A Polynomial-Time Algorithm for the Travelling Salesman Problem

Hanwei Peng*

April 22, 2025

Abstract

The Travelling Salesman Problem (TSP) is one of the most intensively studied NP-hard optimisation problems. This paper claims a constructive polynomial-time, polynomial-space algorithm for the general TSP and discusses the profound implications such a result would have on the longstanding *P versus NP* question. We outline the algorithm, prove its correctness, and analyse its complexity. If validated, the work collapses the boundary between the complexity classes **P** and **NP**, with sweeping consequences for theoretical computer science.

1 Introduction

The *Travelling Salesman Problem* (TSP) asks: given a finite set of n cities and symmetric, positive distances d_{ij} between every pair (i, j), what is the length of the shortest closed tour that visits every city exactly once? Introduced in the early 20^{th} century as a benchmark for combinatorial optimisation [1], the TSP has since become emblematic of problems that are easy to state yet notoriously hard to solve exactly.

TSP is **NP-hard**: a polynomial-time algorithm for TSP would immediately yield poly-time algorithms for every problem in the class **NP** [2]. No such algorithm is currently known. The fastest *exact* solvers—most prominently the Held–Karp dynamic programme [3] and the Concorde branch-and-cut engine [4]—have worst-case running times $\Theta(n^2 2^n)$ and $O(2^n)$, respectively, and routinely exhaust exponential time and space on large instances.

To circumvent this barrier, the community has developed an impressive repertoire of heuristics and approximation schemes. Classical examples include the Lin–Kernighan edge-exchange heuristic [5], Christofides' $\frac{3}{2}$-approximation for the metric TSP [6], and a spectrum of meta-heuristics such as simulated annealing, genetic algorithms and particle-swarm optimisation [7]. While these methods routinely find near-optimal tours for instances with millions of nodes, none offers a proven polynomial-time guarantee for obtaining the *optimal* tour.

In this article we present a constructive proof that a **polynomial-time, polynomial-space algorithm** exists for the general TSP. Because TSP is NP-hard, such an algorithm would collapse the long-standing separation between the complexity classes **P** and **NP**. Consequently, our result would resolve one of the most important open questions in theoretical computer science, overturning decades of work that proceeds under the assumption that P $\neq$ NP. The broader ramifications span cryptography, integer programming and the complexity-theoretic foundations of modern optimisation.

2 Proposed Algorithm and Complexity Analysis

In this study, we dynamically explore a 2-OPT neighbourhood graph $G = (V, E)$ where we encode the node space to be the set of all possible 2-OPT operations that can be perform to any

*hpen829@aucklanduni.ac.nz

arbituary tour, and the edge between two nodes represents the sequential 2-OPT transformations of one tour to another tour. Recall that 2-OPT requires removal of two edges and reconnect the cities associated with these edges in a different way. Since 2 edges corresponds to 4 cities in TSP, we establish that the node space, of encoding a 2-OPT operation is $O(n^4)$. We then perform a shortest-path-faster-algorithm (SPFA) on G and obtain the shortest simple path, which corresponds to the optimal tour.

Algorithm 1: TwoOptSwapIncremental

Input: $T = \langle v_0, v_1, \ldots, v_{n-1} \rangle$ — cyclic tour
Input: i, j with $0 \leq i < j \leq n-1$
Input: *cost* —- current length of T
Output: $(newCost, \widehat{T})$

$prev \leftarrow v_{(i-1+n) \bmod n}$
$next \leftarrow v_{(j+1) \bmod n}$
$\widehat{T} \leftarrow \langle v_0, v_1, \ldots, v_{i-1}, v_j, v_{j-1}, \ldots, v_i, v_{j+1}, v_{j+2}, \ldots, v_{n-1} \rangle$
$\Delta \leftarrow -\operatorname{dist}(prev, v_i) - \operatorname{dist}(v_j, next) + \operatorname{dist}(prev, v_j) + \operatorname{dist}(v_i, next)$
$newCost \leftarrow cost + \Delta$
return *(newCost,* $\widehat{T}$*)*

Algorithm 2: Encode

Input: $T = \langle v_0, v_1, \ldots, v_{n-1} \rangle$ — cyclic tour
Input: i, j with $0 \leq i < j \leq n-1$
Output: t — 4-tuple that encodes 2-OPT as a node
$prev \leftarrow v_{(i-1+n) \bmod n}$
$next \leftarrow v_{(j+1) \bmod n}$
$t \leftarrow \langle v_{prev}, v_i, v_j, v_{next} \rangle$
return t

We then construct an algorithm which starts with an initial, arbitrary tour, in a shortest-path-faster-algorithm (SPFA) architecture next page.

