Dynamic Programming Approaches for the Traveling Salesman Problem with Drone

Paper Author: Bouman

Journal: Networks

Published: 2018 **DOI:** 10.1002/net.21864

Author: Chen Huaneng **Email:** huanengchen@foxmail.com **Date:** 2025-11-29

1 Problem Description

该研究[1]针对的问题是 Traveling Salesman Problem with Drone (TSP-D)，目标函数是求卡车和无人机返回仓库的最短时间，单次无人机飞行只能够服务一个顾客，允许无人机的配送路径是个环（即卡车在某个顾客节点等待无人机服务后返回），且卡车和无人机会合的节点只能够是顾客节点或者仓库。该研究不考虑无人机和卡车的载重限制，也不考虑无人机的续航时间限制。该研究用到的符号及其含义如表 table 1所示。

表 1: TSP-D 用到的符号及含义

符号	含义
V	位置节点集合，包含仓库 v_0 和 $n-1$ 个顾客位置，即 $V=\{v_0, v_1, ..., v_{n-1}\}$
n	位置节点总数（包含仓库 v_0）
$c(v,w)$	卡车从位置 v 到位置 w 的行驶时间（或距离），$v,w \in V$
$c^d(v,w)$	无人机从位置 v 到位置 w 的飞行时间（或距离），$v,w \in V$
α	无人机与卡车的速度关联常数，满足 $\alpha c^d(v,w) \geq c(v,w)$ 对所有 $v,w \in V$，表示无人机最大速度为卡车的 α 倍
o	单个操作（operation），由起始节点、终止节点、最多一个无人机节点（由无人机单独服务的节点）和若干卡车节点（卡车单独服务的节点）组成，当没有无人机节点时，卡车搭载无人机进行服务，当起始节点和终止节点一致且没有卡车节点但有无人机节点时，卡车在起始节点等待无人机服务完后会合
$t(o)$	单个操作 o 的持续时间，取卡车行驶总时间与无人机飞行总时间的最大值
S	V 的子集，用于表示动态规划中已覆盖的节点集合

2 Solution Approaches

该研究的解决方法主要基于 Bellman-Held-Karp dynamic programming algorithm for the TSP[1]进行改良[2-3]，该方法分为三个阶段：

1. 对于任意一个起始节点、终止节点以及中间卡车服务的节点集合，枚举所有这样组合的卡车行驶的最短路

[1]关于 Bellman-Held-Karp 算法的进一步理解可以参考本文的末尾的附录部分，该文档来源于Nagoya University 的 Serge Richard 教授。

2. 在阶段一的卡车行驶最短路径的基础上引入无人机节点，然后计算最小时间花费的覆盖所有服务节点集合的可行操作（efficient operation）

3. 计算满足服务所有顾客节点且起始节点和终止节点都在仓库的最优的可行操作排序

2.1 Dynamic Programming Approach for The Standard TSP

通常的 Traveling Salesman Problem (TSP) 是找到一条从任意一个起点 v 出发，访问所有 V 中的节点并且最终回到起点 v 的最短路。路径的权重可以是距离、时间、排放量等可以被分解到每一步的量，表示为 $D_{\text{TSP}}(V, v)$。

接下来简单介绍一下 Bellman-Held-Karp (BHK) 算法，对于一个包含了 v 和 $w \in S \setminus \{w\}$ 的集合 $S \subset V$，用 $D_{\text{TSP}}(S, w)$ 表示从起点 v 出发，访问 S 中所有节点并到达终点 w 的最短路所需要的花费（时间或者距离等）。而 $D_{\text{TSP}}(S, w)$ 又可以分解为一个更小的子问题（subproblem），即该问题可以由从某个节点 $u \in S$ 到达终点 w 的弧和从 v 出发访问集合 $S \setminus \{w\}$ 中所有节点并到达终点 u 的最短路组合而成。因此解决问题 $D_{\text{TSP}}(S, w)$ 可以变成解决考虑所有弧 $(u, w) : u \in S \setminus \{w\}$ 和子问题 $D_{\text{TSP}}(S \setminus \{w\}, u)$ 的所有组合。可以通过递归公式来表达上述过程：

$$D_{\text{TSP}}(S, w) = \begin{cases} \infty & \text{if } w \notin S \\ c(v, w) & \text{if } S = \{w\} \\ \min_{u \in S} \{D_{\text{TSP}}(S \setminus \{w\}, u) + c(u, w)\} & \text{otherwise} \end{cases} \tag{1}$$

由于 V 的所有可能的子集数量为 2^n，并且 w 的选择最多有 n 种（即最多有 n 个地点作为可能的终点），因此最多可能有 $n \cdot 2^n$ 个子问题。对于解决每个递归的子问题时，最多考虑 n 次递归（因为每次递归都减少一个节点，最多有 n 个节点），所以该算法的时间复杂度为 $O(n^2 \cdot 2^n)$。

