# Fine-Grained Complexity of $k$-OPT in Bounded-Degree Graphs for Solving TSP

**Édouard Bonnet**
ENS Lyon, LIP, Lyon, France
edouard.bonnet@ens-lyon.fr

**Yoichi Iwata**
National Institute of Informatics, Tokyo, Japan
yiwata@nii.ac.jp

**Bart M.P. Jansen**
Eindhoven University of Technology, Eindhoven, The Netherlands
b.m.p.jansen@tue.nl

**Łukasz Kowalik**
Institute of Informatics, University of Warsaw, Poland
kowalik@mimuw.edu.pl

## Abstract

The Traveling Salesman Problem asks to find a minimum-weight Hamiltonian cycle in an edge-weighted complete graph. Local search is a widely-employed strategy for finding good solutions to TSP. A popular neighborhood operator for local search is $k$-opt, which turns a Hamiltonian cycle $\mathcal{C}$ into a new Hamiltonian cycle $\mathcal{C}'$ by replacing $k$ edges. We analyze the problem of determining whether the weight of a given cycle can be decreased by a $k$-opt move. Earlier work has shown that (i) assuming the Exponential Time Hypothesis, there is no algorithm that can detect whether or not a given Hamiltonian cycle $\mathcal{C}$ in an $n$-vertex input can be improved by a $k$-opt move in time $f(k)n^{o(k/\log k)}$ for any function $f$, while (ii) it is possible to improve on the brute-force running time of $\mathcal{O}(n^k)$ and save linear factors in the exponent. Modern TSP heuristics are very successful at identifying the *most promising* edges to be used in $k$-opt moves, and experiments show that very good global solutions can already be reached using only the top-$\mathcal{O}(1)$ most promising edges incident to each vertex. This leads to the following question: can improving $k$-opt moves be found efficiently in graphs of bounded degree? We answer this question in various regimes, presenting new algorithms and conditional lower bounds. We show that the aforementioned ETH lower bound also holds for graphs of maximum degree three, but that in bounded-degree graphs the best improving $k$-move can be found in time $\mathcal{O}(n^{(23/135+\epsilon_k)k})$, where $\lim_{k\to\infty}\epsilon_k = 0$. This improves upon the best-known bounds for general graphs. Due to its practical importance, we devote special attention to the range of $k$ in which improving $k$-moves in bounded-degree graphs can be found in *quasi-linear* time. For $k \leq 7$, we give quasi-linear time algorithms for general weights. For $k = 8$ we obtain a quasi-linear time algorithm when the weights are bounded by $\mathcal{O}(\operatorname{polylog} n)$. On the other hand, based on established fine-grained complexity hypotheses about the impossibility of detecting a triangle in edge-linear time, we prove that the $k = 9$ case does not admit quasi-linear time algorithms. Hence we fully characterize the values of $k$ for which quasi-linear time algorithms exist for polylogarithmic weights on bounded-degree graphs.

**2012 ACM Subject Classification** Theory of computation → Graph algorithms analysis; Theory of computation → Parameterized complexity and exact algorithms

**Keywords and phrases** traveling salesman problem, k-OPT, bounded degree

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2019.21

**Funding** *Yoichi Iwata*: Supported by JSPS KAKENHI Grant Number JP17K12643.
*Bart M.P. Jansen*: Supported by ERC Starting Grant 803421 ReduceSearch.
*Łukasz Kowalik*: Supported by ERC Starting Grant TOTAL (grant agreement No 677651).

**Acknowledgements** This research was initiated at the Shonan Meeting *Parameterized Graph Algorithms & Data Reduction: Theory Meets Practice* held during March 4-8, 2019 in Shonan Village Center, Japan. Yoichi Iwata thanks Kaggle Traveling Santa 2018 Competition for motivating him to study practical TSP heuristics. He also thanks Shogo Murai for valuable discussion about the possibility of faster $k$-OPT algorithms.

# 1 Introduction

## 1.1 Motivation

The Traveling Salesman Problem (TSP) hardly needs an introduction; it is one of the most important problems in combinatorial optimization, which asks to find a Hamiltonian cycle of minimum weight in an edge-weighted complete graph. Local search is widely used in practical TSP solvers [10, 11]. The most commonly used neighborhood is a *$k$-move* (or $k$-opt move). A $k$-move on a Hamiltonian cycle $\mathcal{C}$ is a pair $(E^-, E^+)$ of edge sets such that $E^- \subseteq E(\mathcal{C})$, $|E^-| = |E^+| = k$ and $(\mathcal{C} \setminus E^-) \cup E^+$ is also a Hamiltonian cycle. Marx [13] showed that finding an improving $k$-move (i.e., a $k$-move that results in a lighter Hamiltonian cycle) is W[1]-hard parameterized by $k$, and this result was refined by Guo et al. [6] to obtain an $f(k)n^{\Omega(k/\log k)}$ lower bound under the Exponential Time Hypothesis (ETH). For small values of $k$, the current fastest running time is $\mathcal{O}(n^k)$ for $k = 2, 3$ (by exhaustive search), $\mathcal{O}(n^3)$ for $k = 4$ [4], and $\mathcal{O}(n^{3.4})$ for $k = 5$ [3]. Moreover, de Berg et al. [4] and Cygan et al. [3] showed that improving the running time to $\mathcal{O}(n^{3-\epsilon})$ for $k = 3$ or $k = 4$ implies a breakthrough result of $\mathcal{O}(n^{3-\delta})$-time algorithm for All-Pairs Shortest Paths.

From the hardness shown by the theoretical studies, it seems that local search can be applied only to small graphs. Nevertheless, state-of-the-art local search TSP solvers can deal with large graphs with tens of thousands of vertices. This is mainly due to the following two heuristics.

1. They sparsify the input graph by picking the top-$d$ important incident edges for each vertex according to an appropriate importance measure. For example, Lin-Kernighan [12] picks the top-5 nearest neighbors, and its extension LKH [8] picks the top-5 $\alpha$-nearest neighbors, where the $\alpha$-distance of an edge is the increase of the Held-Karp lower bound [7] by including the edge. The empirical evaluation by Helsgaun [8] showed that the sparsification by the $\alpha$-nearest neighbors can preserve almost optimal solutions.
2. They mainly focus on *sequential* $k$-moves. In general, $E^- \cup E^+$ is a set of edge-disjoint closed walks, each of which alternately uses edges in $E^-$ and $E^+$. If it consists of a single closed walk, the move is called sequential. Graphs of maximum degree $d$ with $n$ vertices have at most $n(2(d-2))^{k-1}$ sequential $k$-moves ($n$ choices for the starting point, 2 choices for the next edge in $E^-$, and at most $d-2$ choices for the next edge in $E^+$), which is linear in $n$ when considering $d$ and $k$ as constants. On the other hand, linear-time computation of non-sequential $k$-moves appears non-trivial. Lin-Kernighan does not search for non-sequential moves at all, and after it finds a local optimum, it applies special non-sequential 4-moves called *double bridges* to get out of the local optimum. LKH-2 [9] improves Lin-Kernighan by heuristically searching for non-sequential moves during the local search.

This state of affairs raises the following questions: what is the complexity of finding improving $k$-moves in bounded-degree graphs? How does the complexity scale with $k$, and can it be done efficiently for small values of $k$? Since improving *sequential moves* can be found in linear time for fixed $k$ and $d$, to answer these questions we have to investigate non-sequential $k$-moves in bounded-degree graphs.

