Average Case Subquadratic Exact and Heuristic Procedures for the Traveling Salesman 2-OPT Neighborhood

Submitted to *INFORMS Journal on Computing*
manuscript (Please, provide the manuscript number!)

Average case sub-quadratic exact and heuristic procedures for the traveling salesman 2-OPT neighborhood

Giuseppe Lancia
Dipartimento di Matematica, Informatica e Fisica, University of Udine, 33100 Udine, Italy, giuseppe.lancia@uniud.it
Paolo Vidoni
Dipartimento di Scienze Economiche e Statistiche, University of Udine, 33100 Udine, Italy, paolo.vidoni@uniud.it

We describe an exact algorithm for finding the best 2-OPT move which, experimentally, was observed to be much faster than the standard quadratic approach for a large part of a best-improvement local search convergence starting at a random tour. To analyze its average-case complexity, we introduce a family of heuristic procedures and discuss their complexity when applied to a random tour in graphs whose edge costs are either uniform random numbers in $[0, 1]$ or Euclidean distances between random points in the plane. We prove that, for any probability p, there is a heuristic in the family which can find the best 2-OPT move with probability at least p in average-time $O(n \log n)$ for uniform instances and $O(n)$ for Euclidean instances. The exact algorithm is then proved to be even faster, in the sense that on those instances on which a heuristic finds the best move, the exact algorithm finds it in a smaller time. We give empirical evidence that a slight variant of our algorithm finds the best move in $O(n)$ time on both types of instances, achieving the best possible performance for this particular problem. Computational experiments are reported to show the effectiveness of our algorithms, both in best-improvement and in first-improvement 2-OPT local search.

Key words: Traveling Salesman; Combinatorial Optimization; 2-OPT Neighborhood; Heuristics; Applied Probability.
History:

1. Introduction

The TSP (Traveling Salesman Problem) is probably the most well-known combinatorial optimization problem (Lawler et al. (1991), Applegate et al. (2006), Gutin and Punnen (2007)). Its objective is to find a shortest Hamiltonian cycle in a complete graph of n nodes weighted on the arcs, and its importance stems from countless applications to all sorts of areas. Due to its relevance, the problem has been extensively studied over the years, and several programs have been designed for both its exact and heuristic solution (Lodi and Punnen (2007)).

Much in the same way as TSP is emblematic of all combinatorial optimization problems, 2-OPT is a prominent example of the concepts of neighborhood and local search procedure for an NP-hard problem. Local search (Aarts and Lenstra (1997), Papadimitriou and Steiglitz (1982)) is a general paradigm for the minimization of an objective function f over a set $\mathcal{S}$ of feasible solutions. The main ingredient of a local search procedure is a map which associates to every solution $x \in \mathcal{S}$ a set $N(x) \subset \mathcal{S}$ called its *neighborhood*. Starting at a solution x^0, local search samples the solutions in $N(x^0)$ looking for a solution x^1 better than x^0, i.e., an *improving solution*. If it finds one, it iterates the same step, this time looking for x^2 in $N(x^1)$, and then it continues the same way until the current solution x^i satisfies $f(x^i) = \min\{f(x)|x \in N(x^i)\}$, i.e., it is a *local optimum*. Replacing x^i with x^{i+1} is called performing a *move* of the search, and the total number of steps to get from x^0 to a local optimum is called the *convergence length*. An move from x^i to an improving solution x which determines the largest decrease in the objective function value is a *best improving move*, and x is a *best improving solution*. Local search can be based on two main strategies, i.e., *first-improvement* and *best-improvement* (also called *steepest-descent*). In first-improvement, x^{i+1} is the first improving solution found in $N(x^i)$ while, in best-improvement, x^{i+1} is a best improving solution in $N(x^i)$. For small-size neighborhoods, such as the one considered in this paper, it becomes feasible to adopt the best-improvement strategy. However, when a neighborhood is very large, best-improvement might become too expensive and first-improvement is the only viable approach.

The 2-OPT neighborhood associates to a TSP solution (also called a *tour*) T each tour T' that can be obtained by replacing two edges in T with two edges that are not in T. In this paper, we describe an effective algorithm for finding a best improving move. Besides best-improvement, our algorithm can easily be employed also for first-improvement. In this case, it has the advantage of looking for improving moves not in a blind way, but guided by a criterion based on their potential quality (i.e., the most promising moves are tried before the others).

The introduction of the 2-OPT neighborhood for the TSP dates back to the late fifties (Flood (1956), Croes (1958)), and still today local search based on this neighborhood is probably the most popular approach for the TSP (especially on large instances), for reasons of simplicity, low time-complexity and overall effectiveness. Indeed, there are some more sophisticated heuristics for the TSP, such as, e.g., 3-OPT (Lin (1965)), or metaheuristics

like genetic algorithms (Potvin (1996), Nagata and Kobayashi (2012)), simulated annealing (Ilhan and Gokmen (2022)) and tabu search (Basu (2012)). The most effective heuristic procedure, i.e., the one for which the trade-off between quality of solutions found and time spent in finding them is the best, is a 3-OPT variant known as Lin-Kernighan's algorithm (Lin and Kernighan (1973), Applegate et al. (2003)). However, all these sophisticated heuristics are somewhat complex to understand and implement, especially in comparison with the simplicity of 2-OPT. This aspect is considered very important in a large part of the industrial world, where in-house software development and maintenance oftentimes lead to the adoption of simple, yet effective, solutions like some basic local search. Indeed, in the case of 2-OPT, the algorithm to find the best move is trivial, i.e., it is a nested-`for` cycle iterating over all pairs of edges in the tour and taking $\Theta(n^2)$ time. Since, generally speaking, quadratic algorithms are considered very effective, not very much research went into trying to speed-up the algorithm for finding the best move. In this work, we will describe simple ways to speed-up, in a mathematically provable way, the standard quadratic procedure for 2-OPT. A probabilistic analysis over random tours and random edge costs shows that our strategies do in fact change the order of complexity from quadratic to sub-quadratic on average. In particular, our algorithm has an average-case complexity of $O(n \log n)$ for uniform random costs instances, and $O(n)$ for Euclidean instances induced by random points in the unit square. Note that $O(n)$ is an optimal complexity for this problem, given that the tour has n edges that must be looked at. The complexity is a combination of the work due to the number of moves evaluated and the work spent in creating/updating the data structure (a heap) used to decide which moves to evaluate. We then provide empirical evidence (but without a proof) that by using a suitable array in place of a heap we can find the best move on a random tour in average time $O(n)$ also for the uniform instances. In this regard, we highlight that our probabilistic results only apply to the first step of the local search, where the initial tour is chosen independently of the edge weights. In subsequent steps this is no longer valid and, as pointed out in the empirical section of the paper, this leads to a deterioration of algorithm performance in the long run.

Most of the literature on 2-OPT focuses on the study of the convergence length and the quality of the local optima that can be obtained. In particular, Chandra et al. (1999) (extending a result of Kern (1989)) have shown that the length of convergence is polynomial on average for random Euclidean instances, while Englert et al. (2014) have shown how to build

very particular Euclidean instances on which the length of the convergence is exponential. As for the local optima quality, Chandra et al. (1999) shows that they are, with high probability, within a constant factor from the global optimum for random Euclidean instances, while worst-case factors depending on n are given in Slootbeek (2017).

With regard to the time spent in finding the best move at each local search iteration, the nested-`for` algorithm is not only worst-case $\Theta(n^2)$, but its average-case is $\Theta(n^2)$ as well. Building on our previous research (Lancia and Vidoni (2020, 2024)) in which we studied how to speed-up some enumerative algorithms looking for a best solution in a polynomially-sized search space such as the 2-OPT neighborhood, we investigate a new greedy strategy for finding the best 2-OPT move. We give empirical and theoretical evidence that, over a long sequence of the tours visited by best-improvement local search starting at a random tour (roughly, for two thirds of the convergence length), our algorithm is on average better than quadratic. If $M(n)$ denotes the average number of moves tested in order to find the best possible one, we show that, with our strategy, at the initial iterations of the local search $M(n)$ is linear for uniform-costs instances, while $M(n)$ is bounded by a constant for random Euclidean instances. The time complexity then takes into account the number of moves tested and the time spent in choosing which moves to try.

To obtain a theoretical justification of the observed time complexity, we have devised a line of proof based on relating our algorithm to a family of heuristics for finding the best move, of which we discuss the average-case running time and the probability of success, and describe how we can control both aspects.

In best-improvement local search on medium- to large-sized instances, our procedure can achieve speedups of three orders of magnitude over the nested-`for` algorithm for most of the convergence. However, while the search progresses and we near the local optimum, our algorithm becomes less effective, so that at some point it might be better to switch back to the $\Theta(n^2)$ enumerative procedure since it does not have the overhead of dealing with any data structure. Experimentally, we determined that this phenomenon happens in the final part of the convergence, and it remains an interesting research question the design of an effective algorithm to find the best 2-OPT move over a nearly locally optimum tour. Finally, when used in first-improvement local search, our procedure has an interesting property. Namely, it combines the advantages of first-improvement and best-improvement, i.e., a certain speed in finding an improving move with a certain quality in the value of the move found. Some

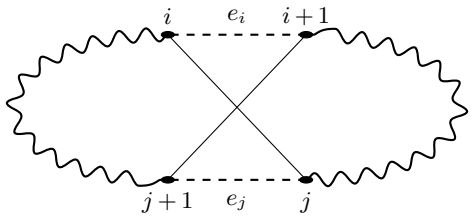
computational experiments will prove this to be a better strategy over the usual "blind" first-improvement.

Paper organization. The remainder of the paper is organized as follows. In Section 2 we describe the general idea for searching the best 2-OPT move, and present the main algorithm. Section 3 is devoted to the probabilistic analysis of our main algorithm and of a family of heuristics for the problem. Section 4 describes a variant of the main algorithm which, empirically, appears to be a $(\log n)$-factor faster on uniform instances. In Section 5 we report on our computational experiments and the statistical results that we obtained. Some conclusions are drawn in Section 6.

2. Our strategy for moves enumeration

Notation and terminology. We assume the set of vertices to be $[n] := \{1, \ldots, n\}$. Let T be the tour on which we seek to determine the best move. T is defined by a permutation of the vertices which, w.l.o.g., we assume to be $T = (1, \ldots, n)$. We denote the edges of the tour by $e_i := \{i, i+1\}$, for $i = 1, \ldots, n$ (where we consider $n+1 := 1$). In this paper we focus on the symmetric TSP, i.e., the graph is undirected, so that the distance between any two nodes is the same in both directions. We denote by $c(i,j) = c(j,i)$ the distance between two generic nodes i and j. The length $c(T)$ of a tour T is the sum of the lengths of the tour edges.

A 2-OPT move $\mu(i,j)$ is identified by e_i and e_j, two non-consecutive edges of the tour called the *pivots* of the move. A pair of pivots is called a *p-pair*. The move $\mu(i,j)$ removes a p-pair $\langle e_i, e_j \rangle$ and replaces them with $\{i,j\}$ and $\{i+1, j+1\}$, yielding the new tour T' (see the figure below).



Given a p-pair $\langle e_i, e_j \rangle$ we define its *removal cost* as $c^{\text{out}}(i,j) := c(e_i) + c(e_j)$ and its *insertion cost* as $c^{\text{in}}(i,j) := c(i,j) + c(i+1, j+1)$. The value of a 2-OPT move $\mu(i,j)$, is defined by

$$\Delta(\mu(i,j)) := c^{\text{out}}(i,j) - c^{\text{in}}(i,j).$$

We say that the move is *improving* if $\Delta(\mu(i,j)) > 0$. An improving move is *best improving* if $\Delta(\mu(i,j)) = \max_{u,v} \Delta(\mu(u,v))$. The goal is to find a best-improving move.

The general strategy. We are going to follow a strategy that allows us not to enumerate all moves, but only those which are "good candidates" to be the best overall. The idea is simple, and it relies on a sequence of iterative improvements in which, at each iteration, there is a certain move (the current "champion") which is the best we have seen so far and which we want to beat. Assume the current champion is $\hat{\mu} := \mu(\hat{\imath}, \hat{\jmath})$. Then, for any move $\mu(i,j)$ better than $\hat{\mu}$ it must be

$$\Delta(\hat{\mu}) < \Delta(\mu(i,j)) = c^{\text{out}}(i,j) - c^{\text{in}}(i,j) \leq c^{\text{out}}(i,j).$$

Based on this trivial observation, we will set-up an enumeration scheme which tests only moves identified by p-pairs for which $c^{\text{out}}(i,j) > \Delta(\hat{\mu})$.

