IEEE Access
Multidisciplinary : Rapid Review : Open Access Journal

Received 2 June 2026, accepted 30 June 2026, date of publication 13 July 2026, date of current version 4 August 2026.

Digital Object Identifier 10.1109/ACCESS.2026.3712495

RESEARCH ARTICLE

How to Initialize Euclidean TSP for Local Search and Does It Matter?

JIMI TUONONEN AND PASI FRÄNTI, (Senior Member, IEEE)
School of Computing, University of Eastern Finland, 80101 Joensuu, Finland

Corresponding author: Jimi Tuononen (jimituo@cs.uef.fi)

This work was supported by the University of Eastern Finland through the Doctoral Programme in Science, Technology and Computing (SCITECO).

ABSTRACT *Lin–Kernighan–Helsgaun* (LKH) heuristic is regarded as the state-of-the-art solution for large-scale traveling salesman problems. Its key components are the k-opt heuristic and the neighborhood graph to speed up the local search. Less attention has been paid to the initialization and its effect on optimization. In principle, any heuristic could be used. However, the problem is that even simple greedy algorithms have quadratic time complexity. An obvious idea is to utilize the neighborhood graph, similar to what is done in the optimization stage. In this paper, we compare candidate graphs and initialization algorithms from the literature and methods implemented (but not documented) in the LKH package. We aim to answer through benchmarking which candidate graphs and initialization methods work well, and how much the choices matter for local search optimization.

INDEX TERMS Euclidean TSP, traveling salesman problem, candidate graph, heuristic, initialization, optimization, local search.

I. INTRODUCTION

The Traveling Salesman Problem (TSP) is one of the most fundamental NP-hard problems in combinatorial optimization. In its most general form, TSP is defined as a weighted graph $G = (V, E)$, where the objective is to find a minimum-cost Hamiltonian cycle that visits each vertex exactly once and returns to the starting point. Because the edge weights may be arbitrary, the general TSP captures a wide range of routing and sequencing applications, but it is also difficult to approximate and solve optimally for large instances.

A more restricted variant is the metric TSP (mTSP), in which the edge weights satisfy the triangle inequality. The mTSP admits stronger approximation guarantees than the general problem, most notably the Christofides algorithm [1], [2], which achieves a 3/2-approximation bound under the triangle inequality.

An even further restriction on the problem, called the eTSP, states that the underlying graph is complete and the points are located in a d-dimensional plane. The distances are measured using the Euclidean norm. This inevitably leads to eTSP being a special case of mTSP. The 2-dimensional eTSP is the variant considered in this paper.

While the general TSP and mTSP lack geometrical structure, the approximation of eTSP is made substantially easier through the abuse of these structures. Discoveries by Arora [3] and, coincidentally, Mitchell [4] proved that a polynomial-time approximation scheme (PTAS) exists for eTSP, i.e., that a $(1+1/c)$-approximation for eTSP is possible in $\mathcal{O}(n(\log n)^{\mathcal{O}(c)})$ time. This means that, while being an NP-hard problem, the approximability of eTSP is much more achievable.

The bounds and dimensionality requirements have been improved in many subsequent works. However, these results are highly theoretical and have not been deployed successfully in practice. Arora's PTAS was attempted in [5], but the theoretical advantages were lost to implementation details. Even in the original work by Arora [3], it is stated that on random instances, which is a frequent testbed for experimental work on the TSP, the algorithm will probably not perform better than Karp's algorithm [6].

Besides the more theoretical methods, exact solution methods also exist. A brute force approach takes $\mathcal{O}(n!)$ time by enumeration of all possible paths. This was improved to $\theta(n^2 2^n)$ in [7] and [8] by a dynamic programming approach.

The associate editor coordinating the review of this manuscript and approving it for publication was Dongxiao Yu.

However, these approaches are ineffective for any problem besides those of a trivial size. The state-of-the-art exact solver for symmetric TSP is Concorde [9], [10], which uses a branch-and-cut framework to compute optimal tours. As an exact solver, Concorde is highly effective, as it is capable of solving medium-sized instances relatively fast. However, in larger instances such as the pla85900 (85,900 cities), the inefficiency is apparent. It was estimated that the solution took approximately 136 CPU years of computing time [9]. An approximation of $0.21 \times 1.24^{\sqrt{n}}$ for Concorde's runtime was made in [11].

The final type of solvers are heuristics, which are practical methods for obtaining high-quality solutions without approximation guarantees. State-of-the-art approaches for large Euclidean TSP instances rely on heuristics and local search methods, which typically produce near-optimal solutions within reasonable computation times. The most famous method is the Lin–Kernighan algorithm, proposed in 1973, which is based on edge exchanges and can be viewed as a variable-k extension of k-opt local search [12]. In a standard k-opt move, k edges of the tour are replaced with k new edges to obtain a shorter tour. Lin–Kernighan generalizes this idea by adaptively choosing the number of exchanged edges during the search.

The Lin–Kernighan algorithm was later refined by Helsgaun into the LKH solver [13] (report available in [14]). Among other improvements, LKH introduced a 5-opt-based move structure. Helsgaun further improved the method in LKH-2 by introducing general k-opt submoves and other refinements to the search procedure [15] (also in the report [16]). All further references to LKH refer to this implementation, which is available online [17].

Another recent implementation of local search with k-opt moves, called Sys2to6, was presented in [18]. However, the approach appears to be slightly less effective than LKH. The paper reports that the method outperforms LKH on clustered instances, but this advantage does not hold under the parameter settings and LKH configuration used in this paper. The same instances are optimized here with better outcomes for LKH in all instances, suggesting that the comparison was affected by the chosen parameter settings (see Sec. V-D for results).

The speedup of local search is a key aspect of making the heuristic efficient. In the simplest implementation of k-opt moves, n^k different combinations of edges have to be considered, making the approach unsuitable for large datasets [14]. LKH and other solutions have paid attention to this, and candidate graphs are used to limit the search. The candidate set (set of neighboring nodes) for each point is constructed using a suitable approach to decide their chances of belonging to the solution. This way, the heuristic can be focused on finding edge exchanges in these localized neighborhoods. A visual example of performing k-opt on random edges and the problem with it is presented in Fig. 1.

Local search starts with an initial solution, making it another design choice in these heuristic algorithms. We next recall existing traditional initialization heuristics. Perhaps the most significant of them, the nearest neighbor heuristic, has a straightforward idea. Starting from a random point, we select the nearest unvisited point and continue the tour there. The process is repeated until there are no unvisited points. This heuristic and its closely related variant, DENN (double-ended nearest neighbor), were analyzed, for example, in [19]. Many other heuristics, such as insertion heuristics (nearest, furthest, cheapest, random), space-filling curves, greedy, double minimum spanning tree, and many more, have been introduced in the literature [9], [20], [21].

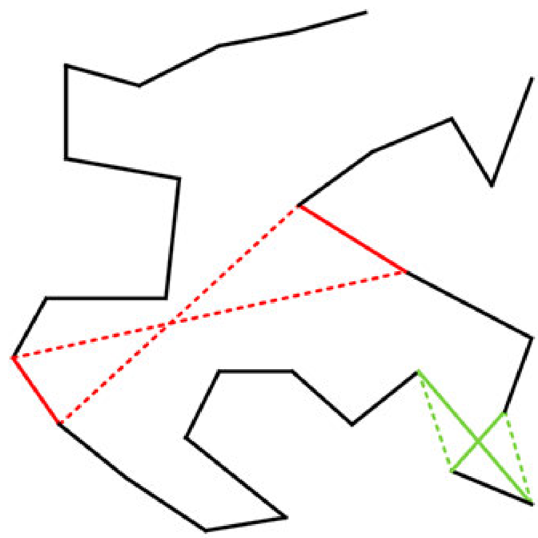

FIGURE 1. Breaking random far-away pairs of edges is rarely good (red). Most improvements are found by making changes in small, localized neighborhoods and breaking non-candidate edges (green).

The problem with the traditional approaches is their often-occurring $\mathcal{O}(n^2)$ time complexity. In addition to optimization, the candidate graph can also be used for constructing the initial tour. In LKH, this has been considered, and different variants using candidate graphs have been implemented. Different approaches can be used when utilizing the graph for initialization. These design choices and another fast graph-based variant were analyzed in a recent study [22]. Another graph-based variant called GrebCap was introduced in [18]. A promising fast non-graph-based variant, the Pair-center algorithm, was developed in [23].

There are different variants for the initialization step of local search, but the research on their effectiveness as part of the optimization is limited. One publication states that local search is typically improved by applying a heuristic instead of random initialization [24]. The outcome was that the final tour was generally better when the heuristic initialization algorithm was used. However, it was noted that the random starting point gave the best solutions when very long computing times were allowed through repeated local searches. It is still an open question whether there are differences when many initialization methods are compared, which is addressed by this paper.

Large-scale TSP requires fast algorithms that operate in sub-quadratic time. A recent publication studied this in the context of the so-called Santa Claus challenge, where an instance of 1.4M points was given to solve within an hour time limit [25]. Design choices were analyzed, and the LKH approach was found to be the most promising. Inspired by that, this paper takes the work further by testing many combinations of candidate graphs and initialization algorithms. These are mostly implemented within LKH, but a few additional initialization options are considered. LKH's optimization procedure has been documented in detail, but these components and their effectiveness have not been shown in practice.

Besides the traditional methods, which are the focus of this paper, deep learning methods have also been attempted in the context of TSP, for example, in [26]. However, that work states that the approach still falls short of standard Operations Research solvers. More notably, [27] introduced NeuroLKH, where the effective LKH search strategy is retained, but the candidate set generation and node penalty estimation are handled using a deep learning approach. This leads to a highly effective hybrid method. However, generalizability remains an important question. The largest TSP instances tested in [27] contain 5,000 points, and it is still unclear how well the approach would work on substantially larger instances, such as the multi-million-point datasets considered in this paper. Additionally, the LKH baseline settings used in that work rely on the alpha candidate set, which, based on the results of this paper, may not be the best design choice for Euclidean instances.

The purpose of this paper is to analyze the role of different candidate graphs and initialization algorithms in local search for the Euclidean TSP. The main contribution is to provide insight through extensive benchmarking, and the paper provides a practical guide for efficient local search optimization. The results show that the choice of candidate graph has a moderate impact on both graph-based initialization and the final tours obtained with LKH. In contrast, the choice of initial solution has only limited significance, as different starting solutions often lead to nearly identical final tour lengths. In practice, shorter initial tours are expected to improve optimization efficiency, since fewer edge exchange moves should be needed to reach a high-quality solution. The results are also analyzed from this perspective.

The paper is organized as follows. Section II describes different candidate graphs, and different initialization algorithms are considered in Sec. III. Section IV summarizes the parameters that are given to LKH for optimization, and Sec. V analyzes the impact of using different candidate sets and initialization algorithms in LKH optimization. Section VI concludes the findings of the experiments and summarizes which approaches work effectively.

