "backwards" by **A2** starting in the root $y \in V_G$ (against edge direction). Initially, $\bar{\Omega} = [0, 0]$.

π: $E_G \to \boldsymbol{R}_+$ is denoted as **edge potential**, shortly **potential**. All edges $e= (p, q) \in E_G$ that have been assigned either to the current partial ***OPG*** Ω developed from start x or to the current partial ***ROPG*** $\bar{\Omega}$ backwards developed from target y get the cost $\pi(e)$ as follows:

a) $e \in E_\Omega$: $\pi(e= (p, q))= C(\mathbf{P}_G(x, q))$ = Cost of the path from x to q via p.

b) $e \in E_{\bar{\Omega}}$: $\pi(e= (p, q))= C(\mathbf{P}_G(p, y))$ = Cost of the path from p to y via q.

$\Pi \subseteq E_G \cup V_G$= "State-membership" of vertices and edges. Usually, Π is implemented as bit marker array.

a) If $e=(p, q) \in \Pi$ then edge e has been fetched from the queue Q and labeled as 'permanent'.

b) If the vertices $p, q \in \Pi$ then they belong to a "permanent" edge (p, q), i.e. $e= (p, q) \in \Pi$.

Q is a priority queue usually organized as binary heap to contain all edges whose potential were improved in the course of the **A2** development. Initially, all start edges (leaving x) and all target edges (incoming to target y) are set into Q. Once, an edge e is fetched from Q its potential $\pi(e)$ can never be improved again. That is why **A2** is called a Label Setting (LS) algorithm different to **A1a** and **A1b** that are LC-algorithms.

L is the minimum potential edge fetched from Q. L is stored to L_Ω if $\mathbf{L} \in E_\Omega$ and stored to $\mathbf{L}_{\bar{\Omega}}$ if $\mathbf{L} \in E_{\bar{\Omega}}$.

$\mathbf{L_1}$, $\mathbf{L_2} \in E_G$ are called "current minimum touch edges" or "bridge edges". They arise in the course of the simultaneous ***OPG*** & ***ROPG*** development corresponding to Fig. 4. If $\mathbf{L_1} \in \Omega$ then $\mathbf{L_2} \in \bar{\Omega}$ and vice versa. Notice: The first touch between ***OPG*** and ***ROPG*** does not necessarily provide the final bridge for the optimal path $\mathbf{P}_G(x, y)$. The end condition is a somewhat more sophisticated (block 4 of Fig. 5).

σ: $E_G \to E_G$ is an edge sequence function describing the current ***OPG*** and ***ROPG*** development:

a) $e \in E_\Omega \Rightarrow \sigma(e) \in E_\Omega$ is the predecessor edge of edge e. Sequence {e=(p, q) ,σ(e), $\sigma(\sigma$(e)) ... (x, z)}, backwards read, describes the edge sequence of an optimal path $\mathbf{P}_G(x, q)$ from x to q within the ***OPG*** Ω of **G**.

b) $e \in E_{\bar{\Omega}} \Rightarrow \sigma(e) \in E_{\bar{\Omega}}$ is the successor edge of edge e. Sequence {e=(p, q) ,σ(e), $\sigma(\sigma$(e)) ..(z, **y**)} describes the edge sequence of an optimal path $\mathbf{P}_G(p, y)$ from p to y within the ***ROPG*** $\bar{\Omega}$ of **G**.

$ec \in E_G$ ('connection edge') is an edge incident to the current edge **L**:

$\mathbf{L} \in E_\Omega$ then $ec \in E_{\bar{\Omega}}$. $L \in E_{\bar{\Omega}}$ then $ec \in E_\Omega$. Edge ec is a bridge candidate to build the optimal path from x to y (via $L \to ec$ (if $L \in \Omega$) or via $ec \to L$ (if $L \in \bar{\Omega}$)).

$cc \in \boldsymbol{R}_+$ are the cost of the trial path via the edges $L \to ec$ ($ec \to L$, respectively), see variable ec.