Throughout the local search we are going to keep the tour edges in an array, sorted by decreasing value of c. Hereafter, we will denote by σ the sorted permutation, i.e., the permutation such that

$$c(e_{\sigma(1)}) \geq c(e_{\sigma(2)}) \geq \cdots \geq c(e_{\sigma(n)})$$

The initial sorting of the edges has a cost $O(n \log n)$ and so it could not be done if we are interested in achieving an $O(n)$ complexity when considering only one iteration of local search. However, when we look at a full local search convergence, the $O(n \log n)$ is paid only at the first iteration, while for all successive iterations the new sorted array can be obtained by updating the previous one in time $O(n)$, with the removal of two elements replaced by two new ones placed in the proper position. Given that the number of iterations is expected to be quite larger than $\log n$ (Kern (1989)), the initial sorting is thus amortized over the convergence, and having the sorted array costs on average linear time per iteration.

Given the sorted array of edges, we can exploit it in order to quickly enumerate all p-pairs whose c^{out} value is above a threshold, in decreasing order of the c^{out} value.

LEMMA 1. *For any threshold $\delta \geq 0$, let $\Pi(\delta)$ be the set of those p-pairs for which $c^{out}(i,j) > \delta$. There is an algorithm to enumerate the elements of $\Pi(\delta)$ from the largest to the smallest in time $\Theta(|\Pi(\delta)| \log n)$.*

PROOF: We discuss an algorithm, whose pseudocode will be given later within the description of our main procedure. The algorithm employs a max-heap, sorted by the c^{out} values, and containing at each step at most $n-1$ p-pairs. Each p-pair $\langle e_i, e_j \rangle$ in the heap is stored with $c(e_i) \geq c(e_j)$, i.e., $\sigma^{-1}(i) < \sigma^{-1}(j)$. There is always at most one p-pair in the heap for each

possible value of the first pivot, i.e., if $\langle e_i, e_j\rangle$ and $\langle e_{i'}, e_{j'}\rangle$ are p-pairs in the heap, then $i \neq i'$, while it could be $j = j'$. We maintain the invariant that if $\langle e_{\sigma(i)}, e_{\sigma(k)}\rangle$ is on the heap, then all p-pairs $\langle e_{\sigma(i)}, e_{\sigma(j)}\rangle$ with $i < j < k$ have already been enumerated. Note that their value is larger than that of $\langle e_{\sigma(i)}, e_{\sigma(k)}\rangle$. The heap is initialized in time $O(\min\{n, |\Pi(\delta)|\})$ by a loop, going from $i = 1$ to $\max\{k : (1 \leq k \leq n-1) \wedge c^{\text{out}}(\sigma(k), \sigma(k+1)) > \delta\}$, which inserts the p-pair $\langle e_{\sigma(i)}, e_{\sigma(i+1)}\rangle$.

At the generic step of the enumeration phase, the next p-pair to be enumerated is the heap root (the highest valued among the p-pairs left), let it be $\langle e_{\sigma(a)}, e_{\sigma(b)}\rangle$. If $c^{\text{out}}(\sigma(a), \sigma(b)) \leq \delta$ then the algorithm stops. Otherwise: (i) if $b < n$, the heap root element is overwritten with $\langle e_{\sigma(a)}, e_{\sigma(b+1)}\rangle$, and its cost is updated with $c^{\text{out}}(\sigma(a), \sigma(b+1))$ while (ii) if $b = n$, the cost of the root element is updated to be $-\infty$. Then, in time $O(\log n)$ the heap property is restored by letting the root element slide down the heap to its correct position (i.e., when it becomes larger than both its sons). Since the algorithm always enumerates the largest p-pair among those left, the p-pairs are enumerated in decreasing order of c^{out} value, and, by the description, at cost of $O(\log n)$ each.

■

Note that, for the sake of simplicity in either a code or a discussion, we sometimes treat as p-pairs also pair of pivots that do share an endpoint (for instance, in the initialization of the above procedure, we have inserted in the heap the elements $\langle e_{\sigma(i)}, e_{\sigma(i+1)}\rangle$ without making sure that the edges are not consecutive in the tour). In general, this can be done with no harm, since: (i) 2-OPT moves for pairs of consecutive pivots exist, but have value 0; (ii) there are only $\Theta(n)$ pairs of consecutive pivots, while there are $\Theta(n^2)$ p-pairs, and so any extra work when we include consecutive pivots within the p-pairs is dominated by the work for the actual p-pairs.

Based on the above lemma, we propose a greedy strategy for finding the best 2-OPT move. The algorithm, called $\mathcal{A}_g$ and described in Procedure 1, follows the criterion of testing always the p-pair of largest $c^{\text{out}}(i, j)$ value among those left to try. As far as the complexity analysis is concerned, step 1. has cost $O(n)$ since building a heap can be done in linear time with respect to the number of its elements by using standard procedures (Cormen et al. (2009)). Then there is a loop repeated M times, where M is the number of p-pairs read from the heap, and each loop iteration has cost $O(\log n)$, due to the call of RestoreHeap$(\cdot)$ in step 10. The procedure RestoreHeap(v) assumes to have a heap in the form of a binary tree in which

Procedure 1 GreedyLargestP-Pair $\mathcal{A}_g$

1. Build a max-heap with elements $[\sigma(i), \sigma(i+1), c^{\text{out}}(\sigma(i), \sigma(i+1))]$,
   where $i = 1, \ldots, n-1$, sorted by the c^{out}-field;
2. Set $\hat{\mu} := \emptyset$ and $\Delta(\hat{\mu}) := -\infty$; `/* first undefined champion */`
3. **while** the c^{out}-value of the heap root element is $> \Delta(\hat{\mu})$ **do**
4. let $[i, j, c^{\text{out}}(i,j)]$ be the heap root, where $i = \sigma(a)$, $j = \sigma(b)$;
5. **if** $\Delta(\mu(i,j)) > \Delta(\hat{\mu})$ **then**
6. $\hat{\mu} := \mu(i,j)$; `/* update the champion */`
7. **if** $b < n$
8. **then** overwrite heap root element with $[i, \sigma(b+1), c^{\text{out}}(i, \sigma(i+1))]$
9. **else** overwrite heap root element with $[i, j, -\infty]$
10. RestoreHeap(root)
11. **endwhile**
12. **return** $\hat{\mu}$;

the key of each node is larger than that of its sons, with, possibly, the exception of v. If that is the case, the heap property is restored by swapping the key of v with that of its largest son, say w, and then calling RestoreHeap(w).

3. Probabilistic analysis

3.1. The general plan

In this section we discuss the average-case complexity of our greedy algorithm, obtaining a theoretical justification of the empirical evidence that it is better than quadratic for a large portion of the convergence to a local optimum. In particular, at the very first steps, when the current solution is a random (or almost random) tour, we observed an average time complexity of $O(n \log n)$ on uniform instances and an average complexity of $O(n)$ on Euclidean instances. Note that $O(n)$ is an optimal result for this problem, since we cannot find the best move in a shorter time than that required to look at all the edges of the tour. When we only measured the total number of moves evaluated to find the best, we observed an average of $O(n)$ moves in uniform instances and $O(1)$ moves in Euclidean instances. The time complexity, however, takes into account also the overhead for managing the heap.

The analysis is relative to the problem of finding the best move on a random tour. In order to explain the observed sub-quadratic complexity, we start by discussing weaker versions

of the algorithm and prove that they run in average sub-quadratic time. These algorithms are heuristics, but we can make their probability of success as high as we please. We then show that the running time of $\mathcal{A}_g$ is dominated by the running time of a heuristic on those instances on which the heuristic succeeds (which, as remarked, can be almost all).

Some preliminaries. In the average-case analysis of an algorithm one considers instances drawn at random according to a certain probability distribution. In our study, an instance is given by $\binom{n}{2}$ non-negative reals (representing the edge costs in a complete undirected graph of n nodes) plus a permutation of $\{1,\ldots,n\}$ identifying a tour in the graph. The size of an instance can be characterized through a parameter n which, for us, is the number of nodes in the graph. By a *random tour* we denote a permutation drawn uniformly at random (u.a.r.) in the set including all the permutations. Hereafter, we assume $n \geq 4$, and, when we talk of a generic instance and of n in the same sentence, n is the instance size. As far as the edge costs are concerned, we consider two types of distributions:

1. **Uniform instances:** A random instance of this type is obtained by setting the cost of each edge $\{i,j\}$ to a value drawn u.a.r. in $[0,1]$. Note that the edge lengths are independent random variables.

2. **Euclidean instances:** A random instance of this type is obtained by drawing u.a.r. n points $P_1,\ldots,P_n$ in the unit square and then setting the cost of each edge $\{i,j\}$ to the Euclidean distance between P_i and P_j. Note that the edge lengths are not independent random variables since triangle inequality must hold.

In the following analysis, we focus on $m^n_{\mathcal{A}}(I)$, which represents the number of moves evaluated by an algorithm $\mathcal{A}$ on an instance I of size n to find the best one. We define the associated random variable $M^n_{\mathcal{A}}$ as the number of moves evaluated by $\mathcal{A}$ on a random instance of size n. The random instance is generated according to a suitable probability distribution corresponding to the uniform or to the Euclidean instances framework. The average-case complexity regarding the moves evaluated by $\mathcal{A}$ is then defined as

$$\bar{M}_{\mathcal{A}}(n) := \mathbb{E}\left[\, M^n_{\mathcal{A}} \,\right],$$

interpreted as a function of the size n. In a similar way, we denote by $t^n_{\mathcal{A}}(I)$ the time (i.e., total number of elementary steps) taken by an algorithm $\mathcal{A}$ on an instance I of size n, by $T^n_{\mathcal{A}}$ the time taken by an algorithm $\mathcal{A}$ on a random instance of size n, and by $\bar{T}_{\mathcal{A}}(n)$ the expected time-complexity of $\mathcal{A}$ as a function of n.

A family of "fixed threshold" heuristics. Let us consider a variant of $\mathcal{A}_g$ in which, instead of testing the p-pairs against the value of the current champion, we test them against a fixed threshold δ_n which depends on the instance size n but not on the instance itself. Essentially, the procedure limits the search of the best move to all and only the p-pairs $\langle e_i, e_j \rangle$ in the tour such that $c^{\text{out}}(i,j) > \delta_n$, which are enumerated sequentially. In particular, with respect to Procedure 1, we add a line 0. in which the algorithm computes a threshold δ_n, and we change line 3. by replacing $\Delta(\hat{\mu})$ with δ_n. Notice that there is an algorithm of this type for each possible function δ_n, and hence we can talk of a family of algorithms. Let us call a generic member of this family $\mathcal{H}(\delta_n)$.

Each algorithm $\mathcal{H}(\delta_n)$ is in fact a heuristic for finding the best 2-OPT move. Indeed, there is no guaranty that it will find the best move, but rather it will find it with a certain probability, depending on δ_n and on the distribution of instances. In particular, $\mathcal{H}(\delta_n)$ may fail to find the best move because of one of two types of errors:

ERR_0: when no p-pair is evaluated (all p-pairs have c^{out}-cost $\leq \delta_n$) and hence no move will be found;

ERR_1: when some p-pairs are evaluated, but the optimal move does not remove a p-pair of c^{out}-cost $> \delta_n$ and so it won't be found.

The probability of failure can be controlled by a proper setting of δ_n. Intuitively, by lowering (increasing) δ_n we decrease (respectively, increase) the probability of errors. At the same time, we increase (respectively, decrease) the average time complexity of the algorithm, since more (respectively, less) p-pairs get evaluated. We will describe a way to balance these two conflicting objectives, namely, having a δ_n large enough so as to guarantee an average sub-quadratic algorithm, but small enough so as the probability of errors can be upper-bounded by any given constant.

Let $\Delta^*(I) := \Delta(\mu^*)$ denote the value of an optimal 2-OPT move μ^* on an instance I. The following is a sufficient, but not necessary, condition for $\mathcal{H}(\delta_n)$ to find the optimal solution:

LEMMA 2. *For every instance I for which $\delta_n < \Delta^*(I)$, $\mathcal{H}(\delta_n)$ finds an optimal solution.*

PROOF: Assume a best move is $\mu^* = \mu(i,j)$. Then, $c^{\text{out}}(i,j) \geq c^{\text{out}}(i,j) - c^{\text{in}}(i,j) = \Delta^*(I) > \delta_n$ and therefore the p-pair $\langle e_i, e_j \rangle$ will be evaluated by $\mathcal{H}(\delta_n)$, thus yielding μ^*. ■

Given an instance I, let us call *good move* any move μ such that $\Delta(\mu) > \delta_n$ (notice that the property of being good for a move depends on δ_n, but, for simplicity, we assume that

δ_n is implicit from the context). Let us also call *good instance* any instance for which there exists at least one good move. Then we have the following

COROLLARY 1. *For every good instance I, $\mathcal{H}(\delta_n)$ finds an optimal solution.*

PROOF: Let μ be a good move in I. Then, $\delta_n < \Delta(\mu) \leq \Delta^*(I)$ and the conclusion follows from Lemma 2. ■

Furthermore, under the conditions of Corollary 1, $\mathcal{A}_g$ evaluates a subset of the moves evaluated by $\mathcal{H}(\delta_n)$, thus running faster than $\mathcal{H}(\delta_n)$.