II. CANDIDATE GRAPHS

Candidate graphs are a way to limit the search space to consider edges associated with each node instead of all edges between all points. The variants in LKH are based on the Delaunay graph [28], quadrants [29], and their combination. Two variants, named Delaunay and Delaunay Pure, are presented in the implementation, utilizing the Delaunay graph. LKH also implements nearest neighbor, alpha [13] (based on alpha-nearness), and Partial Optimization Metaheuristic Under Special Intensification Conditions (POPMUSIC) [30] candidate graphs. This means that we are analyzing a total of seven LKH graphs. It is important to note that, excluding alpha and POPMUSIC, the other candidate graphs can only be used for instances given by coordinates, i.e., they are mapped to some space. The two mentioned candidate sets are applicable in general, as noted by the author of LKH.

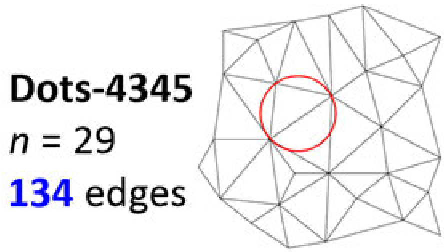

FIGURE 2. An example of the Delaunay graph for a Mopsi dots instance. Edges are formed if any three points form a circle where no other point lies. Each of the edges can be traveled in both directions.

The Delaunay graph is based on the Voronoi diagram, which divides the space by drawing borders between the points so that it corresponds to the nearest neighbor mapping of every space point to its nearest reference point (node). Each node conquers its region in this space. Connecting neighboring nodes creates a so-called Delaunay triangulation, and the graph created from these connections is called the Delaunay graph. While the size of the graph can become overwhelming in higher-dimensional spaces, it can be constructed in $\mathcal{O}(n \log n)$ time in 2-D space. An example of a Delaunay triangulation is given in Fig. 2. The Delaunay averages six candidates per point on bigger datasets [18]. Small instances have a higher proportion of points along the edges where fewer connections are made.

The candidate graphs generated by LKH are given in Fig. 3. Note that by default, the LKH sets are considered as non-symmetric graphs. The standard Delaunay graph is, by definition, a symmetric graph. However, LKH prunes the edges with its maximum candidates setting, which is set to five by default. The difference between the two Delaunay graphs in LKH is that for Delaunay Pure, the optional suffix pure was used. It is noted in the user guide that the suffix should force the candidate graph generation to only use Delaunay edges. However, both of the LKH variants are augmented with additional edges when compared to the

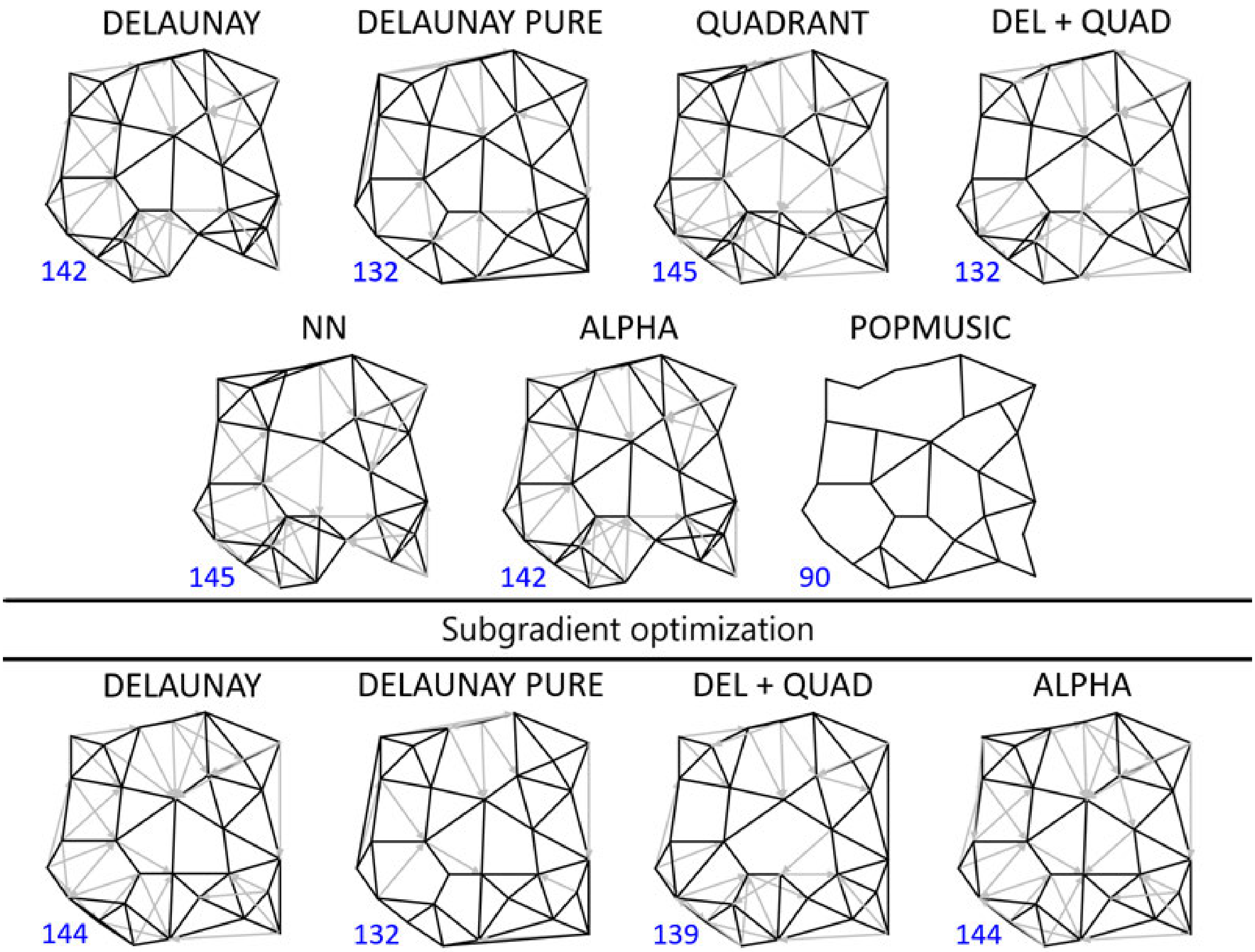

FIGURE 3. Different candidate graphs generated by LKH. The maximum number of candidates was specified to be five (default value). Subgradient optimization can be used to tune the candidate set. Blue numbers indicate the total number of edges. The subgradient optimization applies to POPMUSIC but does not change the structure, only the costs of the edges.

Delaunay in Fig. 2. Presumably, this is done when the number of edges for the given node is under some threshold.

Johnson has suggested using quadrant-based neighbors [29]. The set is generated by selecting the least costly neighbor edges from each of the geometric quadrants around the node. This equates to a maximum of four edges in 2-D space, where each edge is taken from the quadrants given by pairs (x, y), (x, -y), (-x, -y), and (-x, y). The documentation of LKH states that if this does not grant enough edges, the candidate set is augmented by the nearest remaining nodes. This means that at least one of the edges is always augmented this way, given the requirement of five candidates. The quadrants have been found useful in clustered instances [15].

The Delaunay and Quadrant combination graph is a straightforward one. The candidate set of Delaunay is augmented with the Quadrant graph by a simple union. This way, the strengths of both graphs can be combined. Delaunay offers a consistent structure, while quadrants provide additional connections in clustered instances. Helsgaun's submission in [25] considered a combination of a maximum of five Delaunay and four quadrant neighbors. In that work, it was noticed that the application of the complete Delaunay graph outperformed the combination by a slight margin. However, as suggested by the author via personal communication and confirmed by testing, a combination of four Delaunay and quadrant neighbors slightly outperforms both variants, as seen in the results. This variant is used by default throughout the results when the graph is not explicitly mentioned.

The alpha candidate set based on a measure called alpha-nearness was advocated by Helsgaun [13]. The alpha-nearness value measures the increase in cost when a minimum 1-tree (a minimum spanning tree with one additional edge, which creates a cycle) is forced to contain an additional edge. This means that if the edge is part of the minimum 1-tree, it is given an alpha-nearness value of zero. The measure was shown to be effective when compared to the nearest neighbor candidates. The alpha measure was made even more effective by applying a transformation to the original cost matrix with subgradient optimization [31]. The drawback with the alpha candidate set is the $\mathcal{O}(n^2)$ time complexity of the method, making it unusable with large datasets.

The subgradient optimization part can also be applied to candidate sets that are compatible with the minimum 1-tree structure. The method computes node penalties, which modify the edge costs used in the minimum 1-tree calculations. Since this measure is based on minimum 1-trees, it does not apply to the Quadrant and nearest neighbor candidate sets, since they do not preserve the required minimum 1-tree structure. The Delaunay graph, on the other hand, can be used because it contains the Euclidean minimum spanning tree. The determination of the penalties is called an ascent and is controlled by the parameter *INITIAL_PERIOD*. The default value is $n/2$, where n is the number of points, but this becomes time-consuming for large datasets. For example, for 1.4M points, ascent with *INITIAL_PERIOD* set to 100 takes approximately eight minutes.

The POPMUSIC candidate set is based on work in [32], where a meta-heuristic was developed to tackle various large combinatorial problems. Later, the approach was specifically adapted to TSP as a method for candidate set generation [30] and as a general heuristic solution [33]. The basic idea is to optimize sub-parts of solutions that form the complete path. This is done by randomly sampling a small subset of points (seed points, default = 10) and optimizing a route through them. The remaining points are then associated with each of the nearest seed points. This is done recursively until the loosely clustered groups reach a small size and can be optimized entirely. Finally, the complete tour is improved by optimizing sub-paths of consecutive points. The candidate set is formed by generating many POPMUSIC solutions (default = 40) and adding the edges that belong to the solutions to the candidate set. The candidate set is then trimmed to include a maximum of five candidates for each point. The process is generally fast with an empirical complexity lower than quadratic, and it can be applied to any type of data, making it a suitable approach to generating candidate sets.

Finally, the k-nearest-neighbor (KNN) candidate set consists of the k least costly neighbor edges, which can be calculated efficiently using the k-d tree data structure [34].

An overview of the number of edges generated for all candidate sets in a large-scale instance is given in Table 1. The Delaunay variants would normally form approximately six candidates per point [18], but they have nearly five edges instead because of the limitation to five candidates. The addition of quadrant neighbors provides a small number of additional edges. Quadrant and KNN graphs form exactly five edges per point. The POPMUSIC candidate set is slightly less dense with 4.76 edges per point.

Some other considerations for candidate graphs could be the Gabriel graph [35] and its heuristic variant, the XNN graph [36]. The Gabriel graph is a subgraph of Delaunay [37], making it possibly too sparse for 2-D instances, but it would also make it a better option in higher-dimensional spaces. Regarding the XNN graph, it was shown in [36] that 97% of the links in the optimal TSP path are included in the XNN graph. However, this was only proven for five instances of the TSPLIB dataset with a modest number of points, so additional analysis is needed for its suitability.

III. INITIALIZATION ALGORITHMS

Most traditional initialization methods are slow, requiring at least $\mathcal{O}(n^2)$ of work. Candidate graphs can be used during initialization as well to make faster methods. Initialization with graphs requires multiple design choices that affect the algorithm's functionality. An overview of the choices is given in [22]. The choices are divided into four different categories: selection strategy, fragments (yes or no), choice of graph, and, in some cases, the dead-end strategy.

TABLE 1. The number of candidate edges in LKH graph variants in the Santa instance (n = 1,437,195). Edges* column shows the effect of subgradient optimization (initial period = 100).

