The k-Opt Algorithm for the Traveling Salesman Problem Has Exponential Running Time for $k \geq 5$

Sophia Heimann
Research Institute for Discrete Mathematics, University of Bonn, Germany

Hung P. Hoang
Algorithms and Complexity Group, Faculty of Informatics, TU Wien, Austria

Stefan Hougardy
Research Institute for Discrete Mathematics and Hausdorff Center for Mathematics, University of Bonn, Germany

Abstract

The k-Opt algorithm is a local search algorithm for the Traveling Salesman Problem. Starting with an initial tour, it iteratively replaces at most k edges in the tour with the same number of edges to obtain a better tour. Krentel (FOCS 1989) showed that the Traveling Salesman Problem with the k-Opt neighborhood is complete for the class PLS (polynomial time local search) and that the k-Opt algorithm can have exponential running time for any pivot rule. However, his proof requires $k \gg 1000$ and has a substantial gap. We show the two properties above for a much smaller value of k, addressing an open question by Monien, Dumrauf, and Tscheuschner (ICALP 2010). In particular, we prove the PLS-completeness for $k \geq 17$ and the exponential running time for $k \geq 5$.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases Traveling Salesman Problem, k-Opt algorithm, PLS-completeness

Digital Object Identifier 10.4230/LIPIcs.ICALP.2024.84

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2402.07061` [7]

Funding *Hung P. Hoang*: Austrian Science Foundation (FWF, project Y1329 START-Programm). *Stefan Hougardy*: Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC-2047/1 – 390685813.

Acknowledgements This work was initiated at the "Discrete Optimization" trimester program of the Hausdorff Institute of Mathematics, University of Bonn, Germany in 2021. We would like to thank the organizers for providing excellent working conditions and inspiring atmosphere.

1 Introduction

The well-known Traveling Salesman Problem (TSP) consists of finding a spanning cycle of an edge weighted complete graph, such that the total edge weight of the cycle is the smallest possible. A popular heuristic for this problem is a local search algorithm called k-Opt. Starting with an arbitrary tour, it iteratively replaces at most k edges in the tour with the same number of edges, as long as the resulting tour has smaller total edge weight. We define TSP/k-Opt to be the problem of finding a local optimum for a TSP instance with the k-Opt algorithm.

A fundamental question in the area of local search algorithms is to determine the number of iterations a given local search algorithm may need in the worst case. A local search algorithm with a specified pivot rule has the *is-exp* property if there exist problem instances and initial solutions for which the local search algorithm requires an exponential number of iterations. For example, it is well known that the Simplex algorithm for linear programming

51st International Colloquium on Automata, Languages, and Programming (ICALP 2024).
Editors: Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson;
Article No. 84; pp. 84:1–84:18


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

has the is-exp property for many different pivot rules [12, 9, 1, 6]. For TSP, Chandra, Karloff, and Tovey [2] showed that TSP/k-Opt has the is-exp property. This even holds for Euclidean TSP with the 2-Opt neighborhood [5].

For the Simplex algorithm it is not known whether there exists a pivot rule that guarantees a polynomial number of iterations. This is one of the most important open problems in the area of linear programming. Contrary to this Krentel [13] proved in 1989 that for sufficiently large values of k, TSP/k-Opt exhibits the *all-exp* property, that is, the k-Opt algorithm requires an exponential number of iterations to find a local optimum, for all possible pivot rules and for infinitely many pairs of a TSP instance and an initial tour. Krentel estimated that his proof yields a value for k between 1,000 and 10,000. By using a straight forward way to implement some missing details in Krentel's proof it was recently shown that his proof yields the value 14,208 for k [8].

Following Krentel's paper there have been claims in other papers [10, 22] through private communication with Krentel that a careful analysis of the original proof can bring down the value to $k = 8$ and conceivably to $k = 6$. However, there has been no available written proof for these claims. In fact, up to date, the 1989 paper of Krentel [13] is the only paper on the topic. Consequently, Monien, Dumrauf, and Tscheuschner [16] posed an open question on the complexity of TSP/k-Opt for $k \ll 1000$.

In this paper, we show that TSP/k-Opt has the all-exp property for much smaller k:

▶ **Theorem 1.** *TSP/k-Opt has the all-exp property for $k \geq 5$.*

Our proof of Theorem 1 is based on a new reduction from the bounded degree Max-Cut problem to TSP (see Section 3) which involves the construction of so called *parity gadgets* (see Section 3.1). With a first such approach we are able to prove the all-exp property of TSP/k-Opt for $k \geq 13$ (see Section 4). To lower the value of k additional ideas are required. First, we exploit the structure of a recent construction of Michel and Scott [15] of a degree-four bounded Max-Cut instance with the all-exp property under the flip neighborhood. Second, we show how to use global properties of our overall reduction to relax some local conditions on our parity gadgets. Combining these two ideas we achieve the value $k \geq 9$ (see Section 5). To arrive at our final result for $k \geq 5$ we have to modify the construction of Michel and Scott [15]. Moreover, we have to combine our parity gadgets with so called *double gadgets* and use a labeling scheme to assign different gadgets at different places in the reduction. These results we present in Section 6.

The second main contribution of our paper is a proof of the following result:

▶ **Theorem 2.** *TSP/k-Opt is PLS-complete for $k \geq 17$.*

The complexity class PLS and the notion of PLS-completeness (for definitions see Section 2) were introduced in 1988 by Johnson, Papdimitriou, and Yannakakis [11] to capture the observation that for many NP-hard problems it is not only difficult to compute a global optimum but even computing a local optimum is also hard. Examples of such problems are the Maximum Satisfiability problem [14], Max-Cut [21], and Set Cover [3]. The PLS-completeness of a problem means that a polynomial time algorithm to find a local optimum for that problem would imply polynomial time algorithms for finding a local optimum for all problems in PLS.

The PLS-completeness of TSP/k-Opt was proved by Krentel [13] for $k \gg 1000$. However, his proof has a substantial gap as he assumes that edges of weight infinity cannot occur in a local optimum. We present in Section 7 the first rigorous proof for the PLS-completeness of TSP/k-Opt and at the same time drastically lower the value of k from Krentel's $k \gg 1000$ [13]

to $k \geq 17$. Our proof uses several of the ideas used in our proof for Theorem 1. But in this case we need to take more care on the order in which the parity gadgets are plugged together in our construction. In addition, we show in Lemma 14 how to assign specific weights to the non-edges in our construction to prove that no local optimum can contain such an edge. We achieve this by defining a weight assignment that exploits the special structure of the TSP instance resulting from our PLS-reduction. This is the first rigorous proof of such a result for the k-Opt algorithm and there seems not to be a generic way to prove it for arbitrary TSP instances (as for example those constructed by Krentel [13]).

2 Preliminaries

2.1 Local search problems and the class PLS

A *local search problem* P is an optimization problem that consists of a set of instances D_P, a finite set of (feasible) solutions $F_P(I)$ for each instance $I \in D_P$, an objective function f_P that assigns an integer value to each instance $I \in D_P$ and solution $s \in F_P(I)$, and a neighborhood $N_P(s, I) \subseteq F_P(I)$ for each solution $s \in F_P(I)$. The size of every solution $s \in F_P(I)$ is bounded by a polynomial in the size of I. The goal is to find a *locally optimal solution* for a given instance I; that is, a solution $s \in F_P(I)$, such that no solution $s' \in N_P(s, I)$ yields a better objective value than $f_P(s, I)$. Formally, this means, for all $s' \in N_P(s, I)$, $f_P(s, I) \leq f_P(s', I)$ if P is a minimization problem, and $f_P(s, I) \geq f_P(s', I)$ if P is a maximization problem.

A *standard local search algorithm* for an instance I proceeds as follows. It starts with some initial solution $s \in F_P(I)$. Then it iteratively visits a neighbor with better objective value, until it reaches a local optimum. If a solution has more than one better neighbor, the algorithm has to choose one by some prespecified rule, often referred as a *pivot rule*.

A local search problem P has the *all-exp* property, if there are infinitely many pairs of an instance I of D_P and an initial solution $s \in F_P(I)$, for which the standard local search algorithm always needs an exponential number of iterations for all possible pivot rules.