## 1.2 Our contributions

We classify the complexity of finding improving $k$-moves in bounded-degree graphs in various regimes. We present improved algorithms that exploit the degree restrictions using the structure of $k$-moves, treewidth bounds, color-coding, and suitable data structures. We also give new lower bounds based on the Exponential Time Hypothesis (ETH) and hypotheses from fine-grained complexity concerning the complexity of detecting triangles. To state our results in more detail, we first introduce the two problem variants we consider; a weak variant to which our lower bounds already apply, and a harder variant which can be solved by our algorithms.

> $k$-opt Detection **Parameter:** $k$.
> **Input:** An undirected graph $G$, a weight function $w\colon E(G) \to \mathbb{Z}$, an integer $k$, and a Hamiltonian cycle $\mathcal{C} \subseteq E(G)$.
> **Question:** Can $\mathcal{C}$ be changed into a Hamiltonian cycle of strictly smaller weight by a $k$-move?

The related optimization problem $k$-opt Optimization is to compute, given a Hamiltonian cycle in the graph, a $k$-move that gives the largest cost improvement, or report that no improving $k$-move exists. With this terminology, we describe our results.

We show that $k$-opt Detection is unlikely to be fixed-parameter tractable on bounded-degree graphs: we give a new constant-degree lower-bound construction to show that there is no function $f$ for which $k$-opt Detection on subcubic graphs with weights $\{1,2\}$ can be solved in time $f(k) \cdot n^{o(k/\log k)}$, unless ETH fails. Hence the running time lower bound for general graphs by Guo et al. [6] continues to hold in this very restricted setting. While the degree restriction does not make the problem fixed-parameter tractable, it is possible to obtain faster algorithms. By adapting the approach of Cygan et al. [3], exploiting the fact that the number of sequential moves is linear in $n$ in bounded-degree graphs, and proving a new upper bound on the pathwidth of an $k$-edge even graph, we show that $k$-opt Optimization in $n$-vertex graphs of maximum degree $\mathcal{O}(1)$ can be solved in time $\mathcal{O}(n^{(23/135+\epsilon_k)k}) = \mathcal{O}(n^{(0.1704+\epsilon_k)k})$, where $\lim_{k\to\infty} \epsilon_k = 0$. This improves on the behavior for general graphs, where the current-best running time [3] is $\mathcal{O}(n^{(1/4+\epsilon_k)k})$.

Since quasi-linear running times are most useful for dealing with large inputs, we perform a fine-grained analysis of the range of $k$ for which improving $k$-moves can be found in time $\mathcal{O}(n \operatorname{polylog} n)$ on $n$-vertex graphs. Observe that in the bounded-degree setting, the number of edges $m$ is $\mathcal{O}(n)$. We prove lower bounds using the hypothesis that detecting a triangle in an unweighted graph cannot be done in nearly-linear time in the number of edges $m$, which was formulated in several ways by Abboud and Vassilevska Williams [1, Conjectures 2–3]. By an efficient reduction from Triangle Detection, we show that an algorithm with running time $\mathcal{O}(n \operatorname{polylog} n)$ for 9-opt Detection in subcubic graphs with weights $\{1,2\}$ implies that a triangle in an $m$-edge graph can be found in time $\mathcal{O}(m \operatorname{polylog} m)$, contradicting popular conjectures. We complement these lower bounds by quasi-linear algorithms for all $k \leq 8$ to obtain a complete dichotomy for the case of integer weights bounded by $\mathcal{O}(\operatorname{polylog} n)$. When the weights are not bounded, we obtain quasi-linear time algorithms for all $k \leq 7$, leaving open only the case $k = 8$.

## 1.3 Organization

Preliminaries are presented in Section 2. In Section 3 we give faster XP algorithms for varying $k$. By refining these ideas, we give quasi-linear-time algorithms for $k \leq 8$ in Section 4. Section 5 gives the reduction from Triangle Detection to establish a superlinear lower bound on subcubic graphs for $k = 9$. In Section 6 we describe the lower bound for varying $k$.

## 2 Preliminaries

Given a graph $G$ edge-weighted by $w\colon E(G) \to \mathbb{Z}$, and a subset $F \subseteq E(G)$ of its edges, $w(F) := \sum_{e \in F} w(e)$. A *$k$-move* on a Hamiltonian cycle $\mathcal{C}$ is pair $(E^-, E^+)$ of edge sets such that $|E^-| = |E^+| = k$ and $(\mathcal{C} \setminus E^-) \cup E^+$ is also a Hamiltonian cycle. A $k$-move is called *improving* if $w((\mathcal{C} \setminus E^-) \cup E^+) < w(\mathcal{C})$, or equivalently and more simply $w(E^+) < w(E^-)$. A necessary condition for a pair $(E^-, E^+)$ to be a $k$-move is that the multiset of endpoints of $E^-$ is equal to the multiset of endpoints of $E^+$. An exchange $(E^-, E^+)$ that satisfies this condition is called a *$k$-swap*. We say that a $k$-swap *results* in the graph $(\mathcal{C} \setminus E^-) \cup E^+$. Note that a $k$-swap always results in a spanning disjoint union of cycles. A $k$-swap resulting in a graph with a single connected component is therefore a $k$-move. An *infeasible* $k$-swap is a $k$-swap which is not a $k$-move.

We say that a $k$-swap $(E^-, E^+)$ *induces* the graph $E^- \cup E^+$. As a slight abuse of notation, a $k$-swap will sometimes directly refer to this graph. A $k$-swap $(E^-, E^+)$ such that all edges $E^- \cup E^+$ are visited by a single closed walk alternating between $E^-$ and $E^+$ is called *sequential*. In particular, in a simple graph, every 2-swap is sequential. One can notice that an infeasible (sequential) 2-swap results in a disjoint union of exactly two cycles. A $k$-move can always be decomposed into sequential $k_i$-swaps (with $\sum k_i = k$) but some $k$-moves cannot be decomposed into sequential $k_i$-moves. The quantity $w(E^-) - w(E^+)$ is called the *gain* of the swap $(E^-, E^+)$. We distinguish *neutral* swaps, with gain 0, *improving* swaps, with strictly positive gain, and *worsening* swaps, with strictly negative gain.

For an integer $n$, we denote $[n] = \{1, \ldots, n\}$. A *$k$-embedding* (or shortly: *embedding*) is an increasing function $f\colon [k] \to [n]$. A *connection $k$-pattern* (or shortly: *connection pattern*) is a perfect matching in the complete graph on the vertex set $[2k]$. A pair $(f, M)$ where $f$ is a $k$-embedding and $M$ is a connection $k$-pattern, is an alternative description of a $k$-swap. Indeed, let $e_1, \ldots, e_n$ be subsequent edges of $\mathcal{C}$. Then, $E^- = \{e_{f(i)}\colon i \in [k]\}$. Vertices of the connection pattern correspond to endpoints of $E^-$, i.e., vertices $2i-1, 2i \in [2k]$ correspond to the left and right (in the clockwise order) endpoint of $e_{f(i)}$, respectively. Thus, edges of the connection pattern correspond to a set $E^+$ of $|M|$ edges in $G$. We say that a $k$-swap $(E^-, E^+)$ *fits into* $M$ if there is an embedding $f$ such that $(f, M)$ describes $(E^-, E^+)$. Note that every pair of an embedding and a connection pattern $(f, M)$ describes exactly one swap $(E^-, E^+)$. Conversely, for a swap $(E^-, E^+)$ the corresponding embedding $f$ is also unique (and determined by $E^-$). However, in case $E^-$ contains incident edges, the swap fits into more than one matching $M$ (see Fig. 1). See [3] for a more formal description of the equivalence.