2.2 First Pass

第一阶段采用 Bellman-Held-Karp 的动态规划的方法计算对于每一个起始节点 v，终止节点 w 和卡车服务的节点集合 S 的不同组合的卡车最短路，同样可以用递归公式表示：

$$D_{\text{T}}(S, v, w) = \begin{cases} \infty & \text{if } w \notin S \\ c(v, w) & \text{if } S = \{w\} \\ \min_{u \in S} \{D_{\text{T}}(S \setminus \{w\}, v, u) + c(u, w)\} & \text{otherwise} \end{cases} \tag{2}$$

由于 S 的最多所有可能的子集数量为 2^n，并且 w 的选择最多有 n 种（即最多有 n 个地点作为可能的终点），v 的选择最多有 n 种（即最多有 n 个地点作为可能的起点），因此最多可能有 $n^2 \cdot 2^n$ 个子问题。对于解决每个递归的子问题时，最多考虑 n 次递归（因为每次递归都减少一个节点，最多有 n 个节点），所以第一阶段的时间复杂度为 $O(n^3 \cdot 2^n)$。

2.3 Second Pass

第二阶段中,通过结合不同的无人机服务节点和第一阶段获得的卡车路径来获得有效的操作(efficient operations)，每一个有效的操作是给定起始节点 v 和终止节点 w，覆盖节点集合 S 的最小花费的路径。具体而言，即在第一阶段的每一个子问题中，对于任意一个无人机服务的节点 $d \in S \setminus \{v, w\}$，结

合卡车从 v 出发，服务节点 $S \setminus \{d\}$ 并最终回到节点 w 的路径进行计算得到有效操作的花费。写成递归方程表示为：

$$D_{\mathrm{OP}}(S, v, w) = \begin{cases} \infty & \text{if } w \notin S \\ D_{\mathrm{T}}(S, v, w) & \text{if } S = \{w\} \\ \min_{d \in S \setminus \{v,w\}} \max\left\{c^d(v,d) + c^d(d,w), D_{\mathrm{T}}(S \setminus \{d\}, v, w)\right\} & \text{otherwise} \end{cases} \tag{3}$$

第二阶段最多可能有 $2^n \cdot n^2$ 个子问题，每个子问题中的无人机可能服务的节点最多可能有 n 种选择，又由于在第一阶段已经计算了相应的 D_{T}，因此第二阶段的时间复杂度为 $O(2^n \cdot n^3)$

2.4 Third Pass

第三阶段的目标是找到一个从仓库出发最终回到仓库并服务所有的顾客节点的有效操作的序列。寻找的思路是通过状态转移，即从状态 $D(\{v_0\}, v_0)$（没有服务任何顾客，最终到达的节点是仓库节点）出发，通过每次增加一个有效操作进行迭代，直到服务所有顾客并回到仓库，即达到状态 $D(V, v_0)$ 时停止。比如从状态 $D(S_1, w_1)$ 到状态 $D(S_2, w_2)$（其中 $S_1 \subsetneq S_2$）只需要增加一个有效操作 $D_{\mathrm{OP}}(S_2 \setminus S_1, w_1, w_2)$，即从 w_1 节点出发服务节点 $S_2 \setminus S_1$ 然后到达 w_2 节点的有效操作。接下来该研究使用了两种方法来找到最优的操作序列，第一种是标准的动态规划的方法，而第二种则采用了 A^* 算法的原理[4-5]。

2.4.1 Standard Dynamic Programming Implementation of Third Pass

第一种方法通过将状态 $D(S, w)$ 用类似于第二阶段的思想拆分成前一个状态和一个有效操作的所有可能组合中的最小值，从而来进行状态转移，其中状态 $D(S, w)$ 表示从仓库 v_0 出发，服务节点 S 最终到达节点 w 所需的花费，同样用递归公式表达为：

$$D(S, w) = \begin{cases} \infty & \text{if } w \notin S \\ c(v_0, w) & \text{if } S = \{w\} \\ \min_{T \subset S} \min_{u \in S} \{D(S \setminus (T \setminus \{u\}), u) + D_{\mathrm{OP}}(T, u, w)\} & \text{otherwise} \end{cases} \tag{4}$$

对于这个方法的时间复杂度，粗略估算应该是 $O(4^n \cdot n^2)$，因为最多有 $2^n \cdot n$ 个子问题需要解决，而解决每个子问题需要最多 $2^n \cdot n$ 的时间（假设更小的子问题已经被解决了），即每个 S 最多有 2^n 个子集 T，而 $u \in S$ 最多有 n 种选择。但是可以证明这个方法的时间复杂度远比粗略估算的要小得多。