A local search problem P is in the class *PLS* [11], if there are three polynomial time algorithms A_P, B_P, C_P such that

- Given an instance $I \in D_P$, A_P returns a solution $s \in F_P(I)$;
- Given an instance $I \in D_P$ and a solution $s \in F_P(I)$, B_P computes the objective value $f_P(s, I)$ of s; and
- Given an instance $I \in D_P$ and a solution $s \in F_P(I)$, C_P returns a neighbor of s with strictly better objective value, if it exists, and "locally optimal", otherwise.

A *PLS-reduction* from a problem $P \in$ PLS to a problem $Q \in$ PLS is a pair of polynomial-time computable functions h and g that satisfy:

1. Given an instance $I \in D_P$, h computes an instance $h(I) \in D_Q$; and
2. Given an instance $I \in D_P$ and a solution $s_q \in F_Q(h(I))$, g returns a solution $s_p \in F_P(I)$ such that if s_q is a local optimum for $h(I)$, then s_p is a local optimum for I.

A problem $Q \in$ PLS is *PLS-complete* [11] if for every problem $P \in$ PLS, there exists a PLS reduction from P to Q.

2.2 TSP/k-Opt

A *spanning cycle*, a *Hamiltonian cycle*, or a *tour* of an undirected graph is a cycle that contains all vertices of the graph.

A TSP instance is a tuple (G, w), where G is a complete undirected graph (V, E), and $w : E \to \mathbb{R}_{\geq 0}$ is a function that assigns a nonnegative weight to each edge of G. The goal is to find a tour of G that minimizes the sum of edge weights in the tour. The definition of the class PLS requires that we have a polynomial time algorithm to find *some* solution. For complete graphs such an algorithm certainly exists. If the graph is not complete then because of the NP-completeness of the Hamiltonian cycle problem we do not know such an algorithm.

A *swap* is a tuple (E_1, E_2) of subsets $E_1, E_2 \subseteq E$, $|E_1| = |E_2|$. We say that it is a swap of $|E_1|$ edges. If $|E_1| \leq k$ for some k, then we call it a k-swap. Performing a swap (E_1, E_2) from a subgraph G' of G refers to the act of removing E_1 from G' and adding E_2 to G'. We also call it swapping E_1 for E_2 in G'. Given a tour τ, a swap (E_1, E_2) is *improving* for τ, if after swapping E_1 for E_2 in τ, we obtain a tour with lower total edge weight.

A *(k-)swap sequence* is a sequence $L = (S_1, \ldots, S_\ell)$, such that each S_i is a (k-)swap. For a tour τ, we denote by τ^L the subgraph obtained from τ by performing $S_1, \ldots, S_\ell$ in their order in L. L is *improving* for a tour τ if each S_i is an improving (k-)swap for $\tau^{(S_1,\ldots,S_{i-1})}$.

The local search problem TSP/k-Opt corresponds to TSP with the k-Opt neighborhood (that is, the neighbors of a tour τ are those that can be obtained from τ by an improving k-swap). The k-Opt algorithm is then the standard local search algorithm for this problem, and an execution of the algorithm corresponds to an improving k-swap sequence.

2.3 Max-Cut/Flip

A Max-Cut instance is a tuple (G, w), where G is an undirected graph (V, E) and $w : E \to \mathbb{R}$ is a function assigning weights to the edges of G. A *cut* (V_1, V_2) of G is a partition of the vertices of G into two disjoint sets V_1 and V_2. The *cut-set* of a cut (V_1, V_2) is the set of edges $xy \in E$ such that $x \in V_1$ and $y \in V_2$. The goal of Max-Cut is to find a cut that maximizes the *value* of the cut, that is the total weight of the edges in the cut-set.

Given a Max-Cut instance and an initial cut, the *flip* of a vertex is a move of that vertex from a set of the cut to the other. The flip of a vertex is *improving*, if it results in an increase in the value of the cut. For a cut σ, its *flip neighborhood* is the set of all cuts obtained from σ by an improving flip. The Max-Cut/Flip problem is the local search problem that corresponds to the Max-Cut problem with the flip neighborhood. We call its standard local search algorithm the *Flip algorithm*. A *flip sequence* is a sequence $(v_1, \ldots, v_\ell)$ of vertices of G. A flip sequence is *improving*, if flipping the vertices in the order in the sequence increases the value of the cut at every step. In other words, an improving flip sequence corresponds to an execution of the Flip algorithm.

Monien and Tscheuschner [17] showed the all-exp property for Max-Cut/Flip even for graphs with bounded degree.

▶ **Theorem 3** ([17, 15]). *Max-Cut/Flip has the all-exp property, even when restricted to instances where all vertices have degree at most four.*

Michel and Scott [15] recently presented an alternative proof for Theorem 3. Interestingly, their construction is highly structured and exhibits a unique property: With a suitable initial cut, there is exactly one (maximal) improving flip sequence, and this sequence has exponential length. We rely on this particular construction and especially the unique property to achieve the low value of k in Theorem 1.

Note that Theorem 3 is tight with respect to the maximum degree, since the Flip algorithm on graphs with maximum degree at most three always terminates after a polynomial number of iterations [20].

3 The main reduction

In this section, we describe the main reduction to TSP/k-Opt from Max-Cut/Flip.

Let (H, w) be a Max-Cut instance. In order to avoid confusion with the vertices and edges in the TSP instance later on, we use *H-vertices* and *H-edges* for the vertices and edges of H. We denote by n and m the number of H-vertices and H-edges, respectively.

We construct from H the corresponding TSP instance as follows. We start with a cycle of $3(n+m)$ edges. We assign $n+m$ edges of this cycle to each of the n H-vertices and the m H-edges, such that any two assigned edges have distance at least two on the cycle.

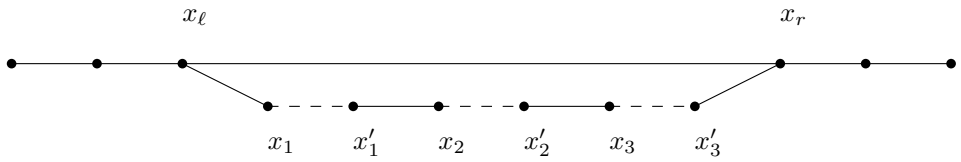

Figure 1 The first-set edge $x_\ell x_r$ and the second-set path $(x_\ell, x_1, x'_1, x_2, x'_2, x_3, x'_3, x_r)$ of an H-vertex x of degree three. The dashed edges are gateways. The other edges of the second-set path are doors.

Next, in the cycle consider an edge that is assigned to an H-vertex x. (Refer to Figure 1 for an illustration of the following concepts.) We label the two incident vertices of this edge x_ℓ and x_r, representing the left and the right vertex of the edge. Let $d(x)$ be the degree of x in H. We add a new path of length $2d(x)+1$ to connect x_ℓ and x_r. We call this new path the *second-set path* of x, while we call the original edge that was assigned to x the *first-set edge* of x. The idea is that the tour can connect x_ℓ and x_r either via the first-set edge or via the second-set path. This simulates whether the H-vertex x is in the first set or second set of the cut for the Max-Cut problem. Let $x_\ell, x_1, x'_1, \ldots, x_{d(x)}, x'_{d(x)}, x_r$ be the labels of the vertices along the second-set path. For $i \in \{1, \ldots, d(x)\}$, we call the edge $x_i x'_i$ a *gateway* of x. The other edges of the second-set path are called the *doors* of x. In other words, we have alternating doors and gateways along the path, with doors at both ends of the path.

For each H-edge xy, we call the edge in the cycle of length $3(n+m)$ assigned to xy the *xy-edge*. We remove a gateway of x, a gateway of y, and the xy-edge, and we connect the six incident vertices of the three removed edges by a *parity gadget*.

The purpose of this parity gadget is to simulate the contribution of the weight of edge xy to the objective of the Max-Cut problem, based on whether x and y are in the same set. We will formally define the parity gadget in Section 3.1.

Finally, for each H-vertex x, we assign an *XOR gadget* to the first-set edge of x and the door of x incident to x_r. The purpose of the XOR gadget is to make sure that we can simulate only one flip in H by a k-swap in the new graph. The formal definition of the XOR gadget and its assignment are discussed in Section 3.2.