Candidate graph	Edges		Edges*	
	Total	Per node	Total	Per node
Delaunay	8,619,818	6.00	n/a	n/a
Delaunay LKH	7,086,635	4.93	7,164,826	4.99
Delaunay Pure	6,960,335	4.84	6,960,335	4.84
Delaunay + Quadrant	7,332,180	5.10	7,407,064	5.15
Quadrant	7,185,975	5.00	n/a	n/a
KNN	7,185,975	5.00	n/a	n/a
POPMUSIC	6,843,858	4.76	6,843,858	4.76

Selection strategy refers to the way the edges are chosen during the search. In the traditional nearest neighbor approach, this is simple: the next edge to use is the nearest neighbor of the current node. The second choice is the use of fragments. Some approaches make use of multiple fragments to build multiple separate paths simultaneously [21]. The third choice is the selected graph, which affects the edges that are used during the process. Finally, when using the multiple fragments approach and depending on the approach, there might be a case where the approach has come to a halt, but we still have unconnected fragments (dead-end). The fragments can be connected in multiple ways. These approaches are called nearest neighbor, re-graph, and full search [22]. There is also another potential step called fragment merging, where small fragments are connected to bigger fragments before the dead-end strategy is used.

Subsection III-A describes initialization algorithms based on candidate graphs. Subsection III-B presents fast TSP initialization methods that do not use candidate graphs. Most of these methods are evaluated in the experiments.

A. INITIALIZATION ALGORITHMS WITH CANDIDATE GRAPHS

The problem with many of the traditional approaches is that they require at least $\mathcal{O}(n^2)$ of work. This is acceptable for datasets up to a medium size, but becomes useless for large-scale datasets. One way to address this problem is to use a candidate graph. The search space can be limited to only considering the edges in the candidate graph (neighborhood graph) instead of all points. K-d trees have also been suggested to speed up nearest-neighbor searches for some of the traditional algorithms [21]. In this work, we focus on different graph-based approaches, especially those that are part of the LKH package [13]. The rest of this section explains all the variants implemented in LKH. They have slight variations in comparison with those described in the literature, but retain the same names. One of the key differences is that the nearest neighbor calculations are always done using the candidate graph.

The greedy implementation in LKH is explained in the following. In the first step, the nearest neighbor for each node is calculated. Each node and the corresponding shortest edge are placed into a min-heap structure. The nodes in the heap are then processed in ascending order. When a node is popped from the heap, it is added to the tour via the shortest edge stored. If the node has a degree less than two, a new nearest neighbor is found, and the node is reinserted into the heap. The nearest neighbor is calculated by doing a breadth-first search on the candidate graph if immediate neighbors are not available.

The nearest neighbor in LKH works like the traditional nearest neighbor approach, but it uses the candidate graph and adds randomization. The method starts from a random node and then continues to the nearest node until all nodes have been visited. The nearest neighbor calculation is done again by doing a breadth-first search on the candidate graph. The randomization is added by using the most favorable edge only 2/3 of the time. This provides a variation of tours, which is favorable for the restart of local search. However, the path length will also be longer because the solution will include unfavorable edges during the building process.

The Boruvka approach is inspired by the old MST algorithm [38], which was then adapted to TSP by [9]. Similar to the greedy approach, the first step includes calculating the nearest neighbor for each node. Instead of using a heap for the processing, the edges are simply sorted as a list and processed in that order. The edges are used to build fragments as long as we are not creating branches or cycles. The key difference to greedy is that the list is not updated, resulting in unconnected fragments. The process is repeated until all the fragments are connected. The new round includes only the nodes that have a degree of less than two.

Quick-Boruvka (Q-Boruvka) is reported somewhat differently in different sources. In [20], the method is explained to have an arbitrary fixed order, while the author states that the nodes are processed in the order of their first coordinate [39]. Nevertheless, the LKH implementation starts by sorting the nodes by their x-axis. This is the key difference of the method: Boruvka sorts the nodes by the distance of the nearest neighbor link, and Q-Boruvka sorts them by the x-axis. Additionally, in Q-Boruvka, the same node is processed immediately again if the node's degree is not two. Finally, in Boruvka, the approach is repeated until we have one complete path. It is stated that two passes through the points are enough to obtain a full path with Q-Boruvka [20]. The LKH implementation for Q-Boruvka does not use repeats but instead relies on the dead-end strategy (see the start of the section) to connect the fragments, which is the same as the greedy approach. The nearest neighbor search is done from the list of all available fragment endpoints in this case.

In [13], Helsgaun introduced a pseudo-random initialization method. The publication does not give it a name. However, the code package refers to it as WALK, which is the terminology we will be using. Helsgaun's idea behind the algorithm was that the approach should be randomized to have diverse initial solutions. This way, the optimization process should find different local optima between subsequent runs. The method is a randomized nearest-neighbor approach with a deterministic element. The candidate edges are chosen if they are usable in the neighborhood. The following pseudocode explains the approach:

Walk:
Choose a random node i.
Repeat
  Choose an unused node i:
  If possible, choose j such that:
    (i, j) is a candidate edge,
    $a(i, j) = 0$, and
    (i, j) belongs to the current best tour.
Otherwise, if possible, choose j such that
(i, j) is a candidate edge.
Otherwise, choose j among the nodes
not already chosen.
Until all nodes have been chosen.

The final graph-based method is from recent literature called Delaunay greedy FM [22], which is another type of greedy tour generator. The work was inspired by [40], where two simple variants of tour generation were experimented with by using the Delaunay graph.

The approach has three steps. The first step is to process all the edges of the candidate set in ascending order. The path generation allows the use of fragments, and each edge is used to extend them if the connection rules allow it. The first step will result in many separated fragments. The second step is to connect small fragments to nearby fragments (fragment merging), as this was noticed to both make the process slightly faster and to provide shorter tours. The final step is to connect the ends of fragments to form a single tour.

B. OTHER INITIALIZATION ALGORITHMS

In addition to the graph-based initialization methods, we next consider a few other fast initialization methods from the literature. Space-filling curves have been considered in [41]. Moore [42] and Sierpiński [43] curve variants are implemented in the LKH software. The idea of space-filling curves is to divide the space into four quadrants, which are recursively divided further until every node has its own cell. The cells are indexed, and the space-filling curve travels through the cells according to the order created by this index. The TSP path is obtained by sorting the nodes according to their index (order in this curve), which requires only $\mathcal{O}(n \log n)$ time.

POPMUSIC initial tour is a byproduct of generating the POPMUSIC candidate graph. The candidate graph is built by generating many POPMUSIC TSP solutions (*default* = 40). One option for the initialization is to use the best tour generated by this method. The POPMUSIC implementation in the LKH package provides tours that are generally 10-20% from the optimum [30]. The results here show that on smaller datasets it can be lower, but can go as high as 35% on the large Santa instance. The POPMUSIC initial tour is always combined with the use of the POPMUSIC candidate set in the results.

Another promising method we consider is the Pair-center method by Formella [23]. The basic idea of the algorithm is a two-step process. In the first step, the clustering phase, a binary tree is created by repeatedly merging the closest pair of points and replacing them with their center point. This process is continued until only a single center point remains, which forms the root of the tree. The original points are located at the leaves. In the second step, the construction phase builds the tour by traversing the binary tree starting from the root. In each step, a point is replaced by the two points that formed it, slowly building up the complete tour. The algorithm provides excellent tours for initialization standards and is thus included in the analysis. There are different versions of the algorithm depending on the implementation details. We selected the PCk variant as it seems to have the best trade-off between speed and quality.

The last method we consider is a completely random initial solution, i.e., a random tour made by shuffling the order. In the work by Perttunen [24], the random initial solution was compared to a heuristic initial solution. The outcome was that the heuristic initial solution performed better in the majority of cases.

IV. LKH PARAMETER SETTINGS

The LKH package is regarded as the state-of-the-art heuristic optimizer and is therefore used to evaluate the effectiveness of different initial tours. The default settings for LKH are tuned to be suitable for small to medium-scale instances. However, with these settings, it is extremely impractical to solve large-scale instances. There are a few reasons for the reduced efficiency. The default candidate set is defined as the alpha candidate set, which has a quadratic time complexity. Therefore, better performance can be gained by changing the candidate set.

Another limiting factor is the local search performed through k-opt operations, and two important parameters related to this are max trials and max swaps. The max trials setting determines the number of attempts to improve the tour using the Lin-Kernighan search procedure [44]. As such, this parameter controls the length of the optimization process. However, the use of this parameter is avoided in most of the experiments, since trials can take different amounts of time depending on the other settings, such as the candidate set and the initial solution. The max swaps parameter specifies the size of the swap stack, which is related to how the k-opt operations are implemented as a sequence of 2-opt moves, or flips. Therefore, it should be set to a sufficiently large static value so that it does not interfere with the optimization process, while also avoiding unnecessary memory usage. By default, the value is based on the number of points.

The tuning of the max trials parameter can be largely avoided by giving LKH a fixed time limit using the TIME_LIMIT or TOTAL_TIME_LIMIT parameters. The former sets a time limit only for the optimization, whereas the latter also includes candidate set generation and initialization.

To allow a fair comparison between the TSPLIB instances, the following strategy was used. Since trials have varying lengths and different amounts of trials are needed in runs, they cannot be relied on directly for allocating computation time. Instead, TOTAL_TIME_LIMIT was used by limiting the total time to one minute per instance on average and distributing this time across the instances according to a quadratic distribution. This choice is motivated by [14], where the average running time of the algorithm is noted to be approximately $O(n^{2.2})$. The distribution was also constrained so that every instance received at least one second of runtime.

Finally, by default, the LKH package runs the optimization multiple times, given by the "RUNS" parameter (default = 10). This is beneficial as different local minima are likely reached with each run. The parameter was kept at the default for small-scale TSP, as runtime poses no issue in these cases. The value was changed to one for TSPLIB and large-scale TSP instances, and repetition was done manually by running LKH many times with different seeds.

The collection of parameters used in the experiments is given in Table 2. The values for the initial tour algorithm, candidate set type, and initial period are changed throughout the experiments. For the Mopsi dots dataset, the other parameters are kept as default, as the optimization is fast. However, LKH does not allow the solving of open-loop TSP by default, so a special version of LKH was used to gather these results (used in [25]).

For large-scale instances, we consulted the author of the software regarding efficient parameter configurations. The key point here is to set the total time limit to one hour to allow a fair comparison between the methods, as well as setting max swaps to 1000. It is important to note that the non-tabulated results always use the Delaunay and Quadrant combination as the default candidate set with subgradient optimization set to 100. Max candidates and extra candidates are set to four in these instances. Finally, a recombination option named CLARIST was used by recommendation from LKH's author, which applies a crossover operation between trials to find alternative tours. Other options include GPX2 and IPT, which may affect the results [45].

TABLE 2. The main parameters for LKH. The experiments were produced with default values for the Mopsi dots dataset. The results include different combinations of candidate set types, initial periods, and initial tour algorithms. The extra candidate parameter is used when the combination of Delaunay and Quadrant is applied. In these cases, the max candidates is also set to four.