The notion of a connection pattern can be extended to represent $k'$-swaps, for $k' < k$, as follows. Note that a matching $N$ in the complete graph on the vertex set $[2k]$ corresponds to an $|N|$-swap if and only if there is a set $\iota(N) \subseteq [k]$ such that $V(N) = \{2i-1, 2i\colon i \in \iota(N)\}$. For a set $X \subseteq [k]$, by $M[X]$ we denote the swap $N$ such that $\iota(N) = X$. We say that a connection pattern $M$ *decomposes* into swaps $N_1, \ldots, N_t$ when $M = \biguplus_{i=1}^{t} N_i$ and each $N_i$ is a connection pattern of a swap. The notion of fitting extends to $k'$-swaps in the natural way.

Consider a connection pattern $N$ of a swap, for $V(N) \subseteq [2k]$. We call $N$ *sequential* if $N \cup \{\{2i-1, 2i\}\colon i \in \iota(N)\}$ forms a simple cycle. In particular, every connection pattern can be decomposed into sequential connection patterns of (possibly shorter) swaps. The correspondence between sequential swaps and sequential connection pattern is somewhat delicate, so let us explain it in detail.

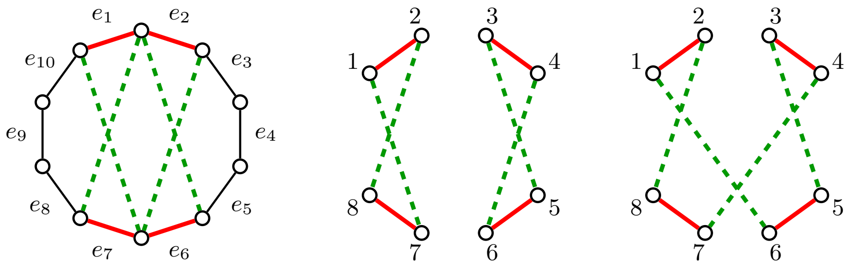



**Figure 1** A sequential swap (left) which fits two connection patterns (center, right). The pattern in the center is not sequential, while the pattern on the right is sequential. On the left the solid red edges are in $E^-$ , the dashed green edges are in $E^+$, and the thin black edges are the remaining edges of the Hamiltonian cycle $\mathcal{C}$. In the central and right pictures, the dashed green edges form some connection patterns.

Let $N$ be a sequential connection pattern, $V(N) \subseteq [2k]$. Recall that for every embedding $f$ there is exactly one $|N|$-swap $(E^-, E^+)$ that fits into $N$. Clearly, this swap is sequential, since every edge in $\{\{2i-1, 2i\} : i \in \iota(N)\}$ corresponds to an edge of $E^-$ and every edge in $N$ corresponds to an edge in $E^+$. Thus the resulting set of edges $E^- \cup E^+$ forms a single closed walk. In particular, if the image of $f$ contains two neigboring indices $i, i+1 \in [n]$, the closed walk is not a simple cycle.

Conversely, it is possible that a sequential swap fits into a connection pattern which is not sequential, see Fig. 1 for an example. However, every sequential $\ell$-swap $(E^-, E^+)$ fits at least one sequential connection pattern. This sequential connection pattern is determined by the closed walk which certifies the sequentiality of the swap. Indeed, let $E^- = \{e_{i_1}, \ldots, e_{i_\ell}\}$, where $i_1, \ldots, i_\ell$ is an increasing sequence. Let $v_0, \ldots, v_{2\ell-1}$ be the closed walk alternating between $E^-$ and $E^+$, in particular assume that $E^- = \{v_i v_{i+1} : i \text{ is even}\}$. Consider any $i = 0, \ldots, \ell - 1$ and the corresponding edge $e_{i_j} = v_{2i} v_{2i+1}$ in $E^-$, for some $j \in [\ell]$. If $v_{2i}$ is the left endpoint of $e_{i_j}$, we put $w_{2i} = 2j - 1$ and $w_{2i+1} = 2j$, otherwise $w_{2i} = 2j$ and $w_{2i+1} = 2j - 1$. Then $w_0, \ldots, w_{2\ell-1}$ is a simple cycle and $N = \{w_i w_{i+1} : i \text{ is odd}\}$ is a sequential connection pattern. By construction, $(E^-, E^+)$ fits $N$, as required. Keeping in mind the nuances in the notions of sequential swaps and corresponding sequential connection patterns, for simplicty, we will often just say ‘a sequential swap $M$’ for a matching $M$, instead of the more formal ‘a sequential connection pattern $M$ of a swap’.

Fix a connection pattern $M$ and let $f : S \to [n]$ be a partial embedding, for some $S \subseteq [k]$. For every $j \in S$, let $v_{2j-1}$ and $v_{2j}$ be the left and right endpoint of $e_{f(j)}$, respectively. We define

$$E_f^- = \{e_{f(i)} \mid i \in S\},$$

$$E_f^+ = \{\{v_{i'}, v_{j'}\} \mid i, j \in S, i' \in \{2i-1, 2i\}, j' \in \{2j-1, 2j\}, \{i', j'\} \in M\}.$$

Then, $\text{gain}_M(f) = w(E_f^-) - w(E_f^+)$.

## 3 Fast XP algorithms

For every fixed integers $k$ and $d$, the number of sequential $k$-swaps in a graph of maximum degree $d$ is $\mathcal{O}(n)$, and we can enumerate all of them in the same running time. Therefore, we can find the best improving $k$-move that can be decomposed into at most $c$ sequential

$k$-swaps in $\mathcal{O}(n^c)$ time. Because $c$ is at most $\lfloor \frac{k}{2} \rfloor$, we obtain an $\mathcal{O}(n^{\lfloor \frac{k}{2} \rfloor})$-time algorithm for $k$-OPT OPTIMIZATION. In what follows, we will improve this naive algorithm. Below we present a relatively simple algorithm which exploits the range tree data structure [15] and achieves running time roughly the same as the more sophisticated algorithm of Cygan et al. [3] for general graphs.

▶ **Theorem 1.** *For every fixed integers $k$, $c$, and $d$, there is an $\mathcal{O}(n^{\lceil \frac{c}{2} \rceil} \operatorname{polylog} n)$-time algorithm to compute the best improving $k$-move that can be decomposed into $c$ sequential swaps in graphs of maximum degree $d$.*

**Proof.** When $c = 1$, we can use the naive algorithm. Suppose $c \geq 2$ and let $h := \lceil \frac{c}{2} \rceil$.

For each possible connection pattern $M$ consisting of $c$ sequential swaps, we find the best embedding as follows. Let $M = \bigcup_{i=1}^{c} N_i$, where each $N_i$ corresponds to a sequential swap. We split $M$ into two parts $M_L = \bigcup_{i=1}^{h} N_i$ and $M_R = \bigcup_{i=h+1}^{c} N_i$ and we define $L = \bigcup_{i=1}^{h} \iota(N_i)$ and $R = \bigcup_{i=h+1}^{c} \iota(N_i)$. Note that $L \uplus R = [k]$. Let $f_L \colon L \to [n]$ and $f_R \colon R \to [n]$ be embeddings of $L$ and $R$, respectively. The union of these two embeddings results in an embedding of $[k]$ if and only if the following conditions hold.

- For each $i \in [k-1]$ with $i \in L$ and $i+1 \in R$, $f_L(i) < f_R(i+1)$ holds.
- For each $i \in [k-1]$ with $i \in R$ and $i+1 \in L$, $f_R(i) < f_L(i+1)$ holds.

