DualOpt: A Dual Divide-and-Optimize Algorithm for the Large-scale Traveling Salesman Problem

Shipei Zhou[1*], Yuandong Ding[1*], Chi Zhang[1], Zhiguang Cao[2], Yan Jin[1†]

[1]School of Computer Science, Huazhong University of Science and Technology, China
[2]School of Computing and Information Systems, Singapore Management University, Singapore
m202273616@hust.edu.cn, yuandong@hust.edu.cn, chizhang02@hust.edu.cn, zgcao@smu.edu.sg,jinyan@mail.hust.edu.cn

Abstract

This paper proposes a dual divide-and-optimize algorithm (DualOpt) for solving the large-scale traveling salesman problem (TSP). DualOpt combines two complementary strategies to improve both solution quality and computational efficiency. The first strategy is a grid-based divide-and-conquer procedure that partitions the TSP into smaller sub-problems, solving them in parallel and iteratively refining the solution by merging nodes and partial routes. The process continues until only one grid remains, yielding a high-quality initial solution. The second strategy involves a path-based divide-and-optimize procedure that further optimizes the solution by dividing it into sub-paths, optimizing each using a neural solver, and merging them back to progressively improve the overall solution. Extensive experiments conducted on two groups of TSP benchmark instances, including randomly generated instances with up to 100,000 nodes and real-world datasets from TSPLIB, demonstrate the effectiveness of DualOpt. The proposed DualOpt achieves highly competitive results compared to 10 state-of-the-art algorithms in the literature. In particular, DualOpt achieves an improvement gap up to 1.40% for the largest instance TSP100K with a remarkable 104x speed-up over the leading heuristic solver LKH3. Additionally, DualOpt demonstrates strong generalization on TSPLIB benchmarks, confirming its capability to tackle diverse real-world TSP applications.

Code — https://github.com/Learning4Optimization-HUST/DualOpt

Introduction

The Traveling Salesman Problem (TSP) is an NP-hard combinatorial optimization problem, which has numerous real-world applications (Madani, Batta, and Karwan 2021; Hacizade and Kaya 2018; Matai, Singh, and Mittal 2010). Let $G = (V, E)$ represent an undirected graph, where $V = \{v_i \mid 1 \leq i \leq N\}$ is the set of nodes and $E = \{e_{ij} \mid 1 \leq i, j \leq N\}$ is the set of edges, with N being the total number of nodes. For each edge e_{ij}, the travel cost $cost(i, j)$ is defined as the Euclidean distance between the nodes v_i and v_j. A special node $v_d \in V$ serves as the depot, from which the salesman starts and ends the trip. A feasible solution of TSP is defined as a Hamiltonian cycle that starts from the depot, visits each node exactly once, and ends at the depot. The objective is to minimize the total travel cost of the solution route τ, denoted as $L(\tau) = \sum_{i=1}^{N-1} cost(\tau_i, \tau_{i+1}) + cost(\tau_N, \tau_1)$, where τ_i represents the $i-th$ node in the route.

Due to its theoretical and practical interest, various traditional exact/heuristic and machine learning-based algorithms have been proposed in the literature. While exact algorithms are generally computationally infeasible for large-scale instances, heuristics can provide near-optimal solutions for TSP with millions of cities, though they cannot guarantee optimality. They often involve time-consuming iterative searches, making them less suitable for time-sensitive tasks. Machine learning based algorithms, on the other hand, offer high computational efficiency and can achieve solution quality comparable to traditional methods for small-scale TSP instances. However, applying them to large-scale TSP, especially those with over 1,000 cities, remains a challenge. One approach is to leverage pre-trained models from small-scale instances for larger ones, but this often results in poor performance due to distribution shift (Joshi et al. 2022). Training models specifically for large-scale TSP is also impractical due to computational resource limitations.

A basic approach to deal with the large-scale routing problem is to apply the general principle of "divide-and-conquer" (Fu, Qiu, and Zha 2021; Pan et al. 2023; Hou et al. 2023; Chen et al. 2023; Ye et al. 2024b; Zheng et al. 2024; Xia et al. 2024). It involves decomposing TSP into a subset of small sub-problems, which can be efficiently solved using traditional or machine learning algorithms to attain sub-solutions. The final solution is then obtained by combining these sub-solutions. This approach can significantly reduce the computational complexity of the large-scale TSP, and yield high-quality solutions in a relatively short time.

In this work, we introduce a novel dual divide-and-optimize algorithm DualOpt based on the divide-and-conquer framework. The first divide-and-optimize procedure partitions the nodes into $M \times M$ equal-sized grids based on their coordinates. The nodes within each grid are

*These authors contributed equally.
†Yan Jin is the corresponding author.

initially solved using the well-known LKH3 solver (Helsgaun 2017). Next, an edge-breaking strategy is proposed to decompose the route into partial routes and nodes, significantly reducing the number of nodes. Subsequently, every four adjacent grids are merged into one larger grid. This process is repeated until all nodes are contained within a single grid, at which point the LKH3 solver is applied to obtain a complete route based on the partial routes and nodes. The second divide-and-optimize procedure further refines the obtained route by partitioning it into non-overlapping subpaths. These subpaths are then optimized in parallel using a neural solver (Kim, Park et al. 2021; Pan et al. 2023; Ye et al. 2024b), ultimately leading to an optimized route. The main contributions of this work are summarized as follows:

- We propose the DualOpt algorithm, which combines two complementary divide-and-conquer frameworks to address the large-scale TSP. The first framework employs grid-based partitioning, while the second applies path-based optimization, significantly improving both solution quality and computational efficiency of large-scale TSP.
- We introduce a novel edge-breaking strategy that decomposes routes into partial routes and nodes. By representing these partial routes solely by their start and end nodes, this strategy considerably reduces the number of nodes, thereby decreasing computational complexity and improving search efficiency.
- We conduct extensive experiments on both randomly generated and real-world large-scale TSP instances. To the best of our knowledge, DualOpt achieves state-of-the-art performance compared to other machine learning-based approaches, particularly excelling in large-scale instances with up to 100,000 nodes.