Theorem (Binomial Theorem)**.**

$$(x + y)^r = \sum_{k=0}^{r} \binom{r}{k} x^k y^{r-k} \tag{5}$$

Theorem. *The TSP-D with n locations can be solved in $O(3^n \cdot n^2)$ time using dynamic programming.*

Proof. 对于公式 (4) 中 $|S| = 1$ 的情况，共有 $O\left(\binom{n}{1} \cdot n\right)$ 个子问题，$|S| = 2$ 时，有 $O\left(\binom{n}{2} \cdot n\right)$ 个子问题，对于 $|S| = i$ 时，有 $O\left(\binom{n}{i} \cdot n\right)$ 个子问题（因为当集合 S 的元素个数为 i 时，可以从 V 集合中选取 i 个，即 $\binom{n}{i}$，然后对于终点 w，最多有 n 种选择）。针对 $|S| = i$ 的情况，因为集合 S 有 2^i 个可能的子集 T，有 $O(n)$ 个可能的 $u \in S$ 可以选择，所以解决这个子问题所需要的时间为 $O\left(n \cdot 2^i\right)$。所以对

于一个集合大小确定为 i 的子集 S 来说，解决集合大小为 i 的所有子问题所需的时间为 $O\left(\binom{n}{i}2^i n^2\right)$。为了解决所有的子问题，需要解决子集大小为 $i = 1, 2, \ldots, n$ 的所有子问题，因此有：

$$O\left(\sum_{i=1}^{n}\binom{n}{i}2^i n^2\right) < O\left(\sum_{i=0}^{n}\binom{n}{i}2^i n^2\right) = O\left(n^2\sum_{i=0}^{n}\binom{n}{i}2^i\cdot 1^{n-i}\right) = O\left((2+1)^n\cdot n^2\right) = O\left(3^n\cdot n^2\right) \quad (6)$$

□

整个算法一共有三个阶段，而前两个阶段的时间复杂度为 $O\left(2^n\cdot n^3\right)$，最后一个阶段的时间复杂度为 $O\left(3^n\cdot n^2\right) = O\left(2^n\cdot\left(\frac{3}{2}\right)^n\cdot n^2\right)$，因此整个算法的时间复杂度主要取决于第三阶段的时间复杂度，所以整个算法的时间复杂度为 $O\left(3^n\cdot n^2\right)$。

2.4.2 A* Implementation of Third Pass

第二种方法则采用了 A^* 寻路算法的思想，由于 section 2.4.1节的第一种方法考虑了很多与最优解不相关的子问题，因此第二种方法考虑通过启发式算法（informed search）的思想来跳过这些不相关的子问题从而减小问题的规模。其思想是通过估计当前的部分解有多好的程度（比如估计距离最优解的下界还有多少）来引导搜索的方向朝最优解的方向进行。

A^* 的关键在于设计一个函数来估计 (estimates) 从当前状态到最终状态 (sink state) 的距离 (distance)。通过仅考虑加总到达当前状态的花费和到达最终状态的函数估计值最小的状态，该算法不会生成（考虑）远离最终状态的状态，从而减小可行解的空间。Dechter 和 Pearl[6] 证明了，当这个函数不会高估（overestimates）到达最终状态的距离时，这个方法能够找到最优解。

为了获得从当前状态到最终状态的路径的下界，该研究通过计算最后一个访问的节点、未访问的节点以及仓库节点组成的图的最小生成树（minimum spanning tree）的长度乘以 $\frac{1}{2+\alpha}$，在作者的另一篇论文[7]中证明了，在距离满足三角不等式的时候，存在一个 TSP 的最小生成树的路径是相应的 TSP-D 路径的 $(2+\alpha)$ 倍近似（approximation），其中 α 是前面提到的关联无人机与卡车速度的常数。

2.5 Restricting the Number of Operations

为了减少第一阶段和第二阶段生成的操作的数量，从而减少需要考虑的子问题的数量，以便降低整个算法的空间复杂度和时间复杂度，该研究通过考虑限制卡车在单独服务时可以访问的顾客节点数量为 k 来实现这个目的。

要将该限制加到阶段一和阶段二可以通过只考虑最多 $k+2$ 个元素的子集 S 来实现，因为起始节点和终止节点不属于卡车单独服务的节点。因此在考虑子问题 $D_{\mathrm{T}}(S, v, w)$ 时，只需要考虑 $|S| \le k+1$ 的情况，因为无人机节点在阶段二才加入，且按照惯例，$v \notin S$ 且 $w \in S$。而在考虑子问题 $D_{\mathrm{OP}}(S, v, w)$ 时，当 $v \neq w$ 时，只考虑 $|S| \le k+2$ 否则考虑 $|S| \le k+1$。