Let G be the resulting graph after all the operations above (see Figure 2 for an example). Except for certain edges in the parity gadgets, which we will specify later, the other edges have weight zero, including the edges in the XOR gadgets, the initial cycle, and the doors. As a TSP instance requires a complete graph, we add the remaining edges with weight ∞ to obtain the final graph G_∞. However, if we start with a tour with a finite total weight, the k-Opt algorithm will never visit a tour that uses an edge with weight ∞. Hence, for the remaining of the reduction, we will argue based only on G.

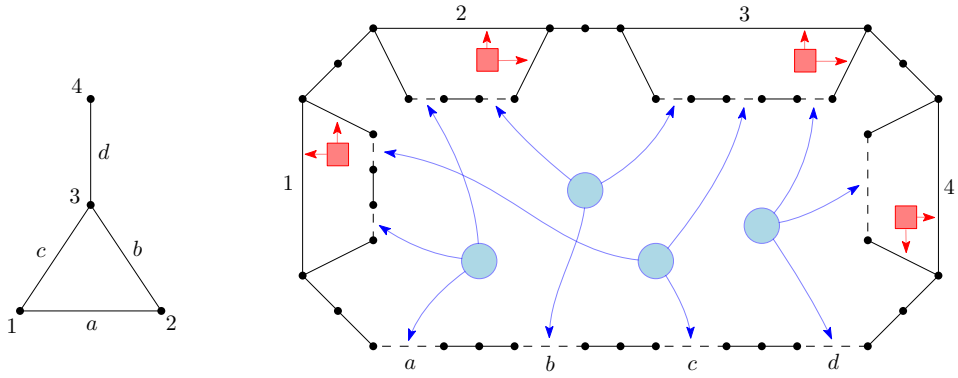

Figure 2 An example of our reduction from a Max-Cut instance (left figure) to a TSP instance (right figure). The parity gadgets are indicated by the blue circles attached to three edges each. The XOR gadgets are indicated by red boxes attached to two edges each.

3.1 Parity gadgets

In this section, we specify the parity gadgets, formally defined as follows.

▶ **Definition 4** (Parity Gadget). *A* parity gadget *is an edge weighted graph containing at least six distinct vertices labeled X, X', Y, Y', Z, Z' that satisfies the following two properties. First there exist at least the following four possibilities to cover the vertices of the parity gadget by vertex disjoint paths with endpoints in the set $\{X, X', Y, Y', Z, Z'\}$:*

(1) *A $\{Z, Z'\}$-path;*

(2) *An $\{X, X'\}$-path and a $\{Z, Z'\}$-path;*

(3) *A $\{Y, Y'\}$-path and a $\{Z, Z'\}$-path; or*

(4) *An $\{X, X'\}$-path, a $\{Y, Y'\}$-path and a $\{Z, Z'\}$-path.*

The four possibilities are called subtour *(1),* subtour *(2),* subtour *(3), and* subtour *(4) (see Figure 3 for an example). We require that in these four cases the cover is unique. A parity gadget may allow more than these four possibilities to cover the vertices by vertex disjoint paths with all endpoints in the set $\{X, X', Y, Y', Z, Z'\}$. Any such cover is called a* subtour *as long as Z and Z' are endpoints of some path(s) in this cover.*

We require in addition if X and X' are in the set of endpoints, then there must exist an $\{X, X'\}$-path in the cover. We require the same for the vertices Y and Y'.

Second, the edges of a parity gadget must allow a partition into three subsets, the same-set edges*, the* different-set edges*, and the remaining edges. The same-set edges have the same weight, which we call the* same-set weight*. Similarly, the different-set edges have the same* different-set weight*. The remaining edges have weight zero.*

We require that subtours (1) and (4) contain exactly one same-set edge and no different-set edge, and that subtours (2) and (3) contain exactly one different-set edge and no same-set edge. See Figure 3 for an example.

As explained before, a parity gadget is used to replace a gateway XX' of an H-vertex x, a gateway YY' of an H-vertex y, and the xy-edge ZZ'. The vertices X, X', Y, Y', Z, and Z' are part of the parity gadget, and the gadget is connected with the rest of G via exactly one incident edge to each of these vertices. We call these six incident edges the *external edges* of the gadget. We define the *internal edges* as the edges within the parity gadget. Further, we say that the gadget is *related* to the H-vertices x and y.

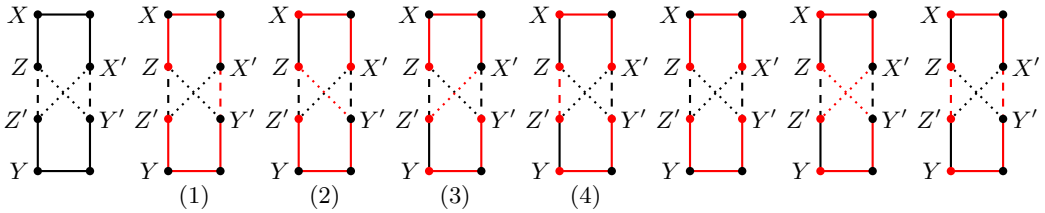

Figure 3 An example of a parity gadget (left figure). The next four figures show the four possibilities, subtour (1)–(4), to cover the vertices of the parity gadget by disjoint paths (red edges and red endpoints). The right three figures show additional subtours. The dashed edges are the same-set edges, the dotted edges are the different-set edges, and the solid edges are the remaining edges.

By construction, the removed edge ZZ' was originally part of a path of length five, say $(Z_1, Z_2, Z, Z', Z_3, Z_4)$. Since Z_2 and Z_3 have degree two in G, any tour of G has to contain Z_2Z and $Z'Z_3$. Therefore, the tour can only contain exactly one internal edge incident to Z and one incident to Z' (which may coincide). This is the reason why in the definition of a parity gadget the set $\{Z, Z'\}$ appears in all four cases.

A subtour containing an $\{X, X'\}$-path (i.e., subtour (2) or subtour (4)) represents that the corresponding H-vertex x is in the second set of the cut; otherwise, x is in the first set. When such a subtour occurs in the gadget, we say the gadget *uses* an $\{X, X'\}$-path. We have a similar representation and notation for the $\{Y, Y'\}$-path.

By definition of a parity gadget the total weight of the tour edges within a parity gadget is the same-set weight, when x and y are in the same set of the cut, and it is the different-set weight, when they are in different sets.

Next, a parity gadget is an (r_x, r_y)-*parity gadget*, if

- We need to remove exactly r_x internal edges and add exactly $r_x - 1$ internal edges to change from subtour (1) to subtour (2) or from subtour (3) to subtour (4);
- We need to remove exactly r_y internal edges and add exactly $r_y - 1$ internal edges to change from subtour (1) to subtour (3) or from subtour (2) to subtour (4);
- In order to change from subtour (1) to subtour (4) or between subtour (2) and subtour (3), we need to remove at least $\max\{r_x, r_y\}$ internal edges and add at least $\max\{r_x, r_y\} - 1$ internal edges.

We call the changes in the first two conditions above and their reverses the *standard subtour changes*.

By definition, a parity gadget may allow more than the four subtours (1)–(4) as possibilities to cover the vertices by disjoint paths with end vertices in the set $\{X, X', Y, Y', Z, Z'\}$. The parity gadget shown in Figure 3 allows for example to cover the vertices by a $\{Z, X'\}$-path and a $\{Z', Y'\}$-path. We say that a parity gadget is a *strict parity gadget*, or simply a *strict gadget*, if subtours (1)–(4) are the only possible subtours for the gadget in G, *after* we equip all gadgets (details are given in Section 3.3) used in the reduction. Thus the property of being strict may depend on the other gadgets used in the reduction.

3.2 XOR gadgets

The remaining gadgets used in the reduction are the XOR gadgets. We generalize these gadgets from the XOR gadget by [18]. See Figure 4(a)–(c) for an illustration for the definition below.

