The Asymmetric Steiner Traveling Salesman Path Problem ASTSPP - Open and Closed Tours' Efficient Determination in General Digraphs

34th Conference of the European Chapter on Combinatorial Optimization, ecco2021, Madrid 10.-11.06.2021,

Peter Hagen Richter
O&S Consultancy, Berlin, Germany, peterhrichter@online.de

1. Introduction

The Asymmetric Steiner Traveling Salesman Path Problem ASTSPP has been unattended in the past despite its vital importance for OR, navigation, logistics, defence, ... We are given a digraph $\vec{G}$ with asymmetric arc weights $\vec{\lambda}$, start point s, target point t, and a subset $\Omega \subseteq V_G$. The objective is to find a shortest route from s to t in $\vec{G}$ visiting all destinations Ω at least once. We start from the Traveling Salesman Problem TSP: "Given undirected complete graph G with edge length $\lambda: E_G \rightarrow \mathbb{R}_0^+$. Find a shortest Hamiltonian tour through V_G!".

TSP is ***NP***-hard, Garey&Johnson 1979. We present an efficient ASTSPP-algorithm called $\mathfrak{A}$ that looks in digraphs $\vec{G}$ for a function $\pi: \{0, 1, ..., |\Omega|+1\} \rightarrow \Omega \cup \{s, t\}$ with $\pi(0) := s$ and $\pi(|\Omega|+1) := t$ minimizing $\sum_{k=0}^{|\Omega|} d_{\vec{G}}(\pi(k), \pi(k+1))$ with shortest distances $d_{\vec{G}}: V_{\vec{G}}^2 \rightarrow \mathbb{R}_{\geq 0}$ resulting to layout $\mathfrak{M} = \Big|\Big|_{k=0}^{|\Omega|} P_{\vec{G}}(\pi(k), \pi(k+1))$ with shortest path $P_{\vec{G}}(x, y)$ from x to y in $\vec{G}$. Table 1 shows the relation of the ASTSPP with other TSP related ***NP***-hard problems in order to show which problems ☑ can now be tackled by ASTSPP algorithm $\mathfrak{A}$.

Table 1. *The TSP Problem Family – An Overview.*

	In graph G or digraph $\vec{G}$ we look for a function π describing a) closed path b) s – t – path / c) closed walk c) s – t – walk via $\Omega \subseteq V_G$	Instance $\mathfrak{I}$=	Round trip?	G directed?	x∈ Ω to visit	Ω= V_G?
TSP	Traveling Salesman Problem	(G= K_n, λ)	y	n	1	Ω= V_G
ATSP	Asymmetric TSP V_G	($\vec{G}$= $\vec{K}_n$, $\vec{\lambda}$)		y		
TSPP	Traveling Salesman Path Problem	(G= K_n, λ, s, t)	n	n	1	
☑ TSWP	Traveling Salesman Walk Problem.				≥ 1	
ATSPP	Asymmetric TSPP	($\vec{G}$= $\vec{K}_n$, $\vec{\lambda}$, s, t)		y	1	
☑ ATSWP	Asymmetric TSWP				≥ 1	
☑ STSP	Steiner Trav. Salesman Problem	(G, λ, Ω)	y	n	≥ 1	Ω⊆ V_G
☑ ASTSP	Asymmetric STSP	($\vec{G}$, $\vec{\lambda}$, Ω)		y		
☑ STSPP	Steiner Trav, Salesman Path Problem	(G, λ, Ω, s, t)	n	n		
☑ ASTSPP	Asymmetric STSPP	($\vec{G}$, $\vec{\lambda}$, Ω, s, t)		y		