这意味着，在前两个阶段的子问题数量为 $O\left(\sum_{i=1}^{k+2} i^2\binom{n}{i}\right)$，而不是 $O\left(n^2\cdot 2^n\right)$[2]。对于第三阶段而言，仍然需要考虑 $O\left(2^n\cdot\right)$ 个子问题，但是解决每个子问题所需要的步骤可以减少，这是因为不需要考虑 S 的所有可能子集，而只需要考虑 $|T| \le k+2$ 的子集，这意味着每个子问题都可以在 $O\left(n^{k+1}\right)$ 时间复杂度内解决，这是因为对于 T 集合来说，最多有 $\sum_{i=0}^{k}\binom{n}{i}$ 种可能的子集，其上界是 $\frac{k+1}{k!}n^k$[3]，而 w 是确定的节点，u 则最多有 n 种可能的选择。因此所有的子集可以在时间复杂度 $O\left(2^n\cdot n^{k+2}\right)$ 内解决，

[2] 对于 $|S| = i$ 时来说，从 V 的 n 个节点中选择 i 个节点作为集合 S，然后在这 S 中对于起始节点和终止节点最多各有 i 种选择，而 $|S|$ 的大小从 1 到 $k+2$，因为除去起始和终止节点，卡车最多只能单独服务 k 个节点。

[3] 这里假设了 $k < \frac{n}{2}$，这样才能确保最大的组合数是 $\binom{n}{k}$

而不是 $O\left(3^n n^2\right)$。如果禁止卡车单独进行服务，即 $k=0$，则总的时间复杂度为 $O\left(2^n \cdot n^3\right)$，这是因为对于 S 来说，一共有 $O\left(2^n \cdot n\right)$ 个子问题，而此时卡车不能单独服务顾客，因此对于 D_{OP} 来说，u 最多有 n 种可能的选择，而 w 是确定的，集合 T 最多只能包含除了 u 和 w 之外的一个节点，且这个节点应该是无人机进行服务的，而这个无人机服务的节点最多也只能有 n 种可能的选择，因此最终的时间复杂度为 $O\left(2^n \cdot n^3\right)$。

Theorem.

$$\sum_{i=0}^{k}\binom{n}{i} < \frac{k+1}{k!}n^k, when\ k < \frac{n}{2}$$

Proof.

$$\sum_{i=0}^{k}\binom{n}{i} < \sum_{i=0}^{k}\binom{n}{k} = (k+1)\frac{n!}{k!(n-k)!} = (k+1)\frac{n\cdot(n-1)\cdots(n-k+1)}{k!} < \frac{(k+1)}{k!}n^k$$

□

假设 k 是一个比较小的值是合理的，因为这样可以使得无人机和卡车更多地进行协同服务，从而更多地减少卡车单独服务的节点，从而减少总的服务时间。

Theorem. *For any instance of the TSP-D with symmetric drone costs, that is, $c^d(v,w) = c^d(w,v)$, there is a TSP-D tour without truck nodes (i.e., every operation consists of a start node, an end node, and possibly a drone node), whose time duration is at most twice the time duration of the optimal TSP-D tour for that instance.*

Proof. 假设 $\hat{O} = (\hat{o}_1, \hat{o}_2, \ldots, \hat{o}_l)$ 为最优的 TSP-D 路径，可以通过将每一个 $\hat{o}_i, i = 1, \ldots, l$ 替换为不存在卡车单独服务节点的 TSP-D 路径 o，从而使得没有卡车单独服务的路径的总时间最多为最优 TSP-D 路径的两倍。

让 o 作为一个有 k 个卡车单独服务节点 $v^1, \ldots, v^k$，起始节点和终止节点分别为 v^0 和 v^{k+1} 的操作，并且无人机服务节点为 v^d。如果 $c^d(v^0, v^d) \le c^d(v^d, v^{k+1})$，通过替换 o 为 $(o^0, o^\cdot \ldots, o^{k+1})$，其中操作 o^0 中，无人机从 v^0 起飞到 v^d 然后再回到 v^0，卡车在 v^0 等待无人机，然后在操作 $o^j, j = 1, \ldots, k+1$ 时，卡车从 v^{j-1} 行驶到 v^j（无人机在卡车上）。否则就将操作 o 替换为 $(o^1, o^2, \ldots, o^{k+1}, o^0)$，其中 o^0 中无人机从 v^{k+1} 飞行到 v^d，卡车在 v^{k+1} 处等待无人机返回，而 $o^j, j = 1, \ldots, k+1$ 则仿照前一种情况。因此有：

$$\sum_{j=0}^{k+1} t(o^j) = \underbrace{2\cdot\min\{c^d(v_0, v_d), c^d(v^d, v^{k+1})\}}_{\le c^d(o)} + \underbrace{\sum_{j=0}^{k+1} c(v_{j-1}, v_j)}_{=c(o)} \le 2\cdot\max\{c(o), c^d(o)\} = 2t(o)$$