Parameter	Default	Mopsi Dots	TSPLIB	Large-scale TSP
INITIAL_TOUR_ALGORITHM	WALK	varies	varies	varies
CANDIDATE_SET_TYPE	ALPHA	varies	varies	varies
INITIAL_PERIOD	n/2	0 or n/2	0, 100 or n/2	0 or 100
MAX_CANDIDATES	5	4 or 5	4 or 5	4 or 5
EXTRA_CANDIDATES	0	0 or 4	0 or 4	0 or 4
MAX_SWAPS	n/2	n/2	1000	1000
MAX_TRIALS	n/2	n/2	-	1000
TOTAL_TIME_LIMIT	-	-	relative to n	1800 or 3600 s
RUNS	10	10	1	1
SEED	1	1–5	1–5	1–5

TABLE 3. Datasets used in the experiments.

Dataset	Type	Distance	Instances	Sizes	Reference
Mopsi Dots	Open loop	Euclidean	6,449	5–31	[47]
TSPLIB	Closed loop	Integral	78	51–18,512	[48]
VLSI	Closed loop	Integral	3	238,025–744,710	[49]
Santa	Closed loop	Euclidean	1	1,437,195	[25]
World-TSP	Closed loop	GEOM	1	1,904,711	[50]
DIMACS	Closed loop	Integral	3	316,228–3,162,278	[51]

V. RESULTS

The computational results are gathered by applying different initialization algorithms while being paired with different candidate graphs. The main idea is to test the methods on a variety of datasets to observe their behavior on datasets with diverse structures and difficulty levels. Size is one of the main factors in making a dataset more difficult [46].

The details of the used datasets are presented in Table 3. The datasets are ordered from easy to challenging. The Mopsi Dots dataset [47] represents easy-to-solve instances that a good method should handle (nearly) optimally. The classical decades-old Traveling Salesman Problem Library (TSPLIB) provides a variety of instances for medium-scale TSP [48]. This paper uses 78 out of the total 111 instances provided in the library, which corresponds to all 2-D eTSP instances. Finally, Santa, World-TSP, the three biggest instances from VLSI, and the three biggest instances (excluding E10M.0) from the DIMACS challenge represent large-scale TSP, where the optimal solutions are unknown.

Many other datasets can be found in the literature, but these were found suitable for the type of testing done in this paper. For example, one recent work done on initialization employs ten different datasets [23]. The first three of the following subsections will analyze the results for each size category. Section V-D includes a deeper analysis of the effect of initialization methods on the optimized result through repeated runs. Section V-E provides brief structural analysis of initial tours by looking at overlapping edges compared to a reference tour. In Section V-F, the speed of optimization is considered.

A. MOPSI DOTS (SMALL)

The results of applying different initialization methods with candidate graphs for the Mopsi Dots dataset are presented in Table 4. The first half of the table shows the results for only initialization, and the second half shows the results after the optimization using LKH.

Firstly, the time taken is negligible for any result in this table, as each instance is handled in 5 to 10 milliseconds on average. Regarding the initialization, none of the applied graph-based methods provide great results without subgradient optimization. Additionally, many of the graph-based methods provide similar results compared to each other when the same candidate graph is used. The selection of the candidate graph mostly dictates the effectiveness. The utilization of subgradient optimization gives a clear advantage to POPMUSIC and alpha graphs.

Other methods not included in the table are Delaunay greedy FM, Pair-center algorithm (PCk variant), and POPMUSIC initial tour, which achieve gaps of 2.18%, 0.81%, and 0.012%, respectively. POPMUSIC has the advantage here as the method generates dozens of different randomized tours, of which the best is chosen. The instances are small, and one of the generated tours is very likely to be close to optimal, if not optimal.

The results after optimization provide a better view of which candidate graph works most effectively. In these small instances, subgradient optimization surprisingly worsens the final results for several graphs. One possible explanation is that the resulting candidate sets become less diverse, reducing the number of available edges for later k-opt moves. However,

TABLE 4. Results on the Mopsi Dots dataset. Values are percentage gaps compared to the optimal solutions. The first part reports the initialization results, while the second part reports results after optimization. The candidate sets are also considered with and without subgradient optimization.

	Candidate graph						
Algorithm	Delaunay LKH	Delaunay Pure	Delaunay + Quadrant	Alpha	Quadrant	KNN	POPM
Initial solution							
Boruvka	21.23%	21.74%	17.13%	26.23%	20.18%	16.87%	4.30%
Q-Boruvka	35.27%	14.10%	17.19%	50.31%	14.93%	13.68%	3.81%
NN	36.35%	24.58%	26.77%	39.52%	26.18%	27.56%	5.52%
Greedy	18.90%	19.72%	16.48%	34.57%	20.01%	19.97%	4.33%
Walk	48.94%	44.82%	51.74%	63.11%	55.89%	66.95%	10.07%
Subgradient optimization = default							
Boruvka	2.19%	6.53%	1.94%	**0.78%**	n/a	n/a	**0.74%**
Q-Boruvka	2.45%	6.69%	2.59%	**1.36%**	n/a	n/a	**0.77%**
NN	2.36%	6.74%	2.39%	**0.72%**	n/a	n/a	**0.72%**
Greedy	2.04%	6.56%	1.78%	**0.72%**	n/a	n/a	**0.62%**
Walk	11.09%	14.16%	12.17%	**10.36%**	n/a	n/a	**4.04%**
After optimization							
Boruvka	0.179%	0.314%	0.115%	0.232%	0.015%	0.017%	**0.013%**
Q-Boruvka	0.174%	0.342%	0.116%	0.268%	0.015%	0.017%	**0.011%**
NN	0.168%	0.321%	0.115%	0.180%	0.016%	0.017%	**0.013%**
Greedy	0.174%	0.317%	0.115%	0.195%	0.017%	0.017%	**0.012%**
Walk	0.160%	0.268%	0.115%	0.254%	0.015%	0.016%	**0.013%**
Subgradient optimization = default							
Boruvka	0.790%	4.352%	0.440%	**0.013%**	n/a	n/a	**0.012%**
Q-Boruvka	0.787%	4.350%	0.439%	**0.013%**	n/a	n/a	**0.013%**
NN	0.788%	4.354%	0.440%	**0.013%**	n/a	n/a	**0.013%**
Greedy	0.790%	4.355%	0.440%	**0.013%**	n/a	n/a	**0.012%**
Walk	0.786%	4.339%	0.440%	**0.012%**	n/a	n/a	**0.013%**

this effect was not investigated further. The Alpha and POPMUSIC graphs provide the best results. The Quadrant and KNN graphs are also nearly as effective. Nevertheless, subgradient optimization does not apply to them, and they are not as effective on the larger datasets.

B. TSPLIB (MEDIUM)

The collection of different initialization methods with candidate graphs for the TSPLIB dataset is given in two tables. Table 5 shows the results for initialization, and Table 6 shows the results after optimization using the LKH package. The results include subgradient optimization on the graph with initial period = 0 (no optimization), initial period = 100, and the default (initial period = n/2).

The results for the initialization are roughly the same as for the Mopsi Dots dataset, albeit somewhat worse. Most importantly, the subgradient optimization is not as effective as in the smaller instances. It remains effective, and yields results below 12% for the best methods in this table, including Boruvka and greedy. Without subgradient optimization, the best results are around 20%. The difference between the slight application of subgradient optimization (initial period = 100) and the default application is nearly negligible. The variation is a 0.5–1 percentage-point improvement in the best case and a slight degradation in the worst case, depending on the connections formed during the initialization step.

The time taken for the methods given in Table 5 is as follows. The initialization time itself is negligible given each method's sub-quadratic time complexity (100-200 ms per instance on average, including the candidate set generation). Alpha is the only exception because the generation requires more time. With that candidate set, the initialization takes nearly three seconds. Additionally, generating the POPMUSIC candidate set takes slightly over half a second on average.

The optimization results show that, without subgradient optimization, POPMUSIC is the best option for the candidate set, although the difference is only minor compared to the Delaunay Pure variant, Delaunay + Quadrant, and the pure Quadrant candidate set. The same conclusion holds for the case with initial period set to 100, but with an even smaller margin. When using the default value for subgradient optimization, the Delaunay + Quadrant combination provides the best results, although only narrowly compared to POPMUSIC. The alpha-nearness candidate set does not perform as well, despite being proposed candidate set for LKH initially [13].

TABLE 5. Initialization results for the TSPLIB dataset given as gap values. Results are reported for subgradient-optimized candidate sets using values 0, 100, and default ($n/2$).

Algorithm	Delaunay LKH	Delaunay Pure	Delaunay + Quadrant	Alpha	Quadrant	KNN	POPM
	Candidate graph						
Subgradient optimization = 0							
Boruvka	19.97%	20.01%	22.74%	20.78%	20.46%	20.24%	17.40%
Q-Boruvka	23.46%	22.22%	28.08%	23.62%	25.61%	30.55%	21.69%
NN	30.64%	35.67%	41.51%	30.63%	37.32%	31.55%	33.05%
Greedy	18.81%	19.69%	20.86%	19.56%	19.12%	19.55%	18.97%
Walk	124.68%	140.23%	140.54%	118.64%	231.64%	219.30%	134.91%
Subgradient optimization = 100							
Boruvka	11.65%	12.78%	11.90%	11.34%	n/a	n/a	12.43%
Q-Boruvka	15.19%	16.17%	20.24%	15.28%	n/a	n/a	15.92%
NN	17.37%	21.29%	23.78%	16.40%	n/a	n/a	21.07%
Greedy	11.20%	12.64%	11.99%	11.45%	n/a	n/a	12.06%
Walk	96.85%	116.17%	109.24%	94.25%	n/a	n/a	111.19%
Subgradient optimization = default							
Boruvka	11.57%	12.58%	11.47%	12.02%	n/a	n/a	11.81%
Q-Boruvka	15.01%	16.45%	19.10%	16.00%	n/a	n/a	16.23%
NN	16.45%	20.89%	23.29%	16.69%	n/a	n/a	19.98%
Greedy	11.55%	12.85%	11.39%	11.94%	n/a	n/a	11.70%
Walk	94.64%	113.82%	107.76%	93.23%	n/a	n/a	108.23%

TABLE 6. Optimization results for the TSPLIB dataset given as gap values. Optimization time was limited to one minute per instance on average. Results are reported for subgradient-optimized candidate sets using values 0, 100, and default ($n/2$).

