A subexponential parameterized algorithm for Subset TSP on planar graphs

Philip N. Klein* Dániel Marx†

Abstract

Given a graph G and a subset S of vertices, the Subset TSP problem asks for a shortest closed walk in G visiting all vertices of S. The problem can be solved in time $2^k \cdot n^{O(1)}$ using the classical dynamic programming algorithms of Bellman and of Held and Karp, where $k = |S|$ and $n = |V(G)|$. Our main result is showing that the problem can be solved in time $(2^{O(\sqrt{k}\log k)} + W) \cdot n^{O(1)}$ if G is a planar graph with weights that are integers no greater than W. While similar speedups have been observed for various paramterized problems on planar graphs, our result cannot be simply obtained as a consequence of bounding the treewidth of G or invoking bidimensionality theory. Our algorithm consists of two steps: (1) find a locally optimal solution, and (2) use it to guide a dynamic program. The proof of correctness of the algorithm depends on a treewidth bound on a graph obtained by combining an optimal solution with a locally optimal solution.

1 Introduction

The Traveling Salesperson Problem (TSP) is one of the most famous and most studied combinatorial optimization problems. Given a pair $(S, d(\cdot,\cdot))$ where S is a finite set and $d : S \times S \longrightarrow \mathbb{R}$ is a weight function, a *tour* is a permutation cycle $(s_0 \dots s_{k-1})$. The weight of a tour is $\sum_i d(s_i, s_{i+1 \pmod k})$, and the goal is to find a minimum-weight tour. We use k for the size of S. An easy reduction from Hamiltonian Cycle shows that the problem is NP-hard even if every weight is 1 or 2. The problem can be solved by enumerating all the $(k-1)!$ tours, or more efficiently, by the classical dynamic programming algorithms of Bellman [4] and Held and Karp [17] solving 2^k subproblems.

Metric TSP is the special case when the pair (S, d) is a metric space, that is, the distances are symmetric and satisfy the triangle inequality. This problem is equivalent to finding a minimum-weight walk in an undirected graph with nonnegative symmetric edge-weights, for one can define S to be the vertex set and $d(u, v)$ to be the u-to-v distance (this is the metric corresponding to the edge-weighted graph). Metric TSP has a polynomial-time $\frac{3}{2}$-approximation due to Christofides [7]. Various special cases of Metric TSP or the related Hamiltonian Cycle problem are known to be solvable in time c^n for different constants $c < 2$ [14, 5, 18, 13]. However, in general graphs, we do not expect algorithms with running time $2^{o(n)}$ for Metric TSP, as this would contradict the Exponential Time Hypothesis (ETH), see, e.g., the survey [27].

The problem becomes significantly easier when the metric is the Euclidean metric between points in the plane. For this problem, polynomial-time approximation schemes are known [1, 29, 30]. Moreover, there is a $2^{O(\sqrt{n}\log n)}$-time algorithm to find the optimum solution [32].

The problem also becomes significantly easier when the metric is given by distances in a planar graph. First, it admits a polynomial-time approximation scheme (PTAS) ([15] for the case of unit weights) and ([2] for general weights) and even a linear-time approximation scheme [22, 24]. For finding an exact solution, the well-known fact that an n-vertex planar graph has treewidth $O(\sqrt{n})$ [31] combined with standard techniques for solving TSP on bounded-treewidth graphs yield a $2^{O(\sqrt{n}\log n)}$-time algorithm [8]. Using noncrossing properties with a Catalan bound yields a $2^{O(\sqrt{n})}$-time algorithm [12].

One motivation for studying Metric TSP where the metric is given by distances in a planar graph is that a road network can be modeled by a planar graph (ignoring overpasses etc.). However, this formulation requires the tour to visit every road intersection, which is not very realistic. This suggests the Subset TSP problem: given a graph G and a subset $S \subseteq V(G)$ of vertices, the task is to find the shortest closed walk visiting every vertex of S (and possibly other vertices).

*Brown University, Providence, Rhode Island, USA. Research supported by National Science Foundation Grant CCF-0964037.

†Institute of Computer Science and Control, Hungarian Academy of Sciences (MTA SZTAKI), Budapest, Hungary. Research supported by the European Research Council (ERC) grant "PARAMTIGHT: Parameterized complexity and the search for tight complexity results," reference 280152 and OTKA grant NK105645.

We refer to the vertices in S as *terminals*.

For general graphs, SUBSET TSP can be modeled by just starting with the metric space corresponding to G and then taking the submetric induced on S. However, even when G is planar, the submetric is not in general the metric corresponding to a planar graph, so SUBSET TSP restricted to planar graphs is a strict generalization of METRIC TSP restricted to planar graphs. Indeed, this problem is also a strict generalization of METRIC TSP restricted to the Euclidean metric of points in the plane: a planar embedded graph can be constructed by taking the line segments between the given points, and introducing new vertices at the crossings.

Do the favorable algorithmic properties of METRIC TSP on planar graphs carry over to SUBSET TSP problem? It has been shown that this generalization has an $O(n \log n)$ approximation scheme [23]. Moreover, using standard techniques for bounded-treewidth graphs, the problem can be solved exactly in time $2^{O(\sqrt{n} \log n)}$. But note that the number k of terminals is likely to be much smaller than the total number of vertices. By disregarding planarity, one can use the classical DP approach on the submetric to solve the problem exactly in $2^k \cdot n^{O(1)}$ time. Can we do better by exploiting planarity? In particular, can we get an exact algorithm whose running time is polynomial in n and whose dependence on k improves over that of the general algorithm? Our main result is a positive answer to this question:

THEOREM 1.1. SUBSET TSP *on a planar n-vertex graph with k terminals can be solved in time $(2^{O(\sqrt{k} \log k)} + W) \cdot n^{O(1)}$ if the weights are integers no greater than W.*

As we have observed above, Theorem 1.1 cannot be obtained by simply observing that the n-vertex planar graph G has treewidth $O(\sqrt{n})$, as this leads only to $2^{O(\sqrt{n} \log n)}$ time algorithms. Bidimensionality theory [10, 9] gives parameterized algorithms on planar graphs with running time of the form $2^{O(\sqrt{k})} \cdot n^{O(1)}$ or $2^{O(\sqrt{k} \log k)} \cdot n^{O(1)}$ for a number of problems such as finding an independent set of size k or finding a cycle of length at most k. The main idea of these algorithms is that either the graph contains an $\Omega(\sqrt{k}) \times \Omega(\sqrt{k})$ grid minor, in which case we can immediately get an answer, or the graph has no such grid, in which case we can conclude that the graph has treewidth $O(\sqrt{k})$ and hence standard techniques on bounded-treewidth graphs can be used. This approach does not seem to work for SUBSET TSP: it is not clear what we can conclude from the existence of an $\Omega(\sqrt{k}) \times \Omega(\sqrt{k})$ grid minor in this problem.

Our algorithm eventually relies on the fact that an $O(k)$-vertex planar graph G has treewidth $O(\sqrt{k})$, but we have to adopt a different viewpoint in order to be able to exploit this combinatorial relation. First, let us review how TSP can be solved in time $2^k \cdot n^{O(1)}$ using dynamic programming [4, 17]. Let us fix an arbitrary start vertex v_0. For every vertex x and subset $S' \subseteq S$, we define the subproblem (x, S'), which is the minimum length of a path from v_0 to x that visits exactly the terminals in S'. It can be shown that if we have solved all the subproblems with $|S'| = i$, then the subproblems with $|S'| = i+1$ can be solved using a simple recurrence relation. The subproblem (x, S) tells us the minimum length of a path from v_0 to x visiting all vertices and it is easy to deduce the minimum length of the tour from these subproblems. The dominating factor of the running time comes from the number of subproblems, which is $k \cdot 2^k$.

We can try to generalize the algorithm described in the previous paragraph the following way. Instead of requiring a path from v_0 to x visiting a certain set S', we require a family of at most p paths with specified endpoints such that these paths together visit every vertex of S'. The family of paths can be specified by $2p$ endpoints, which adds a factor of $k^{O(p)}$ to the number of subproblems; as long as $p = O(\sqrt{k})$, this is only $2^{O(\sqrt{k} \log k)}$. One could expect that by keeping track of $O(\sqrt{k})$ paths, the algorithm becomes more powerful and perhaps finds the optimum solution more efficiently. While this might be true, the main bottleneck of the algorithm is the number of possible such sets S', which is still 2^k. Thus if we want to improve the running time beyond 2^k, we need to find a way of considering only a restricted collection $\mathcal{S}$ of subsets S', instead of all 2^k possible subsets of S.

One of the main technical ideas of the paper appears in the way this restricted collection $\mathcal{S}$ of subsets is constructed. The construction is based on a *locally optimal* solution. First, by the symmetry of the weight function, the weight of a tour is the same as the weight of its reversal. Up to reversal, a tour is specified by an undirected graph on vertex set S whose edges form a simple cycle. Therefore, we henceforth use the term *S-tour* (or just *tour* if the choice of S is clear) to refer to such a graph.