Graph	G or K_n= (V_G, E_G), λ: $E_G \rightarrow \mathbb{R}_{\geq 0}$,
Digraph	$\vec{G}$ or $\vec{K}_n$= (V_G, $E_{\vec{G}}$), $\vec{\lambda}$: $E_{\vec{G}} \rightarrow \mathbb{R}_{\geq 0}$,
Destinations	Ω ⊆ V_G, start s, target t,
Result:	Function $\boldsymbol{\pi}$: {1, 2, …|Ω|} → Ω,
☑:	Problems also efficiently tackled by algorithm $\mathfrak{A}$.

x∈ Ω to be visited:
1 – exactly once
≥ 1 – at least once

Since any algorithm that can process digraphs $\vec{G}$ (like $\mathfrak{A}$) is also able to cope with undirected graphs ($\vec{\lambda}(x, y) = \vec{\lambda}(y, x) = \lambda(x, y)$) we can refer the following problems ☑ to be solved ASTSPP algorithm $\mathfrak{A}$:

TSWP:
Given: G= K_n, λ, {s, t}⊂ V_G,
Sought: Shortest s-t-walk meeting all x∈ V_G at least once.
Call: $\mathfrak{A}$(G, λ, V_G, s, t, …).

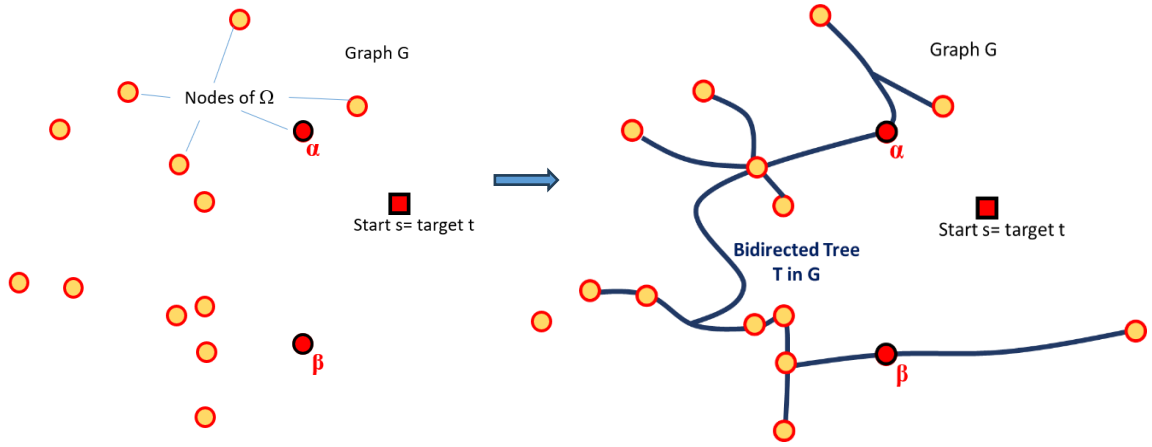
ATSWP:
Given: $\vec{G}$= $\vec{K}_n$, $\vec{\lambda}$, {s, t}⊂ V_G,
Sought: Shortest s-t-walk meeting all x∈ V_G at least once.
Call: $\mathfrak{A}$($\vec{G}$, $\vec{\lambda}$, V_G, s, t, …).

STSP:
Given: G, λ, Ω⊂ V_G,
Sought: Shortest tour meeting all x∈ Ω at least once.
Call: $\mathfrak{A}$(G, λ, Ω, s, t, …) with an arbitrary s= t ∈ V_G.

ASTSP:
Given: $\vec{G}$, Ω⊂ V_G, $\vec{\lambda}$,
Sought: Shortest tour meeting all x∈ Ω at least once.
Call: $\mathfrak{A}$($\vec{G}$, $\vec{\lambda}$, Ω, s, t, …) with an arbitrary s= t ∈ V_G..