Related Work

Here we present representative traditional and machine learning-based algorithms to solve TSP, and then focus on the divide-and-conquer algorithms that are more effective for solving large-scale TSP over 1000 nodes.

Traditional Algorithms Traditional algorithms can be roughly classified into two categories: exact and heuristic algorithms (Gutin and Punnen 2006; Accorsi and Vigo 2021). Concorde (Applegate et al. 2009) is one of the best exact solvers, which models TSP as a mixed-integer programming problem and solves it using a branch-and-cut (Toth and Vigo 2002). Exact algorithms can theoretically guarantee optimal solutions for instances of limited size, but are impractical for solving large instances due to their inherent exponential complexity. LKH3 (Helsgaun 2017) is one of the state-of-the-art heuristics that uses the k-opt operators to find neighboring solutions, which is guided by a α-nearness measure based on the minimum spanning tree. The heuristics are the most widely used algorithms in practice, yet they are still time consuming to obtain high-quality solutions when solving problems with tens of thousands of nodes.

Machine Learning-based Algorithms In recent years, machine learning-based algorithms have attracted more interest in solving the TSP. Depending on how solutions are constructed, they can be broadly categorized into *end-to-end* approaches and *search-based* approaches.

End-to-end approaches generate solutions from scratch. The Attention Model (Kool, van Hoof, and Welling 2019) utilizes the Transformer architecture (Vaswani et al. 2017) and is trained with REINFORCE (Wiering and Van Otterlo 2012) using a greedy rollout baseline. POMO (Kwon et al. 2020) improves on this by selecting multiple nodes as starting points, using symmetries in solution representation, and employing a shared baseline to enhance REINFORCE training. DIFUSCO (Sun and Yang 2023) uses graph-based denoising diffusion models to generate solutions, which are further optimized by local search with k-opt operators. Although DIFUSCO can handle problems with up to 10,000 nodes, it is less suitable for time-sensitive scenarios. Pointerformer (Jin et al. 2023) incorporates a reversible residual network in the encoder and a multi-pointer network in the decoder, allowing it to solve problems with up to 1000 nodes. *Search-based* approaches start with a feasible solution and iteratively apply predefined rules to improve it. NeuRewriter (Chen and Tian 2019) iteratively rewrites local components through a region-picking and rule-picking process, with the model trained using Advantage Actor-Critic, and the reduced cost per iteration serves as the reward. NeuroLKH (Xin et al. 2021) enhances the traditional LKH3 solver by employing a sparse graph network trained through supervised learning to simultaneously generate edge scores and node penalties, which guide the improvement process. DeepACO (Ye et al. 2024a) integrates neural enhancements into Ant Colony Optimization (ACO) algorithms, further improving its performance.

Machine learning-based algorithms perform well on TSP instances with up to 1,000 nodes, but struggle with larger instances due to the exponential increase in memory requirements and computation time as the number of nodes grows. This leads to memory and time constraints during training, making it difficult to converge to near-optimal solutions. To address this challenge in solving large-scale problems with thousands or more nodes, the divide-and-conquer strategy is often employed. It is always combined with traditional or learning-based algorithms to generate high-quality solutions in a relatively short time. GCN-MCTS (Fu, Qiu, and Zha 2021) applies graph sampling to construct fixed-size sub-problems, solved by graph convolutional networks, with heatmaps guiding Monte-Carlo Tree Search (MCTS). H-TSP (Pan et al. 2023) hierarchically builds TSP solutions using a two-level policy: the upper-level selects sub-problems, while the lower-level generates and merges open-loop routes. ExtNCO (Chen et al. 2023) uses LocKMeans with $o(n)$ complexity to divide nodes into sub-problems, solving them with neural combinatorial optimization and merging solutions via a minimum spanning tree. GLOP (Ye et al. 2024b) learns global partition heatmaps to decompose large-scale problems and introduces a scalable real-time solver for small Shortest Hamiltonian Path problems. UDC (Zheng et al. 2024) proposes a Divide-Conquer-Reunion framework using efficient Graph Neural Networks for division and fixed-length solvers for sub-problems. Soft-

Dist (Xia et al. 2024) demonstrates that a simple baseline method outperforms complex machine learning approaches in heatmap generation, and the heatmap-guided MCTS paradigm is inferior to the LKH3 heuristic despite leveraging hand-crafted strategies.

The Proposed DualOpt Algorithm

The proposed DualOpt algorithm is based on and extends the basic divide-and-conquer method by incorporating a grid-based procedure and a path-based procedure to improve efficiency. Each procedure operates on two levels: the first level is responsible for generating sub-problems, while the second level focuses on solving these sub-problems.

Algorithm 1: The Proposed DualOpt Algorithm for the Large-scale TSP

Input: TSP instance $V = \{v_1, v_2, \cdots, v_N\}$
Output: Solution route τ

$PartialRoutesSet\ \Upsilon \leftarrow \emptyset$;
$NodesSet\ \aleph \leftarrow V$;
repeat
  $Grids \leftarrow Partition(\Upsilon, \aleph)$;
  $Routes \leftarrow SolveGridsParallel(\Upsilon, \aleph)$;
  $\Upsilon', \aleph' \leftarrow EdgeBreaking(Routes)$;
  $\Upsilon \leftarrow \Upsilon'$;
  $\aleph \leftarrow \aleph'$;
until *predefined iterations reached*;
$\tau \leftarrow SolveReducedProb(\Upsilon, \aleph)$;
repeat
  $SubPath \leftarrow DivideSolution(\tau)$;
  $SubPath' \leftarrow OptimizeSubPathBatch(SubPath)$;
  $\tau' \leftarrow MergeSubPath(SubPath')$;
  $\tau \leftarrow \tau'$;
until *predefined iterations reached*;
return τ