A popular heuristic for symmetric TSP is based on iteratively improving a tour by making small changes. For a number c, a c-change in a tour consists of removing c edges to split the tour into c paths and then adding c new edges to reconnect the endpoints of these paths and obtain a new tour. We say that a tour is a *c-opt* tour with respect to the metric $d(\cdot, \cdot)$ if no such move can strictly improve the length of the tour. Finding 2-opt or 3-opt tours form the basis of many heuristic

algorithms [26, 19, 20, 21].

The following bound is the main combinatorial result of the paper: it bounds the treewidth of combining an optimal tour and a 4-opt tour. For technical reasons, we need to impose an additional condition on the 4-opt tour. We say a tour is *non-self-crossing* with respect to the planar embedded graph G if it corresponds to a non-self-crossing walk in G (see Section 2 for details).

THEOREM 1.2. *Let G be a planar embedded graph, let S be a set of k terminals, and let $(S, d(\cdot,\cdot))$ be the corresponding metric space. Let T_4 be an S-tour that is 4-opt with respect to $d(\cdot,\cdot)$ and non-self-crossing with respect to G. Then there is an optimal tour T_{opt} with respect to $d(\cdot,\cdot)$ such that the k-vertex graph formed by the union of the cycles T_4 and T_{opt} has treewidth at most $\alpha\sqrt{k}$, where α is a universal constant.*

Note that this is a purely combinatorial statement and gives no algorithm for finding the optimum or computing a tree decomposition of the union. Nevertheless, we conclude from Theorem 1.2 that it is sufficient to form the restricted collection $\mathcal{S}$ of subsets of S by taking every possible union of $O(\sqrt{k})$ consecutive segments of T_4. For this choice, the size of $\mathcal{S}$ is $k^{O(\sqrt{k})} = 2^{O(\sqrt{k}\log k)}$. A subproblem is defined by specifying $O(\sqrt{k})$ endpoints for the paths and selecting a member of $\mathcal{S}$, hence the number of subproblems is also $2^{O(\sqrt{k}\log k)}$.

In summary, the algorithm works as follows. First, we find a non-self-crossing 4-opt S-tour in polynomial time by iterative improvement. Then we construct a collection of subsets by taking every possible union of $O(\sqrt{k})$ consecutive segments of T_4 and define $2^{O(\sqrt{k}\log k)}$ types of partial solutions; each type is defined by fixing the endpoints of $O(\sqrt{k})$ paths and selecting a subset from our collection. Starting with some trivial partial solutions, we present a way of combining partial solutions to obtain larger partial solutions. We use the tree decomposition given by Theorem 1.2 to prove that this process of combining partial solutions will eventually lead to an optimum solution for the whole problem.

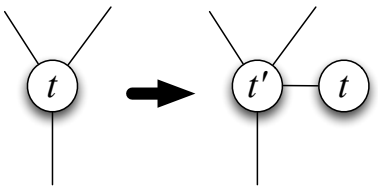
Strictly speaking, our algorithm cannot be called dynamic programming. The usual meaning of dynamic programming is that we specify some number of well-defined subproblems and then we provide a correct answer for each of these subproblems in some order, based on the answers for earlier subproblems. In our algorithm, we define types of partial solutions, but we do not claim in any way that we find an optimal partial solution for each type. What we show is that the operation of combining partial solutions gives optimal partial solutions for certain types, including the type that represents the whole problem. More precisely, the types that are guaranteed to be solved optimally depend on the tree decomposition given by Theorem 1.2. As this is a purely existential statement, we do not know during the execution of the algorithm which are the types that are guaranteed to be solved optimally, with the exception of the one representing the whole, which is always solved optimally.

Let us emphasize that, even though treewidth appears in a crucial way in the analysis of our algorithm, it is never explicitly used by the algorithm. In particular, we do not find tree decompositions or solve any problem using dynamic programming on a given tree decomposition. The tree decomposition is used only in the analysis of the algorithm to argue that we get optimal partial solutions for certain types. This way of using a tree decomposition without actually having the decomposition at hand is very similar to how consistency-based algorithms solve CSP instances of bounded width, see, e.g., [3, 6].

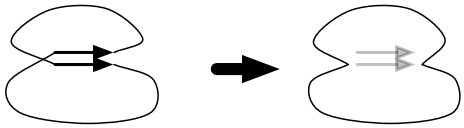
We would like to dispel a possible source of confusion. Based on Theorem 1.2 and the way we construct the collection $\mathcal{S}$ using $O(\sqrt{k})$ segments of T_4, the reader might have the impression that the optimum solution can be obtained from T_4 by $O(\sqrt{k})$ changes. If this were true, then any $O(\sqrt{k})$-change optimal tour would be globally optimal, and then we could obtain an optimum solution in a much simpler way, without the complicated process of combining partial solutions. However, we show an example where even an $\Omega(k)$-change optimal tour is not globally optimal. Intuitively, even though the locally optimal tour can be considered "similar" to an optimal tour, the similarity can be more subtle than what can be expressed by a simple change of ordering.

As noted earlier, SUBSET TSP in the metric arising from a planar graph is a generalization of the problem of finding a minimum-length tour of points in the plane. Our algorithm can therefore be applied to this problem. Note that a 4-opt solution for such a Euclidean instance is automatically non-self-crossing. For such an instance, we do not have a good bound on the number of iterations of local search required to achieve a 4-opt solution and therefore on the running time (although reportedly local search tends to terminate quickly in practice). However, the running time of our algorithm if given a 4-opt solution matches the $2^{O(\sqrt{k}\log k)}$ running time of [32] up to the constant hidden by the big-O.

The paper is organized as follows. Section 2 introduces notation and contains some preliminary results on non-self-crossing tours, local search, and tree decompositions. The main algorithm is described in Section 3. The proof of Theorem 1.2 appears in Section 4.

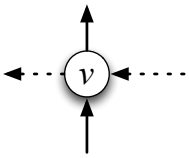


(a) Replacing a terminal t with a degree-1 terminal t'.

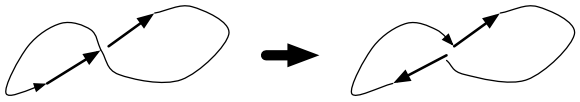


(b) This figure shows that we may assume that a realization uses each edge at most once in each direction. By requiring that each edge is one of several parallel edges, we can therefore assume that a realization uses each edge at most once.

Figure 1: Two transformations



(a) A crossing.



(b) Uncrossing a self-crossing walk.

Figure 2: Crossing and uncrossing

2 Preliminaries

We present some of the background and technical tools necessary for our algorithm in this section. We formally discuss the fact that tours can be represented in two different ways (ordering of the terminals or a closed walk in the graph), define local optimality, and introduce treewidth.

2.1 Tours and realizations An *S-tour* is a graph with vertex set S whose edges form a simple cycle visiting all vertices. Let G be a graph whose vertex set includes S. We refer to the vertices of S as *terminals*. A *realization* of an S-tour in G is a walk that visits the terminals in the same cyclic order as the S-tour.

It is convenient to impose the requirement on G that each terminal is adjacent in G to exactly one other vertex. If some terminal t has more than one neighbor, replace it with an artificial node t' and attach t to t' via a zero-weight edge (see Figure 1a). Under this requirement, we can restrict our attention to realizations in which each terminal occurs exactly once.

It is also convenient to require that each edge of G is one of four parallel edges. Under this requirement, we can restrict our attention to realizations in which each edge occurs at most once. (The argument uses the modification shown in Figure 1b.) In fact, when considering a pair of tours, we can assume in addition that the tours are edge-disjoint.

Now let us also assume G is planar embedded. Let a_1va_2 and b_1vb_2 be two-edge paths sharing the same middle vertex v. We say these two paths form a *crossing* at v if in the cyclic order of edges about v, the edges alternate between edges from $\{a_1v, a_2v\}$ and edges from $\{b_1v, b_2v\}$ (see Figure 2a). We say a walk is *self-crossing* if it contains subpaths that form a crossing. Given a closed walk that is self-crossing, one can obtain a closed walk that is non-self-crossing and uses the same edges by repeating the step illustrated in Figure 2b. Note that when this step is applied to the realization of an S-tour, the result might be a realization of a different S-tour. We say an S-tour is *non-self-crossing* with respect to a graph G if the S-tour has a non-self-crossing realization in G.

Now let G have nonnegative edge-weights. For a set A of edges with weights, $w(A)$ denotes the total weight. Define the metric $d(\cdot,\cdot)$ by letting $d(s,s')$ to be the s-to-s' distance in G. Given the graph G and any S-tour T, we can find a realization W of minimum weight by finding shortest paths between vertices of S consecutive in the S-tour. Then we can uncross this this realization to obtain a non-self-crossing closed walk W'. However, after uncrossing, the walk W' might visit the vertices of S in a different order than T. Nevertheless, based on W' we can define an S-tour T' such that W' is a realization of T.