We can efficiently compute a pair of embeddings satisfying these conditions using an orthogonal range maximum data structure as follows. Let $\{l_1, \ldots, l_p\} = \{i \colon l_i \in L \text{ and } l_i + 1 \in R\}$ and let $\{r_1, \ldots, r_q\} = \{i \colon r_i - 1 \in R \text{ and } r_i \in L\}$. We first enumerate all the $|L|$-swaps that fit into $M_L$ and all the $|R|$-swaps that fit into $M_R$, in $\mathcal{O}(n^h)$ time. For each such $|L|$-swap $(f_L, M_L)$, we create a $(p+q)$-dimensional point $(f_L(l_1), \ldots, f_L(l_p), f_L(r_1), \ldots, f_L(r_q))$ with a priority $\text{gain}_{M_L}(f_L)$, and we collect these points into a data structure. It stores $\mathcal{O}(n^h)$ points. For each $|R|$-swap $(f_R, M_R)$, we query for the embedding $f_L$ of maximum priority satisfying $f_L(l_i) < f_R(l_i + 1)$ for every $i \in [p]$ and $f_R(r_i - 1) < f_L(r_i)$ for every $i \in [q]$, and we answer the pair maximizing the total gain, i.e., the sum $\text{gain}_{M_L}(f_L) + \text{gain}_{M_R}(f_R)$. Using the range tree data structure [15], each query takes $\mathcal{O}(\log^{p+q} n^h) = \mathcal{O}(\operatorname{polylog} n)$ time, so the total running time is $\mathcal{O}(n^h \operatorname{polylog} n)$. ◀

Since $c \leq \lfloor \frac{k}{2} \rfloor$ we get the following corollary.

▶ **Corollary 2.** *For all fixed integers $k$ and $d$, $k$-OPT OPTIMIZATION in graphs of maximum degree $d$ can be solved in time $\mathcal{O}(n^{\lceil \frac{k-1}{4} \rceil} \operatorname{polylog} n)$.*

Let us take another look at the proof of Theorem 1. Recall that for merging embeddings $f_L$ and $f_R$, we were interested only in values $f_L(i)$ for $i \in L$ such that $i+1 \in R$ or $i-1 \in R$. The embeddings of the remaining elements of $L$ were forgotten at that stage, but we knew that it is possible to embed them and we stored the gain of embedding them. This suggests the following, different approach. We decompose the connection pattern into sequential swaps and we scan the swaps in a carefully chosen order. Assume we scanned $t$ swaps already and there are $c-t$ swaps ahead. Assume that only $p \ll t$ of the $t$ 'boundary' swaps interact with the remaining $c-t$ swaps, where two swaps $N_1$ and $N_2$ interact when there is $i \in \iota(N_1)$ such that $i-1 \in \iota(N_2)$ or $i+1 \in \iota(N_2)$. Then it suffices to compute, for every embedding $f_L$ of the $p$ swaps, the gain of the best (i.e., giving the highest gain) embedding $g_L$ of the $t$ swaps, such that $f_L$ matches $g_L$ on the boundary swaps. This amounts to $\mathcal{O}(n^p)$ values to compute, since each sequential swap can be embedded in $\mathcal{O}(n)$ ways, if $k$ and the maximum degree are $\mathcal{O}(1)$. The idea is to (1) compute these values fast (in time linear in their number) using analogous values computed for the prefix of $t-1$ swaps, (2) find an order of swaps so that $p$ is always small, namely $p \leq (23/135 + \epsilon_k)k$. The readers familiar with the notion of pathwidth recognize that $p$ here is just the pathwidth of the graph obtained from the path $1, 2, \ldots, k$ by identifying vertices in the set $\iota(N)$ for every sequential swap $N$ in $M$, and that (2) is just dynamic programming over path decomposition. The resulting algorithm is summarized in Theorem 3, and due to space limits, its formal proof is skipped here and will be included in the full version.

▶ **Theorem 3.** *For all fixed integers $k$ and $d$, $k$-OPT OPTIMIZATION in graphs of maximum degree $d$ can be solved in time $\mathcal{O}(n^{(23/135+\epsilon_k)k}) = \mathcal{O}(n^{(0.1704+\epsilon_k)k})$, where $\lim_{k\to\infty} \epsilon_k = 0$.*

## 4 Fast algorithms for small $k$

Note that the algorithm for $k$-OPT OPTIMIZATION from Corollary 2 is quasi-linear for $k \leq 5$. In this section we extend the quasi-linear-time solvability to $k \leq 7$ for $k$-OPT DETECTION. Under an additional assumption of bounded weights, we are able to reach quasi-linear time for $k = 8$ as well, but the details of this part are deferred to the full version because of space constraints. To be precise, in the $k = 7$ case we prove the following stronger statement than just finding an arbitrary improving $k$-move.

▶ **Theorem 4.** *For $k \leq 7$, there is a quasi-linear-time algorithm to compute the best improving $k$-move in bounded-degree graphs under the assumption that there are no improving $k'$-moves for $k' < k$.*

We say that a connection pattern $M$ of $k$-swaps is *reducible* if it can be decomposed into two moves. Note that if $M$ is improving, then at least one of the two moves is improving, contradicting the assumption of Theorem 4.

▷ Observation 5. If there are no improving $k'$-moves for $k' < k$, then no improving $k$-swap fits into a reducible connection pattern.

Before we formulate our algorithm, we need two lemmas. We can prove these lemmas by case analysis, and because of the space constraints, their proofs are skipped here and will be included in the full version. Let $M[X]$ and $M[Y]$ be two swaps in a connection pattern $M$, for some disjoint $X, Y \subseteq [k]$. *Interaction* between $M[X]$ and $M[Y]$ is any $i \in [k-1]$ such that $i \in X$ and $i + 1 \in Y$ or $i \in Y$ and $i + 1 \in X$.

▶ **Lemma 6.** *For any $k \geq 6$, there is no feasible and irreducible connection $k$-pattern that contains two 2-swaps that interact at least twice.*

Let $M$ be a connection pattern, i.e., a perfect matching on vertices $[2k]$. We say that $M'$ is obtained from $M$ by swapping $i$ and $i + 1$, for $i \in [k]$, when $M'$ is obtained from $M$ by swapping the mates of $2i - 1$ and $2i + 1$ and swapping the mates of $2i$ and $2i + 2$.

▶ **Lemma 7.** *Let $M$ be a feasible irreducible connection $k$-pattern. Assume that $M$ decomposes into three sequential swaps $M[X]$, $M[Y]$, and $M[Z]$, such that $|X| = |Y| = 2$. If there is exactly one index $i \in [k-1]$ with $i \in X$ and $i+1 \in Y$ or $i \in Y$ and $i+1 \in X$, the connection pattern $M'$ obtained from $M$ by swapping $i$ and $i + 1$ is either feasible or reducible.*

Now we are ready to describe the algorithm from Theorem 4 (see also Pseudocode 1). For each feasible and irreducible connection $k$-pattern $M$, we compute the best embedding as follows. If $M$ consists of at most two sequential swaps, we can use the algorithm in Theorem 1. Otherwise, $M$ consists of three sequential swaps $M[X]$, $M[Y]$, $M[Z]$ such that

**Pseudocode 1** Quasi-linear-time algorithm for $k \leq 7$

**for** each feasible irreducible connection $k$-pattern $M$ **do**
    **if** $M$ consists of at most two sequential swaps **then**
        Apply the algorithm in Theorem 1.
    **else**
        Let $M = M[X] \uplus M[Y] \uplus M[Z]$ where $|X| = |Y| = 2$ and $|Z| = k - 4$.
        **if** there are no interactions between $X$ and $Y$ **then**
            **for** each embedding $f_Z$ for $Z$ **do**
                Independently compute the best embeddings $f_X$ for $X$ and $f_Y$ for $Y$.
        **else**
            Relax the constraint $f_X(i) < f_Y(i+1)$ to $f_X(i) \neq f_Y(i+1)$.
            **for** each embedding $f_Z$ for $Z$ **do**
                Compute the best pair $(f_X, f_Y)$ satisfying the relaxed constraints.