The whole procedure of DualOpt is summarized in Algorithm 1. It starts by partitioning the TSP instance into smaller grids, which reduces the problem size and allows for parallel solving. In each iteration, the partial routes and nodes within each grid are solved in parallel to generate a solution for that portion of the TSP. Next, an edge-breaking procedure is applied to divide each route into smaller partial routes and nodes. The grids are then merged into larger ones using a grid merging procedure. This grid-based iteration continues until only one grid remains, forming a reduced problem that consists of a number of partial routes and nodes. From this reduced problem, a high-quality initial solution of is constructed. To further refine the solution, it is divided into subpaths, which are optimized in batches and then merged. This iterative refinement, performed with varying partition sizes, progressively improves the solution quality.

A Grid-based Divide-and-Conquer Procedure

To decompose the large-scale TSP for efficient solving without significantly downgrading solution quality, we employ an iterative grid decomposition strategy, denoted as the Grid-based Divide-and-Conquer Procedure, as depicted in Algorithm 2. Initially, the node set $\aleph$ contains all the nodes of the TSP instance, while the partial route set Υ is initialized as empty. In each iteration, the 2D space is discretized into an evenly spaced grid of size $K = 2^{N_{iter}-iter} \times 2^{N_{iter}-iter}$ grids. The node set $\aleph$ and partial route set Υ are then divided into K subsets $\{(\Upsilon_1, \aleph_1), \ldots, (\Upsilon_K, \aleph_K)\}$ based on their positions within the grid. Each of these subsets is solved in parallel using the well-known TSP solver LKH3 (Helsgaun 2017), resulting in K small routes $\{R_1, \ldots, R_K\}$. If the current iteration (*iter*) is not the last one, the algorithm proceeds by applying a proposed edge-breaking strategy in parallel, which breaks a subset of the edges of the routes and updates Υ and $\aleph$ with new sets of partial routes and nodes $\{(\Upsilon'_1, \aleph'_1), \ldots, (\Upsilon'_K, \aleph'_K)\}$. As the iterations progress, K decreases and the grid size increases correspondingly until only one grid remains. At this final stage, a reduced problem consisting of partial routes and nodes is formed, from which an initial high-quality TSP solution τ is obtained.

Algorithm 2: Grid-based Divide-and-Conquer

Input: TSP instance $V = \{v_1, v_2, ..., v_N\}$, number of iterations N_{iter}
Output: Solution route τ

$iter \leftarrow 1$;
$NodesSet\ \aleph \leftarrow V$;
$PartialRoutesSet\ \Upsilon \leftarrow \emptyset$;
while $iter \leq N_{iter}$ **do**
  $K \leftarrow 2^{N_{iter}-iter} \times 2^{N_{iter}-iter}$; /*Calculate the number of grids*/
  $\{(\Upsilon_1, \aleph_1), \ldots, (\Upsilon_K, \aleph_K)\} \leftarrow Partition(\Upsilon, \aleph, K)$; /*Partition nodes and partial routes into grids*/
  $\{R_1, ..., R_K\} \leftarrow SolveGridsParallel(\{(\Upsilon_1, \aleph_1), \ldots, (\Upsilon_K, \aleph_K)\})$; /*Solve the TSP for each grid in parallel*/
  if $iter \neq N_{iter}$ **then**
    $\{(\Upsilon'_1, \aleph'_1), \ldots, (\Upsilon'_K, \aleph'_K)\} \leftarrow BreakEdgesParallel(\{R_1, \ldots, R_K\}$; /*Break edges into partial routes and nodes in parallel*/
    Υ is updated by $\{\Upsilon'_1, \ldots, \Upsilon'_K\}$;
    $\aleph$ is updated by $\{\aleph'_1, \ldots, \aleph'_K\}$;
  else
    $\tau \leftarrow SolveReducedProb(\Upsilon, \aleph)$; /*Solve the reduced problem to get a complete route*/
  $iter \leftarrow iter + 1$;
return τ

Figure 1 illustrates this procedure with $N_{iter} = 3$. In the first iteration, K is 16, partitioning the 2D space into 16 evenly spaced grids. The LKH3 generates 16 localized

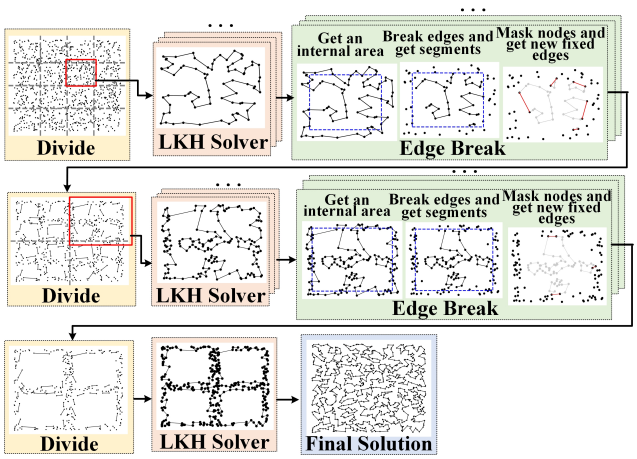

Figure 1: An illustration of the Grid-based Divide-and-Conquer Procedure with $N_{iter} = 3$.

routes per grid, which are then decomposed into partial routes and separate nodes. In the second iteration, K is reduced to 4, resulting in 4 larger grids where the LKH3 processes partial routes with fixed nodes. In the final iteration, K is reduced to 1, merging the grids into a single one. As a result, the number of nodes decreases while the number of partial routes increases. This reduction in problem scale enables the LKH3 solver to efficiently find a TSP solution with high quality.