PROPOSITION 2.1. *Given a graph G and an S-tour T, we can find in polynomial time a non-self-crossing S-tour T' with cost not more than the cost of T.*

2.2 Locally optimal tours Given two S-tours T_1 and T_2, the *distance* between T_1 and T_2 is defined as $|E(T_1) \setminus E(T_2)| = |E(T_2) \setminus E(T_1)|$. We say that an S-tour T is *c-opt* if there is no tour T' at distance at most c with $w(T') < w(T)$. For a fixed constant c, one can use brute force to check in polynomial time if a tour is c-opt, and if not, find a tour with strictly

smaller weight. By repeatedly searching the distance-c neighborhood and replacing the current solution if a better solution is found, eventually one arrives to a c-opt solution. The sequence of improvements until we reach a c-opt solution can be very long, but if the weights are positive integers, then we can bound the length of the sequence by the maximum weight of a tour.

PROPOSITION 2.2. *Let G be a weighted graph where every weight is a positive integer not larger than W. Then a c-opt S-tour can be found in time $(|S|c)^c|V(G)|^{O(1)}W$.*

Note that finding a better solution in the c-change neighborhood is W[1]-hard [28] parameterized by c, thus it seems that c has to appear in the exponent of the running time.

In our algorithm, we need to work with non-self-crossing S-tours. After each step of the iterative improvement, we can find a non-self-crossing S-tour of the same cost using Proposition 2.1. Therefore, by interleaving the local improvement and the uncrossing step, we get the following algorithm:

PROPOSITION 2.3. *Let G be a weighted planar embedded graph where every weight is a positive integer not larger than W. Then a non-self-crossing c-opt S-tour can be found in time $(|S|c)^4|V(G)|^{O(1)}W$.*

2.3 Treewidth We recall the most important notions related to treewidth in this section.

DEFINITION 2.1. *A* tree decomposition *of a graph G is a pair $(T, \mathcal{B})$ in which T is a tree and $\mathcal{B} = \{B_t | t \in V(T)\}$ is a family of subsets of $V(G)$ such that*

1. *$\bigcup_{t \in V(T)} B_i = V$;*
2. *for each edge $e = uv \in E(G)$, there exists an $t \in V(T)$ such that both u and v belong to B_t; and*
3. *the set of nodes $\{t \in V(T) \mid v \in B_t\}$ forms a connected subtree of T for every $v \in V(G)$.*

To distinguish between vertices of the original graph G and vertices of T in the tree decomposition, we call vertices of T nodes *and their corresponding B_i's* bags. *The* width *of the tree decomposition is the maximum size of a bag in $\mathcal{B}$ minus* 1. *The* treewidth *of a graph G, denoted by* $\mathrm{tw}(G)$, *is the minimum width over all possible tree decompositions of G.*

A tree decomposition is *nice* [25] if it has the following property: every node $t \in V(T)$ is either

- a *leaf node* (t has no children and $|B_t| = 1$),
- a *join node* (t has two children t', t'' and $B_{t'} = B_{t''}$),
- a *forget node* (t has a single child t' and $B_t \subseteq B_{t'}$, $|B_t| = |B_{t'}| - 1$), or
- an *introduce node* (t has a single child t' and $B_t \supseteq B_{t'}$, $|B_t| = |B_{t'}| + 1$).

It is known that every tree decomposition can be turned into a nice tree decomposition of the same width in polynomial time. We often assume that the tree is rooted.

We need the following bound on the treewidth of planar graphs:

THEOREM 2.1. ([31]) *The treewidth of a planar graph on k vertices is $O(\sqrt{k})$.*

Tutte proved that any 2-connected graph has a decomposition into 3-connected components. This result can be stated in modern terms using tree decompositions the following way. First, we need the definition of torso. Let $(T, \mathcal{B})$ be a tree decomposition. The *torso at t* is defined as the supergraph of $G[B_t]$ obtained by adding a clique on $B_t \cap B_{t'}$ for every neighbor t' of t in T. The following theorem states that every graph can be decomposed into 3-connected components. We will use this statement in the proof of Theorem 1.2, when we focus on 3-connected parts of a certain graph.

THEOREM 2.2. (CF. [11, SECTION 12, EX. 20] [16]) *Every 2-connected graph has a tree decomposition $(T, \mathcal{B})$, where*

- *$|B_t \cap B_{t'}| = 2$ for every $tt' \in E(T)$,*
- *for every $t \in V(T)$, the torso at t is either 3-connected or is a complete graph of size at most 3.*

A *2-separator* is a set $Z = \{x, y\}$ of vertices whose removal splits the graph into at least two components. Note that if $(T, \mathcal{B})$ is the tree decomposition of a 2-connected graph given by Theorem 2.2, then $B_{t'} \cap B_{t''}$ is a 2-separator for every $t't'' \in E(T)$, and conversely, every 2-separator can be obtained this way.

The following statement appears implictly in earlier work: it allows us to bound treewidth by bounding the treewidth of 3-connected components. We provide a short proof for completeness.

PROPOSITION 2.4. *Let $(T, \mathcal{B})$ be a tree decomposition of a graph G such that the torso at every $t \in V(T)$ has treewidth at most w. Then G has treewidth at most w.*

Proof. Let t be a node of T such that $|B_t| > w + 1$. Let G_t^* be the torso at t. By assumption, G_t^* has a tree decomposition $(T_t^*, \mathcal{B}_t^*)$ of width at most w, which is also a tree decomposition of $G[B_t]$. We modify $(T, \mathcal{B})$ to obtain another decomposition $(T', \mathcal{B}')$ of G. We

construct T' by removing node t, adding the tree T_t^*, and then connecting T_t^* to $T \setminus t$ the following way. For a neighbor t' of t in T, let $Z_t = B_t \cap B_{t'}$; note that Z_t induces a clique in G_t^*. It is well-know that every clique is covered by some bag of the tree decomposition; let $t^* \in V(T_t^*)$ be such that $Z_t \subseteq B_t$. Then we make t' adjacent to t^*. Repeating this process for every neighbor t' of t, we indeed get a tree decomposition $(T', \mathcal{B}')$ of G. For every $t^* \in T_t^*$, the size of B_{t^*} is at most $w+1$, thus $(T', \mathcal{B})$ has one fewer bags with size greater than $w+1$. Therefore, repeating this process for every such bag gives the desired tree decomposition. □

3 Algorithm

We describe our main algorithm in Section 3.1. The algorithm takes a set of "admissible types" as an input. In Section 3.2, we define a particular set of admissible types and we prove (using Theorem 1.2) that the algorithm finds an optimum solution if this set is given in the input.

3.1 Building partial solutions Let $(S, d(\cdot,\cdot))$ be a submetric space of the metric space defined by distances in a planar graph with edge-weights. Let G be the complete edge-weighted graph with vertex set S where the weight of ss' is $d(s,s')$. We assume $|S| \geq 3$.

A *partial solution* H is a subgraph of G that is either the vertex-disjoint union of paths or a cycle containing all $k = |S|$ vertices. We denote by $s(H)$ the nonisolated vertices of H, which we refer to as the set of vertices *visited* by the partial solution. We denote by $\epsilon(H)$ the endpoints of the paths (of length at least 1) in H; if H is a cycle visiting all vertices, then $\epsilon(H) = \emptyset$. The *weight* $w(H)$ of a partial solution H is the total weight of the edges in H. Given a partial solution H, we define a matching $m(H)$ on the set $\epsilon(H)$ as follows: $x, y \in \epsilon(H)$ are matched in $m(H)$ if H contains a path with endpoints x and y. Clearly, $m(H)$ is a perfect matching of $\epsilon(H)$. The *type* of the solution is the pair $(s(H), m(H))$.

Note that the type does not describe which path visits which vertices, it describes only the total set of vertices visited and the endpoints of the paths (see Figure 3). The number of edges of the subproblem can be deduced from the type: it is exactly $s(H)$ minus the number of edges in $m(H)$. Observe that for every partial solution with at most two edges, there is no other partial solution with the same type. However, this is not true for partial solutions with 3 edges: the partial solution consisting of the path $v_1v_2v_3v_4$ and the partial solution consisting of the path $v_1v_3v_2v_4$ have the same type $(\{v_1, v_2, v_3, v_4\}, \{v_1v_4\})$.