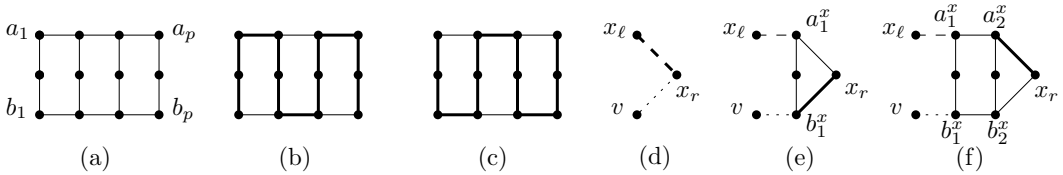

Figure 4 The XOR gadget of order four (a) and its two subtours ((b) and (c)). (d)–(f) are examples of assigning the XOR gadgets of order from zero to two to the H-vertex x, where dashed edges, dotted edges, and bold edges represent the left first-set edge, the door closest to x_r, and the right first-set edge, respectively.

▶ **Definition 5** (XOR Gadget). *Let p be a nonnegative integer. The* XOR gadget *of order p is a graph containing two paths $(a_1, \ldots, a_p)$ and $(b_1, \ldots, b_p)$, and for $i \in \{1, \ldots, p\}$, there is a path of length two with a_i and b_i as endpoints. A* subtour *of the XOR gadget is a spanning path with two endpoints in the set $\{a_1, a_p, b_1, b_p\}$. For convenience, when $p = 0$, both the XOR gadget of order zero and its only subtour are defined to be the empty graph.*

It is easy to see that an XOR gadget has two subtours, except when $p \leq 1$, in which case, it has only one subtour. Further, for $p \geq 2$, changing from one subtour to the other requires a swap of $p - 1$ edges.

We define the *assigning* of the XOR gadget of some order p to an H-vertex x as follows. (See Figure 4(d)–(f) for an illustration.) Recall that $x_\ell x_r$ is the first-set edge of x, and let vx_r be the incident door of x to x_r. We subdivide the two edges above into paths of length $p+1$, $(x_\ell, a_1^x, \ldots, a_p^x, x_r)$ and $(v, b_1^x, \ldots, b_p^x, x_r)$. Then for $i \in \{1, \ldots, p\}$, we connect a_i^x and b_i^x with a path of length two. Note that when $p = 0$, we do nothing. Further note that when we remove the edges incident to x_ℓ, x_r, and v in the construction above, we obtain the XOR gadget of order p as defined in Definition 5.

We call these incident edges to x_ℓ, x_r, and v the *external edges* of the XOR gadget, and we call the other edges in the construction the *internal edges* of the gadget. For convenience, we still refer to the external edge incident to v (i.e., vb_1^x for $p \geq 1$ and vx_r for $p = 0$) as a door of x. Additionally, we call it the *closest door to x_r*. We call the external edge incident to x_ℓ (i.e., $x_\ell a_1^x$ for $p \geq 1$ and $x_\ell x_r$ for $p = 0$) the *left first-set edge* of x. We define the *right first-set edge* of x as $x_r x_\ell$ if $p = 0$, $x_r a_p^x$ if p is positive and even, and $x_r b_p^x$ if p is odd. Lastly, we call the other external edge incident to x_r the *right second-set edge* of x.

We define an *incident edge* of a nonempty subtour of the XOR gadget to be an external edge incident to an endpoint of the subtour. The incident edge of an empty subtour (i.e., when $p = 0$) is defined to be simply an external edge of the XOR gadget.

Based on the definitions above, it is easy to verify the following.

▶ **Observation 6.** *For the XOR gadget assigned to an H-vertex x, one subtour of the gadget is incident to the left and right first-set edges of x, and another subtour of the gadget is incident to the closest door to x_r and the right second-set edge of x. The two subtours above are identical, if the order of the gadget is at most one. Otherwise, they are distinct.*

3.3 Equipping gadgets

Within our reduction we will use different parity gadgets at different places and XOR gadgets of different orders. The exact specification of which parity gadget we use at what place and the XOR gadget of which order is used for which H-vertex will be called the *equipping* of gadgets. We equip the gadgets through a labeling scheme. Specifically, for an H-vertex x

and an incident H-edge z, a labeling L assigns an integer label to the pair (x, z). We say the label is incident to x and to z. Additionally, L also assigns an integer label to each H-vertex. We also say this label is incident to x. We call a labeling L *valid*, if for every H-edge xy, there exists a $(L(x, xy), L(y, xy))$-parity gadget, and for every H-vertex x, $L(x) \geq 0$.

Then we equip the gadgets based on a valid labeling L as follows. For an H-edge xy, we equip the $(L(x, xy), L(y, xy))$-parity gadget to xy. Recall that $w(xy)$ is the weight of xy. If $w(xy) \geq 0$, the same-set weight of the gadget is $w(xy)$, and its different-set weight is zero. Otherwise, the same-set weight of the gadget is zero, and its different-set weight is $-w(xy)$. Finally, for an H-vertex x, we assign the XOR gadget of order $L(x)$ to x, and all edges involved in this assignment have weight zero. We call this a *gadget arrangement* corresponding to L. Note that this construction implies that edge weights in the TSP instance are nonnegative.

For each H-vertex x, we define the *label sum* of x to be the sum of the labels incident to x, i.e., $L(x) + \sum_{y:xy \in E(H)} L(x, xy)$. We say a labeling is an s-labeling, if each H-vertex has label sum s.

3.4 Initial tour

To complete the description of the TSP/k-Opt instance, we specify the initial tour of G. We obtain this tour from the initial cut of H as follows. The tour contains all incident edges of degree-two vertices. For an H-vertex x, if x is in the first set, we include the left and right first-set edges of x in the tour. Further, we use the subtour of the XOR gadget assigned to x, such that this subtour is incident to the left and right first-set edges of x. If x is in the second set, we include all the doors and the right second-set edge of x in the tour. Moreover, we use the subtour of the XOR gadget assigned to x, such that this subtour is incident to the right second-set edge of x and the closest door to x_r.

For an H-edge xy, in the corresponding gadget, we use the subtour (1) if x and y are in the first set, subtour (2) if x is in the second set and y in the first set, subtour (3) if x is in the first set and y in the second set, and subtour (4) if x and y are in the second set.

By the construction of G and the definition of the subtours, it can be verified that we obtain a tour of G.

3.5 Correspondence between flip sequences and swap sequences

Let us assume that we have a valid s-labeling L, and that the parity gadgets in the corresponding gadget arrangement are strict. Then we have the following correspondence (for a proof see [7]).

▶ **Lemma 7.** *Let I be a Max-Cut/Flip instance and σ be an initial cut for I. Suppose for some s, there is a valid s-labeling L for I such that all gadgets in the gadget arrangement in L are strict. Then we can reduce I to a TSP/k-Opt instance I', for $k = s + 1$, and obtain an initial tour τ from σ, such that there is a one-to-one correspondence between improving flip sequences for I and σ and improving k-swap sequences for I' and τ.*

4 All-exp property for $k \geq 13$

In this section, we prove the following statement.

▶ **Lemma 8.** *TSP/k-Opt has the all-exp property for $k \geq 13$.*

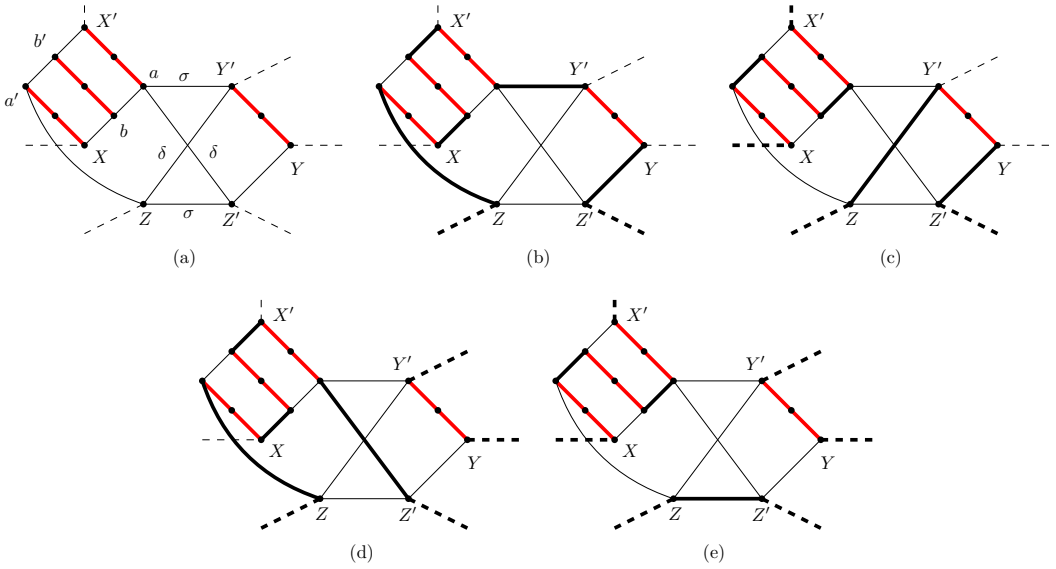

Figure 5 (a) A (4,2)-simple gadget. Bold red edges are incident to degree-two vertices. Dashed edges are external edges. $Y'a$ and ZZ' are same-set edges with same-set weight σ, while $Z'a$ and $Y'Z$ are different-set edges with different-set weight δ. (b)–(e) show the subtours (1)–(4).