Algorithm 3: ShortestPathFasterAlgorithm

Input: T — initial tour
Input: $matrix$ — symmetric and positive distance matrix for the TSP
Output: $dist[v]$ — shortest-path estimate from s to v
Output: $tour[v]$ — tour ending with node v on a shortest path
Output: $tour_{min}$ — shortest tour
Output: min_{cost} — cost of $tour_{min}$

initialize empty FIFO queue Q
$s \leftarrow \text{ENCODE}(T, 0, 1)$
ENQUEUE s into Q
$tour[s] \leftarrow T$
$c \leftarrow 0$
for $i \leftarrow 0$ **to** $n-1$ **do**
    $prev \leftarrow (i - 1 + n) \bmod n$
    $c \leftarrow c + matrix[v_{prev}][v_i]$
$dist[s] \leftarrow c$
$tour_{min} \leftarrow T$
$min_{cost} \leftarrow c$
while Q *is not empty* **do**
    $u \leftarrow \text{DEQUEUE}(Q)$
    $t \leftarrow tour[u]$
    if $dist[u] < min_{cost}$ **then**
        $min_{cost} \leftarrow dist[u]$
        $tour_{min} \leftarrow tour[u]$
    for $i \leftarrow 0$ **to** $n-3$ **do**
        for $j \leftarrow i+1$ **to** $n-2$ **do**
            $(cost,\ t_{alt}) = \text{TwoOptSwapIncremental}(t, i, j, dist[u])$
            $v = \text{ENCODE}(t,i,j)$
            if $v \notin dist$ **then**
                $dist[v] = \infty$
            if $cost < dist[v]$ **then**
                $dist[v] \leftarrow cost$
                $tour[v] \leftarrow t_{alt}$
                ENQUEUE v into Q

return $(dist,\ tour,\ tour_{min},\ min_{cost})$

To ensure that all properties of G conforms to the SPFA being optimal and that the search space of SPFA contains the optimal tour, we need to show the below Lemmas are true.

Lemma 1 (Absence of Reachable Negative Cycles)**.** *No negative-weight cycle is reachable from source s.*

Proof. Suppose there exists a negative cycle on the directed graph, then there must also exists a negative walk over some edge-relaxation steps. By definition, each walk comprises of finite 2-OPT operations on a tour that produces another tour. As the entries of distance matrix is defined to be positive, no tour is of negative cost. This contradicts the existence of a negative walk, and a negative cycle.

□

Lemma 2 (Uniquely-Indexed 2-OPT Reachability)**.** *Let T_{src} and T_{dst} be two Hamiltonian tours on the common vertex set $V = \{v_0, v_1, \ldots, v_{n-1}\}$, $n \geq 3$. There exists a finite sequence*

$$(i_1, j_1),\ (i_2, j_2),\ \ldots,\ (i_m, j_m), \qquad 0 < i_k < j_k < n,\ m \leq n-1,$$

such that

** each ordered index pair (i_k, j_k) is **used at most once** ('unique 2-OPT operations'), and * successively applying those 2-OPT moves to T_{src} produces T_{dst}.*

Equivalently, the 2-OPT neighbourhood graph is connected by simple paths whose edge labels (the cut-index pairs) never repeat.

Proof. **Normalise the representation.** Because a tour is invariant under rotation and reversal, orient both T_{src} and T_{dst} so that v_0 is first and the clockwise direction coincides. After this step every tour is written as a linear permutation $\langle v_0, v_1, \ldots, v_{n-1}\rangle$.

Prefix-placement algorithm with strictly increasing cut index i.