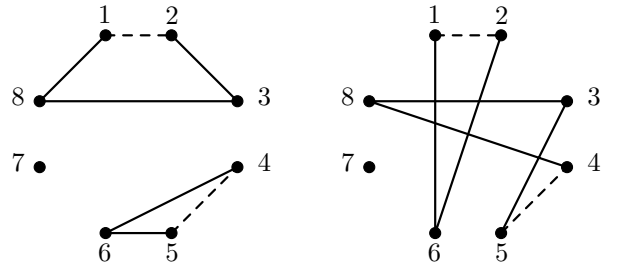


Figure 3: Two partial solutions of type $(\{1,2,3,4,5,6,8\}, \{12,45\})$.

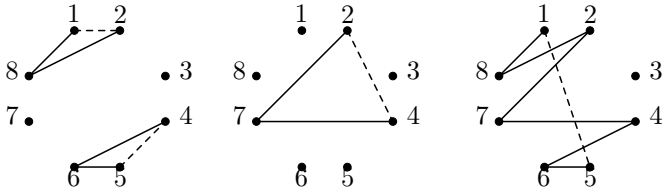


Figure 4: The union of a partial solution of type $(\{1,2,4,5,6,8\}, \{12,45\})$ and a partial solution of type $(\{2,4,7\}, \{24\})$ has type $(\{1,2,4,5,6,7,8\}, \{15\})$.

We say that two partial solutions H_1 and H_2 are *mergeable* if they are edge disjoint and their union $H_1 \cup H_2$ is also a partial solution (see Figure 4). The following lemma shows that the definition of type contains all the information that we need to know when merging partial solutions.

LEMMA 3.1. *Let H_1 and H_2 be two mergeable partial solutions. Let H_1' and H_2' be two partial solutions that have the same type as H_1 and H_2, respectively. Then H_1' and H_2' are mergeable and $H_1 \cup H_2$ has the same type as $H_1' \cup H_2'$.*

Proof. We first show that H_1' and H_2' are edge disjoint. Suppose that there is an edge $xy \in E(H_1') \cap E(H_2')$. This means that $x, y \in s(H_1') \cap s(H_2') = s(H_1) \cap s(H_2)$. That is, x and y have degree at least 1 in both H_1 and H_2. As H_1 and H_2 are edge disjoint, it follows that x and y have degree at least 2 in $H_1 \cup H_2$. As $H_1 \cup H_2$ is a partial solution, x and y have degree exactly 2 in $H_1 \cup H_2$, and hence have degree exactly 1 in both H_1 and H_2. That is, $x, y \in \epsilon(H_1) \cap \epsilon(H_2) = \epsilon(H_1') \cap \epsilon(H_2')$. Now the facts $x, y \in \epsilon(H_1')$ and $xy \in E(H_1')$ imply that $xy \in m(H_1') = m(H_1)$. Similarly, we get that $xy \in m(H_2') = m(H_2)$. As H_1 and H_2 are edge disjoint, the paths of H_1 and H_2 with endpoints x and y form a cycle in $H_1 \cup H_2$. This is only possible if $H_1 \cup H_2$ is a cycle containing every vertex, which means that $m(H_1)$ and $m(H_2)$ contain only the single edge xy. Then $m(H_1')$ and $m(H_2')$ also contain only the single edge xy and it follows that H_1'

and H_2' contain only the single edge xy. Therefore, $s(H_1') = s(H_2') = s(H_1) = s(H_2) = \{x, y\}$ and hence $s(H_1 \cup H_2) = \{x, y\}$, contradicting that $H_1 \cup H_2$ visits every vertex and the assumption $|S| \geq 3$.

Clearly, $s(H_1 \cup H_2) = s(H_1) \cup s(H_2) = s(H_1') \cup s(H_2') = s(H_1' \cup H_2')$. It is easy to see that the components of $H_1 \cup H_2$ are described by the components of the graph $m(H_1) \cup m(H_2)$ in the sense that for every path of $H_1 \cup H_2$, there is a path of $m(H_1) \cup m(H_2)$ with the same endpoints and vice versa. (If $H_1 \cup H_2$ is a cycle visiting every vertex, then $m(H_1) \cup m(H_2)$ is a single cycle). In other words, $m(H_1 \cup H_2)$ is the matching defined by the endpoints of the paths in $m(H_1) \cup m(H_2)$. As $m(H_1') \cup m(H_2') = m(H_1) \cup m(H_2)$, we can deduce that the components of $H_1' \cup H_2'$ are paths with the same pair of endpoints as in $H_1 \cup H_2$. Therefore, $H_1' \cup H_2'$ is also a partial solution and $m(H_1' \cup H_2') = m(H_1 \cup H_2)$. □

Our algorithm starts with an initial set $\mathcal{P}$ of partial solutions (e.g., those containing only a single edge) and then creates new partial solutions by taking the union of mergeable pairs. For each type, we keep only one partial solution: the one with the smallest weight found so far. By considering the mergeable pairs in increasing order of the number of edges (see later for details), one can show that the running time is polynomial in the number of types. However, the number of types is bounded from below by the number of subsets of S, which is 2^k, and the number of matchings, which is $2^{\Omega(k \log k)}$, thus there is no hope of obtaining a $2^{O(\sqrt{k} \log k)}$ time algorithm if we consider every possible type.

In order to decrease the running time, we restrict our attention to a subset of all possible types. Let $\mathcal{T}$ be a set of types, which we call the *admissible types.* Our algorithm merges two partial solutions only if the type of the resulting new partial solution is admissible. Since the running time can be bounded by the number of admissible types, it suffices to show that we can construct a set $\mathcal{T}$ of admissible types that is sufficiently small to keep the running time $2^{O(\sqrt{k} \log k)}$, but at the same time sufficiently large to guarantee that an optimum solution is found. Section 3.2 presents a way of construction such a collection.

The algorithm BuildSolutions is presented formally in Algorithm 1. Given a set $\mathcal{T}$ of admissible types and a set $\mathcal{P}$ of admissible partial solutions, the algorithm repeatedly merges partial solutions to obtain new partial solutions having admissible types. From each admissible type, the algorithm keeps only the best solution of that type found so far. The algorithm performs the merges in a specific order to avoid the situation where two partial solutions H_1 and H_2 are merged, but later a there emerges another partial solution H_1' of the same type as H_1 and having smaller weight. For increasing values of i, the algorithm considers merges where H_1 and H_2 have less than i edges, but $H_1 \cup H_2$ has at least i edges. This ensures that after the first time a partial solution H is used in a merge, no new partial solution with the same type as H is created. The result $H_1 \cup H_2$ of the merge is introduced into $\mathcal{P}$ if either $\mathcal{P}$ has no partial solution of this type, or the partial solution of this type in $\mathcal{P}$ has strictly larger weight. At the end of the algorithm, if $\mathcal{P}$ contains a solution of type $(S, \emptyset)$ (that is, a cycle visiting every vertex), then the algorithm returns it. Note that, depending on the set $\mathcal{T}$, the algorithm might not be able to create such a solution and even if it creates one, there is no guarantee that it is a minimum weight cycle visiting every vertex. The size of $\mathcal{P}$ is always at most $|\mathcal{T}|$. For each value of i, the algorithm performs at most $|\mathcal{P}|^2$ merge operations. We therefore obtain the following.

Proposition 3.1. *The number of steps of* BuildSolutions *is polynomial in* $|\mathcal{T}|$ *and* k.

3.2 Admissible types We construct a set $\mathcal{T}$ of admissible types in the following way. First, we compute a non-self-crossing 4-opt tour T_4 using the algorithm of Proposition 2.1. A continuous sequence of vertices visited by T_4 will be called a *segment* of T_4. Let $D := \max\{4, \lceil \alpha\sqrt{k} \rceil + 1\} = O(\sqrt{k})$, where α is the constant in Theorem 1.2. The type $\tau = (S', M)$ appears in $\mathcal{T}$ if and only if

- S' can be formed as the union of at most D segments of T_4, and
- M has at most D edges.

The hidden constant in the big-O notation comes from Theorem 1.2. It is clear that $|\mathcal{T}| = k^{O(\sqrt{k})} = 2^{O(\sqrt{k} \log k)}$ and $\mathcal{T}$ can be constructed in time polynomial in $\mathcal{T}$ and k. We initialize $\mathcal{P}$ by introducing every partial solution with only one edge (there are $\binom{k}{2}$ of them). For notational convenience, we also add the empty partial solution of type $(\emptyset, \emptyset)$ into $\mathcal{P}$. We run the algorithm BuildSolutions on the sets $\mathcal{T}$ and $\mathcal{P}$. The correctness of the algorithm crucially relies on Theorem 1.2, stated in the introduction. We are now ready to show that OptimalTSP (Algorithm 2) finds an optimum solution, proving Theorem 1.1. The proof shows that partial solutions corresponding to the tree decomposition given by Theorem 1.2 have to eventually appear in $\mathcal{P}$.