The LKH3 Solver The LKH3 solver, a state-of-the-art heuristic for the TSP, is built upon the foundational Lin-Kernighan heuristic, incorporating several advanced components. Among these, the use of a 1-tree structure is particularly notable, as it provides a lower bound that guides the search process effectively. The algorithm dynamically employs K-opt neighborhood operators, where multiple edges are iteratively removed and reconnected to explore potential routes within the solution space. The efficiency of LKH3 is further enhanced by using candidate sets, which strategically limits the number of neighborhood operators, thereby reducing search overhead.

The number of nodes within each grid, typically ranging from a few hundreds to approximately one thousand, is determined by both the instance scale and the number of grids K. The input to the solver includes not only the coordinates of nodes but also the fixed partial routes. LKH3 excels in handling node scales within this range, delivering high-quality solutions with remarkable speed, particularly for routing problems involving fixed edges. Therefore, LKH3 was selected as the subsolver for this study.

Edge-breaking Strategy Recall that multiple routes are generated, one for each grid within the 2D space. When merging and optimizing these routes, it is essential to reschedule the surrounding nodes of each grid in conjunction with the nodes of adjacent grids, while excluding the internal nodes from this rescheduling process. To achieve this, we propose an edge-breaking strategy that effectively reduces the problem's scale. Specifically, for each grid, we define an internal grid that maintains a fixed spacing from the outer grid. This spacing is calculated as spacing = $\frac{x_{\max}-x_{\min}}{2^{N_{\text{iter}}+2}}$, where $x_{\max}$ and $x_{\min}$ denote the maximum and minimum x-coordinates of the grid, respectively. We then remove all edges that lie outside the internal grid, as well as those crossing into it. As a result, the nodes and edges within the internal grid form several partial routes that each may consist of more than two connected nodes, along with isolated nodes.

A Path-based Divide-and-Optimize Procedure

Based on the solution derived from the grid-based divide-and-conquer procedure, further optimization is achieved through the path-based divide-and-optimize procedure, as detailed in Algorithm 3. Following previous studies (Kim, Park et al. 2021; Ye et al. 2024b; Zheng et al. 2024), this iterative algorithm aims to refine an initial solution τ by partitioning it into smaller sub-paths, optimizing these sub-paths individually, and then merging them to generate an improved solution τ^*.

The procedure begins by iterating over a specified set of sub-path lengths $len_1, \ldots, len_m$ and iterations $iter_1, \ldots, iter_m$. For each sub-path length len, the initial solution τ is divided into sub-paths of the designated length. If the length of the last sub-path is shorter than the designated length, it remains unchanged. Each sub-path can be considered as an open-loop TSP with two fixed endpoints. These sub-paths undergo a distribution normalization of vertex coordinates to maintain consistency with the original TSP instance, resulting in normalized sub-paths denoted as $SubPathSet^{'}$. Following normalization, a neural solver is employed in batch mode to optimize the normalized sub-paths, and sub-paths are updated by the policy when they are better than the current ones, producing a set of sub-paths $SubPathSet^*$. These sub-paths are then merged to form an improved solution τ^* by connecting their fixed endpoints in their original order. To ensure continuous refinement and overlap during the iterations, the starting point κ for sub-path division is incremented by $\max(1, \frac{len}{iter})$. This iterative refinement process is repeated for each combination of sub-path length and iteration count, progressively refining the manageable sub-paths of the initial solution τ to yield a highly optimized solution τ^*.

Neural Solver The underlying solver of the path-based divide-and-optimize procedure is an attention-based neural network, which can efficiently solve the obtained sub-paths through batch parallelism.

The neural network uses an encoder-decoder architecture. The encoder employs self-attention layers to embed the input node sequence, while the decoder generates the node sequence auto-regressively. Each sub-path π can be regarded as an open-loop TSP with specified start and end nodes. We employ a modified context embedding inspired by (Kim, Park et al. 2021), defined as $h_{(c)}^{(L)} = [\overline{h}^{(L)}, h_{\pi_{pre}}^{(L)}, h_{\pi_{des}}^{(L)}]$. Here, h denotes a high-dimensional embedding vector from the encoder, and L represents the number of multi-head attention layers. $\overline{h}^{(L)}$ is the mean of the final node embed-

Algorithm 3: Path-based Divide-and-Optimize

Input: Solution $\tau = \{\tau_1, \tau_2, ..., \tau_N\}$, a set of sub-path lengths$\{len_1, \ldots, len_m\}$, and a set of iterations $\{iter_1, \ldots, iter_m\}$
Output: Optimized Solution τ^*

for $len \leftarrow len_1$ ***to*** len_m **do**
  $\kappa \leftarrow 1$;
  for $iter \leftarrow iter_1$ ***to*** $iter_m$ **do**
    $SubPathSet \leftarrow \{\tau_{\kappa:\kappa+len}, \tau_{\kappa+len:\kappa+2len}, \ldots\}$;
    $SubPathSet^{'} \leftarrow NormalizeVertexCoordinate(SubPathSet)$;
    $SubPathSet^{*} \leftarrow NeuralSolverBatch(SubPathSet^{'})$;
    $\tau^* \leftarrow MergeSubPath(SubPathSet^{*})$;
    $\kappa \leftarrow \kappa + max(1, \frac{len}{iter})$;
    $\tau \leftarrow \tau^*$
return τ^*

dings, $h^{(L)}_{\pi_{pre}}$ is the embedding of the previously selected node, and $h^{(L)}_{\pi_{des}}$ is embedding of the end node of the sub-path. Moreover, following insights from previous studies (Kim, Park et al. 2021; Cheng et al. 2023; Ye et al. 2024b), we leverage the symmetry of sub-paths. Specifically, reversing the traversal direction of a sub-path (moving backward versus forward) yields an equivalent sequence, which helps improve the network's robustness and efficiency.