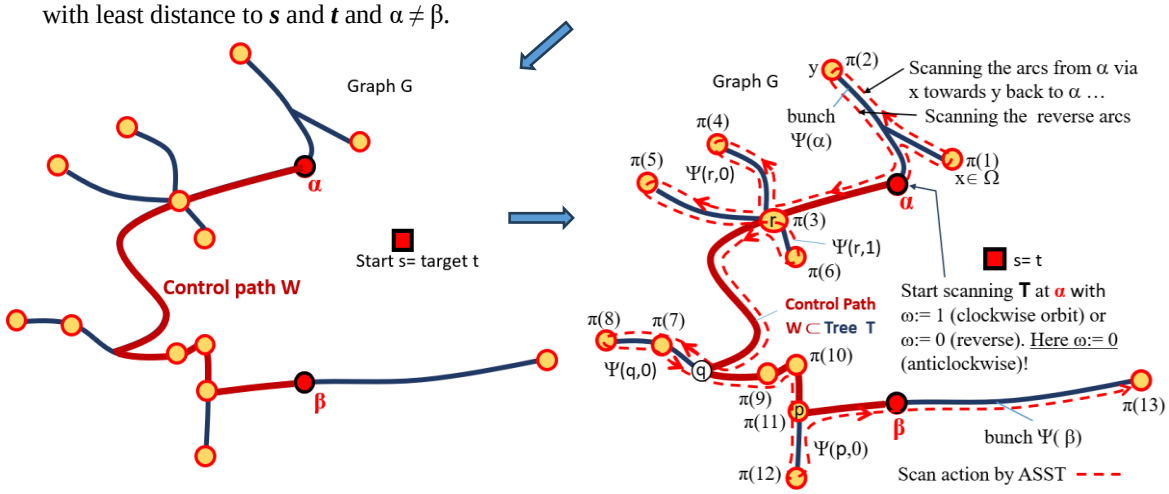
STSPP:
Given: G, Ω⊂ V_G, λ, {s, t}⊂ V_G
Sought: Shortest s-t-walk meeting all x∈ V_G at least once.
Call: $\mathfrak{A}$(G, λ, Ω, s, t, …).

Figure 1 below gives a overview about the strategy algorithm 𝔄 proceeds.



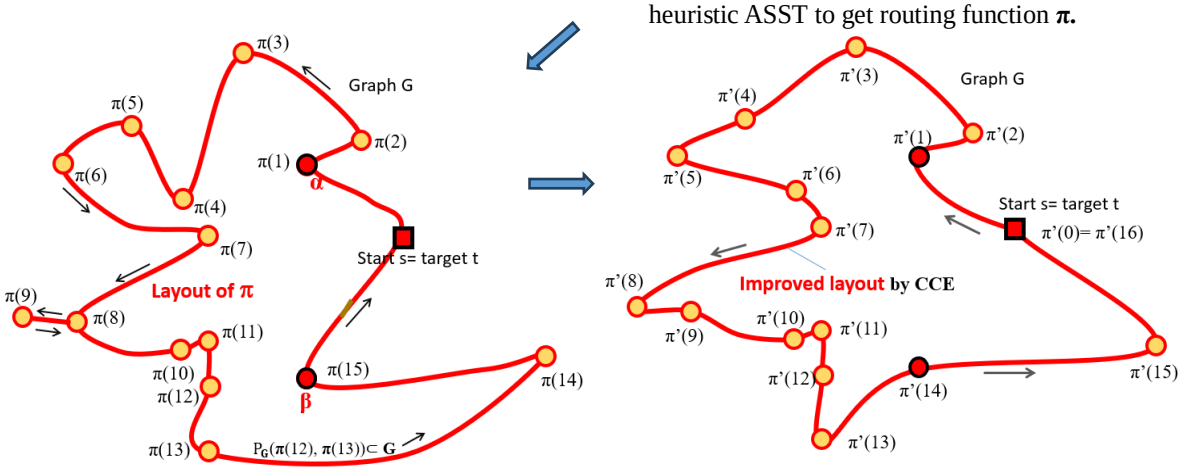
A Take Ω and determine scan-entry α and exit β with least distance to ***s*** and ***t*** and α ≠ β.