Algorithm 1 BUILDSOLUTIONS$(\mathcal{T}, \mathcal{P})$

Input:
$\mathcal{T}$: set of admissible types
$\mathcal{P}$: initial set of admissible partial solutions

for $i = 2$ to k
  for every pair $H_1, H_2 \in \mathcal{P}$ with $|E(H_1)|, |E(H_2)| < i$
    if H_1 and H_2 are mergeable and $|E(H_1 \cup H_2)| \geq i$
      let $\tau = (S', M)$ be the type of $H_1 \cup H_2$
      if $\tau \in \mathcal{T}$ and $\mathcal{P}$ contains no partial solution of type τ
        $\mathcal{P} := \mathcal{P} \cup \{H_1 \cup H_2\}$
      if there is a $H \in \mathcal{P}$ having type τ and $w(H) > w(H_1 \cup H_2)$
        $\mathcal{P} := (\mathcal{P} \setminus \{H\}) \cup \{H_1 \cup H_2\}$
return the partial solution in $\mathcal{P}$ having type $(S, \emptyset)$ (if exists)

Algorithm 2 OPTIMALTSP(G, S)

Input:
G: a planar graph.
S: k-element subset of $V(G)$.

Compute the metric $d(\cdot, \cdot)$ on S.
Find a non-self-crossing 4-opt tour T_4.
Let $D := \max\{4, \lceil \alpha\sqrt{k} \rceil + 1\}$. {$\alpha$ is the constant in Theorem 1.2}
Let $\mathcal{T}$ contain every type $\tau = (S', M)$ where
  – S' is the union of at most D segments of T_4, and
  – M is a matching of at most D edges.
Let $\mathcal{P}$ contain
  – the empty partial solution, and
  – the $\binom{k}{2}$ partial solutions with one edge each.
return BUILDSOLUTIONS$(\mathcal{T}, \mathcal{P})$.

For the reader's convenience, we restate the theorem:

> SUBSET TSP on a planar n-vertex graph with k terminals can be solved in time $(2^{O(\sqrt{k}\log k)} + W) \cdot n^{O(1)}$ if the weights are integers no greater than W.

Proof. of Theorem 1.1. We show that the algorithm OPTIMALTSP solves the problem. The bound on the running time is straightforward: the non-self-crossing 4-opt solution T_4 can be found in time $k^{O(1)} \cdot W$ using Proposition 2.3, the set $\mathcal{T}$ has size $2^{O(\sqrt{k}\log k)}$, and the number of steps of BUILDSOLUTIONS is polynomial in $\mathcal{T}$ and k. To argue that the algorithm finds an optimum solution, let T_{opt} be the optimum solution and let α be the constant in Theorem 1.2. Let (T, B_t) be a rooted nice tree decomposition of $T_4 \cup T_{\text{opt}}$ (the graph representing the combination of the two tours) having width at most $\alpha\sqrt{k}$. For every node t, let V_t be the union of $B_{t'}$ for every descendant t' of t (including t itself). We define $P_t = T_{\text{opt}}[V_t]$ and P_t^- as P_t minus the edges induced by B_t. Observe that P_t^- is also a partial solution.

CLAIM 3.2. *For every $t \in V(T)$, the partial solutions P_t and P_t^- are admissible.*

Proof. Let S_t and S_t^- be the set of nonisolated vertices of P_t and P_t^-, respectively. Note that these sets are subsets of V_t, but can be proper subsets as, e.g., a vertex $v \in B_t$ can have both of its neighbors in T_{opt} outside V_t, making v isolated in P_t. The set S_t induces a set of d disjoint paths on T_4. We claim that each of the $2d$ endpoints of these paths is either in B_t or is a neighbor of B_t in T_4. Indeed, if a vertex v is in $V_t \setminus N[B_t]$, then its neighbors are in $V_t \setminus B_t$, and all these vertices are visited by P_t, thus v cannot be an endpoint. Therefore, we can bound the number $2d$ of endpoints by $|B_t|$ plus the number of neighbors of B_t in T_4. More tightly, observe that if $v \in B_t$ is an endpoint of a path in $G[S_t]$ and v_1, v_2 are the neighbors of v in T_4, then v_1 and v_2 cannot be both endpoints. Thus we can be bound the number $2d$ of endpoints by $2|B_t|$, that is, S_t induces a set of at most $d \leq |B_t| \leq \alpha\sqrt{k} + 1 \leq D$ paths in T_4. This means that P_t is admissible. The same argument shows that P_t^- is

admissible as well. ⌟

By the way we initialized $\mathcal{P}$, every partial solution with at most one edge is in $\mathcal{P}$. Moreover, in iteration $i = 2$ of BuildSolutions, we merge these partial solutions every possible way and therefore every partial solution with at most two edges is in $\mathcal{P}$. (This is no longer true for partial solutions consisting of 3 edges: the partial solution consisting of the path $v_1v_2v_3v_4$ and the partial solution consisting of the path $v_1v_3v_2v_4$ have the same type, hence they cannot both appear in $\mathcal{P}$). Furthermore, as $D \geq 4$ (defined in Step 3 of Algorithm 2), every partial solution with at most 2 edges is admissible.

Claim 3.3. *For every $t \in V(T)$,*

- *after iteration $i = |E(P_t)|$, there is a $Q_t \in \mathcal{P}$ with the same type as P_t and $w(Q_t) \leq w(P_t)$,*
- *after iteration $i = |E(P_t^-)|$, there is a $Q_t^- \in \mathcal{P}$ with the same type as P_t^- and $w(Q_t^-) \leq w(P_t^-)$.*

Proof. Note that if a partial solution with i edges appears in $\mathcal{P}$ at the end of iteration i, then it remains in $\mathcal{P}$ until the end of BuildSolutions: in iterations larger then i, only types corresponding to more than i edges are updated. We prove the statement by induction on the tree decomposition. Let us assume that the statement is true for every child t' of t. We consider the different cases corresponding to the type of the node t.

- *Node t is a leaf node.* In this case $V_t = B_t$ has size 1, thus P_t and P_t^- has no edges. As $\mathcal{P}$ contains the empty solution, the statement holds.

- *Node t is an introduce node with child t'.* Let $B_t \setminus B_{t'} = \{v\}$. Observe that $P_t^- = P_{t'}^-$, thus the statement for P_t^- follows from the induction hypothesis. Similarly, if v is isolated in P_t, then $P_t = P_{t'}$ and the statement follows. Suppose now that v has 1 or 2 edges incident to it in P_t; let H_v be the partial solution containing only these (at most two) edges. Note that $P_t = P_{t'} \cup H_v$. As H_v has at most two edges, H_v is in $\mathcal{P}$ at the end of iteration $i = 2$ by our observation before. If $|E(P_t)| = |E(P_{t'})|$, then $P_t = P_{t'}$ and the statement follows from the induction hypothesis. Otherwise, after iteration $i = |E(P_{t'})| < |E(P_t)|$, there is a partial solution $Q_{t'}$ in $\mathcal{P}$ having the same type as $P_{t'}$ and $w(Q_{t'}) \leq w(P_{t'})$. By Lemma 3.1, $Q_{t'}$ and H_v are mergeable and $Q_{t'} \cup H_v$ has the same type as $P_{t'} \cup H_v = P_t$. Since $Q_{t'}$ and H_v are mergeable and they both appear in $\mathcal{P}$ at the beginning of iteration $i = |E(P_t)|$, there is a set $Q_t \in \mathcal{P}$ at the end of iteration $i = |E(P_t)$ that has the same type as $Q_{t'} \cup H_v$ (and therefore as P_t) and has $w(Q_t) \leq w(Q_{t'} \cup H_v) \leq w(P_{t'}) + w(H_v) = w(P_t)$. The existence of such a Q_t is exactly what we had to show.

- *Node t is a forget node with child t'.* Let $B_{t'} \setminus B_t = \{v\}$. Observe that $P_t = P_{t'}$, thus the statement for P_t follows from the induction hypothesis. Similarly, if v is isolated in P_t^-, then $P_t^- = P_{t'}^-$ and the statement follows. Suppose now that v has 1 or 2 edges incident to it in P_t^-; let H_v be the partial solution containing only these edges. We have $P_t^- = P_{t'}^- \cup H_v$. From this point, we can argue as in the case of introduce nodes.