$X \uplus Y \uplus Z = [k]$, $|X| = |Y| = 2$ and $|Z| = k - 4$. For each embedding $f_X : X \to [n]$ of $X = \{i, j\}$ we create a 2-dimensional point $(f_X(i), f_X(j))$ with priority $\mathrm{gain}_X(f_X)$ and we put all the points in a range tree data structure $D_X$ [15]. We build an analogous data structure for $Y$. Next, for each embedding $f_Z$ for $Z$, we compute the best pair of embeddings $(f_X, f_Y)$ for $X$ and $Y$ as follows.

If there are no interactions between $X$ and $Y$, we can find the best pair in $\mathcal{O}(\mathrm{polylog}\, n)$ time by independently picking the best embeddings for $X$ and $Y$ by querying the range trees $D_X$ and $D_Y$. Indeed, first note that there is no index $i \in [k-1]$ such that $X = \{i, i+1\}$ because in such a case, both the 2-swap and the remaining $(k-2)$-swap have to be feasible (similarly for $Y$). Since there are no interactions between $X$ and $Y$, we must have $i - 1 \in Z \cup \{0\}$ and $i+1 \in Z \cup \{k+1\}$ for every $i \in X \cup Y$. To find the best embedding $f_X$ of $X = \{i, j\}$, we query $D_X$ with the constraints $f_Z(i-1) < f_X(i) < f_Z(i+1)$ and $f_Z(j-1) < f_X(j) < f_Z(j+1)$, where we define $f_Z(0) := 0$ and $f_Z(k+1) := n+1$. We proceed analogously for $Y$.

Finally, assume there are interactions between $X$ and $Y$, so from Lemma 6, there is exactly one interaction. W.l.o.g. $i \in X$ and $i+1 \in Y$. Note that $i-1 \in Z \cup \{0\}$ and $i+2 \in Z \cup \{k+1\}$. We first relax the constraint $f_Z(i-1) < f_X(i) < f_Y(i+1) < f_Z(i+2)$, where we define $f_Z(0) := 0$ and $f_Z(k+1) := n+1$, to three constraints $f_Z(i-1) < f_X(i) < f_Z(i+2)$, $f_Z(i-1) < f_Y(i+1) < f_Z(i+2)$, and $f_X(i) \neq f_Y(i+1)$. We then drop the disturbing inequality constraint $f_X(i) \neq f_Y(i+1)$ by color-coding[1]. We color each vertex in $[n]$ in red or blue, and we independently pick the best embedding for $X$ (resp. $Y$) that uses only red (resp. blue) vertices. By using a family of perfect hash functions [5], we can construct a set of $\mathcal{O}(\log^2 n)$ colorings such that, for every pair of embeddings $f_X$ and $f_Y$, there is at least one coloring that colors all the vertices in $f_X$ red and all the vertices in $f_Y$ blue.

We now obtain the best pair of embeddings $(f_X, f_Y)$ satisfying the relaxed constraints. If the obtained $k$-swap is not improving, we immediately know that there are no improving $k$-moves that fit into $M$. If it is improving and satisfies the original constraint, we are done. Finally, if it is improving but does not satisfy the original constraint, it fits into the connection pattern $M'$ that is obtained from $M$ by swapping $i$ and $i+1$. By Lemma 7, $M'$

[1] Instead of color-coding, we can adapt the range tree to support orthogonal range maximum queries with an additional constraint of the form $x \neq i$ by keeping one additional point in each node. With this approach, we can avoid the additional $\log^2 n$ factor. Because this paper does not focus on optimizing the $\mathrm{polylog}\, n$ factor, we do not touch on the details.

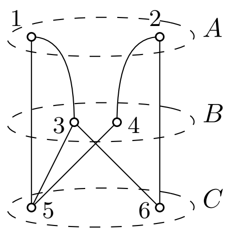



**Figure 2** An instance of TRIANGLE DETECTION

is either feasible or reducible. Because no improving $k$-swaps fit into reducible connection patterns, $M'$ has to be feasible. We therefore obtain a $k$-move that is as good as the best $k$-move that fits into $M$. This completes the proof of Theorem 4.

We finally consider the case of $k = 8$. Note that, because Lemma 6 and 7 do not assume $k \le 7$, the above algorithm can also compute the best improving $k$-move that can be decomposed into three sequential swaps of size $(2, 2, k-4)$ for any fixed $k$ under the same assumption. Moreover, any connection patterns of 8-moves consisting of four 2-swaps are reducible because it always induces a pair of two 2-swaps that interact at least twice. The remaining case for $k = 8$ is only when the 8-move can be decomposed into three sequential swaps of size $(2, 3, 3)$. In order to tackle this case, we exploit the bounded-weight assumption as follows. For each connection pattern $M = M[X] \uplus M[Y] \uplus M[Z]$ with $|X| = 2$ and $|Y| = |Z| = 3$, and for each embedding $f_Z$ for $Z$, we want to compute the best pair of embeddings $f_X$ for $X$ and $f_Y$ for $Y$. When all the weights are integers from $[W]$, the gain of $(f_X, M[X])$ is an integer from $[-2W, 2W]$, and the gain $(f_Y, M[Y])$ is an integer from $[-3W, 3W]$. We therefore have only $\mathcal{O}(W^2)$ pairs of gains. By guessing the pair of gains, the query of finding the *best* pair can be reduced to the query of finding an *arbitrary* pair, and the latter query can be efficiently answered by adapting the range tree. This leads to the following algorithm, whose detailed description is skipped here and will be included in the full version.

▶ **Theorem 8.** *When all the weights are integers from $[W]$, there is an $\mathcal{O}(W^2 n \operatorname{polylog} n)$-time algorithm to compute the best improving 8-move under the assumption that there are no improving $k'$-moves for $k' < 8$.*

## 5 Lower bound for $k = 9$

The starting point for our reduction is the following problem (see Fig. 2 for an exemplary instance).

> TRIANGLE DETECTION **Parameter:** $m := |E(H)|$.
> **Input:** An undirected graph $H$ whose vertex set $V(H)$ is partitioned into $A \cup B \cup C$.
> **Question:** Is there a triple $(a, b, c) \in A \times B \times C$ such that $\{ab, ac, bc\} \subseteq E(H)$?

We assume without loss of generality that $A$, $B$, and $C$ are three independent sets, so that finding such a triple is equivalent to finding a triangle in the graph $H$. By simple reductions that incur only a constant blow-up in the number of vertices and edges, this problem is equivalent to determining whether a graph has a triangle or not.

▷ Assumption 1 (Triangle hypothesis [1]). There is a fixed $\delta > 0$ such that, in the Word RAM model with words of $\mathcal{O}(\log n)$ bits, any algorithm requires $m^{1+\delta-o(1)}$ time in expectation to detect whether an $m$-edge graph contains a triangle.