The network employs a single-step constructive policy aimed at optimizing the solution by generating sequences in both forward and backward directions. The policy is defined as follows, where s represents a sub-path with a start node π_1 and an end node π_n. The policy $p_\theta(\boldsymbol{\pi_f}, \boldsymbol{\pi_b}|s)$ is parameterized by θ and specifies a stochastic policy for constructing the forward solution $\boldsymbol{\pi_f}$ and the backward solution $\boldsymbol{\pi_b}$:

$$p_\theta(\boldsymbol{\pi_f}, \boldsymbol{\pi_b}|s) = p_\theta(\boldsymbol{\pi_f}|s)p_\theta(\boldsymbol{\pi_b}|s) = \prod_{t=1}^{n-1} p_\theta(\pi_{1+t}|s, \boldsymbol{\pi_{1:t}}, \pi_n) \times \prod_{t=1}^{n-2} p_\theta(\pi_{n-t}|s, \boldsymbol{\pi_{n:n-t+1}}, \pi_1). \tag{1}$$

The model is trained using REINFORCE with a shared baseline. The objective is to minimize the expected length of the solutions to the sub-problems. To achieve this, the loss function is defined as the expected length of these solutions. The policy gradient is then computed as follows:

$$\begin{aligned}\nabla\mathcal{L}(\theta|s) =& \mathrm{E}_{p_\theta(\boldsymbol{\pi_f}|s)}[R(\boldsymbol{\pi_f}) - b(s)]\nabla \log p_\theta(\boldsymbol{\pi_f}|s) \\ &+ \mathrm{E}_{p_\theta(\boldsymbol{\pi_b}|s)}[R(\boldsymbol{\pi_b}) - b(s)]\nabla \log p_\theta(\boldsymbol{\pi_b}|s).\end{aligned} \tag{2}$$

The shared baseline $b(s)$ is employed to reduce variance and enhance training stability. This baseline is computed by averaging the path lengths obtained from two greedy rollouts. Note that during model training, both forward and backward solutions are utilized. However, during inference, only the superior trajectory between the two solutions is selected to ensure the best possible outcome.

Node Coordinate Normalization To improve the robustness of the model with respect to the sub-paths, we normalize the node coordinates to be uniformly distributed within the range $[0, 1]$ (Fu, Qiu, and Zha 2021). We define $x_{max} = \max_{i\in G} x_i$, $x_{min} = \min_{i\in G} x_i$, $y_{max} = \max_{i\in G} y_i$, $y_{min} = \min_{i\in G} y_i$ as the maximum and minimum values of the horizontal and vertical coordinates of all N nodes in the TSP instance. For each node in the sub-path, we convert its coordinate (x_i, y_i) to $(x_i^{'}, y_i^{'})$ as shown in Eq. (3), ensuring that all coordinates fall within the range $[0, 1]$.

$$\begin{aligned} x_i^{'} &= \frac{x_i - x_{\min}}{\max(x_{\max} - x_{\min}, y_{\max} - y_{\min})}, \\ y_i^{'} &= \frac{y_i - y_{\min}}{\max(x_{\max} - x_{\min}, y_{\max} - y_{\min})}. \end{aligned} \tag{3}$$

Experiments

To evaluate the performance of our proposed DualOpt, we conducted extensive experiments on the large-scale TSP instances, with up to 100,000 nodes. We compared the performance of DualOpt with that of state-of-the-art algorithms from the literature. All experiments, including rerunning the baseline algorithms, were executed on a machine with an RTX 3080(10GB) GPU and a 12-core Intel(R) Xeon(R) Platinum 8255C CPU.

Our evaluation focused on two groups of the large-scale instances: randomly generated instances **TSP_random** and the well-known real-world benchmark **TSPLIB**. The **TSP_random** represents random Euclidean instances, with node coordinates uniformly sampled from a unit square $[0, 1]^2$. This group includes seven datasets, each corresponding to a different number of nodes, denoted as TSPN, i.e., TSP1K (1,000 nodes), TSP2K, TSP5K, TSP10K, TSP20K, TSP50K, and TSP100K (100,000 nodes). For a fair comparison, we used the same test instances provided by Fu et al. (Fu, Qiu, and Zha 2021) and Pan et al. (Pan et al. 2023). Each dataset contains 16 instances, except TSP1K, which includes 128 instances, and TSP100K, which contains one instance. The **TSPLIB** is a well-known real-world benchmark (Reinelt 1991) that consists of 100 instances with diverse node distributions derived from practical applications, with sizes ranging from 14 to 85,900 nodes. For our experiments, we select the ten largest TSPLIB instances, each containing more than 5,000 nodes.

Baselines We compare DualOpt against ten leading TSP algorithms in the literature. **Traditional Algorithms:** 1) Concorde (Cook et al. 2011): One of the best exact solvers. 2) LKH3 (Helsgaun 2017): One of the highly optimized heuristic solvers. **Machine Learning Based Algorithms:** 1) POMO (Kwon et al. 2020): Feature an *end-to-end* model based on Attention Model, comparable to LKH3 for small-scale TSP instances. 2) DIMES (Qiu, Sun, and Yang 2022): Introduce a compact continuous space for parameterizing the underlying distribution of candidate solutions for solving large-scale TSP with up to 10K nodes. 3) DIFUSO (**?**): Leverage a graph-based denoising diffusion model to solve

large-scale TSP with up to 10K nodes. 4) SIL (Luo et al. 2024): Develop an efficient self-improved mechanism that enables direct model training on large-scale problem instances without labeled data, handling TSP instances with up to 100K nodes. **Divide-and-Conquer Algorithms:** 1) GCN+MCTS (Fu, Qiu, and Zha 2021): Combine a GCN model trained with supervised learning and MCTS to solve large-scale TSP involving up to 10K nodes with a long searching time. 2) SoftDist (Xia et al. 2024): Critically evaluate machine learning guided heatmap generation, the heatmap-guided MCTS paradigm for large-scale TSP. 3) H-TSP (Pan et al. 2023): Hierarchically construct a solution of a TSP instance with up to 10K nodes based on two-level policies. 4) GLOP (Ye et al. 2024b): Decompose large routing problems into Shortest Hamiltonian Path Problems, further improved by local policy. It is the first neural solver to effectively scale to TSP with up to 100K nodes.