B Build the Steiner **T**ree T= **T**(Ω) spanning Ω.



C Build the control path **W**= $P_T(\alpha, \beta) \subset$ **T.**

D Scan T from α to β adhering to **W** with meta-heuristic ASST to get routing function ***π.***



E Layout of routing ***π*** with interlinked walks $P_G(\pi(i), \pi(i+1)) \subset$ **G**, here without CCE.

F ***π*** honed by meta-heuristic CCE and TSA running scanning of **T** corresponding to D).

Figure 1. *Summarized strategy of ASTSPP algorithm 𝔄 via interim findings A …F*

The success of algorithm 𝔄 relies on novel and very efficient meta-heuristics. Explaining them below leads to understanding of 𝔄.

2. Meta-Heuristics used by Algorithm $\mathfrak{A}$

2.1 ASST – Advanced Scan of Spanning Trees

After a scan-entry α and a scan-exit β were determined (Figure 1 A) and after the locations $\Omega \subset V_G$ have been connected to a *Steiner tree T* (Figure 1 B) and after the *Control path* $W = P_T(\alpha, \beta)$ has been determined (Figure 1 C) ASST scans tree T adhering at W corresponding to Figure 1 D) as follows: ASST traces W from α and β padding π consecutively with each $x \in \Omega$ only if met for the first time. If ASST encounters a crossing $z \in V_W$ being the root of one or two bunches (left, right, or left and right of W) $\mathfrak{A}$ proceeds scanning T depending on the last used scan direction (clockwise $\omega = 1$, anticlockwise $\omega = 0$): If currently holds $\omega = 1$ ($\omega = 0$) the left (right) bunch will be scanned first clockwise (anti-clockwise). If the bunch has been scanned and root z will be encountered again, ω will be changed to 0 (1) to scan the opposite bunch anti-clockwise (clockwise). Figure 1 E) shows the result if π is transformed to the catenation of the corresponding shortest paths $\mathfrak{M} = \Big|\Big|_{k=0}^{|\Omega|} P_{\vec{G}}(\pi(k), \pi(k+1))$.

2.2 CCE – Confined Complete Enumeration

Meta-heuristic ASST fills π which each new location being encountered for the first time. Although the length C(π) of routing π is sufficient enough, improvements mostly are possible with low effort as following: Considering the current partial bijection $\pi: \{1, 2, \ldots, \sigma\} \rightarrow \Omega$ with $\sigma \leq |\Omega|$, each time ASST finds a location $x \in \Omega$ and sets it into π by π(++σ):=x, CCE rearranges the current last δ entities of π within the sequence π(σ- δ), π(σ- δ+1), π(σ- δ+2),..., π(σ-1), π(σ), π(σ +1) where the inner nodes are taken for δ! permutations for the last δ service nodes to get the least cost π′: {1, 2, …, σ, σ +1} → Ω with weight $\sum_{k=\sigma-\delta}^{\sigma} d_G(\pi'(k), \pi'(k+1))$ for its current length σ+1. That is, while *Tour*() scans T using ASST to fill π, CCE optimizes the sequence of the last but one $2 \leq \delta \leq 5$ nodes contained already in π each time a new $x \in \Omega$ is added with π(σ+1):= x to the end of π. Because $\delta \leq 5$ time effort to calculate δ! permutations is neglectable. Figure 1 F) should indicate how CCE improves the layout $\mathfrak{M}$ to $\mathfrak{M}$` with decreased cost C(π').