Description A2, Fig. 5

Block 1: Set the potential π(e) of all edges to infinite (max. float number) and σ(e) to e. Assign x to Ω and y to $\bar{\Omega}$. Label x and y as "permanent". Clear priority queue Q. Set L_1 and L_2 to "not found" (-1) and c* to infinite. Set the potential of all edges leaving x (arriving y) to their length, put them into Q, assign them (including their nodes) as belonging to the ***OPG*** Ω (***ROPG*** $\bar{\Omega}$, respectively).

Block 2: If Q is empty (no further development is possible) go to Block 13.

Block 3: Fetch the minimum potential edge L from Q. Label L and its vertices as "permanent". Store the incident nodes of L a and b so that b is situated at the brink of the ***OPG***, ***ROPG*** respectively (see Fig. 4). Store the opposite root r= y (r= x) if $L \in E_\Omega$ ($L \in E_{\bar{\Omega}}$, respect.).

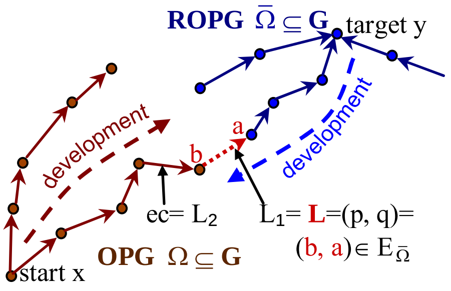


Fig. 4 Edges L_1 and L_2 connecting Ω and $\bar{\Omega}$

Block 4: The end condition occurs if the cost c* of the stored path (related to the bridge edges L_1 and L_2) is lower than the current connection cost related to the current bridge edges L_Ω and $L_{\bar{\Omega}}$.

Block 5: If L doesn't touch a vertex b belonging to the other sub-graph (Ω, $\bar{\Omega}$) proceed with Block 9.

Block 6: Node **b** of L belongs to the other sub-graph than L belongs to.

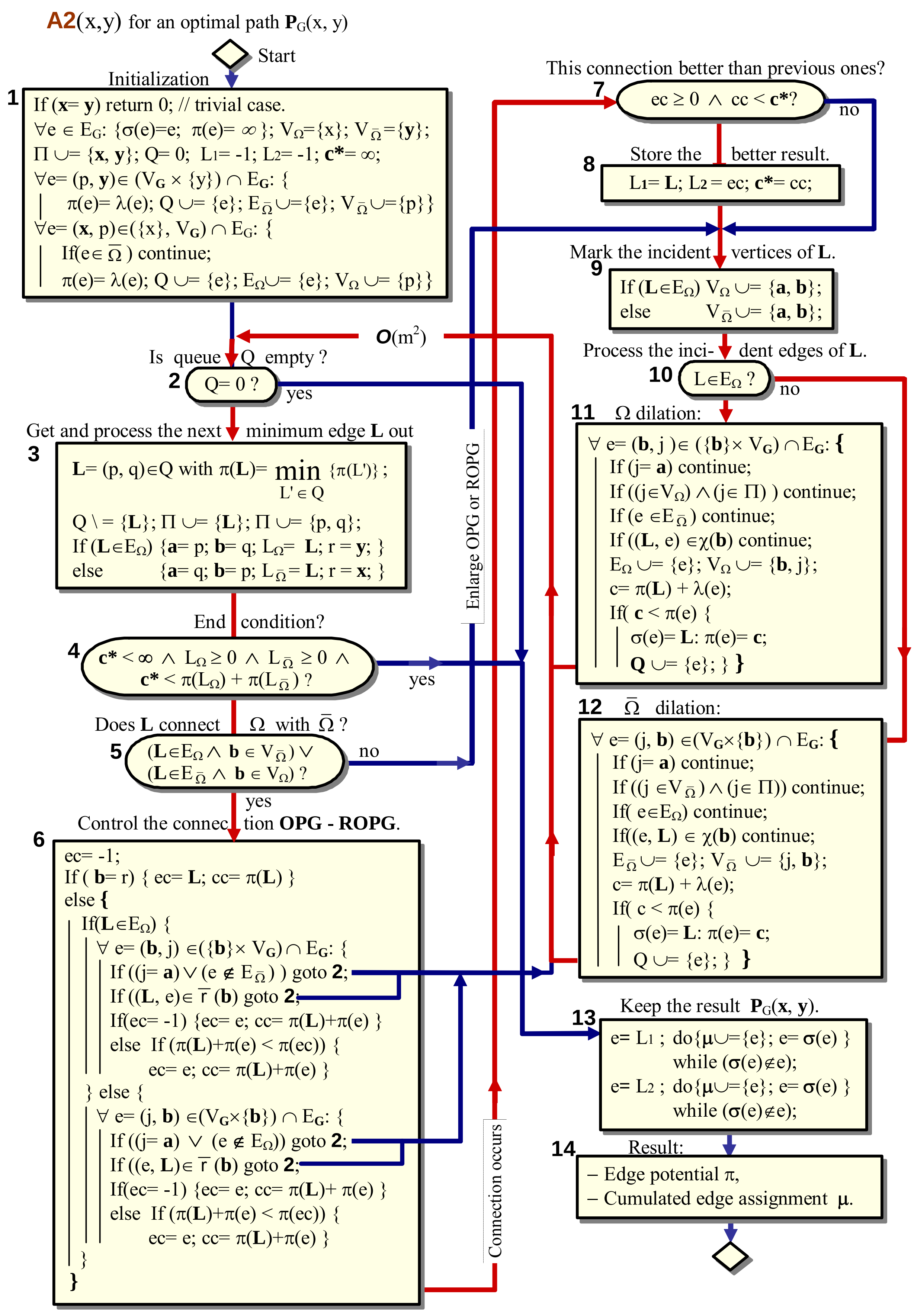
a) $L \in E_\Omega$: Take all outgoing edges e of b belonging to $E_{\bar{\Omega}}$ that observes turn restrictions, i.e. $(L, e) \notin \chi$. If the sum of the potential of the bridge edges L and e is smaller than that stored in cc then store ec= e (and cost cc= $\pi(\mathbf{L})$+ π(e)) . Regard (L, ec) as a new bridge candidate to provide the path $\mathbf{P}_G(x, y)$.