LEMMA 3. *For every good instance I, it is $t^n_{\mathcal{A}_g}(I) \leq t^n_{\mathcal{H}(\delta_n)}(I)$.*

PROOF: Since there exist good moves, it is $\delta_n \leq \Delta^*(I)$. Let $\langle e_{i_1}, e_{j_1}\rangle, \langle e_{i_2}, e_{j_2}\rangle, \ldots, \langle e_{i_k}, e_{j_k}\rangle$ be the sequence of p-pairs tested by $\mathcal{A}_g$. Note that $c^{\text{out}}(i_1, j_1) \geq \cdots \geq c^{\text{out}}(i_k, j_k)$. Independently of which p-pair evaluation yielded the optimal move, since $\langle e_{i_k}, e_{j_k}\rangle$ was eventually evaluated it must be $c^{\text{out}}(i_k, j_k) \geq \Delta^*(I)$. Therefore $c^{\text{out}}(i_t, j_t) \geq c^{\text{out}}(i_k, j_k) \geq \Delta^*(I) > \delta_n$ for all $t = 1, \ldots, k$, which implies that all moves evaluated by $\mathcal{A}_g$ are also evaluated by $\mathcal{H}(\delta_n)$. Since the work paid for each move evaluated is the same for $\mathcal{A}_g$ and $\mathcal{H}(\delta_n)$, we conclude that $t^n_{\mathcal{A}_g}(I) \leq t^n_{\mathcal{H}(\delta_n)}(I)$. ■

The following lemma is useful for evaluating the average-case complexity of $\mathcal{H}(\delta_n)$ for every distribution over the instances.

LEMMA 4. *Let E_n be the edge set of the complete graph. Denote by C_e the random variable representing the cost of any edge $e \in E_n$ in a random instance. If δ_n is chosen so that, for every $a, b \in E_n$ which do not share an endpoint, it is $\mathbf{Pr}[\, C_a + C_b > \delta_n \,] = O(n^{-r})$, where $r \in (0, 2]$, then $\bar{M}_{\mathcal{H}(\delta_n)}(n) = O(n^{2-r})$.*

PROOF: Let $\mathcal{P}_n$ be the set of pairs of edges in the complete graph that do not share an endpoint. It is easy to see that $|\mathcal{P}_n| = \Theta(n^4)$. For each $\{a, b\} \in \mathcal{P}_n$, consider the indicator variable $X_{\{a,b\}}$ which is 1 if both edges a and b are in the tour (i.e., they are a p-pair) and their cumulative length is greater than δ_n. These two events are independent. Since there are $\Theta(n^2)$ p-pairs in a tour, the probability for $\{a, b\}$ to be a p-pair is $\Theta(n^2)/\Theta(n^4) = \Theta(n^{-2})$. We get

$$\mathbb{E}\left[\, X_{\{a,b\}} \,\right] = \mathbf{Pr}\left[\, X_{\{a,b\}} = 1 \,\right] = \Theta(n^{-2})\, \mathbf{Pr}\left[\, C_a + C_b > \delta_n \,\right] = O(n^{-2-r}).$$

Recalling that $M^n_{\mathcal{H}(\delta_n)} = \sum_{\{a,b\}\in\mathcal{P}_n} X_{\{a,b\}}$ is the random variable representing how many moves get evaluated by $\mathcal{H}(\delta_n)$, we obtain

$$\bar{M}_{\mathcal{H}(\delta_n)}(n) = \mathbb{E}\left[M^n_{\mathcal{H}(\delta_n)} \right] = \sum_{\{a,b\}\in\mathcal{P}_n} \mathbb{E}\left[X_{\{a,b\}} \right] = \Theta(n^4) \times O(n^{-2-r}) = O(n^{2-r}).$$

■

THEOREM 1. *Let $r \in (0,2]$. If δ_n is chosen so that for every p-pair $\langle a,b\rangle$ it is $\mathbf{Pr}[\, C_a + C_b > \delta_n \,] = O(n^{-r})$, then $\bar{T}_{\mathcal{H}(\delta_n)}(n) = O(n^{2-r}\log n)$ if $r \le 1$, while $\bar{T}_{\mathcal{H}(\delta_n)}(n) = O(n)$ if $r > 1$.*

PROOF: $\mathcal{H}(\delta_n)$ pays an $O(n)$ work to initialize the data structure. Furthermore, by Lemma 4, the average number of moves evaluated by $\mathcal{H}(\delta_n)$ is $O(n^{2-r})$. Since each evaluation costs $O(\log n)$ work, we have $\bar{T}_{\mathcal{H}(\delta_n)}(n) = O(n + n^{2-r}\log n) = O(\max\{n, n^{2-r}\log n\})$. The conclusion follows by noticing that the first terms dominates for $r > 1$, and the second for $r \le 1$.

■

A final simple claim, useful for our probabilistic analysis, states the independence of the random costs of two pivots in a p-pair.

CLAIM 1. *Let $a, b \in E_n$ be two edges which do not share an endpoint. Then C_a and C_b are independent random variables, under both the uniform and Euclidean random distributions.*

In the next two sections, we study our algorithms with respect to uniform and Euclidean random instances. For both types of distributions, we use the following approach:

1. We set δ_n so that, for all p-pairs $\langle a,b\rangle$, it is $\mathbf{Pr}\,[\, C_a + C_b > \delta_n \,] = \alpha n^{-r}$ for some constant $\alpha > 0$ and $r \in (0,2]$.

2. We describe a specific type of good moves and show that, asymptotically in n, the probability of having no good moves of our type tends to 0 for increasing α. This implies that for every $p \in [0,1)$ we can find an α to set δ_n so that, asymptotically, the probability for an instance to be good is grater than p.

3. We conclude that $\mathcal{H}(\delta_n)$ is a heuristic whose average-case complexity is sub-quadratic that succeeds on at least a fraction p of instances. This implies that, by Lemma 3, for at least a fraction p of all instances $\mathcal{A}_g$ is dominated by an algorithm of sub-quadratic average-case complexity, where p can be made as close to 1 as we want.

3.2. Uniform random costs

We start by a simple result on uniform random variables.

LEMMA 5. *Let X and Y be random variables drawn u.a.r. in $[0,1]$ and let $0 \leq \tau \leq 2$. Then*

$$\mathbf{Pr}[\, X + Y > \tau \,] = \begin{cases} 1 - \frac{\tau^2}{2} & \text{if } 0 \leq \tau \leq 1 \\ \frac{(2-\tau)^2}{2} & \text{if } 1 < \tau \leq 2 \end{cases}$$

PROOF: Since the sum of independent uniform random variables in $[0,1]$ follows the well-known Irwin-Hall distribution (Irwin (1927), Hall (1927)), the conclusion is immediate. ■

LEMMA 6. *Let $\alpha > 0$ be a constant and define*

$$\delta_n := 2 - \alpha\, n^{-1/2}.$$

Then, under the uniform distribution setting for random instances, the average-case complexity of $\mathcal{H}(\delta_n)$ satisfies $\bar{T}_{\mathcal{H}(\delta_n)}(n) = O(n \log n)$.

PROOF: Since we are interested in the asymptotic growth of complexity, assume n large enough so that $\delta_n > 1$. By Lemma 5, for each pair of edges a, b of the complete graph it is

$$\mathbf{Pr}[\, C_a + C_b > \delta_n \,] = \frac{(2-\delta_n)^2}{2} = \frac{\alpha^2}{2n} = O(n^{-1})$$

Then, by Theorem 1, it is $\bar{T}_{\mathcal{H}(\delta_n)}(n) = O(n \log n)$. ■

Let us call an edge $\{i, j\}$ *long* if

$$c(i,j) > \frac{2+\delta_n}{4} = 1 - \left(\frac{\alpha}{4}\right) n^{-1/2}$$

and *short* if

$$c(i,j) < \frac{2-\delta_n}{4} = \left(\frac{\alpha}{4}\right) n^{-1/2}.$$

The specific type of good moves that we consider, called *Long-Short moves* (LS-moves), are those that replace two long edges with two short ones. Indeed, for any such move μ, it is

$$\Delta(\mu) > 2(2+\delta_n)/4 - 2(2-\delta_n)/4 = \delta_n.$$

THEOREM 2. *For each $\alpha > 0$ denote by $P_0(\alpha, n)$ the probability that there is no LS-move in a random tour of n nodes on a random uniform instance. Then*

$$\lim_{n \to \infty} P_0(\alpha, n) \leq \frac{1}{e^{(\alpha/8)^4}}.$$

PROOF: Let $p_n := (\alpha/4)n^{-1/2}$ be the probability for an edge to be long, which is the same as the probability to be short. Let, as usual, the tour be $T = (1, \ldots, n, 1)$. Then, $P_0(\alpha, n)$ is the probability that there is no cycle $(i, i+1, j+1, j, i)$ of four edges (two in the tour and two not in the tour) such that $\{i, i+1\}$ and $\{j, j+1\}$ are long while $\{i+1, j+1\}$ and $\{i, j\}$ are short.

We can then think of two Bernoulli trials, in sequence, where the first trials determine long edges along the tour and the second trials determine good moves for pairs of long edges along the tour. To obtain independence for the second trials, we will consider moves that remove either two odd-indexed edges or two even-indexed edges. Let us focus on the odd-indexed edges. The first set of Bernoulli trials is repeated $n/2$ times, i.e., for all edges $\{i, i+1\}$ where i is odd, and the probability of success is p_n. We have a success if the edge $\{i, i+1\}$ is long. Assume there have been k successes altogether. Then, the second Bernoulli trials are repeated $\binom{k}{2}$ times, one for each pair $\{i, i+1\}$, $\{j, j+1\}$ of long edges . The probability of success is p_n^2, and there is a success if both $\{i+1, j+1\}$ and $\{i, j\}$ receive a short length. Note that these trials are independent, since for every two pairs s and q of long tour edges, the sets of non-tour edges that define the 2-OPT move for s and for q are disjoint.

The probability of having no LS-moves at all is upper bounded by the probability of having no LS-moves of the above type, i.e., having zero successes in the second Bernoulli trials. By the law of binomial distributions, this probability is

$$P_0'(\alpha, n) = \sum_{k=0}^{n/2} \binom{\frac{n}{2}}{k} p_n^k (1-p_n)^{\frac{n}{2}-k} (1-p_n^2)^{\binom{k}{2}}.$$

Let $S_n \sim \text{Binomial}(n/2, p_n)$. Then $P_0'(\alpha, n)$ can be written as

$$P_0'(\alpha, n) = \mathbb{E}\left[\left(1 - \alpha^2/(16n)\right)^{\binom{S_n}{2}} \right].$$

Now fix $\delta \in (0, \frac{1}{4})$, and define $a_n = \alpha\sqrt{n}/8 - n^{1/4+\delta}$ and $b_n = \alpha\sqrt{n}/8 + n^{1/4+\delta}$. Also, consider the event $A_n = \{S_n \in [a_n, b_n]\}$, and let its complement be A_n^c. By the Chebyshev's inequality,

$$\mathbf{Pr}\left[\, A_n^c \,\right] \leq \frac{\mathbf{Var}(S_n)}{n^{\frac{1}{2}+2\delta}} \leq \frac{C}{n^{2\delta}}$$

for some absolute constant $C > 0$ and hence $\mathbf{Pr}\left[\, A_n^c \,\right] \to 0$ as $n \to \infty$. Then by noting that $k \mapsto (1-p^2)^{\binom{k}{2}}$ is decreasing in k, we get

$$(1-\alpha^2/(16n))^{\binom{b_n}{2}} \mathbf{Pr}\left[\, A_n \,\right] \leq P_0'(\alpha, n) \leq (1-\alpha^2/(16n))^{\binom{a_n}{2}} \mathbf{Pr}\left[\, A_n \,\right] + \mathbf{Pr}\left[\, A_n^c \,\right],$$

and it is easy to check that both the lower and upper bound converge to $e^{-\alpha^4/2^{11}}$, which is the limit of $P'_0(\alpha,n)$. Since $P_0(\alpha,n) \le P'_0(\alpha,n)$ and $e^{-\alpha^4/2^{11}} < e^{-(\alpha/8)^4}$, the conclusion follows. ■

COROLLARY 2. *For each $p \in [0,1)$ there exist $\alpha > 0$ and $n_0 \in \mathbb{N}$ such that, for each $n \ge n_0$ and uniform random instance of size n, it is* $\mathbf{Pr}[$ *The instance is good* $] > p$.