Algorithm	Delaunay LKH	Delaunay Pure	Delaunay + Quadrant	Alpha	Quadrant	KNN	POPMUSIC
	Candidate graph						
Subgradient optimization = 0							
Boruvka	0.187%	0.046%	0.050%	0.181%	0.051%	0.650%	0.033%
Q-Boruvka	0.103%	0.038%	0.040%	0.199%	0.054%	0.565%	0.041%
NN	0.135%	0.058%	0.033%	0.220%	0.036%	0.377%	0.040%
Greedy	0.272%	0.041%	0.046%	0.165%	0.039%	0.458%	0.039%
Walk	0.147%	0.040%	0.035%	0.268%	0.046%	0.568%	0.031%
Subgradient optimization = 100							
Boruvka	0.050%	0.044%	0.041%	0.071%	n/a	n/a	0.030%
Q-Boruvka	0.045%	0.036%	0.039%	0.065%	n/a	n/a	0.027%
NN	0.055%	0.029%	0.048%	0.067%	n/a	n/a	0.036%
Greedy	0.048%	0.034%	0.038%	0.043%	n/a	n/a	0.034%
Walk	0.051%	0.029%	0.037%	0.070%	n/a	n/a	0.035%
Subgradient optimization = default							
Boruvka	0.050%	0.029%	0.030%	0.043%	n/a	n/a	0.036%
Q-Boruvka	0.057%	0.039%	0.034%	0.064%	n/a	n/a	0.031%
NN	0.038%	0.039%	0.034%	0.047%	n/a	n/a	0.039%
Greedy	0.047%	0.031%	0.034%	0.044%	n/a	n/a	0.035%
Walk	0.044%	0.036%	0.027%	0.035%	n/a	n/a	0.036%

The experimental setup for TSPLIB instances is explained in Sec. IV. In short, each instance is given one minute on average to allow a fair comparison across all combinations, with the time distributed according to a quadratic function of the instance size. This means that the remaining time available for optimization varies depending on which candidate graph is used. The initialization times are practically equal across initialization methods, and the main differences occur during subgradient optimization, which is the most time-consuming part of the setup process. Subgradient optimization requires, on average, roughly six seconds for the Delaunay Pure and POPMUSIC candidate sets, eight seconds for the Delaunay LKH and Delaunay + Quadrant candidate sets, and 15 seconds for the alpha candidate set. However, this time limit is still substantial on average, and most of the instances are solved to optimality, excluding some of the larger instances in the dataset.

TABLE 7. Initialization results for the Santa instance. Tour length is given in kilometers.

	Candidate graph				
Algorithm	Del Pure	Del + Quad	Quad	KNN	POPM
Subgradient optimization = 0					
Boruvka	139 471	163 132	154 614	132 712	148 403
Q-Boruvka	154 409	197 504	189 082	156 604	176 256
NN	179 228	243 286	231 795	142 676	206 141
Greedy	132 460	150 480	146 858	132 065	139 550
Walk	455 729	513 198	57M	70M	506 758
Subgradient optimization = 100					
Boruvka	128 798	133 029	n/a	n/a	130 399
Q-Boruvka	144 222	167 546	n/a	n/a	157 463
NN	158 508	185 250	n/a	n/a	168 197
Greedy	127 038	129 775	n/a	n/a	127 846
Walk	468 314	521 043	n/a	n/a	525 311

C. SANTA AND WORLD-TSP (LARGE)

The result for initialization in the Santa instance is given in Table 7. The time taken for each entry is not tabulated, but it solely depends on the candidate graph. Del Pure and Quadrant graphs are the fastest to generate, and the resulting initialization is processed in 30 seconds. Delaunay and Quadrant combination takes one minute, the KNN graph about 2 minutes, and POPMUSIC is the slowest, taking around nine minutes. The application of subgradient optimization is also time-consuming, adding approximately eight to nine minutes for each candidate graph, depending on the number of edges.

The quality of the resulting initial solutions varies. The walk approach results in longer paths. For the other methods, Greedy achieves the shortest tours, followed by Boruvka and Q-Boruvka. The other methods provide initial tours of length 139,647 km (Delaunay greedy FM), 112,767 km (Pair-center), and 143,239 km (POPMUSIC). These methods complete the initialization in 17 seconds, one and a half minutes, and nine minutes, respectively. The Pair-center algorithm gives a significantly shorter tour compared to others. The initialization result is only meaningful if it also provides better paths after the optimization step. The result of applying both the initialization and optimization steps for the Santa instance is therefore given in Table 8. Each result includes the time taken for the initialization, and the optimization is applied until a total time of one hour is reached. Notably, all the results are close to each other, except that the KNN candidate graph is considerably worse. For the graph-based initializations, the choice of method seems not to change the outcome, as the results are very close to each other. The candidate graph has a slightly more noticeable impact. All the other results are within a 0.5% gap margin from each other, excluding the KNN candidate graph with almost a 9% gap. Delaunay + Quadrant and POPMUSIC graphs give the best results, below 109,000 kilometers. Subgradient optimization provides only a minor improvement. In general, the improvement is around 0.2% in the gap value. It is also important to note that in these cases, optimization starts approximately eight minutes later.

TABLE 8. Optimization results for the Santa instance. The running time is fixed to one hour for each method. Tour length is given in kilometers.

	Candidate graph				
Algorithm	Del Pure	Del + Quad	Quad	KNN	POPM
Subgradient optimization = 0					
Boruvka	109 523	109 059	109 309	118 551	109 100
Q-Boruvka	109 223	109 026	109 316	119 341	109 075
NN	109 303	109 057	109 228	–	109 030
Greedy	109 454	109 013	109 259	118 461	109 122
Walk	109 977	108 976	109 247	4M	108 956
Subgradient optimization = 100					
Boruvka	109 254	108 906	n/a	n/a	108 883
Q-Boruvka	109 090	108 816	n/a	n/a	108 892
NN	109 425	108 852	n/a	n/a	108 858
Greedy	109 360	108 856	n/a	n/a	108 966
Walk	109 585	108 817	n/a	n/a	108 911

The other analyzed methods, Delaunay greedy FM, Pair-center algorithm (PCk variant), and POPMUSIC initial tour, provide results of 109,024, 108,993, and 108,833 kilometers, respectively, after optimization (initial period = 100). These results are nearly identical to the tabulated results. The best initialization result obtained by the Pair-center algorithm does not achieve the best result after the optimization. The overall structure of the initial tour seems to be more important than the length.

The results for the World-TSP instance are given in Table 9 as millions of kilometers. The initialization results for this instance were omitted because the ranking of the methods was nearly identical to that of the Santa instance. The optimization results are also nearly the same. For optimization, it is challenging to determine which initialization to prefer, as they all yield similar results.

The other analyzed methods, Delaunay greedy FM, Pair-center algorithm (PCk variant), and POPMUSIC initial tour, result in lengths of 7536, 7535, and 7548, respectively. Therefore, the tabulated methods are slightly better after the optimization. The candidate graph selection is clear. The Delaunay and Quadrant combination consistently gives the best result. Although POPMUSIC candidates performed well in other instances, it is not the best choice for the World-TSP. Like in the Santa instance, subgradient optimization is beneficial for World-TSP.

For context, the best-reported tour for the World-TSP instance has a length of 7,515,755,956 kilometers [50]. This means that the best result from Table 9 (7,529,618,750), for example, has a gap of 0.18% to the best-reported tour. It is essential to note that the tours calculated here were limited to an hour of computing time. The record was discovered on February 15, 2021, by Keld Helsgaun, the author of LKH, and has remained very stable over the years. Only an improvement of approximately 200,000 kilometers has been made since 2008. The current best lower bound, found by the Concorde TSP solver, is 7,512,218,268. This bound shows that Helsgaun's record tour has a length of at most 0.047% longer than the length of the optimal tour [50].

TABLE 9. Optimization results for the World-TSP instance. The running time is fixed to one hour for each method. Tour length is given in millions of kilometers.

	Candidate graph				
Algorithm	Del Pure	Del + Quad	Quad	KNN	POPM
Subgradient optimization = 0					
Boruvka	7,542	7,548	7,548	7,669	7,546
Q-Boruvka	7,540	7,544	7,546	7,661	7,555
NN	7,552	7,543	7,553	–	7,552
Greedy	7,547	7,543	7,541	7,667	7,549
Walk	7,560	7,544	7,549	12M	7,552
Subgradient optimization = 100					
Boruvka	7,550	7,535	n/a	n/a	7,537
Q-Boruvka	7,534	**7,531**	n/a	n/a	7,544
NN	7,549	**7,530**	n/a	n/a	7,542
Greedy	7,533	7,533	n/a	n/a	7,543
Walk	7,541	**7,530**	n/a	n/a	7,552

D. IMPACT OF INITIALIZATION ON THE FINAL TOUR

The results obtained with different candidate graphs and initialization algorithms vary, but no single combination is consistently superior across all datasets. To further examine the effect of initialization, the candidate graph was fixed as either the Delaunay + Quadrant combination or POPMUSIC. Five independent runs of initialization followed by optimization were then performed on a range of instances.

The LKH package was run on large instances using the same parameter settings as before, with a total time limit of one hour. Instances with fewer than one million points were limited to 30 minutes of runtime. For the TSPLIB dataset, the computation time was limited to one minute per instance on average, following the setup described in Sec. IV. Based on the results reported by Helsgaun, most TSPLIB instances are solved optimally within this time frame [14].

The initialization results using the Delaunay + Quadrant candidate set are presented in Table 10. Switching to POPMUSIC as the candidate graph during initialization makes only a minor difference. The initial tours are slightly shorter for the methods that make use of the candidate graph, since the POPMUSIC candidate set is slightly sparser and therefore more selective in which edges are included. The additional results using POPMUSIC as the candidate set are provided in Table 16 in the Appendix. The main takeaway is that the different initialization methods produce tours of relatively similar quality across the datasets. In other words, their ranking remains nearly unchanged. Here, the initial tours mainly serve as a baseline for evaluating the effect of the subsequent LKH optimization.

The ranking results after optimization are presented in Tables 11 and 12 for the Delaunay + Quadrant and POPMUSIC candidate sets, respectively. First, the variation between runs is very small, indicating that LKH provides highly stable optimization results. For example, in the Santa instance in Table 11, the largest variation occurs with Boruvka, where the best and worst runs differ by 121 km, corresponding to about 0.1% of the average tour length of 108,874 km. The smallest variation occurs with Q-Boruvka, where the difference between the best and worst runs is only 48 km.

Second, and more importantly, the choice of initialization method has only a limited effect on the optimized result. One could expect shorter initial tours to also lead to shorter final tours. However, the ranking of the methods changes considerably when the initialization and optimization results are compared. The exception is the random initialization, which performs noticeably worse and is treated as an outlier in the following analysis.

The main difference is between the two candidate sets. The Delaunay + Quadrant set performs slightly better overall, with an average gap of 0.18% across all initialization methods, compared to 0.21% for POPMUSIC. The best individual result is also obtained with Delaunay + Quadrant, where Walk reaches a gap of 0.167%. With POPMUSIC, the best result is obtained with Pair-center, reaching a gap of 0.180%. The result for Walk is somewhat surprising, since it produces initial tours that are several times longer than the best-known tour. Additionally, extending the time limit could further favor the Delaunay + Quadrant set, since it is denser than POPMUSIC (Table 1) and is therefore expected to provide more opportunities for improving moves.

Another notable pattern is that the Delaunay + Quadrant set favors the longer initial solutions in general and does not perform best when paired with the shortest Pair-center initial solution. In contrast, for the POPMUSIC set, the situation is the opposite. There, the length of the initial tour is a good indicator of the quality of the final solution, and starting from Pair-center initialization results in the best optimization performance.

The key takeaway is that the shortest initial tour does not necessarily lead to the best final result. Instead, the outcome

TABLE 10. Ranking of initialization methods when Delaunay + Quad is used as the candidate set. Gap values are reported as percentages relative to the best-known tours.