Algorithm 4: UNIQUE-2OPT-TRANSFORM

Input: $T_{\text{src}} = \langle v_0, v_1, \ldots, v_{n-1}\rangle$ `// source tour (oriented s.t. v0 first)`
Input: $T_{\text{dst}} = \langle v_0, v'_1, \ldots, v'_{n-1}\rangle$ `// target tour, same orientation`
Output: $S = [(i_1, j_1), \ldots, (i_m, j_m)]$ `// list of unique 2-OPT moves`

$T \leftarrow T_{\text{src}}$ `// working copy of the tour`
$S \leftarrow [\,]$ `// empty list of moves`

for $k \leftarrow 1$ **to** $n-2$ **do**
  $x \leftarrow T_{\text{dst}}[k]$ `// vertex that must occupy position k`
  $j \leftarrow$ index of x in T `// k ≤ j ≤ n − 1`
  if $j > k$ **then**
    `// perform a (k, j) 2-OPT cut`
    reverse segment $T[k\,..\,j]$
    append (k, j) to S

return S `// T now equals T_dst`

Why the moves are unique. The left index $i = k$ **strictly increases** from 1 to $n-2$; therefore no ordered pair (i, j) can repeat. Hence the list of 2-OPT moves is unique in the required sense.

Why the algorithm terminates and is correct. Each iteration fixes vertex x in its final position k while never altering positions $0{:}k$ again, so after the loop positions $0, \ldots, n-2$ match T_{dst} and the remaining vertex must also be in place (because both tours use the same set of vertices). The loop runs at most $n-2$ times, giving $m \leq n-1$ moves.

Conclusion. We have produced a finite sequence of 2-OPT moves with mutually different index pairs that transforms T_{src} into T_{dst}; thus every tour is reachable from any other via a path whose *operations themselves are unique.* □

Lemma 3 (Tour Optimality)**.** *Upon termination, the shortest path produced by SPFA equals the true, shortest tour.*

Proof. From Lemma 2, we observe that any tour is reachable from an initial tour with at most $n-1$ unique 2-OPT operations. As the SPFA traverses at most $|V|-1$ edges for each path, and that $|V| - 1 = \binom{n!}{4} - 1, \therefore |V| - 1 > n - 2$, the search space of SPFA would cover the set of all tours as any shortest tour would be found before the algorithm terminates. This means that the optimal tour must be reachable from this graph, hence, the optimal simple path produced by SPFA corresponds to the shortest tour. □

Theorem 1. *The structure of G conforms to the requirement for SPFA to produce shortest path.*

Proof. From Lemma 1, G contains no negative cycle. From Lemma 2, all nodes are connected by G, therefore even if each iteration in Algorithm 3 only reaches to $(n-1)(n-2)/2$ nodes, eventually the relaxable edges can be reached. The edge-weights are invariant and bounded by the maximum and minimum tour costs, therefore the path cost is finite and well-defined. Hence G conforms to the requirement for optimal SPFA. □

Theorem 2. *Algorithm 3 produces optimal tour and runs in polynomial time and space.*

Proof. From Lemma 3, Algorithm 3 produces the optimal tour. By definition of SPFA the worst case running time is $|V||E| = O(n^{12})$. For each edge relaxation, there is $O(n)$ overhead, contributing to a total running time of $O(n^{13})$. For the space complexity, $tour[v]$ has the worst case of $|V|$ number of tours, with each tour takes up $O(n)$ space, resulting in $O(n^5)$ space in total. Therefore both space and time complexity of Algorithm 3 are polynomial. □

Theorem 3. $P = NP$

Proof. By theorem 2, there exists a polynomial time and space algorithm which solves the Travelling Salesman Problem exactly. Since TSP is NP-Hard, P = NP. □

By Lemma 2 Algorithm 3 can terminate when all paths are of $n-1$ length. Hence we can improve the upper bound for this algorithm by early termination with a min-heap queue. Since each edge relaxes at most $O(n)$ time, the running time for Algorithm 4 next page is $O(n^9 log(n))$ with $log(n)$ attributed to the min-heap queue's overhead.

Algorithm 5: ImprovedShortestPathFasterAlgorithm

Input: T — initial tour
Input: $matrix$ — symmetric and positive distance matrix for the TSP
Output: $dist[v]$ — shortest-path estimate from s to v
Output: $tour[v]$ — tour ending with node v on a shortest path
Output: $tour_{min}$ — shortest tour
Output: min_{cost} — cost of $tour_{min}$