PROOF: Let $\alpha > 0$ be such that $e^{-(\alpha/8)^4} < 1-p$, i.e., $\alpha > 8\sqrt[4]{\ln(1-p)^{-1}}$. Then

$$\begin{aligned}\lim_{n\to\infty} \mathbf{Pr}[\,\text{There is at least one LS-move}\,] &= 1 - \lim_{n\to\infty} P_0(\alpha,n) \\ &\ge 1 - \frac{1}{e^{(\alpha/8)^4}} \\ &> p\end{aligned}$$

and therefore, from some n_0 on, it is $\mathbf{Pr}[\,\text{There is a least one LS-move}\,] > p$. Since LS-moves are good moves, the conclusion follows. ■

Notice that we are discussing a lower bound to the probability of some specific good moves, and these are in turn a subset of all good moves. Thus, it is possible to obtain the same probability of no errors with an α smaller than that suggested by the corollary, as we will see in our computational experiments.

Finally, the following theorem bounds probabilistically the time complexity of $\mathcal{A}_g$ on uniform instances via $O(n \log n)$ functions.

THEOREM 3. *Consider the uniform distribution setting for random instances. Then, for each $p \in [0,1)$ there exists an algorithm $\mathcal{H}(\delta_n)$, with $\bar{T}_{\mathcal{H}(\delta_n)}(n) = O(n \log n)$, and an integer n_0 such that, for each $n \ge n_0$, it is* $\mathbf{Pr}[\,T^n_{\mathcal{A}_g} \le T^n_{\mathcal{H}(\delta_n)}\,] > p$.

PROOF: By Corollary 2 we can find $\bar{\alpha} > 0$ and n_0 such that, for $n \ge n_0$, the probability of a good instance is greater than p. Set $\delta_n := 1 - \bar{\alpha} n^{-1/2}$ and consider $\mathcal{H}(\delta_n)$. By Lemma 6 and Theorem 1, we have $\bar{T}_{\mathcal{H}(\delta_n)}(n) = O(n \log n)$. Since $T^n_{\mathcal{A}}$ is defined as the (random) number of steps executed by an algorithm $\mathcal{A}$ on a (random) instance of size n, we can usefully distinguish between the case in which the instance is good and the case in which it is not good, so that

$$\begin{aligned}\mathbf{Pr}[\,T^n_{\mathcal{A}_g} \le T^n_{\mathcal{H}(\delta_n)}\,] = {} & \mathbf{Pr}[\,\text{instance is good}\,] \times \mathbf{Pr}[\,T^n_{\mathcal{A}_g} \le T^n_{\mathcal{H}(\delta_n)} \,|\, \text{instance is good}\,] \\ & + \mathbf{Pr}[\,\text{instance is not good}\,] \times \mathbf{Pr}[\,T^n_{\mathcal{A}_g} \le T^n_{\mathcal{H}(\delta_n)} \,|\, \text{instance is not good}\,].\end{aligned}$$

By Lemma 3, $\mathbf{Pr}[\,T^n_{\mathcal{A}_g} \le T^n_{\mathcal{H}(\delta_n)} \,|\, \text{instance is good}\,] = 1$ and then, $\forall n \ge n_0$,

$$\begin{aligned}\mathbf{Pr}[\,T^n_{\mathcal{A}_g} \le T^n_{\mathcal{H}(\delta_n)}\,] &\ge \mathbf{Pr}[\,\text{instance is good}\,] \times \mathbf{Pr}[\,T^n_{\mathcal{A}_g} \le T^n_{\mathcal{H}(\delta_n)} \,|\, \text{instance is good}\,] \\ &= \mathbf{Pr}[\,\text{instance is good}\,] \\ &> p.\end{aligned}$$

■

3.3. Random Euclidean instances

We start with the following result on the distance between random points in the unit square.

LEMMA 7. *Let* $1.055 < d \le \sqrt{2}$ *and let* D *be the distance between two random points drawn uniformly in the unit square. Then* $\mathbf{Pr}\,[\,D > d\,] \le \frac{7}{16}\left(1 - \sqrt{d^2-1}\right)^4$.

PROOF: Consider Figure 1. In order for two points to have distance greater than d, they must not fall within a circle of ray $d/2$. We draw such a circle with center in $(0,0)$ and look at the intersections of the circle and the unit square. From Pythagoras' theorem, we get

$$y = \frac{1}{2}\sqrt{d^2-1}$$

and then

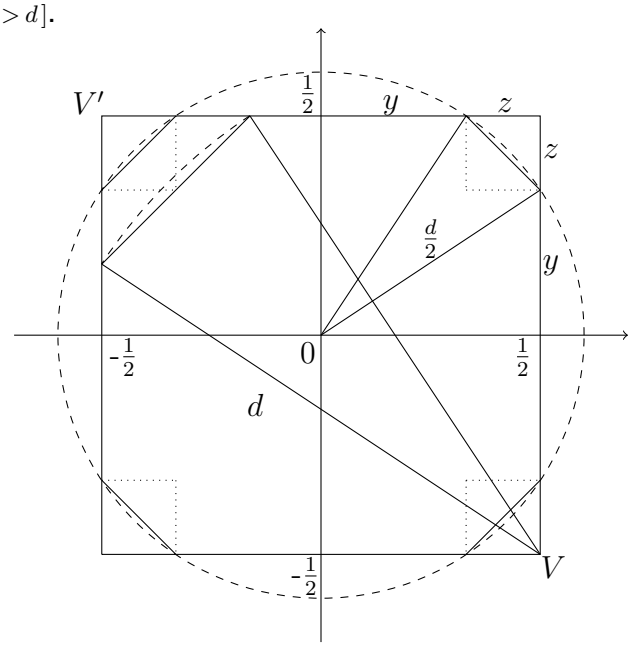
$$z = \frac{1}{2} - y = \frac{1}{2}\left(1 - \sqrt{d^2-1}\right).$$

For two points to have distance greater than d at least one of them should fall out of the circle, i.e., in the corners, each of which is an area of "triangular" shape but with a curve basis. We are going to relax this, and require that the point must fall within one of the four triangles with two sides of length z in the corners (this way we are overestimating the probability). Let us call $T^1, \ldots, T^4$ these squares, starting from the top-left and proceeding counter-clockwise.

Once a point is in $T \in \{T^1, \ldots, T^4\}$, the other point must be at distance greater than d from it. The distance between two points in triangles which are not opposite to each other is at most d, as it can be checked by noticing that $\sqrt{1+z^2} \le d$ is always satisfied for $d \ge 1.055$ (where $\sqrt{1+z^2}$ is the maximum distance between two non-opposite triangles T). Therefore we have to look for the second point in the opposite corner.

Since in T the farthest from other points is precisely the corner vertex, say V, we draw a circle C_V of ray d and center V. Let V' be the opposite corner, and let l be the distance

Figure 1 **Study of $\mathbf{Pr}[D > d]$.**



between V' and the intersection of the circle C_V with the square. By Pythagoras' theorem we get

$$l = 1 - \sqrt{d^2 - 1} = 2z.$$

Let $L^1, \ldots, L^4$ be the triangle with two sides of length l in the corners. From the previous discussion, if we pick a sequence X, X' of two points at random, in order to have $D > d$ there must exist opposite corners i and j such that either X in T^i and X' is in L^j, or X is in $L^i \setminus T^i$ and X' is in T^j (notice that since we used chords instead of circle arcs in these areas, these are necessary, but not sufficient conditions, and so we are overestimating the probability that $D > d$). For fixed opposite i, j, we have

$$\mathbf{Pr}\left[(X \in T^i \wedge X' \in L^j) \vee (X \in L^i \setminus T^i \wedge X' \in T^j) \right] = \frac{z^2}{2} \cdot \frac{4z^2}{2} + \frac{3z^2}{2} \cdot \frac{z^2}{2} = \frac{7}{4} z^4.$$

Therefore, for the probability that the above situation is realized at one of the 4 possible pairs of opposite corners, we obtain

$$\mathbf{Pr}[D > d] \leq 7z^4 = \frac{7}{16} \left(1 - \sqrt{d^2 - 1}\right)^4.$$

■

In our computational experiments we observed that, for Euclidean instances, the algorithm $\mathcal{A}_g$ evaluates an average number of moves which does not increase with n. Thus, we calibrate δ_n so that only a fraction $O(n^{-2})$ of the p-pairs pass the test and get evaluated.

LEMMA 8. *Let $\alpha > 0$ be a constant and define*

$$\delta_n := 2\sqrt{2} - \alpha\, n^{-1/4}.$$

Then, under the Euclidean distribution setting for random instances, the average-case complexity of $\mathcal{H}(\delta_n)$ satisfies $\bar{T}_{\mathcal{H}(\delta_n)}(n) = O(n)$.

PROOF: Notice that, for each p-pair $\langle a, b\rangle$, the pivots do not share endpoints, and therefore the lengths C_a and C_b are independent random variables. Furthermore, since it is always $C_a \leq \sqrt{2}$ and $C_b \leq \sqrt{2}$, the event $C_a + C_b > \delta_n$ implies both $C_a > \sqrt{2} - \alpha n^{-1/4}$ and $C_b > \sqrt{2} - \alpha n^{-1/4}$. We obtain

$$\begin{aligned}
\mathbf{Pr}\,[\,C_a + C_b > \delta_n\,] &\leq \mathbf{Pr}\left[\left(C_a > \sqrt{2} - \alpha n^{-1/4}\right) \wedge \left(C_b > \sqrt{2} - \alpha n^{-1/4}\right)\right] \\
&= \mathbf{Pr}\left[C_a > \sqrt{2} - \alpha n^{-1/4}\right]^2 \qquad (\text{ by independence of } C_a, C_b\ ) \\
&= O(n^{-1})^2 \qquad (\text{ by Lemma 7 }) \\
&= O(n^{-2}).
\end{aligned}$$

By Theorem 1, the average time complexity of $\mathcal{H}(\delta_n)$ is then $O(n)$. ■

Instead of discussing how the setting of the constant α affects the probability of having good moves, in the following we find it is easier to rewrite

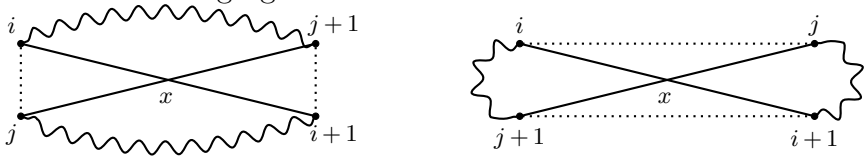
$$\delta_n = 2\sqrt{2} - (10\sqrt{2})\lambda\, n^{-1/4}$$

for a constant $\lambda > 0$ and discuss the constant λ.

We first recall a very basic property of tours on Euclidean instances.

LEMMA 9. *Let $T = (1, \ldots, n)$ be a tour on a Euclidean instance. Assume that edges $\{i, i+1\}$ and $\{j, j+1\}$ cross. Then the 2-OPT move $\mu(i, j)$ has value > 0. Furthermore if $\min\{c(i, i+1), c(j, j+1)\} > l > u > \max\{c(i, j), c(i+1, j+1)\}$, then $\Delta(\mu(i, j)) > 2(l - u)$.*

PROOF: Consider the following figure



(a) high-valued move (b) low-valued move

Denote the Euclidean distance between any two points a, b in the plane by $||\overline{ab}||$. Let x be the point in which $\{i, i+1\}$ and $\{j, j+1\}$ intersect. By triangle inequality $||\overline{ix}|| + ||\overline{xj}|| > ||\overline{ij}|| =: c(i,j)$ and $||\overline{(i+1)x}|| + ||\overline{x(j+1)}|| > ||\overline{(i+1)(j+1)}|| =: c(i+1, j+1)$, so that

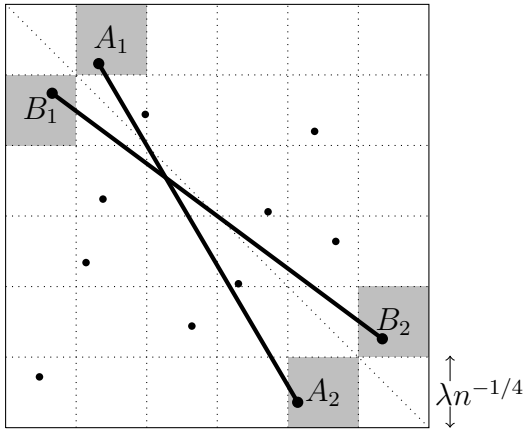
$$\begin{aligned}\Delta(\mu(i,j)) &= c(i,i+1) + c(j,j+1) - c(i,j) - c(i+1,j+1) \\ &= ||\overline{ix}|| + ||\overline{x(i+1)}|| + ||\overline{jx}|| + ||\overline{x(j+1)}|| - c(i,j) - c(i+1,j+1) \\ &> 0.\end{aligned}$$

The second part of the claim is obvious. ■

Notice that a pair of crossing edges implies an improving move, but the move's value could be high or not so high, depending on how small or large the angle $\widehat{ixj}$ is. In the previous figure, left, the angle is small and the move has a high value, while it is less so in the figure on the right.