□

图 1: v_1 和 v_2 分别为两个顾客节点，黑色矩形块为仓库，假设卡车和无人机的速度一致且均为 1

通过图 figure 1可以说明上面这个上界是紧的（tight）。在这个例子中，假设无人机和卡车的速度相等且两个顾客距离仓库的距离一致，最短的配送应该是无人机和卡车分别负责配送一个顾客，然后同时回到仓库，总耗时为 2，当限制卡车不能进行单独服务时，卡车和无人机一起进行配送，需要的总时间为 4，或者无人机进行单独配送，而卡车在仓库不动，同样也需要总时间为 4，因此上面给出的上界是紧的。

参考文献

[1] BOUMAN P, AGATZ N, SCHMIDT M. Dynamic programming approaches for the traveling salesman problem with drone[J/OL]. Networks, 2018, 72(4): 528-542. DOI: 10.1002/net.21864.

[2] BELLMAN R. Dynamic Programming Treatment of the Travelling Salesman Problem[J/OL]. J. ACM, 1962, 9(1): 61-63. DOI: 10.1145/321105.321111.

[3] HELD M, KARP R M. A dynamic programming approach to sequencing problems[C/OL]//Proceedings of the 1961 16th ACM National Meeting On -. Not Known: ACM Press, 1961: 71.201-71.204. DOI: 10.1145/800029.808532.

[4] HART P E, NILSSON N J, RAPHAEL B. Correction to "A Formal Basis for the Heuristic Determination of Minimum Cost Paths"[J/OL]. SIGART Bull., 1972(37): 28-29. DOI: 10.1145/1056777.1056779.

[5] HART P E, NILSSON N J, RAPHAEL B. A Formal Basis for the Heuristic Determination of Minimum Cost Paths[J/OL]. IEEE Transactions on Systems Science and Cybernetics, 1968, 4(2): 100-107. DOI: 10.1109/TSSC.1968.300136.

[6] DECHTER R, PEARL J. Generalized best-first search strategies and the optimality of A*[J/OL]. J. ACM, 1985, 32(3): 505-536. DOI: 10.1145/3828.3830.

[7] AGATZ N, BOUMAN P, SCHMIDT M. Optimization Approaches for the Traveling Salesman Problem with Drone[J/OL]. Transportation Science, 2018/07//Jul/Aug2018, 52(4): 965-981. DOI: 10.1287/trsc.2017.0791.

Travelling Salesman Problem and Bellman-Held-Karp Algorithm

Quang Nhat Nguyen

May 10, 2020

1 Definitions

Definition 1.1 (Travelling Salesman Problem)**.** *Let $G = (V, E, \omega)$ be a* weighted *Hamiltonian graph, with $V = \{x_1, x_2, \ldots, x_n\}$, $i : E \to V \times V$ determines the edges, and $\omega : E \to \mathbb{R}_+$ determines the weighted edge lengths. The problem of finding its shortest Hamiltonian cycle is called the Travelling Salesman Problem (abbrv. TSP).*

Note: *If G is undirected, the TSP is called* symmetric TSP *(sTSP). If G is directed, the TSP is called* asymmetric TSP *(aTSP).*

This is one of the classical problems in Computer Science, and is an introductory problem to Dynamic Programming. The need of computer power arises when we consider a large graph (eg. one that contains hundreds or thousands of vertices) and want to find its shortest Hamiltonian cycle. Before discussing the approaches, let us introduce the *distance matrix.*

Definition 1.2 (Distance matrix)**.** *The $n \times n$ matrix D with elements*

$$D_{ij} := \min\{\omega(e) \,|\, e \in E,\, i(e) = (x_i, x_j)\} \quad \textit{if there exists such } e$$
$$D_{ij} := \infty \quad \textit{if there is no such } e$$

is called the distance matrix *of the weighted graph G.*

In simpler terms, the element D_{ij} denotes the shortest edge that originates at x_i and terminates at x_j. It can be seen that if the graph G is undirected, the matrix D is symmetric. If there is no edge from x_i to x_j, $D_{ij} = \infty$. If there are multiple edges from x_i to x_j, we only consider the shortest edge because our concern is the shortest Hamiltonian cycle.

A naïve approach to this problem would be for the computer to consider all permutations of the vertices, see if one permutation can make a cycle, and if it does then record the cycle's length for comparison. This method has a time complexity of $O(n!)$, and thus is not desirable when dealing with a large database.

Proposition 1.3. *One can reduce the time complexity of $O(n!)$ of the naïve method to $O(n^2 \times 2^n)$ by implementing Bellman-Held-Karp Algorithm.*

Let us discuss this algorithm in the next section.

2 Bellman-Held-Karp Algorithm

First, let us acknowledge the following.