2.3 TSE – Tree Structure Adaption

Some Steiner trees $T(\Omega) \subset G$ have substructures, where it is obvious that their modifications would improve π. Proposed is an efficient method that analyses T already during each execution of ASST to generate a well-founded proposal $\mathcal{p}$ to change $T \overset{\mathcal{p}}{\rightarrow} \mathbf{T'}$ in order to get an "improved" tree $\mathbf{T'}$ such that ASST applied for T′ again provides better results. The four criteria $\chi_1, \ldots \chi_4$ evaluate different cuts of T between some $\mathbb{p} \in V_T$ and $\mathbb{q} \in V_T$ to generate a proposal $\mathcal{p} = ((\mathbb{p}, \mathbb{q}), (w, z))$ applied to T for a new tree T′ as follows:

$T' := (T \setminus \underbrace{\overline{P}_T(\mathbb{p}, \mathbb{q})}_{\text{remove}}) \cup \underbrace{P_G(w', z')}_{\text{add}}$ with (w', z'):= ***bridge***(w, z). I.e. the bidirectional path $\overline{P}_T(\mathbb{p}, \mathbb{q}) \subseteq T$ is to replace by path $\overline{P}_G(\textbf{\textit{bridge}}(w, z)) \subseteq \mathbf{G}$ where function ***bridge***() is necessary to avoid cycles in T′, more detailed in ASTSPP publication on Research Gate.

3. Algorithm $\mathfrak{A}$

Figure 2 shows how $\mathfrak{A}$ proceeds using an outer cycle, executed with different pairs (α, β), and an inner cycle that successively improves the tree T → T' → T'' … till there is no tree improvement proposal or the cycle limit Γ is exhausted.

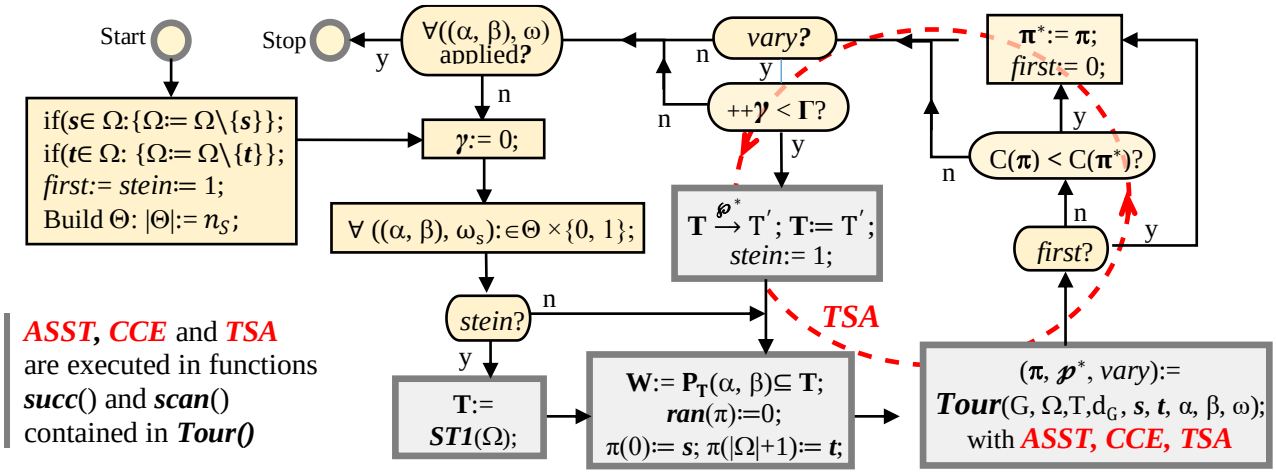


Figure 2. *ASTSPP algorithm $\mathfrak{A}$ corresponding to its implementation*

ST1(Ω) – Build up Steiner Tree T *Tour() – Scanning T using ASST, CCE, TSA*	*Θ – priority queue containing pairs (scan-entry α, scan-exit β) with least distance to start s and target t* *γ - parameter to confine the TSA cycle* *ω – orbit direction ASST has to begin with*