Now we want to describe the specific type of good moves that we will use for the analysis. Consider Figure 2, showing the unit square which has been divided into $(n^{1/4}/\lambda) \times (n^{1/4}/\lambda) = \sqrt{n}/\lambda^2$ squares, each of side $\lambda n^{-1/4}$. Four of these squares are special, and they are labeled A_1, A_2 and B_1, B_2. An instance is a complete graph K_n made of n points and all line segments between them. In the figure, we show some of the points and edges.

Figure 2 **Explaining D-edges and D-crosses**



Call *D-edge* (for Diagonal-edge) an edge of K_n which is either $A_1 \bullet\!\!-\!\!\bullet A_2$ or $B_1 \bullet\!\!-\!\!\bullet B_2$. Furthermore, call *D-cross* (for Diagonal-cross) a pair of edges, one of which is $A_1 \bullet\!\!\rightarrow\!\!\bullet A_2$ and the other is $B_1 \bullet\!\!\rightarrow\!\!\bullet B_2$. Finally, call *C-edge* (for Corner-edge) an edge whose endpoints are both in $A_i \cup B_i$ for $i = 1, 2$. Intuitively, D-edges are "long" and C-edges are "short".

In a random instance, the tour T is identified by a random permutation of the nodes of K_n which, after relabeling, we assume as usual to be $(1,\ldots,n)$. For each D-cross contained in the tour, if the D-cross is traversed in the right order then there is a move which can replace two D-edges with two C-edges. For instance, a right order for the D-cross in the figure would be if the node in A_1 is labeled i, that in A_2 is $i+1$, the node in B_1 is j and that in B_2 is $j+1$, for some i and j. We denote this order as $(A_1 \to A_2 \rightsquigarrow B_1 \to B_2)$. In the analysis, we consider the specific type of good moves that replace a D-cross with two C-edges, which we call *D-uncrossing* moves. Since each D-edge is long at least $\sqrt{2}-3\sqrt{2}\lambda n^{-1/4}$ and each C-edge is long at most $2\sqrt{2}\lambda n^{-1/4}$, by Lemma 9 these moves would have value greater than

$$2(\sqrt{2}-3\sqrt{2}\lambda n^{-1/4})-4\sqrt{2}\lambda n^{-1/4}=2\sqrt{2}-10\sqrt{2}\lambda n^{-1/4}=\delta_n \tag{1}$$

i.e., they are in fact good moves.

LEMMA 10. *Consider a random tour $T=(1,\ldots,n)$ under the Euclidean distributional setting. Then, for each i, it is $\mathbf{Pr}[\,(i\in A_1)\wedge(i+1\in A_2)\,]=\lambda^4/n$.*

PROOF: The probability of a point drawn at random to fall in A_k, for $k=1,2$, is $\lambda^2/\sqrt{n}$. Since the points were drawn independently of each other, the conclusion follows. ■

LEMMA 11. *Consider a random tour T under the Euclidean distributional setting. Let $\mathcal{E}_A$ be the event "T does not traverse any $A_1\to A_2$ D-edge ". Then*

$$\mathbf{Pr}[\,\mathcal{E}_A\,]\le\left(1-\frac{\lambda^4}{n}\right)^{n/2}.$$

PROOF: Let $T=(1,\ldots,n)$ and for each i consider the event $D_i:=(i\in A_1)\wedge(i+1\in A_2)$. By Lemma 10, it is $\mathbf{Pr}[\,\neg D_i\,]=1-\lambda^4/n$. Furthermore, it is $\mathcal{E}_A=\neg D_1\wedge\neg D_2\wedge\cdots\wedge\neg D_n$.

Let us look at the odd-indexed edges of the tour, i.e., edges $\{i,i+1\}$ for $i=1,3,5,\ldots$. Since these edges are disjoint, the events $D_1,D_3,D_5,\ldots$ are independent. The probability that none of them occurs is $\mathbf{Pr}[\,\neg D_1\wedge\neg D_3\wedge\cdots\,]=(1-\lambda^4/n)^{n/2}$. Since $\mathcal{E}_A\implies\neg D_1\wedge\neg D_3\wedge\cdots$, the result follows. ■

COROLLARY 3. *Consider a random tour T under the Euclidean distributional setting. Let $\mathcal{E}_{AB}$ be the event "T does not contain any D-cross $(A_1\to A_2\rightsquigarrow B_1\to B_2)$". Then $\mathbf{Pr}[\,\mathcal{E}_{AB}\,]\le 2(1-\lambda^4/n)^{n/2}$.*

PROOF: Let $\mathcal{E}_A$ be the event "T does not traverse any $A_1 \to A_2$ D-edge" and $\mathcal{E}_B$ be the event "T does not traverse any $B_1 \to B_2$ D-edge". We have $\mathcal{E}_{AB} = \mathcal{E}_A \vee \mathcal{E}_B$. Furthermore, $\mathbf{Pr}[\mathcal{E}_A] = \mathbf{Pr}[\mathcal{E}_B]$ and so $\mathbf{Pr}[\mathcal{E}_{AB}] = \mathbf{Pr}[\mathcal{E}_A] + \mathbf{Pr}[\mathcal{E}_B] - \mathbf{Pr}[\mathcal{E}_A \cap \mathcal{E}_B] \leq 2\,\mathbf{Pr}[\mathcal{E}_A]$. Then the conclusion follows from Lemma 11. ■

THEOREM 4. *For each $p \in [0,1)$ there exist a value for $\lambda > 0$ and an integer n_0 such that, for each $n \geq n_0$ and Euclidean random instance of size n, it is $\mathbf{Pr}[$ The instance is good $] > p$.*

PROOF: Let λ be such that $2/\sqrt{e^{\lambda^4}} < 1 - p$, i.e.,

$$\lambda > \sqrt[4]{2 \ln \left(\frac{2}{1-p} \right)}.$$

It is

$$\begin{aligned} \lim_{n\to\infty} \mathbf{Pr}[\,\text{there are D-uncrossings}\,] &\geq 1 - \lim_{n\to\infty} 2(1-\lambda^4/n)^{n/2} \\ &= 1 - 2/\sqrt{e^{\lambda^4}} \\ &> p \end{aligned}$$

and therefore, from some n_0 on, it is $\mathbf{Pr}[$ there are D-uncrossings $] > p$. Since the D-uncrossings are good moves, the conclusion follows. ■

Finally, the following theorem bounds probabilistically the complexity of $\mathcal{A}_g$ on Euclidean instances via linear functions.

THEOREM 5. *Consider the Euclidean distributional setting for random instances. Then, for each $p \in [0,1)$ there exists an algorithm $\mathcal{H}(\delta_n)$, with $\bar{T}_{\mathcal{H}(\delta_n)}(n) = O(n)$, and an integer n_0 such that, for each $n \geq n_0$, it is $\mathbf{Pr}[T^n_{\mathcal{A}_g} \leq T^n_{\mathcal{H}(\delta_n)}] > p$.*

PROOF: The proof follows the exact same lines as the proof of Theorem 3. ■

4. A different greedy criterion

In this section we describe a variant of our algorithm which follows a different greedy criterion. Namely, instead of enumerating the moves by maximizing the total cost of both pivots, it first maximizes the cost of *the largest* pivot, and, once this pivot is fixed, the sum of the costs. The advantage in this approach is that each p-pair to try can be enumerated in time $O(1)$ instead than $O(\log n)$ by using the sorted array of tour edges. Indeed, with

Procedure 2 GreedyLargestPivot $\mathcal{A}_L$

1. Set $\hat{\mu} := \emptyset$ and $\Delta(\hat{\mu}) := -\infty$; `/* first undefined champion */`
2. $a := 1$;
3. **while** $(a \leq n-1) \wedge (c(e_{\sigma(a)}) > \Delta(\hat{\mu})/2)$ **do**
4. $\quad b := a+1$;
5. $\quad$ **while** $(b \leq n) \wedge (c^{\text{out}}(\sigma(a), \sigma(b)) > \Delta(\hat{\mu}))$ **do**
6. $\quad\quad$ **if** $\Delta(\mu(\sigma(a), \sigma(b)) > \Delta(\hat{\mu})$ **then**
7. $\quad\quad\quad \hat{\mu} := \mu(\sigma(a), \sigma(b))$; `/* update the champion */`
8. $\quad\quad b := b+1$
9. $\quad$ **endwhile**
10. $\quad a := a+1$
11. **endwhile**
12. **return** $\hat{\mu}$;

respect to the sorting σ, the moves are enumerated in lexicographic order, i.e., first all pairs $\{\langle e_{\sigma(1)}, e_{\sigma(2)}\rangle, \langle e_{\sigma(1)}, e_{\sigma(3)}\rangle, \ldots, \langle e_{\sigma(1)}, e_{\sigma(r_1)}\rangle\}$ which contain $e_{\sigma(1)}$ (the largest tour edge). Here, r_1 is the largest index such that $c^{\text{out}}(\sigma(1), \sigma(r_1)) > \Delta(\hat{\mu})$. Then, all pairs which contain $e_{\sigma(2)}$, i.e., $\{\langle e_{\sigma(2)}, e_{\sigma(3)}\rangle, \langle e_{\sigma(2)}, e_{\sigma(4)}\rangle, \ldots, \langle e_{\sigma(2)}, e_{\sigma(r_2)}\rangle\}$, where r_2 is the largest index such that $c^{\text{out}}(\sigma(2), \sigma(r_2)) > \Delta(\hat{\mu})$, and so on. Note that $r_1 \geq r_2 \geq \cdots$. The algorithm, called $\mathcal{A}_L$, where L stands for lexicographic, is described in Procedure 2. In the algorithm there are two nested loops. The outer loop picks, following the order determined by σ, the highest valued pivot, say e_i with $i = \sigma(a)$, terminating when its cost is $\leq \Delta(\hat{\mu})/2$. The second pivot, say e_j with $j = \sigma(b)$, is taken by the inner loop over $b = a+1, a+2, \ldots$ up to the last value such that $c^{\text{out}}(i,j) > \Delta(\hat{\mu})$. Note that the p-pairs that have the same first pivot are enumerated by decreasing value of c^{out}, but, overall, the full list of all p-pairs produced might not be sorted.

Our computational experiments have shown that $\mathcal{A}_L$ displays the same behavior as $\mathcal{A}_g$ with respect to the average number of moves evaluated, i.e., $O(n)$ for uniform instances and $O(1)$ for Euclidean instances. Given that the time to produce each move to try next is $O(1)$, and that keeping the sorted array requires a cost $O(n)$ per local search iteration (with the exception of the $O(n \log n)$ sorting cost for the very first iteration), we experimentally observed an average time $O(n)$ per iteration, independently of the instance type. Therefore, based on empirical evidence, we can conjecture that $\mathcal{A}_L$, unlike $\mathcal{A}_g$, presents an average

time complexity $O(n)$ also for uniform instances. However, to obtain formal proof of the improvement in the average time complexity observed in the uniform case, the approach considered in the previous section, based on a fixed-threshold version $\mathcal{H}_L(\delta_n)$ of $\mathcal{A}_L$, cannot be used.

The point is that, although most of the lemmas and theorems on which we based our construction still hold, the very important one (Lemma 3), which would allow us to upper bound the running time of $\mathcal{A}_L$ by the running time of its heuristic version $\mathcal{H}_L(\delta_n)$, does not. Indeed, it is now possible to have a good instance for which the algorithm $\mathcal{H}_L(\delta_n)$ explores fewer moves to find the optimal move than $\mathcal{A}_L$ does. For an example, assume to have a graph and a tour such that: the cost of the largest tour edge is 100, there are two edges, $\{i, i+1\}$ and $\{j, j+1\}$, of cost 91, and all remaining tour edges have cost lower than 80. The cost of all non-tour edges is large (>100) except for edges $\{i, j\}$ and $\{i+1, j+1\}$ which have cost 1. In this instance, there is only one improving move, which has value 180, and so, for any $\delta_n < 180$, the instance is good. Take $\delta_n = 180 - \epsilon$. Then, while $\mathcal{A}_L$ would enumerate all $n-1$ p-pairs which contain the largest pivot, the only two p-pairs that $\mathcal{H}_L(\delta_n)$ would enumerate in this block are the largest pivot with $\{i, i+1\}$ and with $\{j, j+1\}$.