Parameter Settings For the grid-based divide-and-conquer procedure, we set the value of N_{iter} for different TSP problem scales, as shown in Table 1. The LKH3 solver is used with its default parameters as specified in the literature (Helsgaun 2017) both in DualOpt and baseline. LKH3 baseline runs instance-by-instance. In the grid-based divide-and-conquer stage, LKH3 sub-solver just run parallel for grids in a single instance. For the path-based divide-and-optimize procedure, we train different neural solvers with graph size $len = \{50, 20, 10\}$, with node coordinates randomly generated from a uniform distribution within a unit square. During the training process, we use the same hyper-parameter settings as those used by Kool et al (Kool, van Hoof, and Welling 2019). And during test time, we set $iter = \{25, 10, 5\}$.

Number of Iterations	Problem Scale of TSP
$N_{iter} = 2$	$N < 5,000$
$N_{iter} = 3$	$5,000 \leq N < 20,000$
$N_{iter} = 4$	$20,000 \leq N < 100,000$
$N_{iter} = 5$	$N \geq 100,000$

Table 1: Parameter setting of N_{iter}.

Comparative Studies Table 2 presents a comparison of 10 leading algorithms and our DualOpt algorithm on randomly distributed datasets **TSP_random**. The "Obj." column shows the average objective lengths of the routes obtained by each algorithm for each instance, while the "Gap" column measures the percentage difference between the average route lengths attained by each algorithm and the LKH3, considered as the ground truth, i.e., $Gap = \frac{\textit{Obj. of Algo. - Obj of LKH3}}{\textit{obj. of LKH3}} \times 100\%$. The "Time" column indicates the average time required to solve each instance. The † following the algorithm indicates that the results of this algorithm are drawn from the literature directly and "-" means the results are not provided in the literature. The "OOM" stands for out of CUDA memory with our platform.

From Table 2, we can make the following comments about the **TSP_random** instances. First, the heuristic solver LKH3 provides the highest quality solutions for all instances except TSP1K and TSP2K, though it requires long time searching. DualOpt matches or improves upon LKH3's results for all instances except TSP5K, and achieves an improved result with an improvement gap up to 1.40% for the largest instance TSP100K, with a remarkable 104x speed-up. Second, DualOpt clearly outperforms machine learning based algorithms in both solution quality and running time. POMO underperforms across all instances, primarily because it cannot be trained directly on large-scale instances, and models trained on small-scale instances fail to generalize effectively to large-scale instances. Compared to the current state-of-the-art SIL, our DualOpt surpasses SIL in all instances except for a tie on TSP1K. Third, DualOpt consistently outperforms all four divide-and-conquer algorithms in terms of the best objective result, achieving better results than GLOP for all instances. These observations highlight the superiority of DualOpt in both solution quality and computational efficiency.

To demonstrate the solver's generalization capabilities, we then directly applied the trained neural solver to the real-world dataset TSPLIB with varied distributions. Table 3 presents the results for the 10 largest instances from TSPLIB. Despite the different node distributions in TSPLIB compared to those in TSP_random, DualOpt demonstrates impressive generalization capabilities. It achieves highly competitive results compared to three leading large-scale TSP algorithms from the literature. This performance highlights DualOpt's effectiveness in handling various types of TSP instances and further underscores its versatility in real-world applications.

An Ablation Study Our DualOpt integrates two important divide-and-conquer procedures to achieve superior performance. To evaluate the impact of each procedure, we conducted an ablation study with two distinct variants of DualOpt: DualOpt$_{w/oPath}$ and DualOpt$_{w/oGrid}$. DualOpt$_{w/oPath}$ employs only the grid-based divide-and-conquer procedure, while DualOpt$_{w/oGrid}$ generates the initial solution using a simple yet effective approach called random insertion, which is then improved using the path-based divide-and-optimize procedure. The results of this ablation study are summarized in Table 4. The comparative results clearly demonstrate the significance of both procedures within DualOpt. Specifically, the original DualOpt consistently outperforms the variants, underscoring the importance of combining both the grid-based and path-based procedures for achieving superior performance.

Conclusion

In this paper, we introduced a novel dual divide-and-optimize algorithm DualOpt to solve large-scale traveling salesman problem, which integrates two complementary strategies: a grid-based divide-and-conquer procedure and a path-based divide-and-optimize procedure. Through extensive experiments on randomly generated instances and real-world datasets, DualOpt consistently achieves highly competitive results with state-of-the-art TSP algorithms in terms of both solution quality and computational efficiency. More-