Instance	Random	Walk	Moore	Sierpinski	NN	Q-Boruvka	Del Greedy FM	Boruvka	Greedy	Pair-center
C316k	88.2k	193.84	53.49	50.77	56.64	37.51	26.57	20.43	17.65	3.44
E1M	73.0k	345.38	37.26	34.24	20.93	12.66	13.39	10.41	10.13	4.77
E3M	130.0k	477.21	37.32	34.28	20.57	12.40	13.24	10.29	10.10	4.70
ARA238025	49.1k	328.75	46.01	44.24	28.67	19.61	18.07	13.62	12.80	4.78
LRA498378	89.4k	168.33	46.99	47.01	28.83	22.84	16.52	13.29	12.27	6.06
LRB744710	78.0k	461.74	44.58	43.32	26.74	18.34	17.48	12.82	11.93	5.14
Santa	367.7k	384.33	69.95	66.72	71.53	54.59	28.81	23.23	19.66	4.01
World-TSP	184.1k	538.87	51.31	49.02	37.42	21.14	17.21	14.65	14.93	4.69
Average gap	**132.4k**	**362.30**	**48.37**	**46.20**	**36.41**	**24.89**	**18.91**	**14.84**	**13.68**	**4.70**
Average rank	**10.00**	**9.00**	**7.63**	**6.88**	**6.50**	**4.75**	**4.25**	**3.00**	**2.00**	**1.00**

TABLE 11. Ranking of initialization methods after optimization when Delaunay + Quad is used as the candidate set. Gap values are reported as percentages relative to the best-known tours. *Produced an exceptionally long tour within the time limit; the gap was excluded from the average gap calculation, while rank 10 was used for the average rank calculation.

Instance	Random	Walk	Moore	Sierpinski	NN	Q-Boruvka	Del Greedy FM	Boruvka	Greedy	Pair-center
C316k	0.57	0.29	0.29	0.32	0.31	0.32	0.41	0.30	0.30	0.37
E1M	0.13	0.04	0.04	0.04	0.04	0.04	0.05	0.04	0.04	0.06
E3M	*	0.13	0.12	0.13	0.14	0.14	0.14	0.13	0.13	0.15
ARA238025	0.06	0.04	0.07	0.06	0.05	0.06	0.09	0.05	0.05	0.08
LRA498378	0.39	0.15	0.14	0.14	0.19	0.16	0.18	0.13	0.14	0.16
LRB744710	0.17	0.07	0.09	0.12	0.09	0.08	0.13	0.08	0.09	0.11
Santa	1.11	0.41	0.39	0.38	0.40	0.39	0.56	0.42	0.39	0.56
World-TSP	0.42	0.21	0.22	0.22	0.21	0.20	0.31	0.22	0.23	0.24
Average gap	**0.41**	**0.17**	**0.17**	**0.18**	**0.18**	**0.17**	**0.23**	**0.17**	**0.17**	**0.22**
Average rank	**9.50**	**3.13**	**3.88**	**4.00**	**4.88**	**4.13**	**8.75**	**4.00**	**4.63**	**8.13**

TABLE 12. Ranking of initialization methods after optimization when POPMUSIC is used as the candidate set. Gap values are reported as percentages relative to the best-known tours.

Instance	Random	Walk	Moore	Sierpinski	NN	Q-Boruvka	Del Greedy FM	Boruvka	Greedy	Pair-center
C316k	0.67	0.37	0.39	0.40	0.38	0.40	0.34	0.37	0.36	0.30
E1M	0.12	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04
E3M	1.49	0.15	0.15	0.15	0.14	0.15	0.14	0.14	0.14	0.16
ARA238025	0.07	0.06	0.07	0.07	0.06	0.04	0.06	0.06	0.06	0.06
LRA498378	0.30	0.16	0.13	0.19	0.13	0.13	0.14	0.14	0.15	0.10
LRB744710	0.20	0.11	0.11	0.10	0.12	0.12	0.12	0.11	0.11	0.11
Santa	1.57	0.48	0.45	0.47	0.44	0.44	0.45	0.52	0.47	0.40
World-TSP	0.52	0.50	0.40	0.42	0.41	0.41	0.38	0.39	0.38	0.27
Average gap	**0.62**	**0.23**	**0.22**	**0.23**	**0.21**	**0.22**	**0.21**	**0.22**	**0.21**	**0.18**
Average rank	**9.75**	**6.75**	**5.75**	**6.25**	**4.75**	**4.38**	**5.00**	**4.13**	**4.13**	**4.13**

depends on the combination of the initialization method and the candidate graph. However, the differences between the optimized tours are very small. In practical applications, any of the methods, excluding random, can therefore be used effectively. For applications targeting the shortest possible tours, Walk, Moore, Sierpinski, Q-Boruvka, and Boruvka are all competitive choices when combined with the Delaunay + Quadrant candidate set.

To provide context for the benchmark values, Table 13 presents the best-known tour lengths for each benchmark instance. These tours were collected from several sources: DIMACS [52], VLSI [49], Santa [53], and World-TSP [50]. The obtained tours are consistently close to the best-known values, indicating that the selected parameter settings are effective. Most of the remaining gap is likely explained by the optimization time limit and minor parameter adjustments. The sources do not report their computational budgets, whereas the present experiments were limited to 30 minutes for instances with fewer than one million points and one hour for larger instances.

Finally, the TSPLIB results are visualized in Figs. 4 and 5 for the Delaunay + Quadrant and POPMUSIC sets, respectively. The results for this batch of 78 TSPLIB instances follow the same general patterns as in the larger instances. The Delaunay + Quadrant set provides better optimization performance on average when combined with longer initial

TABLE 13. Comparison to the best-known tours. Best LKH refers to the best result obtained in the experiments within the limited optimization time of either 30 or 60 minutes. The number in parentheses after the instance name indicates the power of ten by which the tour lengths have been divided for representation purposes.

Instance	Best-known	Best LKH	Gap to best-known
C316k (3)	186,871	187,313	0.237%
E1M (3)	713,188	713,444	0.036%
E3M (3)	1,267,295	1,268,868	0.124%
ARA238025 (0)	578,761	578,960	0.034%
LRA498378 (0)	2,168,039	2,170,241	0.102%
LRB744710 (0)	1,611,232	1,612,087	0.053%
Santa (0)	108,416	108,802	0.356%
World-TSP (6)	7,516	7,530	0.186%

tours, whereas the POPMUSIC set performs better with shorter initial tours. However, in this test setup, where the time per instance is strictly limited instead of stopping after a failed trial, the POPMUSIC set performs slightly better overall.

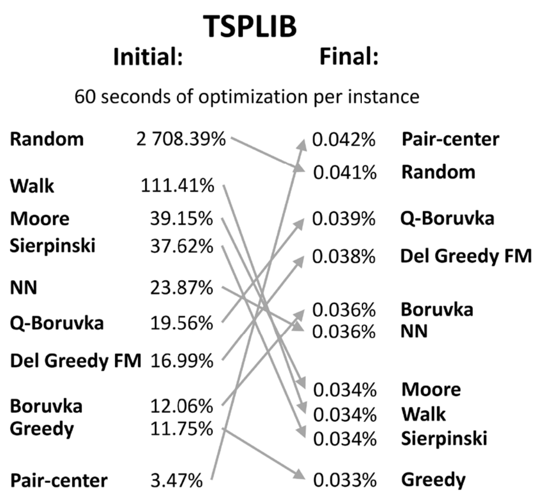

FIGURE 4. Comparison of initial tours and their optimized final paths on the TSPLIB dataset when Delaunay + Quadrant is used as the candidate set. The results are expressed as gaps relative to the optimal tours. Each result is averaged over five repeated runs with different seeds.

Our results support the findings of Perttunen, who showed that heuristic initial solutions generally improve TSP optimization performance compared to random starts [24]. In that study, random initialization became competitive only when a large number of repetitions were allowed (50 or even 100), whereas heuristic starts performed better under typical conditions.

The experimental setups of the two studies differ in several respects. Perttunen used significantly smaller instances and compared one heuristic against random initialization, whereas our experiments include larger benchmark instances, many initialization methods, and a more effective optimization procedure. Despite these differences, both studies support the same conclusion regarding the benefit of heuristic

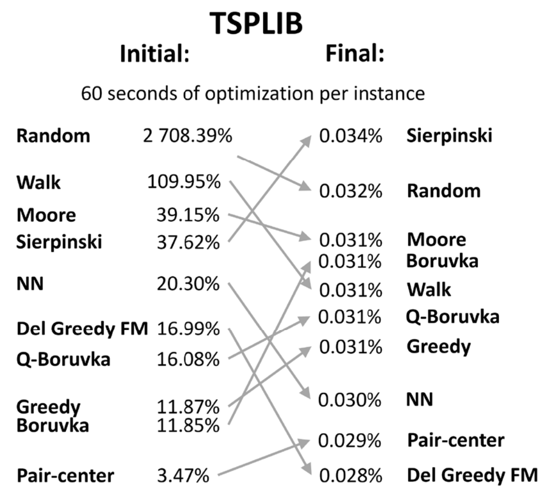

FIGURE 5. Comparison of initial tours and their optimized final paths on the TSPLIB dataset when POPMUSIC is used as the candidate set. The results are expressed as gaps relative to the optimal tours. Each result is averaged over five repeated runs with different seeds.

initial solutions over random initialization. However, we did not replicate the condition where Perttunen found the slight advantage of the random initialization. A practical limitation of repeated local search is the resulting optimization time, which, for larger instances such as in this paper, extends the time from hours to days and longer.

E. STRUCTURAL ANALYSIS OF INITIAL TOURS

To further explain the differences in optimization performance, we analyze the structure of the initial solutions. The first quality indicator is the length of the initial tour, reported in Table 10. As a second approach, we compare how many edges the initial tours share with a high-quality reference tour. The order of points for the best-known tour was available only for the World-TSP instance. Therefore, for each instance, a high-quality reference tour was produced by running LKH for three hours, using Walk as the initial solution and Delaunay + Quadrant with subgradient optimization as the candidate set. The lengths of these tours are presented in Table 14.

TABLE 14. High-quality reference tours for edge overlap comparison. Produced by running LKH for 3 hours for each instance. The number in parentheses after the instance name indicates the power of ten by which the tour lengths have been divided for representation purposes.