In the next session, dedicated to numerical experiments, we evaluate also this alternative greedy algorithm, and compare it with those defined previously. An in-depth discussion of the results of the computational experiments is presented. Although the results, for the uniform case, seem promising, the possible theoretical justification would require, as already mentioned, a different approach that goes beyond the objectives of the present paper.

5. Computational experiments and statistics

In this section we report on our extensive computational experiments. All tests were run on a `Intel®Core™` i7-1065 CPU under Linux Ubuntu, equipped with 16GB RAM at 1.30GHz clock. The programs were implemented in C and compiled under `gcc 5.4.0`.

5.1. Best move from a random tour

In this section we compare experimentally the greedy algorithm ($\mathcal{A}_g$), the lexicographic algorithm ($\mathcal{A}_L$), the heuristic $\mathcal{H}(\delta_n)$ and complete enumeration (CE) by looking at how many moves they evaluate on average over 1000 runs. In particular, we generate 100 random instances and for each of them we generate 10 random tours on which we determine the best 2-OPT move.

Table 1 Average number of moves evaluated for finding the best move on a random tour. Results for uniform instances.

n	CE	$\mathcal{A}_g$	$\mathcal{A}_L$	$\mathcal{H}(\delta_n)$	$\frac{\mathcal{A}_L}{\mathcal{A}_g}$
2,000	1,999,000	3,080	3,813	7,230	1.23
4,000	7,998,000	6,101	7,738	14,469	1.26
6,000	17,997,000	9,284	11,741	21,614	1.26
8,000	31,996,000	12,066	15,403	28,943	1.27
10,000	49,995,000	15,489	19,709	36,187	1.27
12,000	71,994,000	18,698	23,754	43,471	1.27
14,000	97,993,000	21,306	27,324	50,625	1.28
16,000	127,992,000	25,172	32,097	57,914	1.27
18,000	161,991,000	27,432	35,331	64,754	1.28
20,000	199,990,000	31,454	39,898	71,726	1.26
22,000	241,989,000	34,993	44,170	79,325	1.26
24,000	287,988,000	36,664	47,107	86,796	1.28

Uniform instances. In Table 1 we report the average number of moves (rounded to integer) evaluated by CE, $\mathcal{A}_g$, $\mathcal{A}_L$ and $\mathcal{H}(\delta_n)$. In the experiment, we set $\delta_n := 2 - 3.8/\sqrt{n}$, where $\alpha = 3.8$ was chosen, after a little tuning, since it is a value large enough to guarantee a good probability of no errors. Indeed, out of 12,000 instances considered, the algorithm $\mathcal{H}(\delta_n)$ *always* found the best move.

The table shows how our approach can achieve a reduction in the number of moves evaluated of three orders of magnitude over complete enumeration, for instances of size ≤ 24000. The table also reports the ratios between the number of moves of the lexicographic and of the greedy algorithm (column $\frac{\mathcal{A}_L}{\mathcal{A}_g}$). Since this ratio stays pretty much constant (around 1.27) it appears from the table that $\mathcal{A}_g$ and $\mathcal{A}_L$ both evaluate a sub-quadratic number of moves of the same asymptotic growth but with a smaller constant for $\mathcal{A}_g$. In Figure 3 we have plotted the same values of Table 1 and we have fitted the dots with functions $\Theta(n)$, namely $1.55\,n$ for the greedy algorithm and $1.97\,n$ for the lexicographic algorithm. The interpolating function for $\mathcal{H}(\delta_n)$ corresponds to its theoretical expected complexity, which, by Lemma 6, is $(\alpha^2/(2n))(n(n-3)/2) \simeq 3.61n$.

Euclidean and TSPLIB geometric instances. We have performed a similar set of experiments on random Euclidean instances. In Table 2 we can see that the greedy algorithm is several orders of magnitude faster than complete enumeration when looking for the best move on a random tour on graphs with up to 24,000 nodes. The values are averages over 1,000 experiments for each size n, exactly as before. The lexicographic algorithm exhibits a similar time complexity, but it is roughly 1.28 times slower than $\mathcal{A}_g$. The fixed threshold algorithm has been run with $\delta_n = 2\sqrt{2} - 5/\sqrt[4]{n}$. We remark that the algorithm $\mathcal{H}(\delta_n)$ found the optimal

Figure 3 **Fittings of the average number of moves evaluated for finding the best move on a random tour (uniform instances). Each dot corresponds to the average over 1000 trials of size n (100 instances, 10 tours per instance).**

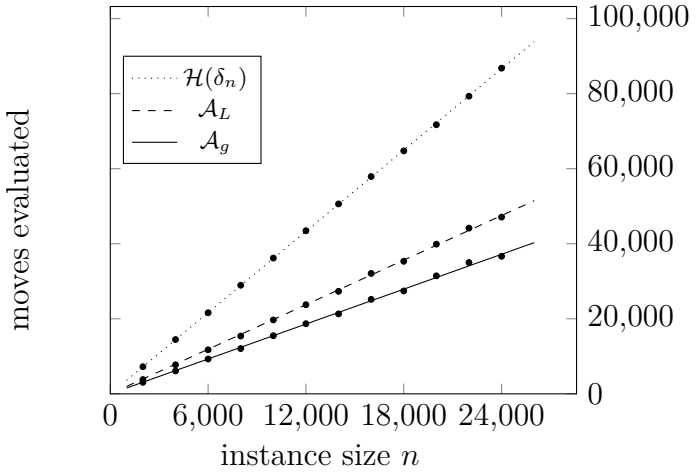


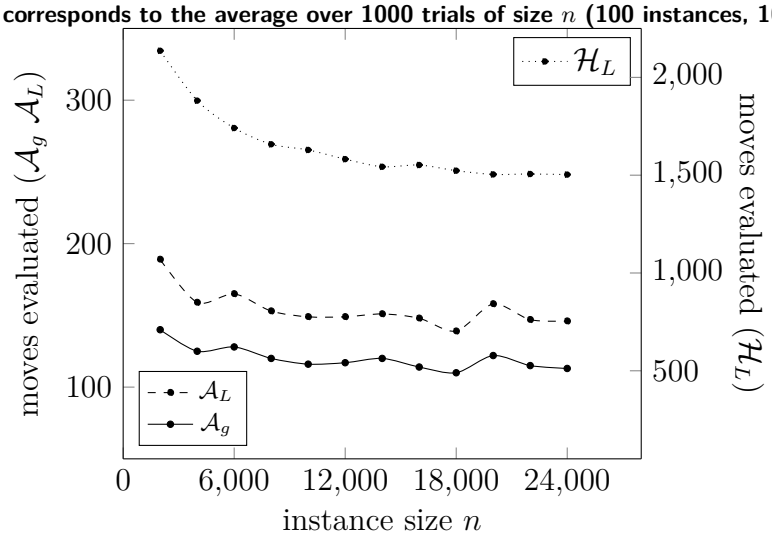
Table 2 **Average number of moves evaluated for finding the best move on a random tour. Results for Euclidean instances.**

n	CE	$\mathcal{A}_g$	$\mathcal{A}_L$	$\mathcal{H}(\delta_n)$	$\frac{\mathcal{A}_L}{\mathcal{A}_g}$
2,000	1,999,000	140.1	189.8	2,135.9	1.35
4,000	7,998,000	125.2	159.7	1,881.7	1.27
6,000	17,997,000	128.2	165.2	1,741.2	1.28
8,000	31,996,000	120.5	153.8	1,658.5	1.27
10,000	49,995,000	116.9	149.7	1,629.7	1.28
12,000	71,994,000	116.8	149,4	1,582.3	1.27
14,000	97,993,000	120.3	151.2	1,543.9	1.25
16,000	127,992,000	114.5	148.1	1,552.5	1.29
18,000	161,991,000	109.9	139.3	1,523.1	1,26
20,000	199,990,000	121.9	158.1	1,504.0	1,29
22,000	241,989,000	115.2	147.3	1,506.5	1.27
24,000	287,988,000	113.3	146.8	1,503.8	1.29

move, over *all* the 12,000 trials. In Figure 4 we can see the data of Table 2 plotted in a graph so that the $O(1)$ behavior of the algorithms can be appreciated graphically. Note that, to obtain a nice representation of all curves within the same figure, we have used two different windows on the y-axis: On the left, we see the y-axis values for $\mathcal{A}_g$ and $\mathcal{A}_L$, while on the right we have the y-axis values for $\mathcal{H}(\delta_n)$.

Some test-bed instances on the repository TSPLIB (Reinelt (1991)) are of geometric nature, and we have tested our algorithms $\mathcal{A}_g$ and $\mathcal{A}_L$ on those as well. In particular, there are some Euclidean instances (but they are not random, they correspond to some networks of world cities), and other are metric, not Euclidean, instances. We have selected the largest such instances (with the exception of `pla85900`, that, with $\geq 85,900$ nodes, was too big

Figure 4 **Average number of moves evaluated for finding the best move on a random tour (Euclidean instances). Each dot corresponds to the average over 1000 trials of size n (100 instances, 10 tour per instance).**



for our computer setting). The results, averages over ten random tours per instance, are reported in Table 3. It can be seen that our method achieves a speed-up of several orders of magnitude in finding the best move on a random tour.

Table 3 **Average number of moves evaluated for finding the best move on a random tour. Results for TSPLIB instances.**

name	n	CE	$\mathcal{A}_g$	$\mathcal{A}_L$	$\frac{\mathcal{A}_g}{\mathcal{A}_L}$
euc2d/rl5915	5,915	17,490,655	233.8	290.5	1.24
euc2d/rl5934	5,934	17,603,211	147.9	179.2	1.21
ceil2d/pla7397	7,397	27,354,106	92.1	119.7	1.29
euc2d/rl11849	11,849	70,193,476	146.3	157.0	1.07
euc2d/usa13509	13,509	91,239,786	127.4	165.2	1.29
euc2d/brd14051	14,051	98,708,275	241.6	282.5	1.17
euc2d/d15112	15,112	114,178,716	185.7	279.6	1.50
euc2d/d18512	18,512	171,337,816	282.7	367.9	1.30
ceil2d/pla33810	33,810	571,541,145	219.4	289.9	1.32

5.2. Best-improvement convergence to a local optimum

Given that finding the best move at the beginning of the local search is much faster with our algorithm than with the standard approach, we were optimistic about the fact that the time for the whole convergence would have been much shorter as well. Unfortunately, this is not the case, since the effectiveness of our approach decreases along the path to the local optimum. Indeed, at some point, our procedure can in fact become slower than complete enumeration since it has the overhead of dealing with a data structure that is not needed by the nested-for algorithm.

The slow-down phenomenon can be explained as follows. While approaching the local optimum, the value of the best move might decrease dramatically. At the beginning of the convergence, there are many improving moves and some of them have a really big value, setting a pretty high threshold for a p-pair to be evaluated. However, when the current tour has already undergone many improvements, most moves are worsening and the few improving moves have a small value. At this point, also the thresholds which determine which p-pairs to evaluate become pretty low and hence many p-pairs get evaluated.

Figure 5 displays this phenomenon quite clearly, comparing $\mathcal{A}_g$ with complete enumeration (CE). On the left we see a best-improvement local search convergence on a uniform instance on 1,000 nodes, lasting about 1,100 steps. On the right is a best-improvement local search convergence on a Euclidean instance on 1,000 nodes, lasting about 1,300 steps.

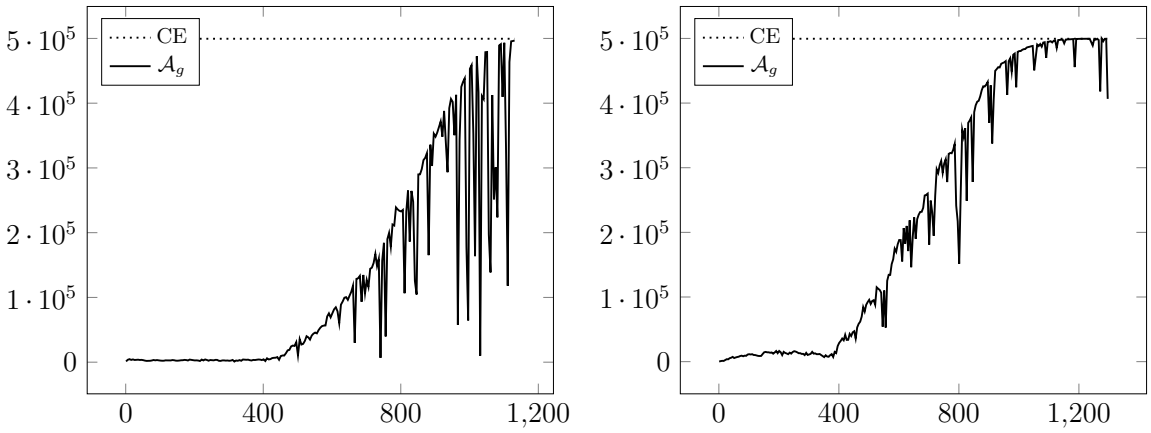


Figure 5 Number of moves evaluated per LS step. $n = 1000$**. Left, UNI instance. Right EUC instance.**

We can see that in both cases, for about 400 steps $\mathcal{A}_g$ evaluates a very small number of moves, much smaller than CE does. During this phase, despite the heap overhead, $\mathcal{A}_g$ is very much faster than CE. Starting at about move 500, however, we see that the number of moves starts to increase significantly, and at about two thirds of the local search $\mathcal{A}_g$ evaluates almost as many moves as CE does. When we take into account the overhead and look at the running times, it then happens that A_g becomes slower than CE.