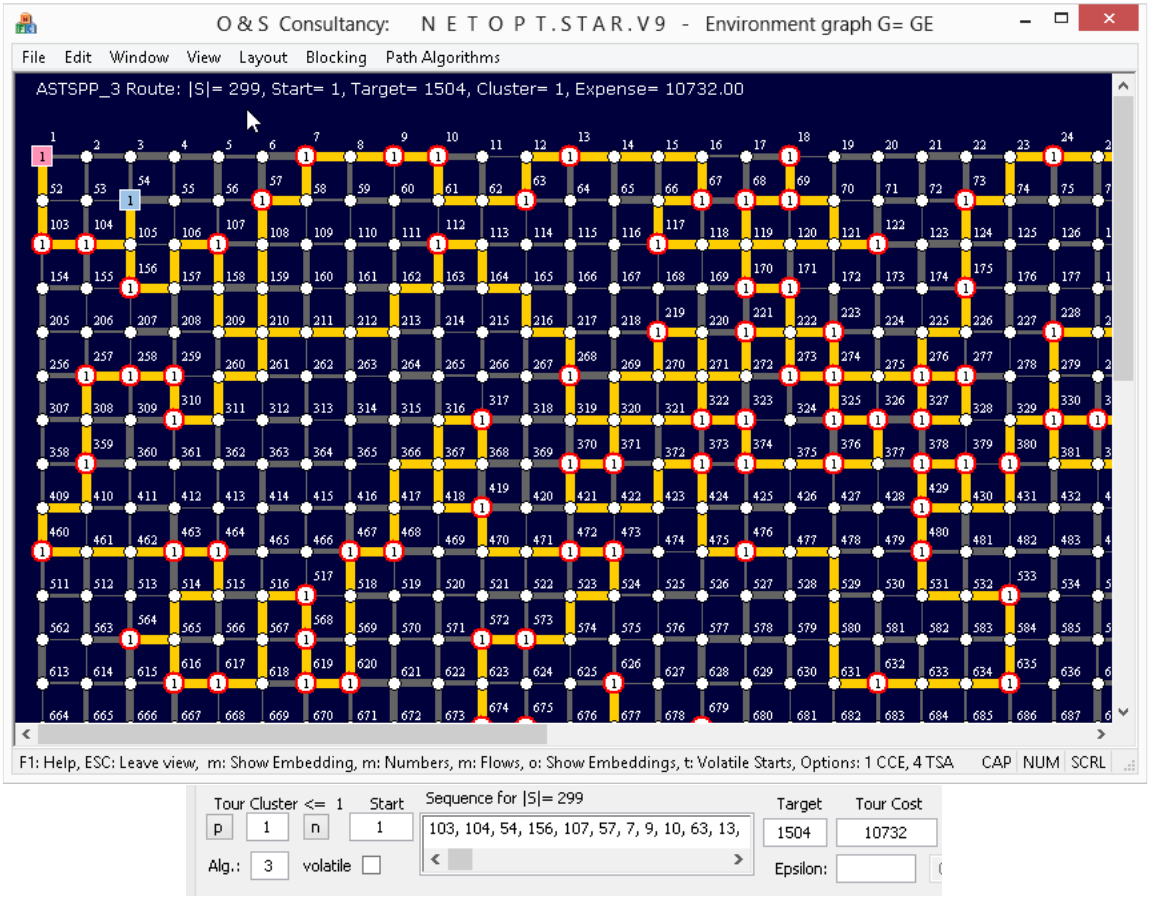


Figure 3 *A part of an ASTSPP-Route by* $\mathfrak{A}$ *represented on the screen for* $|\Omega| = 299$. *The arc lengths there don't correspond to the real random cost* λ: $E_G \rightarrow \mathbb{R}_{\geq 0}$. *Although not represented each arc is coupled with its reverse one generally with different length.*

Figure 3 shows the layout for $|\Omega| = 299$ determined in a digraph $\vec{G}$ with $\vec{\lambda}$: $E_{G2} \rightarrow [0, 40]$, $|V_G| = 1.504$ and $|E_G| = 5.864$ (high density $D = \frac{|E_G|/2}{|V_G|} = 1{,}95$!). To not exceed the concern of this summary, for the test on different random graphs and the corresponding result evaluation we refer to the ASTSPP publication on Research Gate.