Instance	Reference-tour length	Gap to best-known
C316k (3)	187,207	0.180%
E1M (3)	713,340	0.021%
E3M (3)	1,267,958	0.052%
ARA238025 (0)	578,921	0.028%
LRA498378 (0)	2,169,667	0.075%
LRB744710 (0)	1,612,015	0.049%
Santa (0)	108,612	0.181%
World-TSP (6)	7,523	0.092%

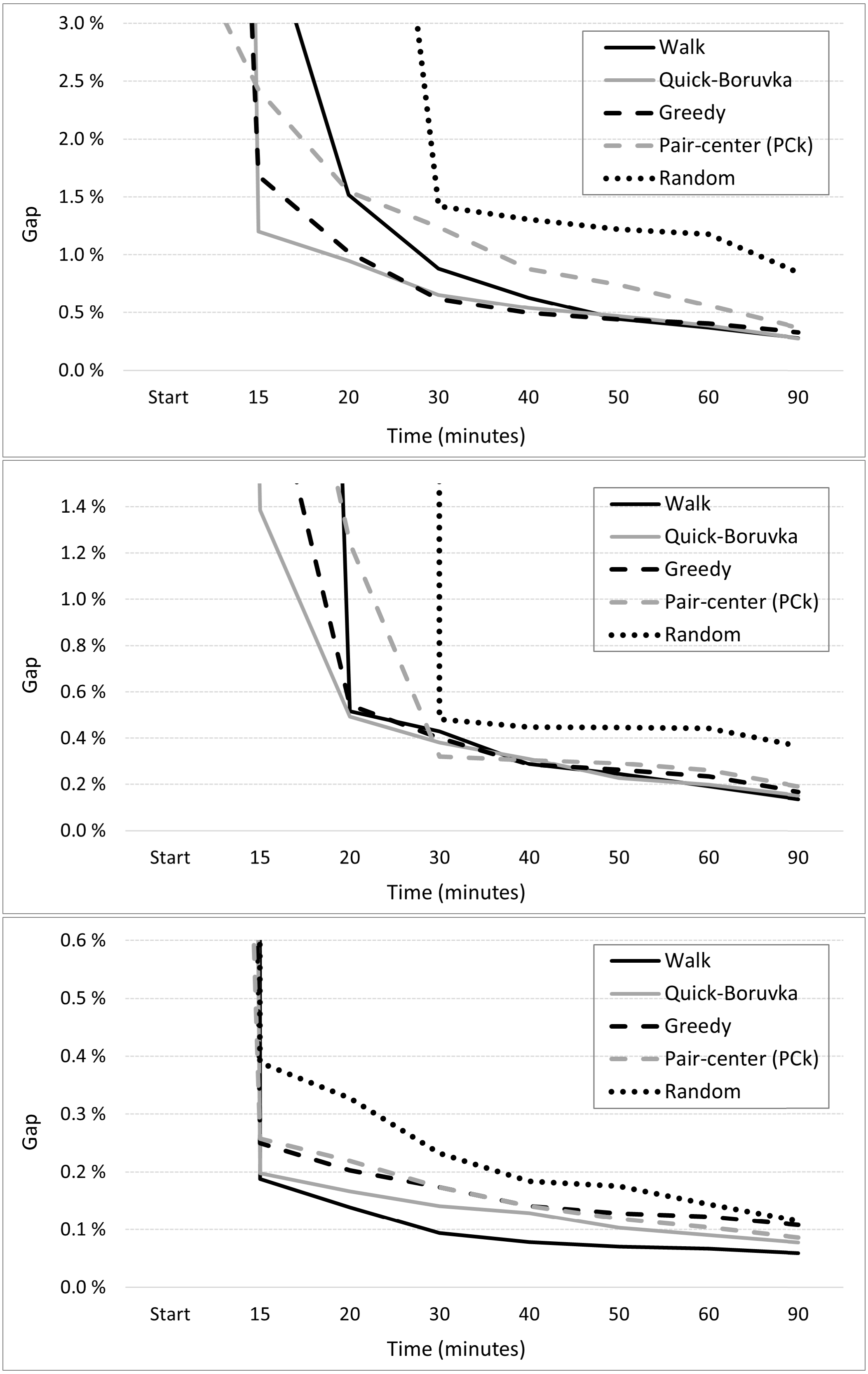

FIGURE 6. Optimization progress for different initial solutions on the Santa, World-TSP, and LRB744710 instances.

Table 15 compares the edge sets of the initial tours with the corresponding reference tours. The constructive initialization methods share between 36% and 85% of their edges with the reference tour. In general, the amount of overlap follows the length of the initial solution. However, Pair-center deviates from this pattern. Although it produces significantly shorter initial tours than the other methods, its edge overlap remains lower than expected. For example, Greedy has an average gap of 13.68% in the initialization benchmark and shares 85% of its edges with the reference tour, whereas Pair-center has a much lower gap of 4.70% but shares only 74% of its edges. Since Greedy constructs the tour directly from the candidate graph, its high edge overlap is expected. The lower overlap of Pair-center suggests that its tours are structurally different, which helps explain some of the difference in optimization behavior.

TABLE 15. Edge overlap between the initial tours and a high-quality reference tour. Averages are computed over the eight large-scale benchmark instances. Values indicate the percentage of edges in the initial tour that are also present in the reference tour. Delaunay + Quadrant was used as the candidate set for graph-based methods. POPMUSIC set changes the percentages only for Walk (+2%), NN (+1%) and Q-Boruvka (+1%).

Initial tour	Avg. overlap	Range
Random	0%	0%
Walk	36%	35–37%
Moore	48%	45–52%
Sierpinski	49%	46–53%
NN	82%	78–85%
Q-Boruvka	72%	68–74%
Del Greedy FM	74%	72–78%
Boruvka	85%	81–87%
Greedy	85%	81–87%
Pair-center	74%	71–79%

The Random approach has practically no overlap with the reference tours, since randomly generated edges are highly unlikely to appear in a high-quality solution. On average, only 2.5 (0%) common edges were found across the tested instances. As a result, the optimization starts from an initial solution with almost no useful structure, which explains the reduced efficiency.

F. OPTIMIZATION SPEED

Since the final tour lengths obtained from different initial solutions are very similar, it is important to analyze the speed of optimization. This was done by running the optimization and reporting the tour length at different timestamps using Santa, World-TSP, and LRB744710 instances as representative examples.

The optimization speed results are shown in Fig. 6. The initialization phase includes the generation of the Delaunay + Quadrant candidate set as well as subgradient optimization using the value 100, which takes approximately five to ten minutes in total, depending on the dataset. Even though the starting lengths vary greatly, all methods converge to similar tours during the first half of optimization. An exception to this is the random initialization, where the optimization process is noticeably slower.

The optimization speed figures help illustrate the results presented in Sec. V-D and Sec. V-E. The random initialization starts with a path that is completely unstructured and therefore more time-consuming to optimize. The Pair-center algorithm provides a short initial solution, but it does not appear to be the most favorable starting point for optimization with the Delaunay + Quadrant set.

Helsgaun [13] states that a significant reduction in runtime may be achieved by choosing initial tours that are close to being optimal. Our results do not support that statement. It is possible that the claim was made under the assumption that the optimal tour is known, and processing can be stopped as soon as it is reached. This would also imply that the optimization would be done for small- to medium-sized instances, where the optimal tour is known.

VI. CONCLUSION

The datasets were split into small-, medium-, and large-scale categories to analyze initialization algorithms and their optimized tours with different candidate graphs in 2-D Euclidean TSP. We found that candidate graphs have a moderate impact on the result.

The choice of the best candidate graph varies depending on data size. Alpha is most effective for small datasets. POPMUSIC is strongest for small and medium-sized datasets. Delaunay + Quadrant becomes most effective for medium and large-sized datasets. Nevertheless, all variants perform quite well except for KNN, which starts to struggle on large-scale datasets.

We considered several fast, sub-quadratic initialization methods. It was found that the effect of initialization on the optimized result is limited when an efficient optimizer such as LKH is applied. The initial tours produced by the methods can differ greatly, but after optimization, the final tours are very similar.

The Pair-center algorithm produced the shortest initial tours, but these did not always translate into the best final tours. A key finding is that the shortest initial tour does not necessarily lead to the best final result. Instead, the optimization results depend on the combination of initialization method and candidate graph. The Delaunay + Quadrant candidate set was the most effective overall compared to POPMUSIC in large-scale TSP. With the described test method, the POPMUSIC set gave slightly better tours in medium-scale TSP.

Additional experiments included a structural analysis of initial tours by comparing the percentage of overlapping edges between the initialization methods and a high-quality reference tour. Furthermore, the optimization speed in large-scale TSP was investigated.

Our results also support the findings of Perttunen, who showed that heuristic initial solutions generally improve optimization performance compared to random starts [24]. Random initialization resulted in fair tours after optimization,

TABLE 16. Ranking of initialization methods using POPMUSIC as the candidate set. The candidate set change affects the results of Walk, NN, Q-Boruvka, Boruvka, and Greedy. Gaps are reported as percentages relative to the best-known tours.

Instance	Random	Walk	Moore	Sierpinski	NN	Q-Boruvka	Del greedy FM	Boruvka	Greedy	Pair-center
C316k (3)	88.2k	167.58	53.49	50.77	40.88	26.04	26.57	17.84	16.06	3.44
E1M (3)	73.0k	362.65	37.26	34.24	17.64	10.16	13.39	10.22	10.10	4.77
E3M (3)	130.0k	492.56	37.32	34.28	17.48	10.00	13.24	10.13	9.90	4.70
ARA238025 (0)	49.1k	304.95	46.01	44.24	24.86	16.10	18.07	13.26	12.73	4.78
LRA498378 (0)	89.4k	175.95	46.99	47.01	26.12	19.57	16.52	12.99	11.55	6.06
LRB744710 (0)	78.0k	466.74	44.58	43.32	23.19	14.96	17.48	12.39	11.71	5.14
Santa (0)	367.7k	378.37	69.95	66.72	53.95	44.64	28.81	20.45	18.09	4.01
World-TSP (6)	184.1k	519.85	51.31	49.02	32.00	15.38	17.21	14.05	13.35	4.69
Average gap	**132.4k**	**358.58**	**48.37**	**46.20**	**29.52**	**19.61**	**18.91**	**13.92**	**12.94**	**4.70**
Average rank	**10.00**	**9.00**	**7.88**	**7.13**	**6.00**	**4.25**	**4.75**	**3.00**	**2.00**	**1.00**

but it is less efficient, and the final tours remained slightly longer than those obtained from heuristic starts.

Further improvements are likely to come from alternative candidate sets or more efficient optimization techniques, although this is challenging, as the complex LKH implementation has remained one of the most effective approaches for decades. Another possible direction is to integrate optimization into the initialization process. This could both accelerate the search, as optimization can be applied to subsets of points, and better model the full tour. Alternatively, one may consider a non-local-search-based or a deep learning approach.

APPENDIX
ADDITIONAL EXPERIMENTAL RESULTS

Table 16 provides additional results for the initialization experiments.

ACKNOWLEDGMENT

The first author would like to thank Keld Helsgaun for insightful discussions regarding LKH and its parameter settings, and Arno Formella for providing the implementation of the Pair-center algorithm and helpful instructions on how to run it properly.

REFERENCES

[1] N. Christofides, "Worst-case analysis of a new heuristic for the travelling salesman problem," Manage. Sci. Res. Group, Graduate School Ind. Admin., Carnegie-Mellon Univ., Pittsburgh, PA, USA, Tech. Rep. 388, 1976.

[2] N. Christofides, "Worst-case analysis of a new heuristic for the travelling salesman problem," *Oper. Res. Forum*, vol. 3, no. 1, p. 20, Mar. 2022.

[3] S. Arora, "Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems," *J. ACM*, vol. 45, no. 5, pp. 753–782, Sep. 1998.

[4] J. S. B. Mitchell, "Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric TSP, k-MST, and related problems," *SIAM J. Comput.*, vol. 28, no. 4, pp. 1298–1309, Jan. 1999.

[5] B. Rodeker, M. V. Cifuentes, and L. Favre, "An empirical analysis of approximation algorithms for Euclidean TSP," in *Proc. Int. Conf. Sci. Comput.*, 2009, pp. 190–196.

[6] R. M. Karp, "A patching algorithm for the nonsymmetric traveling-salesman problem," *SIAM J. Comput.*, vol. 8, no. 4, pp. 561–573, Nov. 1979.

[7] M. Held and R. M. Karp, "A dynamic programming approach to sequencing problems," *J. Soc. Ind. Appl. Math.*, vol. 10, no. 1, pp. 196–210, Mar. 1962.