With respect to the number of moves evaluated, the behavior of $\mathcal{A}_g$ and $\mathcal{A}_L$ is the same, but when we look at the running times we expect $\mathcal{A}_L$ to be faster since it does not require the heap. We have therefore compared the running times of $\mathcal{A}_g$, $\mathcal{A}_L$ and CE over full local search runs. In Figure 6 we can see the running times for the above two instances, which are good representatives of the general case.

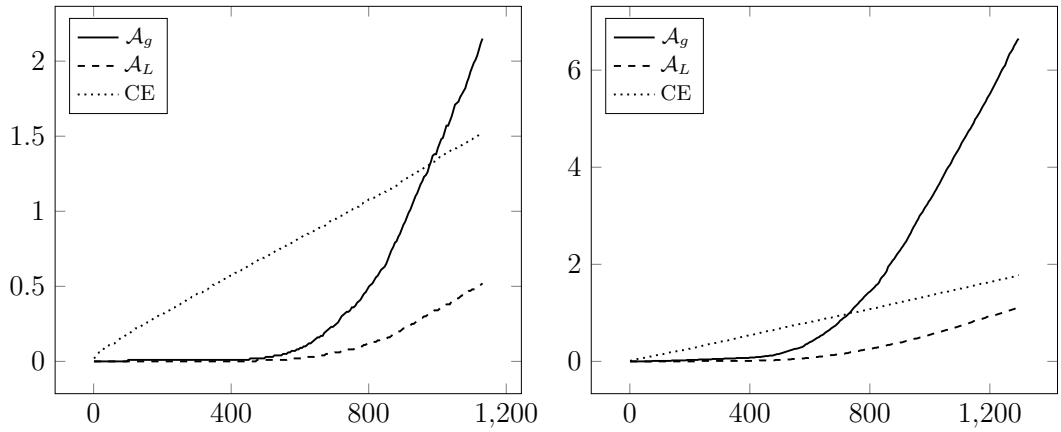


Figure 6 **Cumulative time (sec) up to LS step. $n = 1000$. Left, UNI instance. Right EUC instance.**

We can notice that if we use $\mathcal{A}_g$ for the whole local search we end up being overall slower than complete enumeration. On the other hand, if we start with $\mathcal{A}_g$ and switch to CE at about halfway through the convergence, we would end up saving roughly half of the processing time. Furthermore, we see that if we use $\mathcal{A}_L$ for the whole search, we have a faster algorithm than the standard enumeration even without switching. This behavior was observed in all our experiments. Indeed, we can do even better than $\mathcal{A}_L$ by switching from $\mathcal{A}_L$ to CE at some point, since when both $\mathcal{A}_L$ and CE evaluate more or less the same number of moves, the overhead of CE is certainly smaller than for $\mathcal{A}_L$. Let us denote by $\hat{\mathcal{A}}_L$ the hybrid version of $\mathcal{A}_L$, consisting of $\mathcal{A}_L$ up to a certain step followed by CE up to the end. In Figure 7, left, we show LS on the TSPLIB instance `pr1002`. CE terminates after 7.3s while $\mathcal{A}_L$ takes 5.8s, i.e., a 21% reduction. On the right we see that $\hat{\mathcal{A}}_L$, where the switch to CE is made around step 600, terminates in 3.9s, i.e., a 47% reduction from CE.

After observing many LS convergences, we derived a very simple rule of thumb for switching to CE. Each convergence takes a total number of steps which is slightly larger than n, and good results can be obtained by switching to CE after $\frac{3}{4}n$ steps. In Table 4 we report the results over instances of size from 500 to 3000, and including some TSPLIB instances. The "hybrid" algorithm $\hat{\mathcal{A}}_L$ consists of $\mathcal{A}_L$ for the first $\frac{3}{4}n$ steps, followed by CE up to the local optimum. For each instance we considered 10 random starting tours and report the average convergence length (column "steps"), the average number of moves per step evaluated by each algorithm (column $\bar{m}(\cdot)$) and the average total time for the convergence (column $\bar{t}(\cdot)$) in seconds. Other than the times, all averages are rounded to integer. The final column ("speed-up") reports how many times the hybrid algorithm is faster than CE. It can be seen that by adopting this simple rule, the running time can be approximately halved.

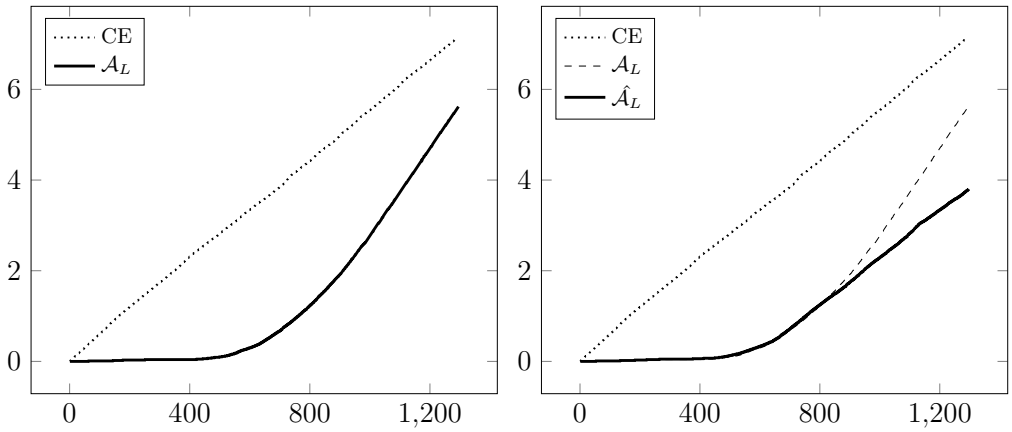


Figure 7 **Cumulative time (sec) up to LS step. TSPLIB instance pr1002, $n = 1002$. Left, $\mathcal{A}_L$ vs CE. Right, $\hat{\mathcal{A}}_L$ vs CE.**

Table 4 **Running times (sec) and move evaluations for best-improvement LS convergences with CE, $\mathcal{A}_L$ and $\hat{\mathcal{A}}_L$. Averages over 10 runs per instance.**

type	n	steps	$\bar{m}(CE)$	$\bar{t}(CE)$	$\bar{m}(\mathcal{A}_L)$	$\bar{t}(\mathcal{A}_L)$	$\bar{m}(\hat{\mathcal{A}}_L)$	$\bar{t}(\hat{\mathcal{A}}_L)$	speed-up
UNI	500	553	124,750	0.72	36,058	0.37	48,456	0.34	2.11×
	1,000	1,147	499,500	6.56	138,478	2.97	194,364	2.74	2.39×
	1,500	1,767	1,124,250	23.67	295,899	10.33	449,733	10.19	2.32×
	2,000	2,437	1,999,000	59.89	555,098	30.12	835,071	27.65	2.15×
	2,500	3,108	3,123,750	129.53	875,875	67.07	1,335,617	60.13	2.16×
	3,000	3,695	4,498,500	238.62	1,194,544	117.06	1,889,970	110.45	2.16×
EUC	500	588	124,750	0.93	58,295	0.59	61,651	0.54	1.72×
	1,000	1,290	499,500	7.15	227,535	5.34	253,496	4.07	1.75×
	1,500	1,972	1,124,250	26.07	495,043	20.08	565,779	14.99	1.73×
	2,000	2,681	1,999,000	73.46	856,554	53.54	1,004,952	37.82	1.94×
	2,500	3,399	3,123,750	141.72	1,324,269	111.10	1,576,564	77.69	1.82×
	3,000	4,156	4,498,500	278.88	1,904,449	218.87	2,298,937	149.34	1.86×
TSPLIB	rat575	1,340	165,025	1.54	78,442	1.38	81,287	0.98	1.57×
	pr1002	2,605	501,501	12.26	227,041	9.93	252,224	6.17	1.98×
	u1432	3,427	1,024,596	33.66	436,540	28.54	472,799	18.15	1.85×
	u2152	5,651	2,314,476	136.52	935,776	124.01	1,138,229	75.91	1.79×
	pr2392	6,576	2,859,636	198.14	1,203,348	167.72	1,444,297	103.85	1.90×
	pcb3038	8,174	4,613,203	496.96	2,007,906	417.04	2,299,242	255.49	1.94×

5.3. Experiments with first-improvement

In first-improvement local search, the time spent per step is much smaller than in best-improvement. On the other hand, also the change in the objective function is smaller, and the convergence to a local optimum generally requires many more steps. This trade-off between time per move and convergence length makes it difficult to consider one type of search surely better than the other.

Our algorithms are easily usable for first-improvement local search. Furthermore, they have a characteristic that might be an advantage over "blind" first-improvement, i.e., we sample

Table 5 First- and Best-improvement LS starting at a random tour. Averages over multiple runs per instance.

		F-I CE				F-I $\mathcal{A}_L$				B-I $\hat{\mathcal{A}}_L$			
type	n	f^*	$\bar{f}$	$\bar{t}$	steps	f^*	$\bar{f}$	$\bar{t}$	steps	f^*	$\bar{f}$	$\bar{t}$	steps
	500	6.02	6.66	0.02	2365	**6.01**	6.50	0.06	1252	6.13	**6.28**	0.26	561
	1000	**8.29**	8.85	0.12	5670	8.32	8.72	0.63	2790	8.44	**8.63**	2.88	1160
U	1500	10.25	10.72	0.36	9242	**10.03**	**10.43**	2.18	4382	10.41	10.53	10.02	1766
	2000	11.66	12.35	0.65	12753	11.84	12.04	5.85	6046	**11.55**	**11.81**	25.58	2437
	2500	13.22	13.62	1.28	17317	13.06	13.40	12.82	7711	**12.74**	**12.98**	54.05	3070
	500	18.29	18.70	0.02	3661	**17.60**	**18.04**	0.14	1690	17.85	18.33	0.59	599
	1000	25.54	26.15	0.11	9208	**24.69**	**25.08**	1.07	3803	25.44	25.62	4.34	1280
E	1500	31.43	31.97	0.22	16057	**30.19**	**30.57**	3.51	6098	30.98	31.10	19.91	1991
	2000	35.94	36.70	0.44	23348	**34.63**	**35.06**	8.33	8582	35.89	36.00	45.91	2710
	2500	40.28	41.18	0.51	29583	**39.24**	**39.54**	17.27	11058	40.15	40.38	88.51	3404
	rat575	7519	7658	0.02	4405	**7334**	**7491**	0.08	1801	7463	7573	0.73	670
	d657	53142	54852	0.03	5183	**51703**	**52866**	0.40	2432	53267	53806	1.35	820
	pr1002	286207	293431	0.06	9283	**275177**	**277871**	1.03	3848	281382	284240	4.38	1288
T	u1060	247260	253046	0.05	11052	**237925**	**240977**	1.41	4109	242578	247453	5.40	1373
	rl1323	293319	309967	0.09	13169	**282121**	**288530**	1.82	5595	296168	302311	10.36	1842
	u1432	172482	175573	0.11	15239	**167760**	**169218**	0.79	5355	170097	171333	12.49	1722
	u2152	73788	76083	0.30	25016	**71363**	**72857**	6.72	8979	72852	73922	57.86	2840
	pr2392	421939	435466	0.44	29542	**406889**	**412659**	15.19	10463	421003	423614	75.38	3275
	Tnm511	**8945401**	9119258	0.01	3098	8958219	**8998822**	0.06	1764	8983466	9073819	0.44	714
	Tnm1021	18570469	18803382	0.04	7029	**18547346**	**18597745**	0.49	3965	18613850	18714185	3.84	1542
H	Tnm1501	27621647	28001323	0.09	13174	**27601361**	**27685473**	1.67	6152	27654368	27761642	13.89	2312
	Tnm2011	37292079	37880141	0.18	18568	**37245854**	**37285407**	5.30	8630	37335996	37699400	38.67	3208
	Tnm2521	46925076	47590156	0.33	23635	**46888936**	**46924132**	10.86	10971	46956008	47259950	79.60	4016

the moves from the most promising to the least promising. Therefore, once an improving move is found, its value is generally much better than the value of a random improving move.