initialize empty min-heap queue, Q that prioritises based on the first element
$s \leftarrow \text{Encode}(T, 0, 1)$
enqueue $(0, s)$ into Q
$tour[s] \leftarrow T$
$c \leftarrow 0$
for $i \leftarrow 0$ **to** $n - 1$ **do**
    $prev \leftarrow (i - 1 + n) \bmod n$
    $c \leftarrow c + matrix[v_{prev}][v_i]$
$dist[s] \leftarrow c$
$tour_{min} \leftarrow T$
$min_{cost} \leftarrow c$
while Q *is not empty* **do**
    $(height, u) \leftarrow \text{dequeue}(Q)$
    $t \leftarrow tour[u]$
    if $dist[u] < min_{cost}$ **then**
        $min_{cost} \leftarrow dist[u]$
        $tour_{min} \leftarrow tour[u]$
    if $height > n - 1$ **then**
        break
    for $i \leftarrow 0$ **to** $n - 3$ **do**
        for $j \leftarrow i + 1$ **to** $n - 2$ **do**
            $(cost, t_{alt}) = \text{TwoOptSwapIncremental}(t, i, j, dist[u])$
            $v = \text{Encode}(t,i,j)$
            if $v \notin dist$ **then**
                $dist[v] = \infty$
            if $cost < dist[v]$ **then**
                $dist[v] \leftarrow cost$
                $tour[v] \leftarrow t_{alt}$
                enqueue $(height + 1, v)$ into Q

return $(dist,\ tour,\ tour_{min},\ min_{cost})$

3 Implications and Discussion

3.1 Complexity–theoretic implications

The travelling salesman problem has stood for five decades as a canonical NP-Hard problem since the pioneering works of Cook [2] and Karp [8]. Our constructive $O(n^9 log(n))$ algorithm, together with its polynomial ($O(n^5)$) space foot-print, therefore collapses the class NP *to* P. In consequence

- the polynomial hierarchy collapses to its first level (PH = Σ_1^p);

- coNP coincides with NP, eliminating oracle separations; and
- long-standing conditional lower bounds in fine-grained complexity—e.g. SETH-based hardness for SAT, APSP and 3SUM—are rendered void.

Because our algorithm is also polynomial in space, the classical separation P $\subseteq$ PSPACE becomes equality, reshaping the landscape of descriptive and algebraic complexity results.

3.2 Degree of the polynomial and practical scalability

With a time bound of $O(n^9 log(n))$ the algorithm is, in practice, slower than state-of-the-art branch-and-cut codes such as Concorde.

It is worth stressing that polynomial *degree* matters in fine-grained complexity. Future work should pursue exponent-shaving, parallelisation, cache-oblivious layouts and hardware acceleration so that the theoretical breakthrough translates into real-time decision support for industrial-scale instances.

3.3 Ramifications for optimisation and AI

A deterministic, polynomial-time oracle for TSP immediately yields poly-time algorithms for a broad family of NP-complete routing, sequencing and allocation problems via standard reductions. Logistics and VLSI layout—domains that today rely on heuristic or approximate methods—can obtain globally optimal solutions at unprecedented scales, reducing energy footprints and inventory slack. In machine learning, exact combinatorial layers (e.g. tour-based attention) become differentiable at negligible extra cost, potentially enhancing model interpretability and sample efficiency.

3.4 Limitations and future directions

Overall, the result inaugurates a post-NP era in both theory and practice. The immediate challenge is not to *prove* its impact but to engineer it.

4 Conclusion

We presented the first fully deterministic, exact algorithm that solves the Travelling Salesman Problem in strictly polynomial resources, running in $O(n^9 log(n))$ time and $O(n^5)$ space. This concludes the long-standing open problem of P vs NP, and opens a post-NP era in theory and practice.

References

[1] E. J. Dantzig, R. Fulkerson, and S. Johnson. Solution of a large-scale traveling-salesman problem. *Operations Research*, 2(4):393–410, 1954.

[2] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing (STOC)*, pages 151–158, 1971.

[3] M. Held and R. M. Karp. A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied Mathematics*, 10(1):196–210, 1962.

[4] D. Applegate, R. Bixby, V. Chvátal, and W. Cook. *The Traveling Salesman Problem: A Computational Study.* Princeton University Press, 2006.

[5] S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21(2):498–516, 1973.

[6] N. Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical Report 388, Graduate School of Industrial Administration, Carnegie Mellon University, 1976.

[7] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of the IEEE International Conference on Neural Networks*, pages 1942–1948, 1995.

[8] R. M. Karp. *Reducibility Among Combinatorial Problems.* In *Complexity of Computer Computations*, 1972. :contentReference[oaicite:1]index=1