It should be noted that one can solve Triangle Detection in time $\mathcal{O}(n^\omega)$ where $n$ is the number of vertices and $\omega \leq 2.373$ is the best-known exponent for matrix multiplication. Alon et al. [2] found an elegant win-win argument to solve Triangle Detection in time $\mathcal{O}(m^{\frac{2\omega}{\omega+1}})$: the 3-vertex paths in which the middle vertex has degree less than $m^{\frac{\omega-1}{\omega+1}}$ can be listed in time $\mathcal{O}(m \cdot m^{\frac{\omega-1}{\omega+1}}) = \mathcal{O}(m^{\frac{2\omega}{\omega+1}})$, and for each, one can check if they form a triangle, whereas the number of vertices of degree greater than $m^{\frac{\omega-1}{\omega+1}}$ is at most $m^{\frac{2}{\omega+1}}$, so one can detect a triangle in time $\mathcal{O}(m^{\frac{2\omega}{\omega+1}})$ in the subgraph that they induce. After more than two decades, this is still the best worst-case running time (when $n^\omega = \Omega(m^{\frac{2\omega}{\omega+1}})$). This suggests that the triangle hypothesis is likely to hold. Moreover, if one thinks that the above scheme yields the best possible running time and that $\omega$ will eventually reach 2, then exponent 4/3 could be the *right answer* for Triangle Detection parameterized by the number of edges. The following is implied by [1, Conjecture 2] (since $\omega \geq 2$), in the regime $m = \Theta(n^{3/2})$ (so that $\mathcal{O}(n^2)$ and $\mathcal{O}(m^{4/3})$ coincide).

▷ Assumption 2. In the Word RAM model with words of $\mathcal{O}(\log n)$ bits, any algorithm requires $m^{4/3-o(1)}$ time in expectation to detect whether an $m$-edge $\Theta(m^{2/3})$-node graph contains a triangle.

We show that Subcubic 9-opt Detection parameterized by the number of vertices is as hard as Triangle Detection parameterized by the number of edges, by providing a linear-time reduction from the latter to the former. In light of Theorem 4, this implies that Bounded-Degree 8-opt Detection is the only remaining open case where a quasi-linear algorithm is not known but also not ruled out by a standard fine-grained complexity assumption.

▶ **Lemma 9.** *There is an $\mathcal{O}(m)$-time reduction from* Triangle Detection *on $m$-edge graphs to* Subcubic 9-opt Detection *on $\mathcal{O}(m)$-vertex undirected graphs with edge weights in $\{1, 2\}$.*

**Proof.** From a tripartitioned instance of Triangle Detection $H = (A \cup B \cup C, E(H))$ with $m$ edges, we build a subcubic graph $G$ with $\Theta(m)$ vertices, an edge-weight function $w : E(G) \to \{1, 2\}$, and a Hamiltonian cycle $\mathcal{C}$. From $\mathcal{C}$, there is a swap of up to 9 edges (i.e., up to 9 deletions and the same number of additions) which results in a lighter Hamiltonian cycle if and only if $H$ has a triangle.

**Overall construction of $G$.**

We will build $G$ by adding chords to the cycle $\mathcal{C}$. Henceforth, a *chord* is an edge of $G$ which is not in $\mathcal{C}$. It is helpful to think $\mathcal{C}$ as a (subdivided) triangle whose three sides correspond to $A$, $B$, and $C$, which we call the $A$-side (left), $B$-side (right), and $C$-side (bottom), respectively. We will only name the edges of $G$ (and not the vertices), since the problem is more efficiently described in terms of edges. We will define some sequential 3-swaps (we recall that a sequential $i$-swap is a closed walk of length $2i$ alternating edges of $E(\mathcal{C})$ and edges of $E(G) \setminus E(\mathcal{C})$). Eventually, all the edges that are not in a described sequential 3-swap are incident to a vertex of degree 2, making them undeletable. (One can also enforce that by subdividing every irrelevant edge once.)

The improving 9-move, should there be a triangle $abc$ in $H$, will consist of a sequence of three 3-swaps. More precisely, it consists of one improving 3-swap, which splits $\mathcal{C}$ into three cycles respectively containing:

**(1)** a part of the vertex gadget of some $a \in A$,

**(2)** the part of the $B$-side below the vertex gadget of $b$, as well as the $C$-side, and
**(3)** the part of the $B$-side above the vertex gadget of some $b \in N_H(a) \cap B$.

This decreases the total weight by 1. Then a neutral 3-swap reconnects (1) and (2) together, but also detaches (4) a part of the vertex gadget of some $c \in N_H(a) \cap C$. Finally a neutral 3-swap glues (3), (1)+(2), and (4) together, provided $bc \in E(H)$. This results in a new Hamiltonian cycle of length $w(\mathcal{C}) - 1$.

There will be relatively few edges of weight 2. To simplify the presentation, every edge is of weight 1, unless specified otherwise. Let $\vec{H}$ be the directed graph obtained from $H$ by orienting its edges from $A$ to $B$, from $B$ to $C$, and from $C$ to $A$. Note that finding a directed triangle in $\vec{H}$ is equivalent to finding a triangle in $H$.

## Vertex scopes, extended scopes, and nested chords.

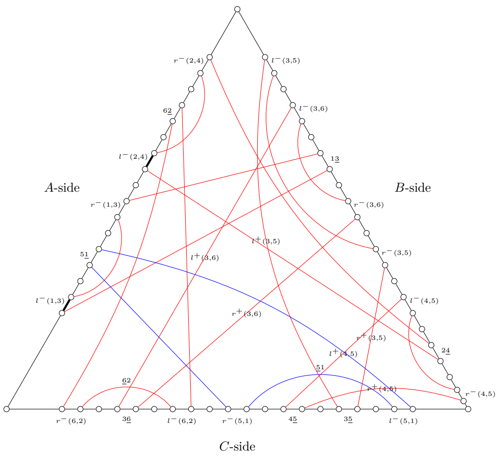
For $(X, Y) \in \{(A, B), (B, C), (C, A)\}$, we set $Z := \{A, B, C\} \setminus \{X, Y\}$ and we do the following as a preparatory step to encode the arcs of $\vec{H}$. Each vertex $v \in X$ is given a (pairwise vertex-disjoint) subpath $I_v$ of $\mathcal{C}$, called the *extended scope* of $v$, with $|I_v| := 6(|N_H(v) \cap Y|) + 3(|N_H(v) \cap Z|) - 1$ vertices. We think of $I_v$ as being displayed from left to right with the leftmost vertex of index 1, and the rightmost one of index $|I_v|$. The extended scopes of the vertices of $A$, $B$, and $C$ occupy respectively the $A$-side, $B$-side, and $C$-side. In what follows, it will be more convenient to have a *circular* notion of *left* and *right*. Starting from the bottom corner of the $A$-side, and going clockwise to the top corner of the $A$-side, then down to the bottom corner of the $B$-side, the relative left and right within the $A$-side and the $B$-side coincide with the usual notion as displayed in Figure 3a. But then closing the loop from the right corner of the $C$-side to its left corner, left and right are switched: the closer to the bottom corner of $A$ (resp. $B$), the more "right" (resp. "left").