b) $L \in E_{\bar{\Omega}}$: Proceed corresponding to a) and Fig. 5.

Block 7: If the solution candidate (L, ec) is better than the previous ones …

Block 8: … store them as L_1 and L_2 with cost c*

Block 9: Mark the incident vertices of **L** belonging to the same sub-graph than L belongs to. Notice: Border vertices belong to V_Ω as well as to $V_{\bar{\Omega}}$.


A2(x,y) for an optimal path P_G(x, y)
Start
Initialization
Is queue Q empty ?
O(m²)
yes
Get and process the next minimum edge L out
End condition?
yes
Does L connect Ω with Ω̄ ?
no
yes
Control the connection OPG - ROPG.
Enlarge OPG or ROPG
Connection occurs
This connection better than previous ones?
no
Store the better result.
Mark the incident vertices of L.
Process the incident edges of L.
no
Ω dilation:
Ω̄ dilation:
Keep the result P_G(x, y).
Result:

```
1
If ($x = y$) return 0; // trivial case.
$\forall e \in E_G$: {$\sigma(e)=e$; $\pi(e)=\infty$}; $V_\Omega=\{x\}$; $V_{\bar\Omega}=\{y\}$;
$\Pi \cup= \{x, y\}$; Q= 0; $L_1= -1$; $L_2= -1$; $c^*= \infty$;
$\forall e= (p, y) \in (V_G \times \{y\}) \cap E_G$: {
|   $\pi(e)= \lambda(e)$; $Q \cup= \{e\}$; $E_{\bar\Omega} \cup= \{e\}$; $V_{\bar\Omega} \cup= \{p\}$}
$\forall e= (x, p) \in (\{x\}, V_G) \cap E_G$: {
|   If($e \in \bar\Omega$) continue;
|   $\pi(e)= \lambda(e)$; $Q \cup= \{e\}$; $E_\Omega \cup= \{e\}$; $V_\Omega \cup= \{p\}$}

2
Q= 0 ?

3
$L= (p, q) \in Q$ with $\pi(L)= \min_{L' \in Q} \{\pi(L')\}$;
$Q \setminus= \{L\}$; $\Pi \cup= \{L\}$; $\Pi \cup= \{p, q\}$;
If ($L \in E_\Omega$) {a= p; b= q; $L_\Omega$= L; r = y; }
else            {a= q; b= p; $L_{\bar\Omega}$= L; r = x; }

4
$c^* < \infty \wedge L_\Omega \ge 0 \wedge L_{\bar\Omega} \ge 0 \wedge$
$c^* < \pi(L_\Omega) + \pi(L_{\bar\Omega})$ ?

5
$(L \in E_\Omega \wedge b \in V_{\bar\Omega}) \vee$
$(L \in E_{\bar\Omega} \wedge b \in V_\Omega)$ ?

6
ec= -1;
If ( b= r) { ec= L; cc= $\pi(L)$ }
else {
  If($L \in E_\Omega$) {
  | $\forall e= (b, j) \in (\{b\} \times V_G) \cap E_G$: {
  | | If (($j= a$) $\vee$ ($e \notin E_{\bar\Omega}$) ) goto 2;
  | | If ((L, e)$\in \bar r$ (b) goto 2;
  | | If(ec= -1) {ec= e; cc= $\pi(L)+\pi(e)$ }
  | | else  If ($\pi(L)+\pi(e) < \pi(ec)$) {
  | |           ec= e; cc= $\pi(L)+\pi(e)$ }
  } else {
  | $\forall e= (j, b) \in (V_G \times \{b\}) \cap E_G$: {
  | | If (($j= a$) $\vee$ ($e \notin E_\Omega$)) goto 2;
  | | If ((e, L)$\in \bar r$ (b) goto 2;
  | | If(ec= -1) {ec= e; cc= $\pi(L)+ \pi(e)$ }
  | | else  If ($\pi(L)+\pi(e) < \pi(ec)$) {
  | |           ec= e; cc= $\pi(L)+\pi(e)$ }
  }
}

7
$ec \ge 0 \wedge cc < c^*$?

8
$L_1$= L; $L_2$ = ec; $c^*$= cc;

9
If ($L \in E_\Omega$) $V_\Omega \cup= \{a, b\}$;
else           $V_{\bar\Omega} \cup= \{a, b\}$;

10
$L \in E_\Omega$ ?

11
$\forall e= (b, j) \in (\{b\} \times V_G) \cap E_G$: {
| If ($j= a$) continue;
| If (($j \in V_\Omega$) $\wedge$ ($j \in \Pi$) ) continue;
| If ($e \in E_{\bar\Omega}$) continue;
| If ((L, e) $\in \chi(b)$ continue;
| $E_\Omega \cup= \{e\}$; $V_\Omega \cup= \{b, j\}$;
| c= $\pi(L) + \lambda(e)$;
| If( c < $\pi(e)$ {
| |  $\sigma(e)$= L: $\pi(e)$= c;
| |  $Q \cup= \{e\}$; } }

12
$\forall e= (j, b) \in (V_G \times \{b\}) \cap E_G$: {
| If ($j= a$) continue;
| If (($j \in V_{\bar\Omega}$) $\wedge$ ($j \in \Pi$)) continue;
| If( $e \in E_\Omega$) continue;
| If((e, L) $\in \chi(b)$ continue;
| $E_{\bar\Omega} \cup= \{e\}$; $V_{\bar\Omega} \cup= \{j, b\}$;
| c= $\pi(L) + \lambda(e)$;
| If( c < $\pi(e)$ {
| |  $\sigma(e)$= L: $\pi(e)$= c;
| |  $Q \cup= \{e\}$; } }

13
e= $L_1$ ; do{$\mu \cup= \{e\}$; e= $\sigma(e)$ }
      while ($\sigma(e) \ne e$);
e= $L_2$ ; do{$\mu \cup= \{e\}$; e= $\sigma(e)$ }
      while ($\sigma(e) \ne e$);

14
– Edge potential $\pi$,
– Cumulated edge assignment $\mu$.
```