- *Node t is a join node with children t' and t''.* Observe that P_t is the disjoint union of $P_{t'}$ and $P_{t''}^-$ (this explains the reason for defining the partial solutions P_t^-: we want to express P_t as the disjoint union of two partial solutions). If P_t is the same as $P_{t'}$ or $P_{t''}^-$, then the statement for P_t follows from the induction hypothesis. Otherwise, $|E(P_t)| > |E(P_{t'})|, |E(P_{t''}^-)|$. By the induction hypothesis, after iteration $i = |E(P_{t'})| < |E(P_t)|$, there is a solution $Q_{t'} \in \mathcal{P}$ having the same type as $P_{t'}$ and $w(Q_{t'}) \leq w(P_{t'})$; and after iteration $i = |E(P_{t''}^-)| < |E(P_t)|$, there is a solution $Q_{t''}^- \in \mathcal{P}$ having the same type as $P_{t''}^-$ and $w(Q_{t''}^-) \leq w(P_{t''}^-)$. By Lemma 3.1, $Q_{t'}$ and $Q_{t''}^-$ are mergeable and $Q_{t'} \cup Q_{t''}^-$ has the same type as $P_{t'} \cup P_{t''}^- = P_t$. Therefore, $Q_{t'}$ and $Q_{t''}^-$ appear in $\mathcal{P}$ at the beginning of iteration $i = |E(P_t)|$ and, as they are mergeable, there is a set $Q_t \in \mathcal{P}$ at the end of iteration $i = |E(P_t)|$ that has the same type as $Q_{t'} \cup Q_{t''}^-$ (a hence as $P_{t'} \cup P_{t''}^- = P_t$) and satisfies $w(Q_t) \leq w(Q_{t'} \cup Q_{t''}^-) \leq w(P_{t'}) + w(P_{t''}^-) = w(P_t)$, what we had to show.

  For the statement on P_t^-, observe that P_t^- is the disjoint union of $P_{t'}^-$ and $P_{t''}^-$. The argument is then the same as for P_t.

⌟

If r is the root of the tree decomposition, then $P_r = T_{\text{opt}}$. Thus Claim 3.3 for P_r implies that at the end of BuildSolutions, the set $\mathcal{P}$ contains a solution having the same type as T_{opt} (that is, type $(V(G), \emptyset)$) and weight not more than the weight of T_{opt}. This means that the algorithm returns an optimum tour. □

3.3 A remark on locally optimal solutions The proof of Theorem 1.1 shows that the vertices visited by every P_t appear on $O(\sqrt{k})$ consecutive segments of T_4.

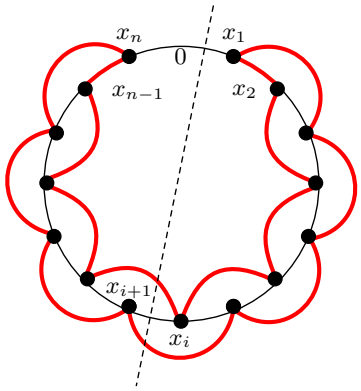


Figure 5: An example of an $\Omega(k)$-opt tour that is not globally optimal.

But this *does not* imply that each path in P_t is the union of $O(\sqrt{k})$ consecutive segments of T_4. It may very well happen that P_t consists of two paths that together visit a consecutive segment of T_4, but one path visits the odd vertices on the segment and the other path visits the even vertices of the segment; therefore, each path may visit $\Omega(k)$ segments of T_4. It is not true in any sense that T_{opt} is constructed from $O(\sqrt{k})$ segments of T_4 or that T_{opt} is within the $O(\sqrt{k})$-change neighborhood of T_4. Our algorithm is more subtle than that: essentially, we try to construct $O(\sqrt{k})$ paths that together cover $O(\sqrt{k})$ segments of T_4.

To show that we indeed need this subtle way of constructing T_{opt}, we present an example where even an $\Omega(k)$-opt tour is not globally optimal. For an odd integer n, let us define the following weighted planar graph on n vertices $x_1, \ldots, x_n$ (see Figure 5):

- For $1 \leq i \leq n-1$, there is an edge $x_i x_{i+1}$ of weight 1.
- For $1 \leq i \leq n-2$, there is an edge $x_i x_{i+2}$ of weight 1.
- There is an edge $x_1 x_n$ of weight 0.

Note that the exact values 0 and 1 do not play an important role here; the example would work with any $\alpha < \beta$. Clearly, there is a tour of weight $n-1$. Consider now the tour T that visits the vertices in the order $x_1, x_3, \ldots, x_n, x_{n-1}, x_{n-3}, \ldots, x_2, x_1$; clearly, this tour has weight n.

We claim that T is $\Omega(n)$-opt. Let T' be a tour with weight smaller than n, that is, of weight exactly $n-1$. This is only possible if T' uses the edge $x_1 x_n$ of weight 0. We show that for every $2 \leq i \leq n-2$, the symmetric difference of T and T' includes at least one edge incident to x_i or x_{i+1}. Suppose not, and consider the cut between $\{x_1, \ldots, x_i\}$ and $\{x_{i+1}, \ldots, x_n\}$ (see Figure 5). Tour T' contains the edge $x_1 x_n$ of this cut and all the other edges of this cut are incident to either x_i or x_{i+1}. If the symmetric difference of T and T' does not contain edges incident to x_i and x_{i+1}, then we know that T' contains the edges $x_{i-1}x_{i+1}$ and $x_i x_{i+2}$, but it does not contain the edge $x_i x_{i+1}$. Therefore, T' contains exactly 3 edges of this cut, which is a contradiction, as every cycle contains an even number of edges of each cut.

Therefore, the symmetric difference contains an edge incident to $\{x_i, x_{i+1}\}$ for every $2 \leq i \leq n-2$, hence the symmetric difference has $\Omega(n)$ edges. That is, the distance of T and T' is $\Omega(n)$. As this is true for every tour T' with smaller cost than T, it follows that T is $\Omega(n)$-opt, yet not globally optimal.

4 Treewidth of the union of a 4-opt tour and an optimal tour

We prove Theorem 1.2 in this section. We introduce a notion of representations, argue that a certain structure called the "grid" cannot appear in a minimal representation, and then show that the lack of a grid implies the required treewidth bound.

4.1 Representations Our main combinatorial result is understanding how a locally optimal tour interacts with a globally optimal tour in a planar graph. There are some number of technical issues that arise, e.g., it is possible that a vertex of the planar graph is visited several times by both tours. We solve these issues in a clean, yet somewhat abstract way: instead of arguing about closed walks in the planar graph G, we argue about cycles in an abstract representation.

Definition 4.1. *Let T_1 and T_2 be two S-tours that are non-self-crossing with respect to G. A* representation *with respect to G of the pair (T_1, T_2) of tours is a triple (G', C_1', C_2') where*

- *G' is a planar embedded 4-regular graph whose vertex set contains S,*
- *for any terminals t, t', the t-to-t' distance in G' is at least the t-to-t' distance in G,*
- *for $i = 1, 2$, C_i' is a simple cycle of G' visiting S in the same order as T_i,*
- *for $i = 1, 2$, the weight of C_i' is the same as T_i, and*
- *C_1' and C_2' are edge-disjoint.*

First we show that any pair of non-self-crossing tours have a representation in the sense of Definition 4.1.

Lemma 4.2. *Any pair (T_1, T_2) of S-tours that are non-self-crossing with respect to a planar embedded graph G has a representation.*

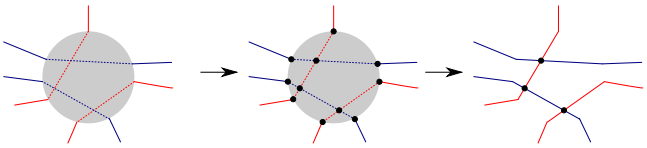


Figure 6: Replacing a high-degree vertex in the proof of Lemma 4.2.

Proof. We assume without loss of generality that G satisfies the two requirements discussed in Section 2.1: each terminal is adjacent to exactly one other vertex, and each edge of G is one of four parallel edges. Let C_1 and C_2 be the non-self-crossing realizations of T_1 and T_2 in G. Under the second requirement, we can assume that C_1 and C_2 are edge-disjoint. Under the first requirement, therefore, we can assume that each terminal has four incident edges of $C_1 \cup C_2$.

We transform G into a four-regular graph G', in several steps. First we delete edges not belonging to C_1 or C_1, and delete those vertices that have no incident edges of C_1 or C_2.

Next, we transform those vertices having degree greater than four. Each such vertex v is replaced with a subgraph as follows (see Figure 6). For each edge e incident to v, an artificial vertex v_e is placed at the endpoint of e, and for each pair e, e' of edges that are consecutive in some C_i, a zero-weight path from v_e to $v_{e'}$ is constructed. The paths are embedded in such a way that intersections of different pairs of paths do not coincide. Additional artificial vertices are created at these intersections to restore planarity. We have thus replaced a vertex v of degree greater than four with vertices of degree two (the vertices v_e) and vertices of degree four (the intersection points).

Since each terminal has degree four, it is unaffected by this transformation. Finally, in the graph as a whole, each vertex v of degree two is spliced out, and the two incident edges e_1 and e_1 are replaced with a single edge whose weight is the sum of the weights of e_1 and e_2. The resulting graph is four-regular. By assumption, the realizations C_1 and C_2 are non-self-crossing, hence every artificial vertex introduced at a crossing has two edges coming from C_1 and two edges coming from C_2. The cycles C_1, C_2 have been replaced with cycles C_1', C_2' of the same total weight. □

The following proposition states a simple property of minimal representations, illustrated in Figure 7.