[8] R. Bellman, "Dynamic programming treatment of the travelling salesman problem," *J. ACM*, vol. 9, no. 1, pp. 61–63, Jan. 1962.

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

[10] W. Cook. *Concorde TSP Solver*. Accessed: Jun. 2, 2025. [Online]. Available: https://www.math.uwaterloo.ca/tsp/concorde.html

[11] H. H. Hoos and T. Stützle, "On the empirical scaling of run-time for finding optimal solutions to the travelling salesman problem," *Eur. J. Oper. Res.*, vol. 238, no. 1, pp. 87–94, Oct. 2014.

[12] S. Lin and B. W. Kernighan, "An effective heuristic algorithm for the traveling-salesman problem," *Oper. Res.*, vol. 21, no. 2, pp. 498–516, Apr. 1973.

[13] K. Helsgaun, "An effective implementation of the Lin–Kernighan traveling salesman heuristic," *Eur. J. Oper. Res.*, vol. 126, no. 1, pp. 106–130, Oct. 2000.

[14] K. Helsgaun, "An effective implementation of the Lin–Kernighan traveling salesman heuristic," Dept. Comput. Sci., Roskilde University, Roskilde, Denmark, Tech. Rep. 81, 1998. [Online]. Available: http://webhotel4.ruc.dk/~keld/research/LKH/LKH-2.0/DOC/LKH_REPORT.pdf

[15] K. Helsgaun, "General k-opt submoves for the Lin–Kernighan TSP heuristic," *Math. Program. Comput.*, vol. 1, nos. 2–3, pp. 119–163, Oct. 2009.

[16] K. Helsgaun, "An effective implementation of k-opt moves for the Lin–Kernighan TSP heuristic," Comput. Sci., Roskilde University, Roskilde, Denmark, Tech. Rep. 109, 2006. [Online]. Available: http://webhotel4.ruc.dk/ keld/research/LKH/KoptReport.pdf

[17] K. Helsgaun. (2022). *LKH-2.0.10*. Accessed: Jun. 2, 2025. [Online]. Available: http://webhotel4.ruc.dk/ keld/research/LKH/

[18] T. Strutz, "Redesigning the wheel for systematic travelling salesmen," *Algorithms*, vol. 16, no. 2, p. 91, Feb. 2023.

[19] G. Kizilateş and F. Nuriyeva, "On the nearest neighbor algorithms for the traveling salesman problem," in *Advances in Computational Science, Engineering and Information Technology*, vol. 225. Heidelberg, Germany: Springer, 2013, pp. 111–118.

[20] D. S. Johnson and L. A. McGeoch, "Experimental analysis of heuristics for the STSP," in *The Traveling Salesman Problem and Its Variations*, vol. 12. Boston, MA, USA: Springer, 2007, pp. 369–443.

[21] J. J. Bentley, "Fast algorithms for geometric traveling salesman problems," *ORSA J. Comput.*, vol. 4, no. 4, pp. 387–411, Nov. 1992.

[22] J. Tuononen and P. Fränti, "Modified greedy Delaunay graph-based method for TSP initialization," in *Proc. IEEE Int. Conf. Prog. Informat. Comput. (PIC)*, Dec. 2024, pp. 177–183.

[23] A. Formella, "Quasi-linear time heuristic to solve the Euclidean traveling salesman problem with low gap," *J. Comput. Sci.*, vol. 82, Oct. 2024, Art. no. 102424.

[24] J. Perttunen, "On the significance of the initial solution in travelling salesman heuristics," *J. Oper. Res. Soc.*, vol. 45, no. 10, pp. 1131–1140, Oct. 1994.

[25] R. Mariescu-Istodor and P. Fränti, "Solving the large-scale TSP problem in 1H: Santa Claus challenge 2020," *Frontiers Robot. AI*, vol. 8, Oct. 2021, Art. no. 689908.
[26] C. K. Joshi, T. Laurent, and X. Bresson, "An efficient graph convolutional network technique for the travelling salesman problem," 2019, *arXiv:1906.01227*.
[27] L. Xin, W. Song, Z. Cao, and J. Zhang, "NeuroLKH: Combining deep learning model with Lin–Kernighan–Helsgaun heuristic for solving the traveling salesman problem," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 34, 2021, pp. 7472–7483.
[28] G. Reinelt, *The Traveling Salesman: Computational Solutions for TSP Applications*. Berlin, Germany: Springer, 1994.
[29] D. S. Johnson, "Local optimization and the traveling salesman problem," in *Automata, Languages and Programming*. Berlin, Germany: Springer, 1990, pp. 446–461.
[30] K. Helsgaun, "Using POPMUSIC for candidate set generation in the Lin–Kernighan–Helsgaun TSP solver," Dept. Comput. Sci., Roskilde University, Roskilde, Denmark, Tech. Rep., Jul. 2018. [Online]. Available: http://webhotel4.ruc.dk/ keld/research/LKH/POPMUSIC_REPORT.pdf
[31] M. Held and R. M. Karp, "The traveling-salesman problem and minimum spanning trees: Part II," *Math. Program.*, vol. 1, no. 1, pp. 6–25, Dec. 1971.
[32] É. D. Taillard and S. Voss, "POPMUSIC—Partial optimization metaheuristic under special intensification conditions," in *Essays and Surveys in Metaheuristics*. Boston, MA, USA: Springer, 2002, pp. 613–629.
[33] É. D. Taillard and K. Helsgaun, "POPMUSIC for the travelling salesman problem," *Eur. J. Oper. Res.*, vol. 272, no. 2, pp. 420–429, Jan. 2019.
[34] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Commun. ACM*, vol. 18, no. 9, pp. 509–517, Sep. 1975.
[35] K. R. Gabriel and R. R. Sokal, "A new statistical approach to geographic variation analysis," *Systematic Zool.*, vol. 18, no. 3, pp. 259–278, Sep. 1969.
[36] P. Fränti, R. Mariescu-Istodor, and C. Zhong, "XNN graph," in *Structural, Syntactic, and Statistical Pattern Recognition*, vol. 10029. Cham, Switzerland: Springer, 2016, pp. 207–217.
[37] D. W. Matula and R. R. Sokal, "Properties of Gabriel graphs relevant to geographic variation research and the clustering of points in the plane," *Geographical Anal.*, vol. 12, no. 3, pp. 205–222, Jul. 1980.
[38] O. Boruvka, "O jistém pproblému minimálním," *Práce Moravské Přírodovědecké Společnosti v Brně*, vol. 3, pp. 37–58, 1926.
[39] D. Applegate, W. Cook, and A. Rohe, "Chained Lin–Kernighan for large traveling salesman problems," *INFORMS J. Comput.*, vol. 15, no. 1, pp. 82–92, Feb. 2003.
[40] J. Tuononen and P. Fränti, "Simple and fast TSP initialization by Delaunay graph," in *Proc. 5th Int. Conf. Image, Video Process., Artif. Intell. (IVPAI)*, Mar. 2024, p. 10.
[41] L. K. Platzman and J. J. Bartholdi, "Spacefilling curves and the planar travelling salesman problem," *J. ACM*, vol. 36, no. 4, pp. 719–737, Oct. 1989.
[42] E. H. Moore, "On certain crinkly curves," *Trans. Amer. Math. Soc.*, vol. 1, no. 1, pp. 72–90, Jan. 1900.
[43] W. Sierpiński, "O pewnej krzywej wypełniającej kwadrat. Sur une nouvelle courbe continue qui remplit toute une aire plane," *Bulletin International de l'Académie des Sciences de Cracovie, Série A*, pp. 462–478, 1912.
[44] D. Applegate, R. Bixby, V. Chvátal, and W. Cook, "Finding tours in the TSP," Forschungsinstitut für Diskrete Mathematik, Universität Bonn, Bonn, Germany, Tech. Rep. 99885, 1999. [Online]. Available: https://www.math.uwaterloo.ca/tsp/methods/papers/lkreport.pdf
[45] R. Tinós, K. Helsgaun, and D. Whitley, "Efficient recombination in the Lin–Kernighan–Helsgaun traveling salesman heuristic," in *Parallel Problem Solving From Nature—PPSN XV*. Cham, Switzerland: Springer, 2018, pp. 95–107.
[46] L. Sengupta and P. Fränti, "Comparison of eleven measures for estimating difficulty of open-loop TSP instances," *Appl. Comput. Intell.*, vol. 1, no. 1, pp. 1–30, 2021.
[47] L. Sengupta, R. Mariescu-Istodor, and P. Fränti, "Which local search operator works best for the open-loop TSP?" *Appl. Sci.*, vol. 9, no. 19, p. 3985, Sep. 2019.
[48] G. Reinelt, "TSPLIB—A traveling salesman problem library," *ORSA J. Comput.*, vol. 3, no. 4, pp. 376–384, Nov. 1991.
[49] University of Waterloo. (2026). *VLSI Data Sets*. Accessed: Feb. 27, 2026. [Online]. Available: https://www.math.uwaterloo.ca/tsp/vlsi/index.html
[50] University of Waterloo, "The World TSP," 2025. Accessed: Apr. 15, 2025. [Online]. Available: https://www.math.uwaterloo.ca/tsp/world/index.html.
[51] DIMACS. *8th DIMACS Implementation Challenge: The Traveling Salesman Problem*. Accessed: Nov. 19, 2025. [Online]. Available: http://dimacs.rutgers.edu/archive/Challenges/TSP
[52] K. Helsgaun. (Feb. 2026). *DIMACS TSP Challenge Results: Current Best Tours Found by LKH*. Accessed: Feb. 27, 2026. [Online]. Available: http://webhotel4.ruc.dk/ keld/research/LKH/DIMACS_results.html
[53] T. Strutz, "Travelling Santa problem: Optimization of a million-households tour within one hour," *Frontiers Robot. AI*, vol. 8, Apr. 2021, Art. no. 652417.

JIMI TUONONEN received the B.Sc. and M.Sc. degrees in computer science from the University of Eastern Finland, Joensuu, Finland, in 2020 and 2022, respectively. He was a Research Assistant with the IMPRO project, in 2022. Since 2023, he has held a Doctoral Researcher position with the University of Eastern Finland. He is currently a Project Researcher with the Green ICT project VICTIS. He is a Doctoral Researcher in computer science with the University of Eastern Finland, Joensuu. His research focuses on algorithmic methods and computational experimentation for combinatorial optimization problems. His research interests include combinatorial optimization, algorithms, exact and heuristic methods for solving hard optimization problems. He has a particular focus on the traveling salesman problem.

PASI FRÄNTI (Senior Member, IEEE) received the M.Sc. and Ph.D. degrees from the University of Turku, Finland, in 1991 and 1994, respectively. He has been a permanent Full Professor with the University of Eastern Finland, since 2000. During his 30 years in research, he has published 130 peer-reviewed journal articles and 187 conference publications. He has supervised 34 Ph.D. Students and is guiding more than ten. He has reviewed 140 journals. His research interests include clustering algorithms, location-based services, analysis, and optimization of health care systems. He was an Associate Editor for *Pattern Recognition Letters*, *Journal of Electronic Imaging*, *Machine Learning with Applications*, *Applied Sciences*, and AI+. He is one of the Founding Editors of *Journal of Applied Computing and Intelligence* (AIMS).