In this section we compare first-improvement with our strategy versus the usual first-improvement based on breaking from the nested-for algorithm of CE as soon as an improving move is found. Note that when CE is used for first-improvement, it is important that at each step of the convergence the nested-for resumes from where it ended at the last step. Otherwise, many p-pairs would be tested again that were not improving at the previous step and are still not improving at the current step, making the search much slower than it should be. We have considered four types of instances, i.e., random uniform (U in the table), random Euclidean (E), TSPLIB instances (T) and some "hard" instances (H), defined in Hougardy and Zhong (2001), which are some instances particularly difficult to solve to optimality. In our tests we have considered sizes which go, roughly, from $n = 500$ to $n = 2500$ with increments of 500, for all types of instances included in the experiments.

We have performed a first set of experiments, whose results are reported in Table 5. These experiments are relative to local search starting from a random tour. In the experiment we have also included the best-improvement strategy to try to assess if or when first-improvement is to be preferred over best improvement. The types of convergence considered are first-improvement with complete enumeration (columns labeled "F-I CE"), with $\mathcal{A}_L$ (labeled "F-I $\mathcal{A}_L$") and best-improvement with $\hat{\mathcal{A}}_L$ (labeled "B-I $\hat{\mathcal{A}}_L$"). For each instance,

Table 6 First- and Best-improvement LS starting at a NN tour. Averages over multiple runs per instance.

		F-I CE				F-I $\mathcal{A}_L$				B-I $\hat{\mathcal{A}}_L$			
type	n	f^*	$\bar{f}$	$\bar{t}$	steps	f^*	$\bar{f}$	$\bar{t}$	steps	f^*	$\bar{f}$	$\bar{t}$	steps
U	1000	3.70	4.01	0.04	104	**3.56**	3.85	0.03	97	3.57	**3.69**	0.15	53
	2000	4.24	4.48	0.21	157	4.06	4.30	0.23	152	**3.94**	**4.02**	0.88	91
	3000	4.43	4.69	0.56	198	4.30	4.51	0.73	182	**3.99**	**4.17**	2.63	108
	4000	4.71	4.91	1.19	236	4.50	4.66	1.15	219	**4.30**	**4.38**	5.48	126
	5000	4.83	5.01	1.80	267	4.61	4.79	2.10	247	**4.41**	**4.46**	9.45	144
E	1000	24.71	24.99	0.04	269	**24.28**	**24.46**	0.22	208	24.44	24.51	1.08	152
	2000	34.76	35.07	0.16	576	33.97	34.14	2.06	458	**33.93**	**34.09**	10.11	322
	3000	42.52	42.99	0.43	858	**41.82**	**42.00**	7.73	652	41.85	42.02	40.22	463
	4000	48.90	49.30	0.95	1098	**47.87**	**48.08**	20.49	850	47.99	**48.08**	87.33	588
	5000	54.85	55.20	1.39	1327	**53.45**	53.76	40.43	1046	53.63	**53.75**	188.86	732
T	u2319	245487	247069	0.16	487	243417	244955	0.22	396	**240007**	**241229**	11.61	282
	pr2392	403571	411427	0.20	598	**396172**	400396	2.27	470	397387	**399683**	13.26	310
	pcb3038	148651	149989	0.40	815	145590	146534	4.97	633	**145541**	**145796**	35.59	422
	fl3795	30087	31003	0.63	864	29483	30009	4.95	622	**29463**	**29854**	55.36	453
	rl5915	605735	613195	2.07	792	**591490**	**595127**	11.35	585	594073	597075	104.51	406
H	Tnm1021	18655658	18745272	0.01	17	**18635276**	18716006	0.02	21	18645515	**18676812**	0.08	12
	Tnm2011	37365235	37462906	0.04	14	**37361409**	37460712	0.05	15	37368017	**37429604**	0.21	13
	Tnm3001	**56079824**	56174525	0.15	19	56125606	56192125	0.17	19	56096925	**56122793**	0.89	14
	Tnm4021	75361046	75486418	0.30	14	75387526	75473227	0.32	15	**75320675**	**75437195**	1.25	11
	Tnm5011	**94106391**	94259213	0.54	13	94168303	94262754	0.43	9	94192902	**94229562**	1.34	8

we have run best-improvement 10 times. Then, we have run first-improvement for as many iterations as possible within a time limit given by the time taken by best-improvement.

In the table, for each row (i.e., instance) and each type of convergence we report: the value f^* of the best local optimum found; the average value $\bar{f}$ of all local optima found, rounded to integer for instances in blocks T and H; the average time $\bar{t}$ of a convergence; the average convergence length (column labeled "steps"), rounded to integer.

For each row, we have highlighted in boldface the best value found in columns f^* and $\bar{f}$. Column $\bar{f}$, in particular, gives a good indication on which type of convergence performs better on average. By looking at the table, it appears clear that first-improvement with $\mathcal{A}_L$ should be preferred over first-improvement with CE. Furthermore, it should also be preferred over best-improvement, with the exception of uniform random instances. The table also suggests that a first-improvement convergence with CE takes approximately twice the number of steps than with $\mathcal{A}_L$, which, in turn, is approximately twice the length of a best-improvement convergence with $\hat{\mathcal{A}}_L$.

A similar set of experiments has then been performed when the starting tours are not random, but are obtained by some tour-constructing heuristic. More specifically, each convergence is started from a tour obtained by running the well known Nearest-Neighbor (NN) heuristic. NN builds a tour by growing a partial path, initially consisting of a random node, and incrementally extending it from an endpoint to the closest unvisited node. When all nodes have been visited, the tour is obtained by connecting the endpoints of the path.

The results of the experiments are reported in Table 6. Since the convergence starting at a nearest-neighbor tour takes much fewer steps than at a random tour, we considered larger size instances than before, namely, n goes from 1,000 to 5,000 in increments of roughly 1,000. The experiments were conducted exactly as before. From the table it appears that, for this type of convergences, best-improvement is a better strategy than first-improvement. Moreover, if we limit the comparison to the two types of first-improvement, we see that using $\mathcal{A}_L$ is better than using CE, since in 18 cases out of 20 $\bar{f}$ is better with $\mathcal{A}_L$ than with CE, and in 17 cases out of 20 also f^* is better.

A peculiarity that appears from the table is that for instances in the family H, the convergence length from a NN tour to a local optimum is very short, much shorter than for instances of similar size in the other families. This phenomenon was not present when starting at a random tour.

6. Conclusions

In this work we have described two exact strategies, $\mathcal{A}_g$ and $\mathcal{A}_L$, for finding the best 2-OPT move in a given tour. We have also described a family of sub-quadratic average-case heuristics for the same problem, which can be tuned in such a way that they succeed with very high probability. Computational experiments and theoretical analysis have shown that our strategies largely outperform the classical two-nested-for algorithm for a good part of a best-improvement local search convergence starting from a random tour. In particular, on a starting random uniform instance, $\mathcal{A}_g$ determines the best move in average time $O(n \log n)$, while on a Euclidean instance, $\mathcal{A}_g$ takes average linear time, which is the best possible complexity for this problem. Empirical evidence suggests that the number of moves evaluated by $\mathcal{A}_L$ has the same order as it has for $\mathcal{A}_g$, but a formal proof might require a different approach that the one we used for $\mathcal{A}_g$. This empirical evidence also implies that $\mathcal{A}_L$ is faster than $\mathcal{A}_g$, since $\mathcal{A}_g$ pays $O(\log n)$ for each move which it evaluates, while $\mathcal{A}_L$ pays only $O(1)$.

We have then discussed how to adjust our procedures to obtain an effective best-improvement local search algorithm, given that the performance worsens while we approach the local optima. We have therefore proposed a hybrid procedure, made by $\mathcal{A}_L$ for the first part followed by the standard complete enumeration algorithm for the rest of the convergence, suggesting a possible point at which we should make the switch. This hybrid procedure allows one to roughly halve the overall running time. In further experiments, we have compared

complete enumeration and $\mathcal{A}_L$ when used in first-improvement local search, and determined that even if this case our algorithm is to be preferred over complete enumeration.

Some directions for future research would be: (i) integrating our algorithm, which is extremely fast in the initial iterations of best-improvement local search, with some ad-hoc improvements for the second part, in order to further improve its overall performance; (ii) trying to obtain a formal proof that $\mathcal{A}_L$ has the expected complexity that the empirical evidence suggests; (iii) perform further experiments (with both first- and best- improvement), including larger instances and varying the many parameters (such as the switch point for the best-improvement hybrid algorithm) in order to determine their best possible tuning.

As a final comment, however, we underline how the objective of this research was to focus on 2-OPT alone, and, in particular, to obtain some theoretical results about the work needed to determine the best move. The goal was not to beat other existing strategies to solve the TSP, since 2-OPT alone can never be competitive with more sophisticated heuristics which include also k-OPT moves for $k > 2$.

References

Aarts E, Lenstra JK, eds. (1997) *Local Search in Combinatorial Optimization* (New York, NY, USA: John Wiley & Sons, Inc.), 1st edition.

Applegate D, Bixby R, Chvatál V, Cook W (2006) *The Traveling Salesman Problem: A Computational Study* (Princeton University Press).

Applegate D, Cook W, Rohe A (2003) Chained lin-kernighan for large traveling salesman problems. *INFORMS Journal on Computing* 15(1:82–92.

Basu S (2012) Tabu search implementation on traveling salesman problem and its variations: A literature survey. *American Journal of Operations Research* 2:163–173.

Chandra B, Karloff H, Tovey C (1999) New results on the old k-OPT algorithm for the traveling salesman problem. *SIAM Journal on Computing* 28(6):1998–2029.

Cormen TH, Leiserson CE, Rivest RL, Stein C (2009) *Introduction to Algorithms, Third Edition* (The MIT Press), 3rd edition.

Croes G (1958) A method for solving traveling-salesman problems. *Operations Research* 6(6):791–812.

Englert M, Roglin H, Vocking B (2014) Worst case and probabilistic analysis of the 2-OPT algorithm for the TSP. *Algorithmica* 68(1):190–264.

Flood MM (1956) The traveling-salesman problem. *Operations Research* 4(1):61–75.

Gutin G, Punnen A, eds. (2007) *The Traveling Salesman Problem and Its Variations*. Combinatorial Optimization (Springer US).

Hall P (1927) The distribution of means for samples of size n drawn from a population in which the variate takes values between 0 and 1, all such values being equally probable. *Biometrika* 19:240–245.

Hougardy S, Zhong X (2001) Hard to solve instances of the euclidean traveling salesman problem. *Mathematical Programming Computation* 13:51–74.

Ilhan I, Gokmen G (2022) A list-based simulated annealing algorithm with crossover operator for the traveling salesman problem. *Neural Computing and Applications* 34:7627–7652.

Irwin J (1927) On the frequency distribution of the means of samples from a population having any law of frequency with finite moments, with special reference to pearson's type ii. *Biometrika* 19:225–239.

Kern W (1989) A probabilistic analysis of the switching algorithm for the euclidean TSP. *Mathematical Programming* 44:213–219.

Lancia G, Vidoni P (2020) Finding the largest triangle in a graph in expected quadratic time. *European Journal of Operational Research* 286(2):458–467.

Lancia G, Vidoni P (2024) Algorithmic strategies for finding the best TSP 2-OPT move in average subquadratic time. *arXiv* submit/5503157:1–28.

Lawler EL, Lenstra JK, Kan AHGR, Shmoys DB, eds. (1991) *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization* (Wiley).

Lin S (1965) Computer solutions of the traveling salesman problem. *The Bell System Technical Journal* 44(10):2245–2269.

Lin S, Kernighan BW (1973) An effective heuristic algorithm for the traveling-salesman problem. *Oper. Res.* 21(2):498–516.

Lodi A, Punnen AP (2007) *TSP Software*, 737–749 (in *The Traveling Salesman Problem and Its Variations*, Gutin G and Punnen AP eds.: Springer US).

Nagata Y, Kobayashi S (2012) A powerful genetic algorithm using edge assembly crossover for the traveling salesman problem. *INFORMS Journal on Computing* 25(2):346–363.

Papadimitriou C, Steiglitz K (1982) *Combinatorial Optimization: Algorithms and Complexity* (Prentice Hall).

Potvin JY (1996) Genetic algorithms for the traveling salesman problem. *Annals of Operations Research* 63:339–370.

Reinelt G (1991) TSPLIB - a traveling salesman problem library. *ORSA Journal on Computing* 3:376–384.

Slootbeek JJA (2017) Average-case analysis of the 2-OPT heuristic for the TSP. *Master Thesis – Applied Mathematics, University of Twente* 1–45.