We start with the parity gadgets. We call a parity gadget a *simple gadget*, if the parity gadget only allows subtours (1)–(4) (and no other subtours), independent of other gadgets equipped in the graph G. If a simple gadget is an (r_x, r_y)-parity gadget we call it an (r_x, r_y)-simple gadget. Note that a simple gadget is a strict gadget. The next result shows that a $(4, 2)$-simple gadget exists (and by symmetry, we also have a $(2, 4)$-simple gadget). The proof is given in [7].

▶ **Lemma 9.** *The gadget as depicted in Figure 5 is a* $(4, 2)$*-simple gadget.*

We can now prove Lemma 8. We use the reduction from Max-Cut/Flip to TSP/k-Opt as described in Section 3. By Theorem 3 we may assume H to have maximum degree four.

We now construct a valid $(k-1)$-labeling. Firstly, we assign an orientation on the H-edges, such that every degree-four H-vertex has exactly two incoming edges and two outgoing edges. Specifically, we repeat the following procedure: Until all H-edges have an orientation, we take a maximal (possibly closed) walk in the subgraph of unoriented H-edges, and we orient the edges along the walk. It is easy to see that the orientation is as desired.

Next, for every directed H-edge z with head x and tail y, we label (x, z) and (y, z) with four and two, respectively. By Lemma 9, there exists a simple gadget corresponding to these labels. Our construction guarantees that up to this point the label sum of every H-vertex is at most 12. Next, for each H-vertex, we assign a nonnegative label to the H-vertex, such that the label sum at the H-vertex is exactly $k - 1$. This is possible because $k \geq 13$. Hence, we obtain a valid $(k - 1)$-labeling.

Further, since we use only simple gadgets, the labeling above satisfies the condition of Lemma 7. Then by Theorem 3 and Lemma 7, we obtain Lemma 8.

5 All-exp property for $k \geq 9$

In this section, we prove the following statement.

▶ **Lemma 10.** *TSP/k-Opt has the all-exp property for* $k \geq 9$.

We use the reduction in Section 3, with two extra ingredients. Firstly, we look into the construction by Michel and Scott [15]. The graph for the instance is constructed inductively as follows (see Figure 6 for an illustration):

- The base graph F_0 consists of a single edge $v_{0,1}v_{0,8}$ with weight 7.
- The graph F_n contains a path of eight new vertices $v_{n,1}, v_{n,2}, v_{n,3}, v_{n,4}, v_{n,5}, v_{n,6}, v_{n,7}, v_{n,8}$ that appear in the path in that order. The weights of the edges along the path from $v_{n,1}$ to $v_{n,8}$ are $7 \cdot 8^n, 5 \cdot 8^n, 5 \cdot 8^n, 3 \cdot 8^n, 3 \cdot 8^n, 8^n, 8^n$. Next, we connect F_n to F_{n-1} as follows: We add edges connecting $v_{n-1,1}$ to $v_{n,2}, v_{n,4}$, and $v_{n,6}$, with weights $8^n, -8^n$, and 8^n, respectively. Finally, we add edges connecting $v_{n-1,8}$ to $v_{n,3}, v_{n,5}$, and $v_{n,7}$, with weights 1, -1, and 1, respectively.

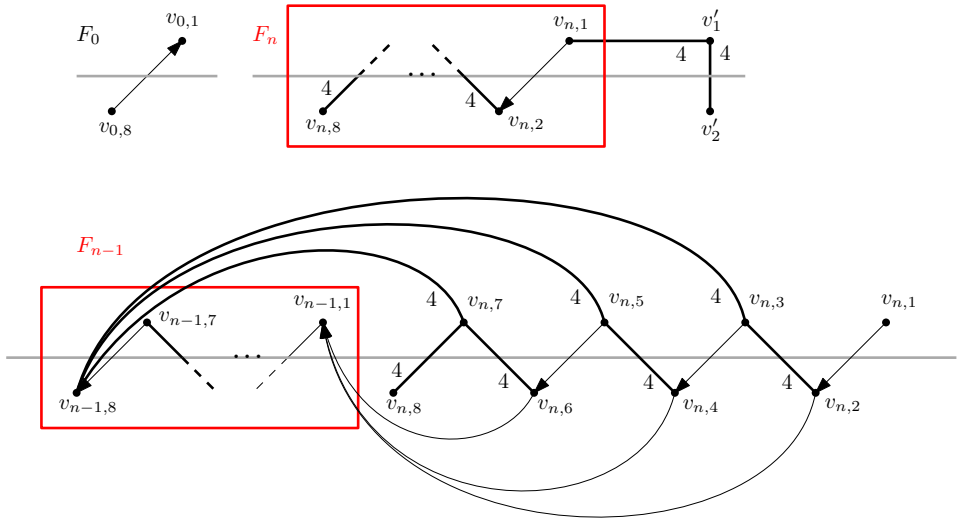

Figure 6 Michel-Scott construction of a Max-Cut instance that has an exponentially long improving flip sequence. Vertices on one side of the horizontal line are in the same set of the initial cut. We show here a valid $(k-1)$-labeling L and its gadget arrangement, for $k \geq 9$. Only labels $L(\cdot, \cdot)$ with value other than two are specified. All labels to vertices have suitable values to ensure the label sum at each vertex is $k-1$. Bold edges indicate simple gadgets, and the directed edges are flexible gadgets. The arrows indicate the direction of the forcing rule (i.e., if the gadget equipped to a directed edge is strict at the tail, then it is forced to be strict at the head).

The final graph H_n consists of all the graphs $F_0, \ldots, F_n$ and the connecting edges, as well as two new vertices v_1' and v_2' and two new edges $v_{n,1}v_1'$ and $v_1'v_2'$ with weights 8^{n+1} and $2 \cdot 8^{n+1}$, respectively. The initial cut of H_n is as follows: One set of the cut contains exactly v_1' and all vertices $v_{i,j}$ such that j is odd. The other set contains the remaining vertices.

Michel and Scott [15] showed that there exists a unique maximal improving flip sequence L_n from the aforementioned cut of H_n. The sequence is described recursively as follows: $L_0 = v_{0,1}v_{0,8}$, and $L_n = v_{n,1}v_{n,2}L_{n-1}v_{n,3}v_{n,4}L_{n-1}v_{n,5}v_{n,6}L_{n-1}v_{n,7}v_{n,8}$.

Secondly, we introduce the *flexible gadget*, a $(2,2)$-parity gadget as depicted in Figure 3. Note that it is not a simple gadget, because besides the subtours (1)–(4), there are other subtours, as shown in Figure 3. However, the following lemma (for a proof see [7]) shows that a suitable equipping of the flexible and simple gadgets can force the flexible gadgets to use only the subtours (1)–(4), and hence, they are strict:

▶ **Lemma 11.** *Suppose in the reduction in Section 3, we only equip simple, flexible, and XOR gadgets. If the H-edges corresponding to the flexible gadgets form a forest in H, then all parity gadgets are strict.*

To prove Lemma 10 we use the reduction from Section 3, with some specialization. Firstly, for the Max-Cut/Flip instance, we use the graph H_n as the graph H, for $n \geq 1$, and we use the corresponding weight and initial cut as described above. Secondly, we use the following labeling L: (See Figure 6 for a depiction of the labeling and gadget assignment.)