PROPOSITION 4.1. *For any pair T_1, T_2 of S-tours, if R is a representation of (T_1, T_2) with respect to G that has the minimum number of vertices, then every vertex of $V(G) \setminus S$ is a crossing.*

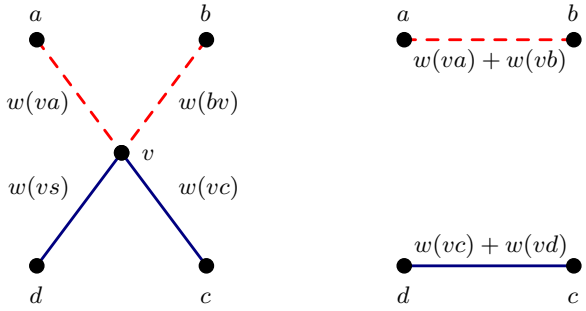


Figure 7: Proposition 4.1: eliminating a vertex that is not a crossing.

4.2 Grids and local improvement The grid is the embedded 16-vertex graph shown in Figure 8a (in this paper we consider only this specific grid and not grids of other sizes). Let (G', C_1', C_2') be a representation. If ϕ is an isomorphism between a subgraph H of G' not containing any terminals and the grid such that the horizontal edges (solid) belong to C_i' and the vertical edges (dashed) belong to C_{3-i}', we say (H, ϕ) is a C_i'-occurence of the grid in G'. Moreover, let C_i' be the cycle (either C_1' or C_2') containing the horizontal edges. Then the edges of C_i' not mapped to the grid form paths connecting the nodes that map to $L = \{(1,1),(1,4),(2,1),(2,4),(3,1),(3,4),(4,1),(4,4)\}$. The *type* of the C_i'-occurence of the grid is the perfect matching on L defined as

$$\left\{ \{\phi(u), \phi(v)\} : \begin{array}{l} \text{there is a } u\text{-to-}v \text{ path using edges} \\ \text{of } C_i' \text{ not mapped to the grid.} \end{array} \right\}$$

For example, Figure 8b illustrates an occurence with type

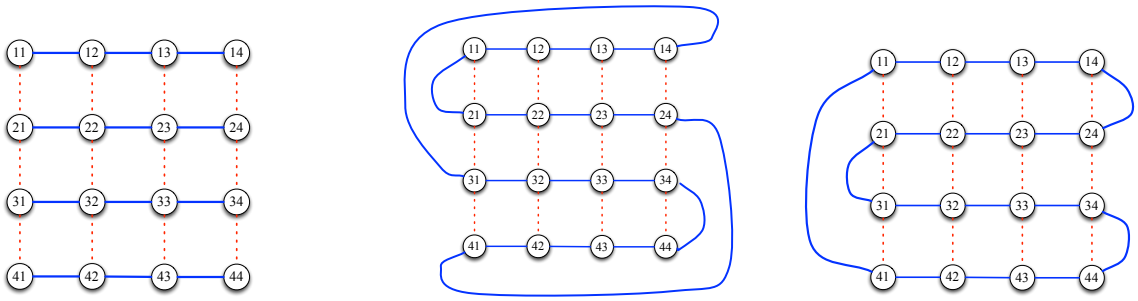
$$\{\{(1,1),(2,1)\},\{(3,4),(4,4)\},\{(1,4),(3,1)\},\{(2,4),(4,1)\}\},$$

which we call type S, and Figure 8c illustrates an occurence of type

$$\{\{(1,1),(4,1)\},\{(2,1),(3,1)\},\{(1,4),(2,4)\},\{(3,4),(4,4)\}\},$$

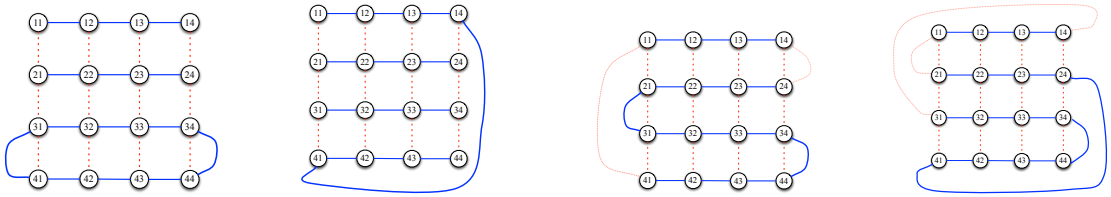
which we call type C. The *mirror image* of a type is the type obtained by swapping $(i,1)$ and $(i,4)$ for $i = 1,2,3,4$.

LEMMA 4.3. *Every type is S type or C type or the mirror image of one of these.*



(a) The grid (b) S type (c) C type

Figure 8: A grid with two different type of cycles.



(a) Subtour (b) Isolation (c) Partial C (d) Partial S

Figure 9: Matching the vertices on the boundary of the grid.

Proof. The proof consists of a case analysis based on two observations. The type cannot correspond to a C'_i containing a cycle that does not include each of the four horizontal lines of the grid; such a cycle is called a *subtour* (see Figure 9a). If there is a simple cycle consisting of some non-grid edges of C'_i together with some grid edges, then the subset of vertices of L that the simple cycle strictly encloses must be paired among themselves, and similarly for the subset of vertices not enclosed by the simple cycle. In particular, each of these subsets must have even cardinality. We say that such a subset is *isolated* (see Figure 9b).

Assume without loss of generality that $(4,4)$ is paired with a vertex in a higher-numbered row of the grid than $(4,1)$ is paired with, else consider the reflection in the rest of the proof.

By the assumption, $(4,4)$ cannot be paired with $(1,4)$. It cannot be matched with $(4,1)$, for this would create a subtour. It cannot be mapped to $(3,1)$, $(2,4)$, or $(1,1)$, for each of these would isolate a subset of odd cardinality. If it were mapped to $(2,1)$, the subset $\{(3,1),(4,1)\}$ would be isolated, so $(4,1)$ would be paired with $(3,1)$, contradicting our assumption. Thus $(4,1)$ must be mapped to $(3,4)$.

Next, $(4,1)$ is paired with something in row 1 or row 2. It cannot be paired with $(2,1)$, else $(3,1)$ would be isolated. It cannot be paired with $(1,4)$, else $\{(1,1),(2,1),(3,1)\}$ would be isolated. It must therefore be paired with $(1,1)$ or with $(2,4)$.

First suppose $(4,1)$ is paired with $(1,1)$. That isolates $\{(1,1),(2,1)\}$, so these two must be paired, which leaves $(1,4)$ and $(2,4)$ so these must be paired. This is the C type.

Now suppose $(4,1)$ is paired with $(2,4)$. Then $(1,1)$ cannot be paired with $(3,1)$, else $(2,1)$ would be isolated, and cannot be paired with $(1,4)$, else a subtour would be formed, so it is paired with $(2,1)$. That leaves $(1,4)$ and $(1,3)$ to be paired, and this is the S type. □

LEMMA 4.4. *Let $(S, d(\cdot,\cdot))$ be a metric space such that there is a planar graph G for which $d(\cdot,\cdot)$ gives the distances between vertices in S. Let T_4 be an S-tour that is 4-opt with respect to $d(\cdot,\cdot)$ and non-self-crossing with respect to G. There is an optimal tour T_{opt} and a representation (G', C'_1, C'_2) of (T_4, T_{opt}) that contains no grid.*

Proof. Among all optimal non-self-crossing tours, let T_{opt} be the one for which the size of the smallest representation of (T_4, T_{opt}) is minimized, and let (G', C'_1, C'_2) be that representation. Assume for a contradiction that there is a grid.

Let (H, ϕ_1) and (H, ϕ_2) be, respectively, a C'_1-occurence and a C'_2-occurence of the grid in the representation. By Lemma 4.3, each occurence has type S or C or a mirror image of one of these. We need to consider four combinations of types: C+C, S+C, S+S, and S+(mirror image of S), ruling out each of these possibil-

ities. That the other combinations cannot occur follows by symmetry.

In each case, the proof is as follows. We show that by moving some edges of the grid from C_1' to C_2' and others from C_2' to C_1', we end up with a different pair of edge-disjoint cycles C_1'' and C_2''. Since the same set of edges are used, the transformation does not increase the total weight: the weight of $C_1'' \cup C_2''$ equals the weight of $C_1' \cup C_2'$. The original cycle C_1' corresponds to a 4-opt tour T_4, and C_2', corresponds to a minimum-weight tour T_{opt}. Because the grid itself contains no terminals and the part of C_1' outside the grid consists of four paths, T_4 can be transformed via a 4-change to a tour whose weight is no greater than that of C_1''. (It might not be transformed to C_1'' itself since C_1'' might not consist of terminal-to-terminal shortest paths.) Since T_4 is a 4-opt tour, therefore, the weight of C_1'' is no less than that of C_1'. Therefore, the weight of C_2'' is no more than that of C_2', so C_2'' corresponds to an optimal tour $\widehat{T}_{\text{opt}}$.