Each vertex $v \in X$ has $|N_H(v) \cap Y|$ nested chords spaced out every three vertices. More precisely, the second vertex of $I_v$ is adjacent to the penultimate, the fifth to the one of index $|I_v| - 4$, the eighth to the one of index $|I_v| - 7$, and so on, until $|N_H(v) \cap Y|$ chords are drawn. Each of these chords is associated to an edge $vy \in E(\{v\}, Y)$, and is denoted by $\underline{vy}$. A vertex just to the right of the left endpoint, or just to the left of the right endpoint, of such a chord will remain of degree 2 in $G$. This is the case of the vertices of index $3, 6, \ldots$ and $|I_v| - 2, |I_v| - 5, \ldots$ in $I_v$. We call $l^-(v, y)$ (resp. $r^-(v, y)$) the edge of $I_v$ incident to both the left endpoint of $\underline{vy}$ and the vertex just to its left (resp. right endpoint of $\underline{vy}$ and the vertex just to its right). Both endpoints of $l^-(v, y)$ and of $r^-(v, y)$ will eventually have degree 3 in $G$.

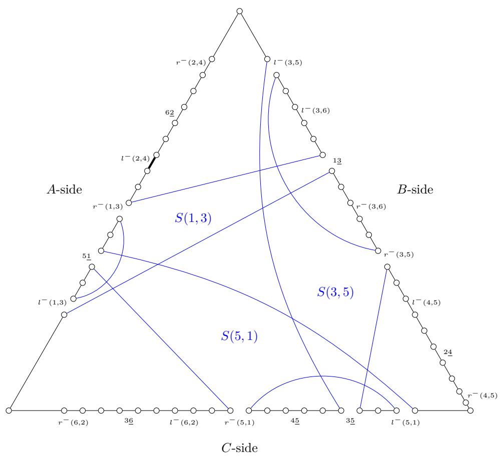
The chord linking the most distant vertices in $I_v$ is called the *outermost* chord, while the one linking the closest pair is called the *innermost* chord. We also say that a chord $e$ is *wider* than a chord $e'$ if $e$ links a farther pair on $I_v$ than $e'$ does. The central path $J_v \subset I_v$ on $|I_v| - (6|N_H(v) \cap Y| - 4) = 3(|N_H(v) \cap Z| + 1)$ vertices, surrounded by the innermost chord, is called the *scope* of $v$. We map in one-to-one correspondence the edges of $E(\{v\}, Z)$ to every three edges of $J_v$ starting from the third edge (that is, the third, sixth, and so on). Note that we have the exact space to do so, since $|J_v| = 3(|N_H(v) \cap Z| + 1)$. We denote by $z\underline{v}$ the edge in $J_v$ corresponding to the edge $vz \in E(\{v\}, Z)$.

## Encoding the arcs of $\vec{H}$.

The last step to encode the arcs of $\vec{H}$, or equivalently the edges of $H$, is the following. Keeping the notations of the previous paragraphs, for every edge $xy \in E(X, Y)$, we add two chords (of weight 1): one chord $l^+(x, y)$ between the left endpoint of $l^-(x, y)$ and the right




**(a)** The construction for the instance of Figure 2. Edges of $\mathcal{C}$ are in black, chords are in red, bold edges are the ones with weight 2. The three chords in blue are the edges to add to perform the neutral 3-swap $S(5,1)$ of $S(C,A)$.




**(b)** The 9-move corresponding to the triangle 135 results in a Hamiltonian cycle using one less edge of weight 2. Note that after the swaps $S(1,3)$ and $S(5,1)$ are performed, the only 3-swap that can reconnect the three cycles into one, is $S(3,5)$, implying the existence of the edge 35, and thereby of the triangle 135.

**Figure 3** Illustration of the reduction (left) and of a potential solution (right).

endpoint of $x\underline{y}$ and one chord $r^+(x,y)$ between the right endpoint of $r^-(x,y)$ and the left endpoint of $x\underline{y}$. We finish the construction of $G$ (and $\mathcal{C}$) by subdividing each edge between consecutive extended scopes once, to make the resulting edges undeletable. The edges $l^-(a,b)$ for $(a,b) \in A \times B$ get weight 2, while all the other edges of $E(G)$ get weight 1. This finishes the construction of $(G, w, \mathcal{C})$. See Figure 3a for an illustration.

### Improving and neutral 3-swaps.

For each $(x,y) \in E(\vec{H})$, we denote by $S(x,y)$ the 3-swap $(\{x\underline{y}, l^-(x,y), r^-(x,y)\}, \{\underline{x}y, l^+(x,y), r^+(x,y)\})$. For $(X,Y) \in \{(A,B), (B,C), (C,A)\}$, we define the set of 3-swaps $S(X,Y) := \bigcup_{xy \in E(X,Y)} S(x,y)$, and $\mathcal{S} := S(A,B) \cup S(B,C) \cup S(C,A)$.

Note that all the 3-swaps of $S(A,B)$ are improving. They gain 1 since $l^-(a,b)$ has weight 2 for any $(a,b) \in A \times B$. On the other hand, all the 3-swaps of $S(B,C)$ and $S(C,A)$ are neutral. The edges added in swaps of $\mathcal{S}$ partition the chords of $G$, and the open neighborhood of the six vertices involved in every swap are six vertices of degree 2 in $G$. Therefore, all the possible swaps are in the set $\mathcal{S}$, they are on vertex-disjoint sets of vertices, and any move is a sequence of 3-swaps of $\mathcal{S}$.

The vertices of $\mathcal{C}$ are incident to at most one chord. Hence the graph $G$ is subcubic. It has $\sum_{v \in V(H)} 1 + |I_v| \leqslant 9|E(H)| + |V(H)| = \Theta(m)$ vertices and $(G, w, \mathcal{C})$ takes $\Theta(m)$-time to build. To summarize, we defined a linear reduction from Triangle Detection with parameter $m$ to Subcubic 9-opt Detection with parameter $n$. So a quasi-linear algorithm for the latter would yield an unlikely quasi-linear algorithm for the former. We now check that the reduction is correct.

**A triangle in $H$ implies an improving 9-move for $(G, w, \mathcal{C})$.**

Let $abc$ be a triangle in $H$. In particular, all three swaps $S(a,b)$, $S(b,c)$, and $S(c,a)$ exist. Performing these three 3-swaps results in a spanning union of (vertex-disjoint) cycles, whose total weight is $w(\mathcal{C}) - 1$. Indeed $S(a,b)$ is swap of weight $-1$, while $S(b,c)$, and $S(c,a)$ are both neutral.

We thus only need to show that the three swaps result in a connected graph (hence, Hamiltonian cycle of lighter weight). By performing the 3-swap $S(a,b)$, we create three components: (1) one on a vertex set $K_{a,b}$ such that $J_a \subseteq K_{a,b} \subseteq I_a$, (2) one containing the scopes of vertices of the $B$-side to the right (lower part) of the scope of $b$, and (3) one containing the scopes of vertices of the $B$-side to the left (upper part) of the scope of $b$. Then the swap $S(c,a)$ glues (1) and (2) together, but also disconnects (4) a cycle on a vertex set $K_{c,a}$ such that $J_c \subseteq K_{c,a} \subseteq I_c$. At this point, there are three cycles: (3), (1)+(2), and (4). It turns out that the 3-swap $S(b,c)$ deletes exactly one edge in each of these three cycles: $b\underline{c}$ in (4), $l^-(b,c)$ in (3), and $r^-(b,c)$ in (1)+(2). Therefore, $S(b,c)$ reconnects these three components into one Hamiltonian cycle.

**An improving k-move for $(G, w, \mathcal{C})$ with $k \leqslant 9$ implies a triangle in $H$.**

We assume that there is an improving $k$-move $\mathcal{M} = (E^-, E^+)$ for $(G, w, \mathcal{C})$ with $k \leqslant 9$. Being improving, the $k$-move has to contain at least one improving 3-swap of $S(A,B)$. Let $S(a,b)$ be a 3-swap of $S(A,B)$ in $\mathcal{M}$ such that for every other (improving) 3-swap $S(a,b')$ in $\mathcal{M}$, the chord $\underline{a}b'$ is wider than $\underline{a}b$. Since $S(a,b)$ exists, it holds in particular that $ab \in E(H)$. Performing $S(a,b)$ results in the union of three cycles: (1) on a vertex set $K_{a,b}$ with $J_a \subseteq K_{a,b} \subseteq I_a$, and cycles (2) and (3) as described in the previous paragraph.