Fig. 5 Double- root edge-queuing label setting algorithm **A2** simultaneously building a partial **OPG** Ω and a partial **ROPG** $\bar\Omega$ for finding an optimal path $P_G(x, y) \subseteq$ **G** considering turn restrictions χ

Block 10: Branch to Block 12 if L belongs to $\bar{\Omega}$.
Block 11: Enlarge the ***OPG*** Ω:
Take those outgoing edges (b, j) of node b that don't belong to $\bar{\Omega}$, whose node j is not labeled as "permanent" ($j \in \Pi$), and that observe turn restrictions: Assign e and its nodes to Ω.. Take e into the queue Q if its potential can be improved by $\pi(L) + \pi(e)$: L is the better predecessor of e than that before.
Block 12: Enlarge the ***ROPG*** $\bar{\Omega}$:
Proceed related to Block 11 above.
Block 13: The optimal path $\mathbf{P}_G(x, y)$ observing turn restriction is being derived from σ and the bridge edges L_1, L_2 to be cumulated into μ .
Block 14: Return π and the newly cumulatedμ.

Performance Test

Instances and measurement:
Testing real-world applications we have constructed a random graph generator capable to produce large two-dimensional grid digraphs generally having edge cost $\lambda((x, y)) \neq \lambda((y, x))$ randomly generated from a closed interval what generally makes invalid the triangle inequality. Turn restrictions at crossings might be set before or during the simulation to prove the algorithm's accuracy (i.e. causing layout changes).
The tests have been conducted on a 2.4 GHz duo-core AMILO Xi 2528 PC without to especially draw advantage of the two-processors existence.
We generated three graphs from which we randomly deleted 12% of their nodes including their incident edges resulting to:
G1: n= 465, m= 1.564;
G2: n= 4.405, m= 15.318; $\lambda: E_G \rightarrow [0, 40]$.
G3: n= 8.805, m= 30.694.

We used as time measurement method the C++ function `GetSystemTime()` that measures the nominal time (not the CPU-time). Each time measurement over the domain S in the charts Fig. 6 and Fig. 7 consists of the value $\frac{1}{8}\sum_{i=1}^{8} t_S$, based on 8 different random clusters of S. Each value t_S itself is the mean of 40 individual measurements each cluster.

Run-time complexity:
Fig. 1 reveals the worst case behavior of algorithm **A-TSP_opt**.
We use $\mathbf{n}= |V_G|$, $\mathbf{m}= |E_{G,}|$, $\mathbf{s}= |S|$:

Block 1: **A1a** and **A1b** run with an effort	$\boldsymbol{O}(m^2)$
Block 2: **A1a** is executed **s**-times:	$\boldsymbol{O}(\mathbf{s}\cdot m^2)$
Block 3: **AP** returns **s**!-times a new permutation:	$\boldsymbol{O}(\mathbf{s}!)$
Block 4: **A2** is executed **s**-times:	$\boldsymbol{O}(\mathbf{s}\cdot m^2)$
Block 5: **A2** is executed twice:	$\boldsymbol{O}(m^2)$
Block 6: **A2** is executed once	$\boldsymbol{O}(m^2)$

$\boldsymbol{O}$(**A-TSP_opt**)= $\boldsymbol{O}(s!) \approx \boldsymbol{O}(s^S)$