Algorithm	TSP1K Obj.	TSP1K Gap(%)	TSP1K Time	TSP2K Obj.	TSP2K Gap(%)	TSP2K Time	TSP5K Obj.	TSP5K Gap(%)	TSP5K Time
LKH3	23.31	0.00	2.4s	32.89	0.00	9.3s	51.52	0.00	1.3m
Concorde	23.12	-0.82	7m	32.44	-1.37	2.8h	53.45	3.75	3.2h
POMO†	30.52	30.93	4.3s	46.49	41.35	35.9s	80.79	56.81	9.6m
DIMES†	23.69	1.63	2.2m	-	-	-	-	-	-
DIFUSCO†	23.39	0.34	11.5s	-	-	-	-	-	-
SIL†	**23.31**	0.00	0.6s	-	-	-	51.92	0.78	28.5s
GCN-MCTS†	23.86	2.36	5.8s	33.42	1.61	3.3m	52.83	2.54	6.3m
SOFTDIST†	23.63	1.37	1.57	-	-	-	-	-	-
H-TSP	24.66	5.79	0.7s	35.22	7.08	1.5s	55.72	8.15	2.3s
GLOP	24.01	3.00	0.4s	33.90	3.07	0.9s	53.49	3.82	1.8s
DualOpt	**23.31**	**0.00**	5.6s	**32.72**	**-0.52**	7.6s	51.56	0.08	13.9s

Method	TSP10K Obj.	TSP10K Gap(%)	TSP10K Time	TSP20K Obj.	TSP20K Gap(%)	TSP20K Time	TSP50K Obj.	TSP50K Gap(%)	TSP50K Time	TSP100K Obj.	TSP100K Gap(%)	TSP100K Time
LKH3	72.96	0.00	6.3m	103.28	0.00	27.4m	164.08	0.00	3.0h	234.098	0.00	14.9h
DIMES†	74.06	1.50	13m	-	-	-	-	-	-	-	-	-
DIFUSCO†	73.62	0.90	3m	-	-	-	-	-	-	-	-	-
SIL†	73.32	0.49	51s	-	-	-	164.53	0.27	3.8m	232.66	-0.61	7.5m
GCN-MCTS†	74.93	2.70	6.5m	-	-	-	-	-	-	-	-	-
SOFTDIST†	74.03	1.40	1m	-	-	-	-	-	-	-	-	-
H-TSP	78.45	7.52	4.8s	110.7	7.18	10.4s		OOM			OOM	
GLOP	75.42	3.37	3.5s	106.7	3.31	7.9s	168.2	2.51	12.1s	237.99	1.66	2.5m
DualOpt	**72.62**	**-0.47**	33.9s	**102.9**	**-0.37**	1m	**162.81**	**-0.77**	6.5m	**230.83**	**-1.40**	8.6m

Table 2: Comparative results with 10 leading algorithms on large-scale *TSP_random* with up to 100K nodes.

Instance	LKH3 Obj.	LKH3 Time	H-TSP Obj.	H-TSP Gap(%)	H-TSP Time	GLOP Obj.	GLOP Gap(%)	GLOP Time	DualOpt Obj.	DualOpt Gap(%)	DualOpt Time
rl5915	572085	9.9m	652336	14.03	9s	628928	9.94	16s	**579152**	1.24	14s
rl5934	559712	9.9m	642214	14.74	9s	616629	10.17	16s	**565912**	1.11	19s
pla7397	23382264	12.4m	25494130	9.03	12s	24990688	6.88	16s	**23536550**	0.66	41s
rl11849	929001	18.9m	1046963	11.2	17s	1006378	8.3	17s	**935904**	0.74	49s
usa13509	20133724	22.6m	21923532	8.89	20s	21023604	4.42	17s	**20248476**	0.57	1.3m
brd14051	474149	23.5m	506211	6.76	20s	491735	3.71	18s	**474559**	0.09	1m
d15112	1588550	25.2m	1696577	6.80	22s	1648777	3.79	18s	**1589033**	0.03	1.2m
d18512	652911	31m	694116	6.31	26s	676840	3.66	21s	**652457**	0.07	1.3m
pla33810	67084217	56.4m		OOM		71934504	7.23	33s	**68083048**	1.49	1.2m
pla85900	144004484	6.5h		OOM		146039168	1.41	1.7m	**145008800**	0.70	3.5m

Table 3: Comparative results on 10 largest instances of *TSPLIB*.

Instance	DualOpt Obj.	DualOpt Gap(%)	DualOpt Time	DualOpt$_{w/oPath}$ Obj.	DualOpt$_{w/oPath}$ Gap(%)	DualOpt$_{w/oPath}$ Time	DualOpt$_{w/oGrid}$ Obj.	DualOpt$_{w/oGrid}$ Gap(%)	DualOpt$_{w/oGrid}$ Time
TSP1K	23.31	0.00	5.6s	23.34	0.13	4.5s	24.56	5.36	5.2s
TSP2K	32.72	-0.52	7.6s	32.75	-0.43	6.3s	34.3	4.29	5.3s
TSP5K	51.56	0.08	13.9s	51.65	0.25	12.1s	54.03	4.87	7.2s
TSP10K	72.62	-0.47	33.9s	72.85	-0.15	31.7s	76.25	4.51	8.2s
TSP20K	102.9	-0.37	1m	103.4	0.12	59s	107.76	4.34	8.7s
TSP50K	162.81	-0.77	6.5m	164	-0.05	6.4m	170.24	3.75	31.3s
TSP100K	230.83	-1.40	8.6m	234.2	0.04	8.5m	240.82	2.87	2.5m

Table 4: An ablation study of DualOpt and its two variants on *TSP_random*

.

over, DualOpt is able to generalize across different TSP distributions, making it a versatile solver for tackling diverse real-world TSP applications. The success of DualOpt paves the way for future research into more sophisticated divide-and-conquer strategies and their application to different types of other combinatorial optimization problems.

Acknowledgements

This work is supported by National Natural Science Foundation 62472189.

References

Accorsi, L.; and Vigo, D. 2021. A fast and scalable heuristic for the solution of large-scale capacitated vehicle routing problems. *Transportation Science*, 55(4): 832–856.

Applegate, D. L.; Bixby, R. E.; Chvátal, V.; Cook, W.; Espinoza, D. G.; Goycoolea, M.; and Helsgaun, K. 2009. Certification of an optimal TSP tour through 85,900 cities. *Operations Research Letters*, 37(1): 11–15.