4. Conclusion

The deterministic $\boldsymbol{O}(n^3)$-algorithm $\mathfrak{A}$ has intensively been tested for open and closed tours. The tests used digital traffic maps $\vec{G}$ without to twist them to complete ones, what enables the consideration of turn-restrictions, activating / deactivating one-way-streets, considering traffic jam, etc.

The new optimization features *Advanced Scan of Spanning Trees* ASST, *Confined Complete Enumeration* CCE and *Tree Structure Adaption* TSA show an impressive optimization potential: Determining the sample standard deviation q_{max} we used large set of random grid graphs with problem size $|\Omega| = 14$ ($|\Omega| > 14$: exuberant time consumption getting optimal results). Algorithm $\mathfrak{A}$ running ASST, CCE and TSA with $|\Theta| = 5$ didn't surpass a sample standard deviation $q_{max} = 1{,}86$ %! Optimization feature CCE ($\delta = 5$) contributes evenly over Ω for the most respectable result improvement.

The introduction of the *Priority Queue* Θ compared to the conventional use of $\mathbb{A} \times \mathbb{B}$ (choosing pairs $(\alpha, \beta) \in \mathbb{A} \times \mathbb{B}$) leads to an essential runtime reduction without to peril getting near-optimal results.

Regarding larger problem sizes $100 \leq |\Omega| \leq 1.000$, the common use of CCE and TSA gets 15% length improvement compared to the sole use of ASST!

Algorithm $\mathfrak{A}$ running on a 2,70 GHz PC using graph $\mathbf{G}_2$ with $|\Theta| = 3$ retains real-time ability ($t \leq 2$ sec) for the following problem sizes: (a) using TSA: $|\Omega| = 115$, (b) not using TSA: $|\Omega| = 1590$. That means that time-critical apps processing higher problem sizes should use CCE without TSA to keep real-time ability but with a loss of about 5% solution quality.
Because $\mathfrak{A}$ tackles the cases

(1) ($\Omega \subset V_G$ & $\Omega = V_G$),
(2) ($\boldsymbol{s \neq t}$ & $\boldsymbol{s = t}$)
(3) (digraphs & undirected graphs)

it efficiently solves not only ASTSPP but also

TSWP, ATSWP, STSP ASTSP, STSPP!

Apart from the algorithm's efficiency, the special peculiarity of algorithm $\mathfrak{A}$ additionally lies in the following characteristics / requirements:

- Graphs are allowed to be directed (G) or undirected ($\vec{G}$).
  ⇨ $\mathfrak{A}$ takes G as $\vec{G}$ with $\vec{\lambda}(x, y) = \vec{\lambda}(y, x) = \lambda(x, y)$).
- $\vec{G}$ can but must not necessarily be a complete graph.
  ⇨ $G \subset K_n$ is the general case.
- Complying with symmetric and asymmetric arc costs.
  ⇨ "Asymmetric" problems are the general ones,
- Routes might visit only a subsets $\Omega \subseteq V_G$.
  ⇨ Not $\Omega = V_G$ but $\Omega \subseteq V_G$ is the normal case.
- Service locations might be visited more than once.
  ⇨ "Walks" for shorter routes require that.
- Walks and tours must be allowed.
  ⇨ "***s-t***-walks" as well tours ***(s= t)*** are to consider.
- $\vec{G}$ can but must not comply with the triangle inequality.
  ⇨ normal case: a graph metric is not required,
- Turn restrictions are to observe for the route calculation.
  ⇨ Edge-queuing shortest path algorithms tackle this.
- Open or closed routes to built with same efficiency.
  ⇨ ***s= t*** or ***s≠ t*** are equally to process.
- Drawing advantage from using planar graphs.
  ⇨ traffic maps are planar,

Algorithm $\mathfrak{A}$ takes all these demands into account!