- For $i \in \{1, \ldots, n\}$ and $q \in \{3, 5, 7\}$, we have $L(v_{i,q}, v_{i,q}v_{i-1,8}) = 4$;
- For $i \in \{1, \ldots, n\}$ and $q \in \{2, 4, 6\}$, we have $L(v_{i,q}, v_{i,q}v_{i,q+1}) = 4$;
- $L(v_{n,8}, v_{n,8}v_{n,7}) = 4$;
- $L(v'_1, v_{n,1}v'_1) = L(v'_1, v'_1v'_2) = 4$;
- $L(x, z) = 2$, for all other pairs of an H-vertex x and an H-edge z not mentioned above and
- $L(v_{n,8}) = L(v_{n,1}) = k - 5$; $L(v'_2) = k - 3$; for other H-vertex x, we have $L(x) = k - 9$. (As $k \geq 9$, these labels are nonnegative.)

For every H-edge xy, if $L(x, xy) = L(y, xy) = 2$, we equip the flexible gadget to xy. Otherwise, we use the $(4, 2)$-simple gadget as described in Lemma 9 instead.

Observe that the labeling above is a valid $(k-1)$-labeling. Further, consider the subgraph of H containing all H-edges corresponding to the flexible gadgets. This subgraph is a forest with the leaves: $v_{0,8}$, $v_{n,1}$, $v_{n,3}$, $v_{n,5}$, $v_{j,3}$, $v_{j,5}$, $v_{j,7}$, and $v_{j,8}$ for $j \in \{1, \ldots, n-1\}$. By Lemma 11, all parity gadgets are strict.

Then Lemma 10 follows from Theorem 3 and Lemma 7.

6 All-exp property for $k \geq 5$

In this section, we prove Theorem 1 that asserts the all-exp property of TSP/k-Opt for $k \geq 5$. This proof is similar to that of Lemma 10 with a few changes. Firstly, we modify the Michel-Scott construction for the Max-Cut/Flip instance. In particular, we replace certain edges by paths of odd length. Secondly, we introduce a new gadget, called the *double gadget* that simulates two adjacent edges simultaneously. Lastly, we do not insist that all gadgets are strict. However, we argue that with our chosen initial tour, we cannot encounter a subtour other than subtours (1)–(4) in any parity gadget by a k-swap sequence.

6.1 Modified Michel-Scott construction

See Figure 7 for a depiction of the modification explained in this section. Recall the construction by Michel and Scott [15] and the unique maximal improving flip sequence L_n in Section 5. Let p be an odd number that is at least three. We observe that for any consecutive pair (v, v') in the sequence L_n, vv' is an edge in H_n and (v', v) is not a contiguous subsequence of L_n. In that case, for such consecutive pair (v, v'), we orient the edge vv' in H_n from v to v'. Then we obtain a partial orientation $\overrightarrow{H}_n$ of H_n.

For $i \in \{0, \ldots, n\}$ and $q \in \{1, \ldots, 7\}$, note that the vertex $v_{i,q}$ only has one out-neighbor $v_{i',q'}$ in $\overrightarrow{H}_n$, for some i', q'. We replace the edge $v_{i,q}v_{i',q'}$ by a path of length p $(v_{i,q}, u^1_{i,q,q'}, \ldots, u^{p-1}_{i,q,q'}, v_{i',q'})$, with $p-1$ new vertices $u^1_{i,q,q'}, \ldots, u^{p-1}_{i,q,q'}$. The weights of the new edges and which set of the cut the new vertices belong to depend on the sign of the weight of the original edges. In particular, let ω be the original weight of $v_{i,q}v_{i',q'}$ and ε be a very small number (say, 2^{-n}), and define $u^0_{i,q,q'} := v_{i,q}$ and $u^p_{i,q,q'} := v_{i',q'}$. If $\omega > 0$, then we assign the weights for the edges along the path in decreasing order: for $j = 0, \ldots, p-1$, the edge $u^j_{i,q,q'}u^{j+1}_{i,q,q'}$ has weight $\omega - j\varepsilon$. Further, we assign $u^j_{i,q,q'}$ to the same set of the cut as $v_{i,q}$ for j even, and to that as $v_{i',q'}$ for j odd. If $\omega < 0$, we assign the weights for the edges along the path in increasing order: for $j = 0, \ldots, p-1$, the edge $u^j_{i,q,q'}u^{j+1}_{i,q,q'}$ has weight

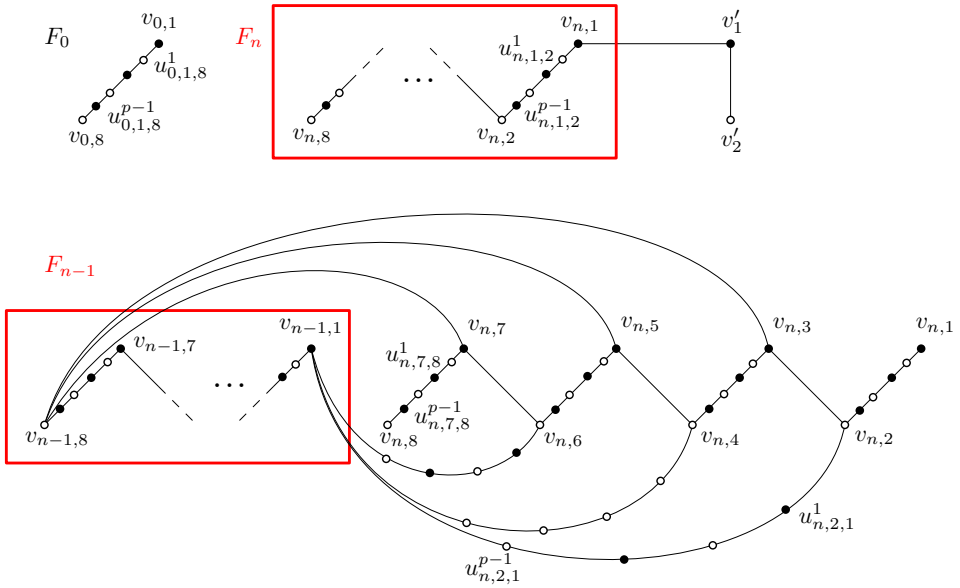

Figure 7 Modified Michel-Scott construction, where we replace certain edges in Figure 6 with paths of length p. Here, we indicate the sets of the initial cut by the colors of the vertices.

$\omega + j\varepsilon$. Further, we assign $u^j_{i,q,q'}$ to the same set of the cut as $v_{i,q}$, for all $j \in [p-1]$. The resulting (undirected) graph, weight, and cut after all replacements are the graph H, the weight w, and the initial cut χ_0 that we will use for the reduction.

We can show (for a proof see [7]) that the uniqueness property carries over to the new instance.

▶ **Lemma 12.** *There is a unique maximal improving flip sequence L'_n from the cut χ_0 of H, and this sequence is exponentially long.*

6.2 Double gadget

Let xy and xt be two H-edges. Denote $z := xy$. A *double gadget* replaces a gateway YY' of y, a gateway TT' of t, the z-edge ZZ', and a subpath (X, X_1, X_2, X') of the second-set path of x, where XX_1 and X_2X' are two gateways of x. Note that the vertices X_1 and X_2 are removed from the graph G, when we equip the double gadget to the pair xy and xt. Further, the xt-edge is not replaced by any gadget. As the third edge in a path of length five in G, the xt-edge is then used in every tour of G. We define the external and internal edges of the double gadget similar to those of parity gadgets.

A double gadget has to guarantee at least eight possible subtours (with subtours defined analogously to subtours defined in Definition 4). A $\{Z, Z'\}$-path is always present in these subtours. The eight subtours corresponds to all possibilities of containing an $\{X, X'\}$-path, a $\{Y, Y'\}$-path, or a $\{T, T'\}$-path.

Let $\sigma(x, y)$ and $\delta(x, y)$ be the same-set and different-set weights for the edge xy, respectively (i.e., they correspond to the weights when x and y are in the same set and in different sets). The numbers $\sigma(x, t)$ and $\delta(x, t)$ are defined analogously. For each of the eight subtours, the total weight of the internal edges in the subtour is the sum of two numbers, a_{xy} and a_{xt}. a_{xy} takes value $\sigma(x, y)$ if an $\{X, X'\}$-path and a $\{Y, Y'\}$-path are both present or both absent; and it takes value $\delta(x, y)$ if exactly one of these paths is present. a_{xt} is defined analogously.