Chen, X.; Li, Y.; Yang, Y.; Zhang, L.; Li, S.; and Pan, G. 2023. Extnco: A Fine-Grained Divide-and-Conquer Approach for Extending Nco to Solve Large-Scale Traveling Salesman Problem. *Available at SSRN 4679437*.

Chen, X.; and Tian, Y. 2019. Learning to perform local rewriting for combinatorial optimization. *Advances in neural information processing systems*, 32.

Cheng, H.; Zheng, H.; Cong, Y.; Jiang, W.; and Pu, S. 2023. Select and optimize: Learning to solve large-scale tsp instances. In *International Conference on Artificial Intelligence and Statistics*, 1219–1231. PMLR.

Cook, W. J.; Applegate, D. L.; Bixby, R. E.; and Chvatal, V. 2011. *The traveling salesman problem: a computational study*. Princeton university press.

Fu, Z.-H.; Qiu, K.-B.; and Zha, H. 2021. Generalize a small pre-trained model to arbitrarily large tsp instances. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, 7474–7482.

Gutin, G.; and Punnen, A. P. 2006. *The traveling salesman problem and its variations*, volume 12. Springer Science & Business Media.

Hacizade, U.; and Kaya, I. 2018. Ga based traveling salesman problem solution and its application to transport routes optimization. *IFAC-PapersOnLine*, 51(30): 620–625.

Helsgaun, K. 2017. An extension of the Lin-Kernighan-Helsgaun TSP solver for constrained traveling salesman and vehicle routing problems. *Roskilde: Roskilde University*, 12.

Hou, Q.; Yang, J.; Su, Y.; Wang, X.; and Deng, Y. 2023. Generalize learned heuristics to solve large-scale vehicle routing problems in real-time. In *The Eleventh International Conference on Learning Representations*.

Jin, Y.; Ding, Y.; Pan, X.; He, K.; Zhao, L.; Qin, T.; Song, L.; and Bian, J. 2023. Pointerformer: Deep reinforced multi-pointer transformer for the traveling salesman problem. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, 8132–8140.

Joshi, C. K.; Cappart, Q.; Rousseau, L.-M.; and Laurent, T. 2022. Learning the travelling salesperson problem requires rethinking generalization. *Constraints*, 27(1-2): 70–98.

Kim, M.; Park, J.; et al. 2021. Learning collaborative policies to solve NP-hard routing problems. *Advances in Neural Information Processing Systems*, 34: 10418–10430.

Kool, W.; van Hoof, H.; and Welling, M. 2019. Attention, Learn to Solve Routing Problems! In *International Conference on Learning Representations*.

Kwon, Y.-D.; Choo, J.; Kim, B.; Yoon, I.; Gwon, Y.; and Min, S. 2020. Pomo: Policy optimization with multiple optima for reinforcement learning. *Advances in Neural Information Processing Systems*, 33: 21188–21198.

Luo, F.; Lin, X.; Wang, Z.; Xialiang, T.; Yuan, M.; and Zhang, Q. 2024. Self-Improved Learning for Scalable Neural Combinatorial Optimization. *arXiv preprint arXiv:2403.19561*.

Madani, A.; Batta, R.; and Karwan, M. 2021. The balancing traveling salesman problem: application to warehouse order picking. *Top*, 29: 442–469.

Matai, R.; Singh, S. P.; and Mittal, M. L. 2010. Traveling salesman problem: an overview of applications, formulations, and solution approaches. *Traveling salesman problem, theory and applications*, 1(1): 1–25.

Pan, X.; Jin, Y.; Ding, Y.; Feng, M.; Zhao, L.; Song, L.; and Bian, J. 2023. H-tsp: Hierarchically solving the large-scale traveling salesman problem. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, 9345–9353.

Qiu, R.; Sun, Z.; and Yang, Y. 2022. Dimes: A differentiable meta solver for combinatorial optimization problems. *Advances in Neural Information Processing Systems*, 35: 25531–25546.

Reinelt, G. 1991. TSPLIB—A traveling salesman problem library. *ORSA journal on computing*, 3(4): 376–384.

Sun, Z.; and Yang, Y. 2023. Difusco: Graph-based diffusion solvers for combinatorial optimization. *Advances in Neural Information Processing Systems*, 36: 3706–3731.

Toth, P.; and Vigo, D. 2002. *The vehicle routing problem*. SIAM.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Wiering, M. A.; and Van Otterlo, M. 2012. Reinforcement learning. *Adaptation, learning, and optimization*, 12(3): 729.

Xia, X., Yifan Xia abd Yang; Liu, Z.; Liu, Z.; Song, L.; and Bian, J. 2024. Position: Rethinking Post-Hoc Search-Based Neural Approaches for Solving Large-Scale Traveling Salesman Problems. In *Proceedings of the 41st International Conference on Machine Learning*, 54178–54190.

Xin, L.; Song, W.; Cao, Z.; and Zhang, J. 2021. Neurolkh: Combining deep learning model with lin-kernighan-helsgaun heuristic for solving the traveling salesman problem. *Advances in Neural Information Processing Systems*, 34: 7472–7483.

Ye, H.; Wang, J.; Cao, Z.; Liang, H.; and Li, Y. 2024a. DeepACO: Neural-enhanced Ant Systems for Combinatorial Optimization. *Advances in Neural Information Processing Systems*, 36.

Ye, H.; Wang, J.; Liang, H.; Cao, Z.; Li, Y.; and Li, F. 2024b. Glop: Learning global partition and local construction for solving large-scale routing problems in real-time. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, 20284–20292.

Zheng, Z.; Zhou, C.; Xialiang, T.; Yuan, M.; and Wang, Z. 2024. UDC: A Unified Neural Divide-and-Conquer Framework for Large-Scale Combinatorial Optimization Problems. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.