Evaluation:
Despite the awful runtime complexity $\boldsymbol{O}(s!)= \boldsymbol{O}(s^s)$, Fig. 6 reveals an excellent time performance of **A-TSP_opt** optimally solving AMSP and ATSP for $|S| \leq 8$ with less than 0.1 sec and for $|\mathbf{S}| \leq 10$ with less than 1 sec what means real-time ability for this problem size. Obviously, the time consumption of the permutation generation for the problem size $\mathbf{s} \leq 8$, Fig. 1, is significantly lower than that of the total path determination by **A1a**, **A1b**, and **A2**. This can be conceived by the nearly linear curve behavior $2 \leq \mathbf{s} \leq 8$ in Fig. 6 within the logarithmic representation for a quadratic function $\boldsymbol{O}(\mathbf{s}\cdot m^2)$ related to the optimal path determination. Fig. 7 reveals the solution time benefit if algorithm **A1a** in Fig. 1 is replaced by the double root algorithm **A2**, see [27]. Of course, the time advantage ends if the permutation generation effort prevails from $|S| > 10$, Fig. 7.
As the performance analysis on large digraphs shows, the selected set of very efficient sub-algorithms constituting algorithm **A-TSP_opt** fulfills the requirements mentioned in chapter 0 for a overwhelming number of civil and military real-time navigation applications where the number of stopovers remains $|S| < 11$. The proposed efficient implementation of turn restrictions enables a widened utilization for traffic and related optimization applications with respect to the ATSP and AMSP.

Conclusion

Due to the implementation and arrangement of the newly designed optimal path algorithms
- edge-queuing label-correcting algorithms **A1a**, **A1b**,
- edge-queuing label-setting algorithm **A2** and
- Trotter's permutetion algorithm **AP**,

the presented algorithm **A-TSP_opt** suffices applications that exactly solve ATSP and AMSP and related problems. Thereby, **A-TSP_opt** considers turn restrictions!
The exact real-time solution of ATSP and AMSP by **A-TSP_opt** is attractive for military and civil routing applications for
- $|\mathbf{S}| \leq 8$ with less than 0.1 sec and for
- $|\mathbf{S}| \leq 10$ with less than 1 sec.

so far the preconditions correspond to the given ones above.

References

[1] K. Bharath-Kumar, J. M. Jaffe: Routing to Multiple Destinations in Computer Networks, IEEE Transactions on Communications, Vol. COM-31, March 1983, p. 343-351

[2] G. Carpaneto, M. Dell'Amico, P. Toth: Exact solution of large-scale, asymmetric traveling salesman problems, ACM Transactions on Mathematical Software (TOMS), Volume 21 , Issue 4 (December 1995) ISSN:0098-3500

[3] G. Carpaneto, M. Dell'Amico, P. Toth: Algorithm 750: CDT: a subroutine for the exact solution of large-scale, asymmetric traveling salesman problems, ACM Transactions on Mathematical Software (TOMS) Volume 21, Issue 4 (December 1995) Pages: 410 - 415 ISSN:0098-3500

[4] Chandra Chekuri, Nitish Korula, Martin Pál: Improved algorithms for orienteering and related problems, Symposium on Discrete Algorithms, Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms, San Francisco, California, Pages 661-670, Year of Publication: 2008

[5] N. Christofides: An Algorithm for the Rural Postman Problem on a directed Graph, Mathematical Programming Study 26 (1986) 155-166

[6] J. Cirasella, D.S. Johnson, L.A. McGeoch, and W. Zhang. 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-Verlag, Berlin, 2001.

[7] E.W. Dijkstra: A Note on two Problems in Connection with Graphs, Numer. Mathematik 1, (1959)/ 269-271

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

[9] G. Gallo, S. Pallottino: A new algorithm to find the shortest paths between all pairs of nodes, Discrte Applied Mathematics 4, 1982, page 23 - 25

[10] M.R. Garey, D.S. Johnson: Computers and Intractability, ISBN 0-7167-1044-7, W.H. Freeman and Company, New York 1979

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

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

[13] M. Held and R. M. Karp: The traveling salesman problem and minimum spanning trees. Operations Res., 18:1138–1162, 1970.

[14] M. Held and R. M. Karp: The traveling salesman problem and minimum spanning trees: Part II. Math. Prog., 1:6–25, 1971.

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

[16] K.Helsgaun: An effective implementation of the Lin-Kernighan traveling salesman heuristic. European J. Op. Res. 126, 2000, 106-130.

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

[18] D. S. Johnson and L. A. McGeoch. 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.