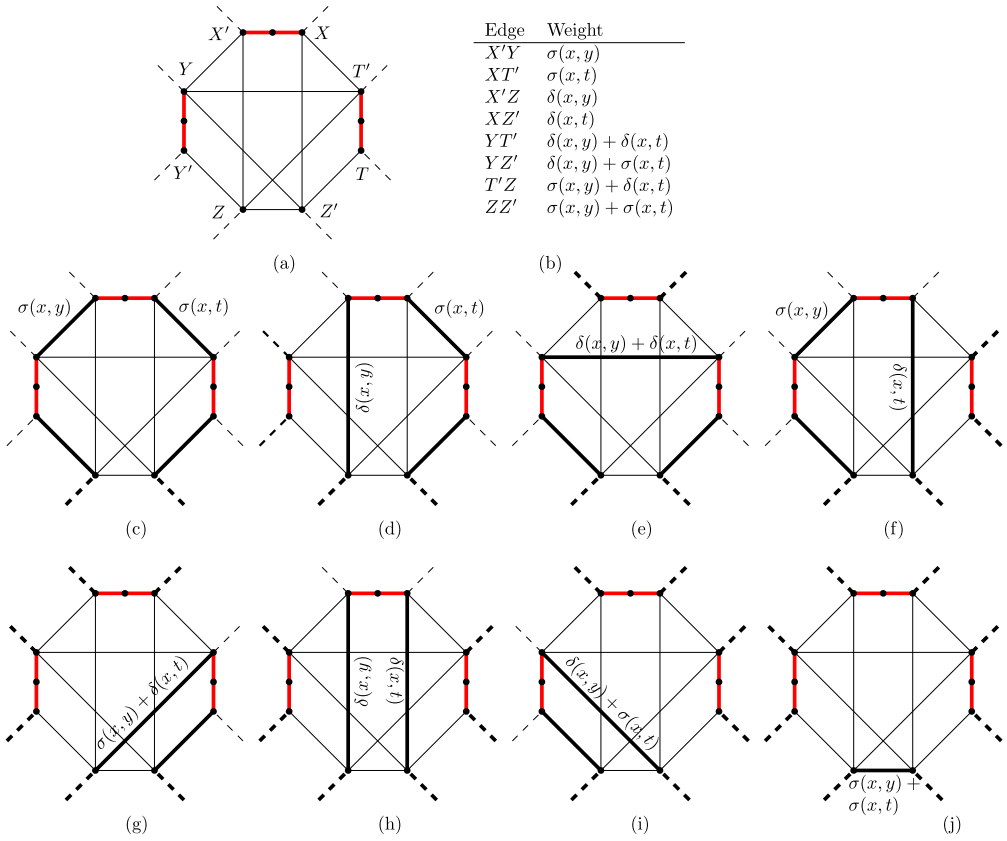

Figure 8 A (2,2,2)-double gadget (a) and the listing of edges with nonzero weights (b). The other panels show the eight subtours when the gadget is locally strict, including the nonzero weights of the edges in the subtours.

We extend the definition of an (r_x, r_y)-parity gadget to an (r_x, r_y, r_t)-double gadget in the obvious way, e.g., r_x is the number of internal edges that have to be removed if in a subtour an $\{X, X'\}$-path is added.

In the following proof, we use the (2,2,2)-double gadget as depicted in Figure 8. One can easily verify that the graph is indeed a (2,2,2)-double gadget. Note that the gadget also allows subtours other than those shown in the figure. However, we will show later that these other subtours do not appear in the improving swap sequence of concern.

6.3 Proof of Theorem 1

We use the reduction described in Section 3, from the Max-Cut instance (H, w) and the initial cut χ_0 indicated in Section 6.1. In the construction of the Max-Cut instance, we use a constant p which is odd and more than $2k$, where we recall that p is the length of the paths that replace certain edges.

We then define a suitable labeling by using the $(2, 2, 2)$-double gadgets, the flexible gadgets, the $(4, 2)$-simple gadgets, and the XOR-gadgets. The crucial idea here is that a double gadget can be used to label two incident edges. As a result, each H-vertex can now be incident to at most three labels, including one label corresponding to the XOR gadget. Accordingly, we can bring the label sum down to as low as four.

Next, we argue that although the flexible gadget and the $(2, 2, 2)$-double gadget allow many subtours, we only encounter a limited subset of these subtours. For this we introduce the notion of local strictness and show that we can get a result similar to Lemma 7. With this we can prove Theorem 1. See [7] for the details of this proof.

7 PLS-Completeness for $k \geq 17$

In this section, we prove the PLS-completeness of TSP/k-Opt for $k \geq 17$. With this result we not only improve the value $k \geq 14,208$ from Krentel [13, 8]. We also present the first rigorous PLS-completeness proof for TSP/k-Opt as Krentel's proof has a substantial gap. He assumes without proof that no edges of infinite weight can appear in a local optimum. The definition of PLS-completeness requires that the function g maps local optima to local optima. Therefore, either one has to show that a local optimum cannot contain edges of infinite weight, or one has to show how to extend the definition of the function g to local optima that contain edges of infinite weight. Both are not done in the paper of Krentel [13], and there is no obvious way how to fill this gap. For our reduction we can prove in Lemma 14 below that local optima cannot contain edges of infinite weight. There seems not to be a generic way to prove such a result for arbitrary TSP instances as for example those constructed by Krentel [13]. A result similar to Lemma 14 was obtained by Papadimitriou [19] for the Lin-Kernighan heuristic.

▶ **Theorem 2.** *TSP/k-Opt is PLS-complete for $k \geq 17$.*

Our proof of Theorem 2 follows closely the proof of Lemma 8. However, there are three key differences. Firstly, while the all-exp property is known to hold for Max-Cut instances with maximum degree four (Theorem 3), the PLS-completeness of Max-Cut/Flip is only known for maximum degree five:

▶ **Theorem 13** ([4]). *Max-Cut/Flip is PLS-complete, even when restricted to graphs of maximum degree five.*

Secondly, we impose certain structure on the graph G in the reduction. Particularly, we specify which gateways a parity gadget can replace.

Lastly, recall that the TSP instance requires a complete graph G_∞, which we obtain from G by adding the missing edges, which we also call the *non-edges*. By choosing suitable weights for the non-edges we will be able to prove in Lemma 14 that no locally optimal tour of G_∞ can contain a non-edge.

Proof of Theorem 2. We use the reduction from Max-Cut to TSP as described in Section 3. By Theorem 13, we can assume in the Max-Cut instance (H, w), that H is a graph of maximum degree five.

We assign an orientation on the H-edges such that each degree-five vertex has in-degree at most three. We can get such an orientation by repeating the following procedure: Until all H-edges have an orientation, we take a maximal (possibly closed) walk in the subgraph of unoriented H-edges, and we orient the edges along the walk. For every directed H-edge z with head x and tail y, we label (x, z) and (y, z) with four and two, respectively. Next, for each H-vertex x, we assign an integer label to x, such that the label sum at x is $k - 1$. This label is nonnegative, as $k \geq 17$. Corresponding to these labels we use the $(4, 2)$-simple gadgets from Lemma 9 and the XOR-gadgets. Hence, this is a valid $(k - 1)$-labeling. We denote it by L.

Next, we specify the gadget arrangement corresponding to L as follows. Recall that n is the number of H-vertices. Let $x^1, \ldots, x^n$ be the H-vertices. For every H-vertex x, we assign the XOR gadget of order $L(x)$ to x. Let ψ be the increasing lexicographical order of the H-edges with respect to the H-vertex indices. That is, for $i < i'$ and $j < j'$, the H-edge $x^i x^{i'}$ precedes the H-edge $x^j x^{j'}$ in the order ψ, if either $i < j$ or $i = j$ and $i' < j'$. Then we equip the parity gadgets according to the labeling L, such that when we go along the second-set path for each H-vertex x from x_ℓ to x_r, the H-edges corresponding to the related gadgets appear in their order in ψ. We equip the gadgets such that for a (4,2)-simple gadget g related to an H-vertex t, either the vertex X or Y in g is adjacent to either t_ℓ or a vertex in a gadget that is also related to t and precedes g in ψ.