Proposition 2.1. *The shortest Hamiltonian cycle does not depend on the choice of the starting vertex.*

This is obvious because the Hamiltonian cycle has cyclic symmetry, thus changing the starting vertex does not change the order of the other vertices in the cycle. Therefore, let us start constructing the desired path from x_1.

Definition 2.2. *Let $S \subset V\backslash\{x_1\} \equiv \{x_2, x_3, \ldots, x_n\}$ be a subset of size s $(1 \leq s \leq n-1)$. For each vertex $x_i \in S$, we define* $\text{cost}(x_i, S)$ *as the length of the shortest path from x_1 to x_i which travels to each of the remaining vertices in S once. More precisely, we implement this function by the following recursion formula:*

$$\text{cost}(x_i, S) = \min_{x_j}\{\text{cost}(x_j, S\backslash\{x_i\}) + D_{ji}\} \tag{2.1}$$

in which $x_j \in S\backslash\{x_i\}$. In case S has size 1, we define:

$$\text{cost}(x_i, S) = D_{1i} \tag{2.2}$$

Using the above recursive formula, we can gradually construct the cost function for subset S of size from 1 to $n-1$. Because of this recursive feature, which means that the current step is the base for the next step, the implementation of this algorithm in a programming language is called Dynamic Programming.

When we reach subset S of size $n-1$, which means $S = V\backslash\{x_1\}$, the only thing left is to find:

$$\begin{aligned}\text{Shortest Hamiltonian cycle} &= \min_{x_i}\{\text{cost}(x_i, S) + D_{i1}\} \\ &\equiv \min_{x_i}\{\text{cost}(x_i, V\backslash\{x_1\}) + D_{i1}\}\end{aligned}$$

with $x_i \in S \equiv V\backslash\{x_1\}$.

Proposition 2.3 (Time complexity of this algorithm)**.** *The time complexity of this algorithm is: $O(n^2 \times 2^n)$.*

Proof. This algorithm considers 2^{n-1} subsets of $V\backslash\{x_1\}$. For each subset, one computes the cost function for all of its elements, which there are no more than n of them. For each execution of the cost function, one again computes no more than n values based on the values obtained from the previous step. In total, the time complexity of this algorithm is: $O(n^2 \times 2^n)$. □

Remark 2.4. *Compared to the naïve method which has time complexity $O(n!)$, the time complexity of this algorithm, $O(n^2 \times 2^n)$, is much better. For example, if $n = 100$:*

$$\begin{aligned}O(n!) = O(100!) &= O(9.33 \times 10^{157}) \\ O(n^2 \times 2^n) = O(100^2 \times 2^{100}) &= O(1.27 \times 10^{34})\end{aligned}$$

which is about 7.35×10^{123}, or 0.7 septillion googols, times faster.

3 Illustration

Let us illustrate the algorithm through the following sample problem.

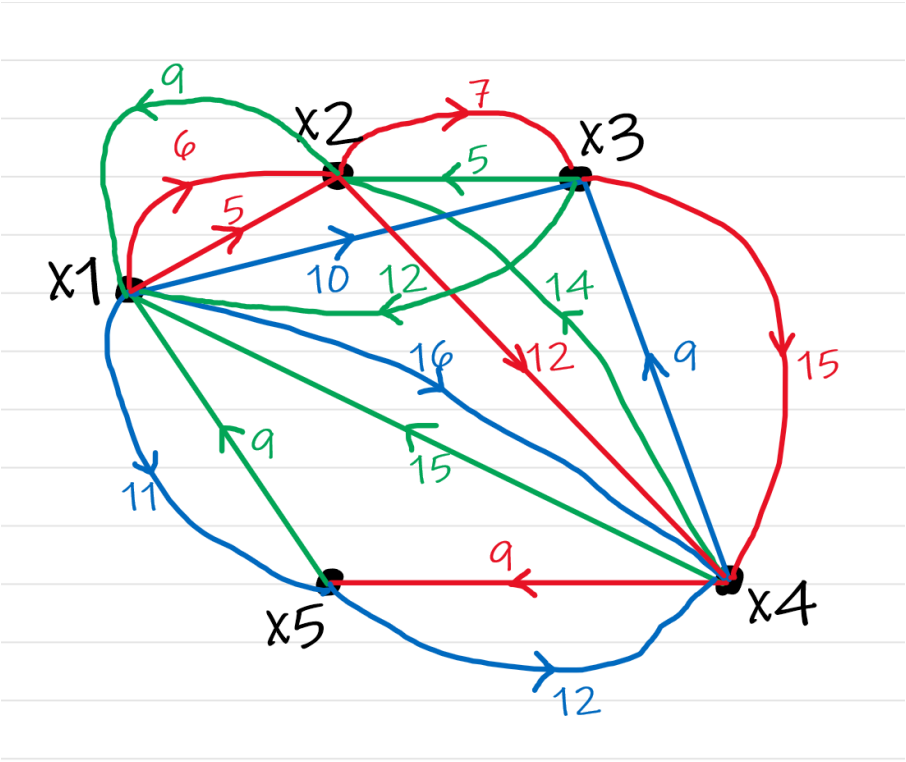

Figure 1: Graph for a sample asymmetric TSP

The distance matrix can be generated as follows:

$$\begin{bmatrix} 0 & 5 & 10 & 16 & 11 \\ 9 & 0 & 7 & 12 & \infty \\ 12 & 5 & 0 & 15 & \infty \\ 15 & 14 & 9 & 0 & 9 \\ 9 & \infty & \infty & 12 & 0 \end{bmatrix}$$