[19] R. Kumar, H. Li : On Asymmetric TSP: Transformation to Symmetric TSP and Performance Bound; Department of Electrical Engineering University of Kentucky Lexington, KY 40506-0046

[20] E.L. Lawler, J.K. Lenstra, E.L. Rinnooy Kan, D. Shmoys: The Traveling Salesman Problem,Wiley, Chichester, 1985

[21] Donald L. Miller and Joseph F. Pekny: Exact Solution of Large Asymmetric Traveling Salesman Problems, Science 15 February 1991: Vol. 251. no. 4995, pp. 754 – 761, DOI: 10.1126/science.251.4995.754

[22] G. Reinelt. TSPLIB – A traveling salesman problem library. ORSA J. Comput., 3(4):376–384, 1991. The TSPLIB website is http://www.iwr.uni-heidelberg.de/iwr/comopt/software/TSPLIB95/.

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

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

[25] P. Richter: Efficient Shortest Path Algorithms for Road Traffic Optimization, 27th International Symposium on Advanced Transportation Applications (ISATA): Dedicated Conference on Advanced Transport Telematics, ISBN 0947719652, Aachen, 31.10.-4.11.1994, 79-86

[26] P. Richter: 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)

[27] P. Richter: 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)

[28] P. Richter: Near Optimal TSP Solution in Real-time - A New Asymmetrical Algorithm, submitted to IMAPP 2009, CCH Hamburg, 14--16 Oct 2009

[29] R. Sedgewick: Permutation Generation Methods, ACM Computing Surveys 9 (1977) 2, 137-164

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

[31] W. Zhang: 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.

[32] W. Zhang: 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.

[33] H.F. Trotter: Algorithm 115, CACM, 1962, referenced in Computer Journal, 14 (1971), Pascal-Code in [13]