By the choice of $b$, the only remaining swaps of $\mathcal{M}$ touching $K_{a,b}$ are in $S(C,A)$. So $\mathcal{M}$ has to contain a neutral 3-swap $S(c,a)$ for some $c \in C$. This implies that $ac \in E(H)$. Performing this swap results in three cycles: (3), (1)+(2), and (4), as described above. To reconnect all three components into one Hamiltonian cycle, the 3-swap has to delete exactly one edge in (3), (1)+(2), and (4). The only 3-swap that does so is $S(b,c)$. This finally implies that $bc \in E(H)$. Thus $abc$ is a triangle in $H$. ◀

We obtain the following theorem as a direct consequence of the previous lemma.

▶ **Theorem 10.** SUBCUBIC 9-OPT DETECTION *requires time:*
**(1)** $n^{1+\delta-o(1)}$ *for a fixed* $\delta > 0$*, under the triangle hypothesis, and*
**(2)** $n^{4/3-o(1)}$*, under the strong triangle hypothesis,*
*in expectation, even in undirected graphs with edge weights in* $\{1, 2\}$*.*

If we use general integral weights and not just $\{1, 2\}$, we can show a stronger lower bound, by reducing from NEGATIVE EDGE-WEIGHTED TRIANGLE. Again, we can assume that the instance is partitioned into three sets $A$, $B$, $C$, and we look for a triangle $abc$ such that $w'(ab) + w'(bc) + w'(ac) < 0$, where $w'$ gives an integral weight to each edge. A truly subcubic (in the number of vertices) algorithm for this problem would imply one for ALL-PAIRS SHORTEST PATHS, which would be considered a major breakthrough. The assumption that such an algorithm is not possible is called the APSP hypothesis.

We only change the above construction in the weight of the edges $l^-(x,y)$. Now each edge $l^-(x,y)$ gets weight $-w'(xy)$. From a NEGATIVE EDGE-WEIGHTED TRIANGLE-instance with $n$ vertices, we obtain an equivalent instance of SUBCUBIC 9-OPT DETECTION with $\mathcal{O}(n^2)$ vertices, in time $\mathcal{O}(n^2)$. So we derive the following.

▶ **Theorem 11.** Subcubic 9-opt Detection *requires time* $n^{3/2-o(1)}$*, under the APSP hypothesis.*

## 6 Lower bound for varying $k$

In this section we describe the main ideas behind the lower bound for $k$-opt Detection in subcubic graphs for varying $k$. The details are deferred to the full version due to space restrictions. The overall approach is similar to the lower bound of Guo et al. [6], in that we give a linear-parameter reduction from the $k$-Partitioned Subgraph Isomorphism problem parameterized by the number of edges $k$. Marx [14] proved that, assuming the Exponential Time Hypothesis, the problem cannot be solved in time $f(k) \cdot n^{o(k/\log k)}$ for any function $f$.

The instance created in the reduction of Guo et al. [6] may contain vertices of arbitrarily large degrees. To obtain such a reduction to $k$-opt Detection in subcubic graphs, an essential ingredient is a *choice gadget* with terminal pairs $(x_0, y_0), \ldots, (x_\ell, y_\ell)$ which enforces that sufficiently cheap Hamiltonian cycles that enter at $x_i$, must leave via the corresponding $y_i$. The gadget can be implemented by suitable weight settings and vertices of degree at most three. This gadget allows us to enforce synchronization properties, which enforce that an improved Hamiltonian cycle first selects which vertices to use in the image of the subgraph isomorphism, and then selects incident edges for each selected vertex. By carefully coordinating the gadgets, this allows us to implement the hardness proof by an edge selector strategy. It leads to a proof of the following theorem.

▶ **Theorem 12.** *There is no function* $f$ *for which* $k$-opt Detection *on* $n$*-vertex graphs of maximum degree* 3 *with edge weights in* $\{1, 2\}$ *can be solved in time* $f(k) \cdot n^{o(k/\log k)}$*, unless ETH fails.*

We remark that the lower bound also holds for *permissive* local search algorithms which output an improved Hamiltonian cycle of arbitrarily large Hamming distance to the starting cycle $\mathcal{C}$, if a cheaper cycle exists in the $k$-opt neighborhood of $\mathcal{C}$.

## References


1 Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *Proc. 55th FOCS*, pages 434–443. IEEE Computer Society, 2014. doi:10.1109/FOCS.2014.53.

2 Noga Alon, Raphael Yuster, and Uri Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997. doi:10.1007/BF02523189.

3 Marek Cygan, Lukasz Kowalik, and Arkadiusz Socala. Improving TSP tours using dynamic programming over tree decompositions. In *Proc. 25th ESA*, volume 87 of *LIPIcs*, pages 30:1–30:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.ESA.2017.30.

4 Mark de Berg, Kevin Buchin, Bart M. P. Jansen, and Gerhard J. Woeginger. Fine-grained complexity analysis of two classic TSP variants. In *ICALP*, volume 55 of *LIPIcs*, pages 5:1–5:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

5 Michael L. Fredman, János Komlós, and Endre Szemerédi. Storing a sparse table with $O(1)$ worst case access time. *J. ACM*, 31(3):538–544, 1984. doi:10.1145/828.1884.

6 Jiong Guo, Sepp Hartung, Rolf Niedermeier, and Ondrej Suchý. The parameterized complexity of local search for TSP, more refined. *Algorithmica*, 67(1):89–110, 2013. doi:10.1007/s00453-012-9685-8.

**7** Michael Held and Richard M. Karp. The traveling-salesman problem and minimum spanning trees: Part II. *Math. Program.*, 1(1):6–25, 1971.

**8** Keld Helsgaun. An effective implementation of the Lin-Kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106 – 130, 2000. doi:10.1016/S0377-2217(99)00284-2.

**9** Keld Helsgaun. General $k$-opt submoves for the Lin-Kernighan TSP heuristic. *Math. Program. Comput.*, 1(2-3):119–163, 2009.

**10** D. S. Johnson and L. A. McGeoch. Experimental analysis of heuristics for the STSP. In G. Gutin and A. Punnen, editors, *The Traveling Salesman Problem and its Variations*, pages 369–443. Kluwer Academic Publishers, Dordrecht, 2002.

**11** D.S. Johnson and L.A McGeoch. The traveling salesman problem: A case study in local optimization. In E. Aarts and J.K. Lenstra, editors, *Local search in combinatorial optimization*, pages 215–310. Wiley, Chichester, 1997.

**12** S. Lin and Brian W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21(2):498–516, 1973. doi:10.1287/opre.21.2.498.

**13** Dániel Marx. Searching the $k$-change neighborhood for TSP is W[1]-hard. *Oper. Res. Lett.*, 36(1):31–36, 2008. doi:10.1016/j.orl.2007.02.008.

**14** Dániel Marx. Can you beat treewidth? *Theory of Computing*, 6(1):85–112, 2010. doi:10.4086/toc.2010.v006a005.

**15** Franco P. Preparata and Michael Ian Shamos. *Computational Geometry - An Introduction*. Texts and Monographs in Computer Science. Springer, 1985.