Now that we have the distance matrix, we can proceed with the recursive formula. First, let us consider subsets S of size 1:

Subset S	cost function	Result
$S = \{x_2\}$	$\text{cost}(x_2, \{x_2\}) = D_{12}$	5
$S = \{x_3\}$	$\text{cost}(x_3, \{x_3\}) = D_{13}$	10
$S = \{x_4\}$	$\text{cost}(x_4, \{x_4\}) = D_{14}$	16
$S = \{x_5\}$	$\text{cost}(x_5, \{x_5\}) = D_{15}$	11

Table 1: Process for subsets S of size 1

Next we consider subsets S of size 2 as follows:

Subset S	cost function	Result
$S = \{x_2, x_3\}$	$\text{cost}(x_2, \{x_2, x_3\}) = \min\{\text{cost}(x_3, \{x_3\}) + D_{32}\} = 10 + 5$	15
	$\text{cost}(x_3, \{x_2, x_3\}) = \min\{\text{cost}(x_2, \{x_2\}) + D_{23}\} = 5 + 7$	12
$S = \{x_2, x_4\}$	$\text{cost}(x_2, \{x_2, x_4\}) = \min\{\text{cost}(x_4, \{x_4\}) + D_{42}\} = 16 + 14$	30
	$\text{cost}(x_4, \{x_2, x_4\}) = \min\{\text{cost}(x_2, \{x_2\}) + D_{24}\} = 5 + 12$	17
$S = \{x_2, x_5\}$	$\text{cost}(x_2, \{x_2, x_5\}) = \min\{\text{cost}(x_5, \{x_5\}) + D_{52}\} = 11 + \infty$	∞
	$\text{cost}(x_5, \{x_2, x_5\}) = \min\{\text{cost}(x_2, \{x_2\}) + D_{25}\} = 5 + \infty$	∞
$S = \{x_3, x_4\}$	$\text{cost}(x_3, \{x_3, x_4\}) = \min\{\text{cost}(x_4, \{x_4\}) + D_{43}\} = 16 + 9$	25
	$\text{cost}(x_4, \{x_3, x_4\}) = \min\{\text{cost}(x_3, \{x_3\}) + D_{34}\} = 10 + 15$	25
$S = \{x_3, x_5\}$	$\text{cost}(x_3, \{x_3, x_5\}) = \min\{\text{cost}(x_5, \{x_5\}) + D_{53}\} = 11 + \infty$	∞
	$\text{cost}(x_5, \{x_3, x_5\}) = \min\{\text{cost}(x_3, \{x_3\}) + D_{35}\} = 10 + \infty$	∞
$S = \{x_4, x_5\}$	$\text{cost}(x_4, \{x_4, x_5\}) = \min\{\text{cost}(x_5, \{x_5\}) + D_{54}\} = 11 + 12$	23
	$\text{cost}(x_5, \{x_4, x_5\}) = \min\{\text{cost}(x_4, \{x_4\}) + D_{45}\} = 16 + 9$	25

Table 2: Process for subsets S of size 2

For the subsets S of size 3, we have the following table:

Subset S	cost function	Result
$S = \{x_2, x_3, x_4\}$	$\text{cost}(x_2, \{x_2, x_3, x_4\}) = \min\{25 + 5, 25 + 14\}$	30
	$\text{cost}(x_3, \{x_2, x_3, x_4\}) = \min\{30 + 7, 17 + 9\}$	26
	$\text{cost}(x_4, \{x_2, x_3, x_4\}) = \min\{15 + 12, 12 + 15\}$	27
$S = \{x_2, x_3, x_5\}$	$\text{cost}(x_2, \{x_2, x_3, x_5\}) = \min\{\infty, \infty\}$	∞
	$\text{cost}(x_3, \{x_2, x_3, x_5\}) = \min\{\infty, \infty\}$	∞
	$\text{cost}(x_5, \{x_2, x_3, x_5\}) = \min\{\infty, \infty\}$	∞
$S = \{x_2, x_4, x_5\}$	$\text{cost}(x_2, \{x_2, x_4, x_5\}) = \min\{23 + 14, \infty\}$	37
	$\text{cost}(x_4, \{x_2, x_4, x_5\}) = \min\{\infty, \infty\}$	∞
	$\text{cost}(x_5, \{x_2, x_4, x_5\}) = \min\{\infty, 17 + 9\}$	26
$S = \{x_3, x_4, x_5\}$	$\text{cost}(x_3, \{x_3, x_4, x_5\}) = \min\{23 + 9, 25 + \infty\}$	32
	$\text{cost}(x_4, \{x_3, x_4, x_5\}) = \min\{\infty, \infty\}$	∞
	$\text{cost}(x_5, \{x_3, x_4, x_5\}) = \min\{25 + \infty, 25 + 9\}$	34