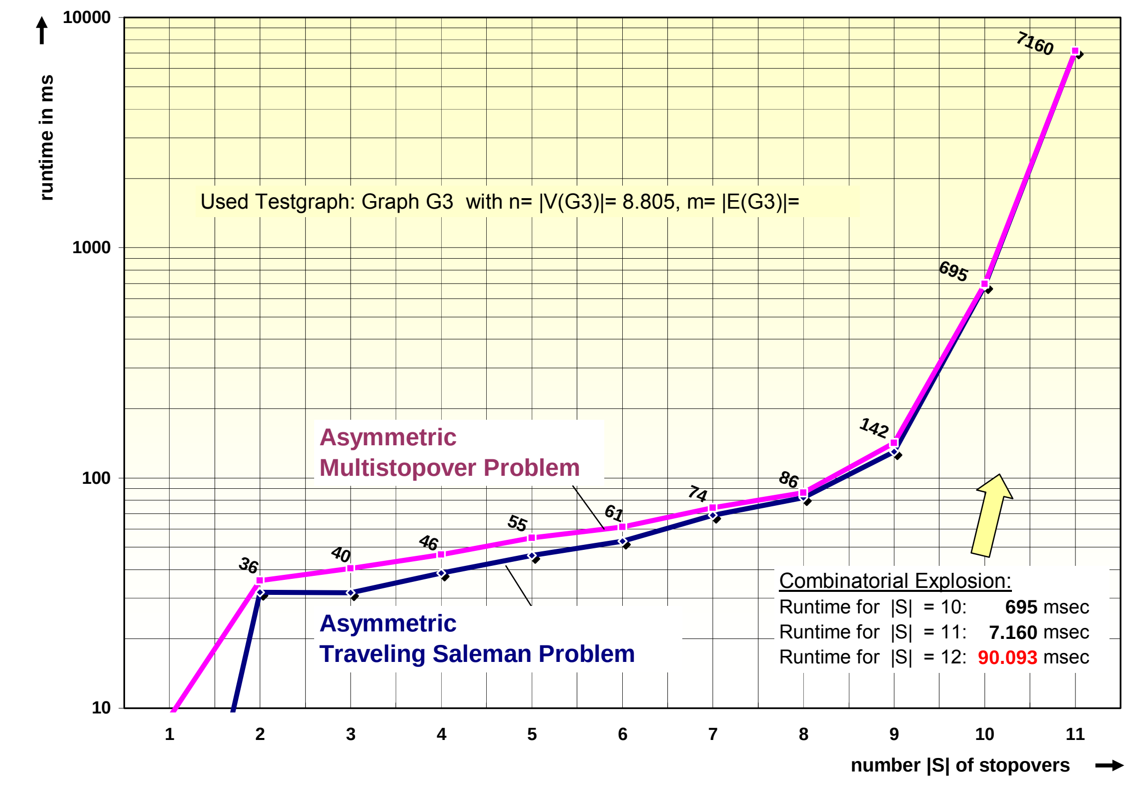


Fig. 6 Performance analysis of algorithm **A-TSP_opt**, Fig. 1.
Real-time ability is assured up to ν= 10 ⇒ nominal time= 695 msec!

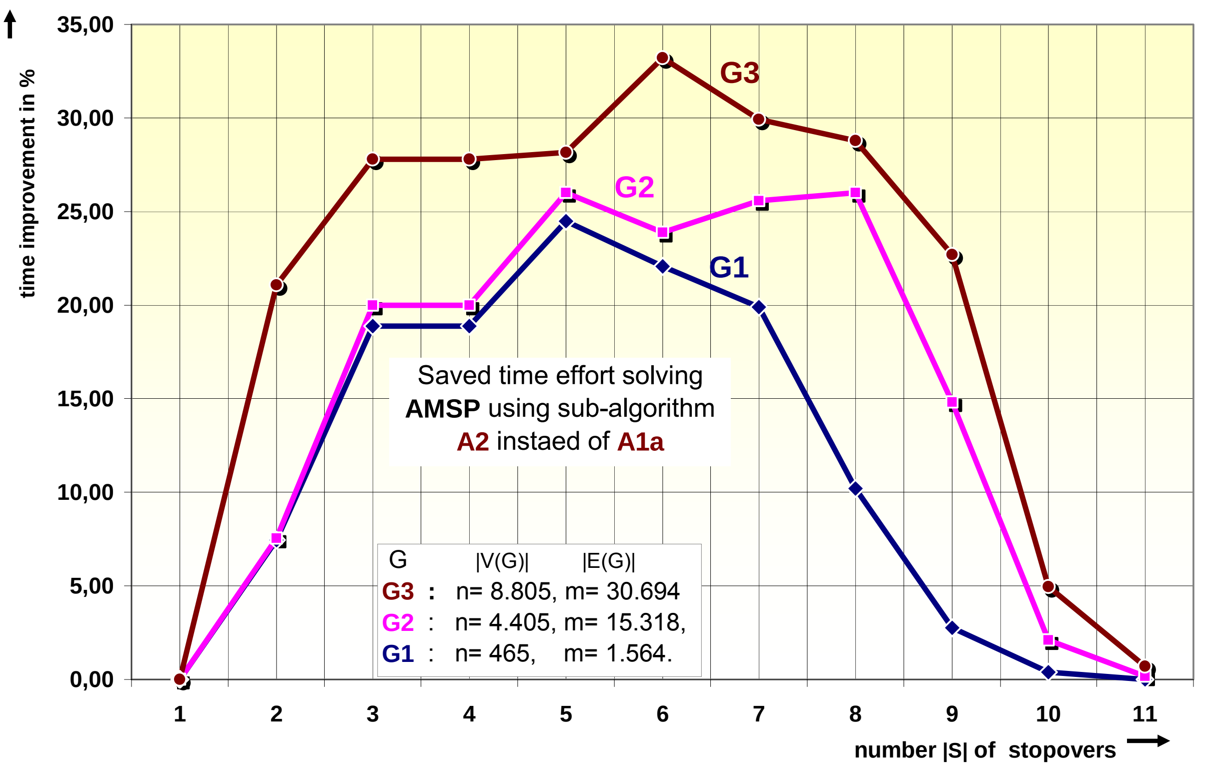


Fig. 7 Asymmetric Multistopover Problem AMSP: Runtime improvement of algorithm **A-TSP_opt** caused by sub-algorithm **A2** replacing sub-algorithm **A1a** in Fig. 1 Block 4, 5 and 6**.**