Let G and w be the resulting graph and edge weight function. We have the following lemma (for a proof see [7]).

▶ **Lemma 14.** *For $k \geq 3$ there exists a complete graph G_∞ with a corresponding edge weight function w' obtained from G and w by adding the missing edges with suitable weights, such that for the TSP/k-Opt instance consisting of (G_∞, w'), all locally optimal tours only contain edges of G.*

For a given tour τ we can map it to a cut σ as follows: For each H-vertex x we put x into the first set if τ uses the left first-set edge. Otherwise we put x into the second set. Assume we have a tour τ that is a local optimum but the corresponding cut is not a local optimum. By Lemma 14 the tour τ contains only edges from G. By using arguments similar to those used in the proof of Lemma 7 we can conclude that τ is not a local optimum, a contradiction.

Then Theorem 2 follows from Theorem 13. ◀

8 Conclusion

We have shown that for $k \geq 5$ the k-Opt algorithm for TSP has the all-exp property, i.e. it has exponential running time for all possible pivot rules (Theorem 1). Moreover, we proved that TSP/k-Opt is PLS-complete for $k \geq 17$ (Theorem 2). In both cases we drastically lowered the so far best known value for k which was $\gg 1000$. It was mentioned (without explaining the details) in [13] that there is a connection between the PLS-completeness of a problem and the all-exp property. This connection was made precise by Schäffer and Yannakakis [21] who introduced the notion of *tight* PLS-completeness. They proved that the tight PLS-completeness of a problem implies the all-exp property. Our PLS-completeness result for TSP/k-Opt relies on the PLS-completeness of Max-Cut/Flip for graphs with maximum degree five [4]. As for the latter the tight PLS-completeness is not known we do not get tight PLS-completeness for TSP/k-Opt.

We put some effort into getting the constant in Theorem 1 as small as possible. In contrast, the constant in Theorem 2 very likely can be lowered to 15 by using our techniques from Section 6. However, this would require substantially more involved proofs. Finally we would like to mention (as already observed by Krentel [13]) that our results also hold for *metric* TSP as one can add a sufficiently large constant to all edge weights.

References

1 D. Avis and V. Chvátal. Notes on Bland's pivoting rule. In M. L. Balinski and A. J. Hoffman, editors, *Mathematical Programming Study 8, Polyhedral Combinatorics: Dedicated to the memory of D.R. Fulkerson*, pages 24–34. North-Holland Publishing Company, Amsterdam, 1978. doi:10.1007/BFb0121192.

2 Barun Chandra, Howard Karloff, and Craig Tovey. New results on the old k-opt algorithm for the traveling salesman problem. *SIAM Journal on Computing*, 28(6):1998–2029, 1999. doi:10.1137/S0097539793251244.

3 Dominic Dumrauf and Tim Süß. On the complexity of local search for weighted standard set problems. In Fernando Ferreira, Benedikt Löwe, Elvira Mayordomo, and Luís Mendes Gomes, editors, *Programs, Proofs, Processes. 6th Conference on Computability in Europe, CiE 2010*, volume 6158 of *Lecture Notes in Computer Science*, pages 132–140, Berlin, Heidelberg, 2010. Springer-Verlag Berlin Heidelberg. doi:10.1007/978-3-642-13962-8_15.

4 Robert Elsässer and Tobias Tscheuschner. Settling the complexity of local max-cut (almost) completely. In Luca Aceto, Monika Henzinger, and Jiří Sgall, editors, *Automata, languages and programming. 38th International Colloquium, ICALP 2011, Proceedings, Part I*, volume 6755 of *Lecture Notes in Computer Science*, pages 171–182. Springer-Verlag Berlin Heidelberg, 2011. doi:10.1007/978-3-642-22006-7_15.

5 Matthias Englert, Heiko Röglin, and Berthold Vöcking. Worst case and probabilistic analysis of the 2-opt algorithm for the TSP. *Algorithmica*, 68:190–264, 2014. doi:10.1007/s00453-013-9801-4.

6 Donald Goldfarb and William Y. Sit. Worst case behavior of the steepest edge simplex method. *Discrete Applied Mathematics*, 1(4):277–285, 1979. doi:10.1016/0166-218X(79)90004-0.

7 Sophia Heimann, Hung P. Hoang, and Stefan Hougardy. The k-opt algorithm for the traveling salesman problem has exponential running time for $k \geq 5$, 2024. doi:10.48550/arXiv.2402.07061.

8 Hung P. Hoang and Stefan Hougardy. On the PLS-completeness of TSP/k-opt. Technical Report No: 231275, Research Institute for Discrete Mathematics, University of Bonn, 2023.

9 R. G. Jeroslow. The simplex algorithm with the pivot rule of maximizing criterion improvement. *Discrete Mathematics*, 4(4):367–377, 1973. doi:10.1016/0012-365X(73)90171-4.

10 David S. Johnson and Lyle A. McGeoch. The traveling salesman problem: a case study. In Emile Aarts and Jan Karel Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 215–310. Princeton University Press, Princeton and Oxford, second edition, 2003. doi:10.2307/j.ctv346t9c.13.

11 David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. How easy is local search? *Journal of Computer and System Sciences*, 37(1):79–100, 1988. doi:10.1016/0022-0000(88)90046-3.

12 Victor Klee and George J. Minty. How good is the simplex algorithm ? In Oved Shisha, editor, *Inequalities—III. Proceedings of the Third Symposium on Inequalities Held at The University of California, Los Angeles. Septemper 1–9, 1969*, pages 159–175, New York and London, 1972. Academic Press.

13 Mark W. Krentel. Structure in locally optimal solutions. In *30th Annual Symposium on Foundations of Computer Science*, pages 216–221. IEEE Computer Society, 1989. doi:10.1109/SFCS.1989.63481.

14 Mark W. Krentel. On finding and verifying locally optimal solutions. *SIAM Journal on Computing*, 19(4):742–749, 1990. doi:10.1137/0219052.

15 Lukas Michel and Alex Scott. Superpolynomial smoothed complexity of 3-flip in local max-cut, 2024. doi:10.48550/arXiv.2310.19594.

16 Burkhard Monien, Dominic Dumrauf, and Tobias Tscheuschner. Local search: simple, successful, but sometimes sluggish. In Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and Paul G. Spirakis, editors, *Automata, languages and programming. 37th International Colloquium, ICALP 2010, Part I*, volume 6198 of *Lecture Notes in Computer Science*, pages 1–17. Springer, Berlin, 2010. doi:10.1007/978-3-642-14165-2_1.

17 Burkhard Monien and Tobias Tscheuschner. On the power of nodes of degree four in the local max-cut problem. In Tiziana Calamoneri and Josep Diaz, editors, *Algorithms and Complexity. 7th International Conference, CIAC 2010*, volume 6078 of *Lecture Notes in Computer Science*, pages 264–275. Springer Berlin Heidelberg, 2010. doi:10.1007/978-3-642-13073-1_24.

18 Christos H. Papadimitriou. The adjacency relation on the traveling salesman polytope is NP-complete. *Mathematical Programming*, 14(1):312–324, 1978. doi:10.1007/BF01588973.

19 Christos H. Papadimitriou. The complexity of the Lin–Kernighan heuristic for the traveling salesman problem. *SIAM Journal on Computing*, 21(3):450–465, 1992. doi:10.1137/0221030.

20 Svatopluk Poljak. Integer linear programs and local search for max-cut. *SIAM Journal on Computing*, 24(4):822–839, 1995. doi:10.1137/S0097539793245350.

21 Alejandro A. Schäffer and Mihalis Yannakakis. Simple local search problems that are hard to solve. *SIAM Journal on Computing*, 20(1):56–87, 1991. doi:10.1137/0220004.

22 Mihalis Yannakakis. Computational complexity. In Emile Aarts and Jan Karel Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 19–55. Princeton University Press, Princeton and Oxford, second edition, 2003. doi:10.2307/j.ctv346t9c.7.