Table 3: Process for subsets S of size 3

Note: Exlanation for the first entry:

$$\begin{aligned}\text{cost}(x_2, \{x_2, x_3, x_4\}) &= \min\{\text{cost}(x_3, \{x_3, x_4\}) + D_{32}, \text{cost}(x_4, \{x_3, x_4\}) + D_{42}\} \\ &= \min\{25 + 5, 25 + 14\} = 30\end{aligned}$$

For the subsets S of size 4, which means $S = \{x_2, x_3, x_4, x_5\}$, we have the following:

cost function	Evaluation	Result
$\text{cost}\{x_2, \{x_2, x_3, x_4, x_5\}\}$	$\min\{\text{cost}(x_3, \{x_3, x_4, x_5\}) + D_{32},$ $\text{cost}(x_4, \{x_3, x_4, x_5\}) + D_{42}, \quad = \min\{32+5, \infty, \infty\}$ $\text{cost}(x_5, \{x_3, x_4, x_5\}) + D_{52}, \}$	37
$\text{cost}\{x_3, \{x_2, x_3, x_4, x_5\}\}$	$\min\{\text{cost}(x_2, \{x_2, x_4, x_5\}) + D_{23},$ $\text{cost}(x_4, \{x_2, x_4, x_5\}) + D_{43}, \quad = \min\{37+7, \infty, \infty\}$ $\text{cost}(x_5, \{x_2, x_4, x_5\}) + D_{53}, \}$	44
$\text{cost}\{x_4, \{x_2, x_3, x_4, x_5\}\}$	$\min\{\text{cost}(x_2, \{x_2, x_3, x_5\}) + D_{24},$ $\text{cost}(x_3, \{x_2, x_3, x_5\}) + D_{34}, \quad = \min\{\infty, \infty, \infty\}$ $\text{cost}(x_5, \{x_2, x_3, x_5\}) + D_{54}, \}$	∞
$\text{cost}\{x_5, \{x_2, x_3, x_4, x_5\}\}$	$\min\{\text{cost}(x_2, \{x_2, x_3, x_4\}) + D_{25},$ $\text{cost}(x_3, \{x_2, x_3, x_4\}) + D_{35}, \quad = \min\{\infty, \infty, 27+9\}$ $\text{cost}(x_4, \{x_2, x_3, x_4\}) + D_{45}, \}$	36

Table 4: Process for subsets S of size 4

For the final step, we have:

$$\begin{aligned}\text{Shortest Hamiltonian cycle} = \min\{&\text{cost}\{x_2, \{x_2, x_3, x_4, x_5\}\} + D_{21},\\ &\text{cost}\{x_3, \{x_2, x_3, x_4, x_5\}\} + D_{31},\\ &\text{cost}\{x_4, \{x_2, x_3, x_4, x_5\}\} + D_{41},\\ &\text{cost}\{x_5, \{x_2, x_3, x_4, x_5\}\} + D_{51}\}\\ = \min\{&37 + 9, 44 + 12, \infty, 36 + 9\}\\ = 45\end{aligned}$$

This result means that this graph has *at least* three Hamiltonian cycles, and the shortest one has length 45. To trace back the order of the vertices in the shortest one, one simply trace back the vertices x_i in the minimal cost functions $\text{cost}(x_i, S)$. In this sample problem, the back-tracing order will be:

$$x_1 \leftarrow x_5 \leftarrow x_4 \leftarrow x_2 \leftarrow x_3 \leftarrow x_1$$

However, this back-tracing order is also correct:

$$x_1 \leftarrow x_5 \leftarrow x_4 \leftarrow x_3 \leftarrow x_2 \leftarrow x_1$$

So, the shortest Hamiltonian cycle has the following forward-going order:

$$x_1 \rightarrow x_3 \rightarrow x_2 \rightarrow x_4 \rightarrow x_5 \rightarrow x_1$$

or:

$$x_1 \rightarrow x_2 \rightarrow x_3 \rightarrow x_4 \rightarrow x_5 \rightarrow x_1$$

One can easily look at Figure 1 and check that the above Hamiltonian cycles both have length 45. This concludes the illustration of Bellman-Held-Karp Algorithm, which is based on Dynamic Programming