Note that (G', C_1'', C_2'') is a representation for $(T_4, \widehat{T}_{\text{opt}})$. We claim that this representation has at least one vertex v of the grid that is not a crossing. It follows from Proposition 4.1 that the representation is not a smallest representation of (G', C_1'', C_2''), a contradiction.

It remains only to show, for each of the type combinations C+C, S+C, S+S, and S+(mirror image of S), how the cycles are transformed. These are shown in Figures 10-13. □

4.3 Bounding treewidth We show next that a representation not having a grid has treewidth $O(\sqrt{k})$ and then we use this to argue that the union of the two S-tours in Theorem 1.2 has bounded treewidth. Let us note that we cannot obtain this bound by simply using the fact that a planar graph not containing a grid minor has bounded treewidth. Our grid (see Figure 8a) is defined as a subgraph, not as a minor, and moreover it has the additional property that it is specified which of the edges come from which of the cycles.

LEMMA 4.5. *Consider a representation $R = (G', C_1, C_2)$ of (T_1, T_2) chosen to minimize the number of vertices of the representation. If the representation contains no grid, then its treewidth is $O(\sqrt{k})$.*

Proof. First, we observe some immediate consequences of minimality.

CLAIM 4.6. *Every face of length less than 4 is incident to S.*

Proof. Consider first a face F of length 2 not incident to S (see Figure 14). The boundary of F contains

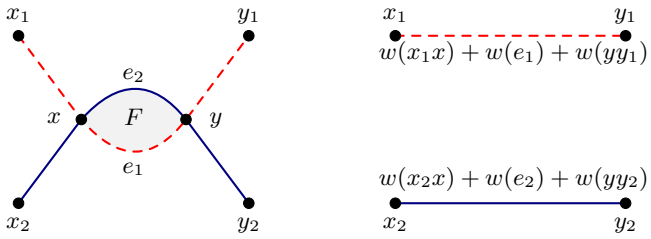


Figure 14: Claim 4.6: eliminating a face of length 2.

one edge from each of C_1 and C_2: otherwise, there would be a cycle of length 2 in C_1 or C_2. As F is not incident to S, the two edges $e_1 \in E(C_1)$ and $e_2 \in E(C_2)$ of the face connect two vertices $x, y \notin S$, which are crossings by Prop. 4.1. Suppose that x_1xyy_1 is a subpath of C_1 containing e_1 and x_2xyy_2 is a subpath of C_2 containing e_2. We modify the representation by removing both e_1 and e_2, adding an edge x_1y_1 of weight $w(x_1x) + w(e_1) + w(yy_1)$ to C_1, and adding an edge x_2y_2 of weight $w(x_2x) + w(e_2) + w(yy_2)$ to C_2. From the fact that both x and y are crossings, it follows that the modified representation is also planar. The other properties of representations can be also verified easily.

Consider now a face F of length 3 not incident to S. By Prop. 4.1, every vertex of F is a crossing. Therefore, the edges of F are alternatingly from C_1 and C_2, implying that the length of F cannot be odd. ⌟

Claim 4.6 states that facial cycles of length less than 4 intersect S, but this does not immediately imply that this is true for every cycle of length less than 4. The following claim proves this stronger statement.

CLAIM 4.7. *Every cycle of length less than 4 contains a vertex of S.*

Proof. Let C be a cycle of length less than 4. Let I be the set of vertices strictly enclosed by C and let O be the set of vertices not enclosed by C. If $I = \emptyset$, then consider a face enclosed by C. The vertices of this face are on C, thus it has at most 3 vertices and then Claim 4.6 implies that C intersects S. Therefore, we can assume that I is nonempty and, by a similar argument, that O is nonempty. This means that each of C_1 and C_2 has at least two edges between I and C, and between O and C. Thus the total degree of the vertices of C is at least $2|C| + 8 \leq 4|C|$, which contradicts $|C| \leq 3$. ⌟

We say that a face is *good* if it has length 4, no terminal is incident to it, and its boundary consists of edges alternatingly from C_1 and C_2; otherwise, we say that the face is *bad*. We use Claim 4.6 and Euler's Formula to give an $O(k)$ bound on the number of bad faces; in

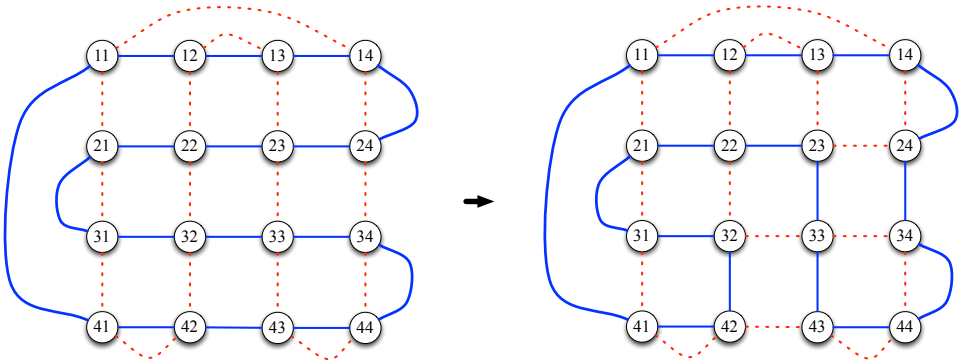


Figure 10: Modification for C+C types

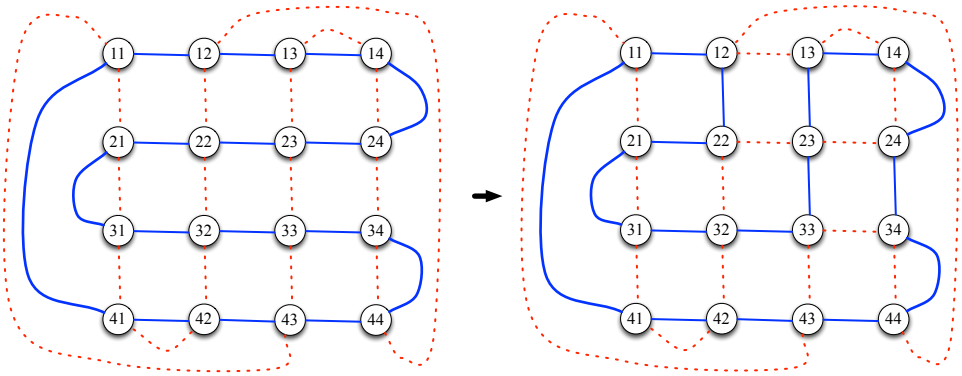


Figure 11: Modification for C+S types

fact, we prove the somewhat stronger statement that the total size of bad faces is bounded by $O(k)$.

Claim 4.8. *Let B be the set of vertices incident to bad faces. Then $|B| = O(k)$.*

Proof. Let n, f, and e be the number of vertices, faces, and edges of R, respectively. As R is 4-regular, we have $e = 2n$. From Euler's Formula, we have $n + f = e + 2 = 2n + 2$, hence $f = n + 2$. Let $\mathcal{F}$ be the set of all faces, and let $\mathcal{F}_{<4}$ and $\mathcal{F}_{\geq 4}$ be the set of faces of length less than 4 and at least 4, respectively. By Claim 4.6, every face of size less than 4 is incident to a vertex in S. As each vertex of S is incident to at most 4 faces, it follows that $|\mathcal{F}_{<4}| = O(k)$. Let $\ell(F)$ be the length of a face and let us define the excess of a face F as $x(F) := \max\{\ell(F) - 4, 0\}$. We can bound the total excess as

$$\begin{aligned}\sum_{F\in\mathcal{F}} x(F) &= \sum_{F\in\mathcal{F}_{\geq 4}} (\ell(F) - 4) \\ &= \sum_{F\in\mathcal{F}_{\geq 4}} \ell(F) - 4|\mathcal{F}_{\geq 4}| \\ &\leq \sum_{F\in\mathcal{F}} \ell(F) - 4(|\mathcal{F}| - |\mathcal{F}_{<4}|) \\ &= 2e - 4f + 4|\mathcal{F}_{<4}| = 4n - 4f + O(k) \\ &= 4(f-2) - 4f + O(k) = O(k).\end{aligned}$$

As $\ell(F) \leq x(F) + 4 \leq 5x(F)$ for a face of length strictly greater than 4, the total length of such faces is at most $5\sum_{F\in\mathcal{F}} x(F) = O(k)$. By Claim 4.6, the total length of faces of length less than 4 is $O(k)$ (as each vertex of S can be incident to at most 4 such faces and each such face has at most 3 vertices). Moreover, if a face of length 4 is not incident to S, then all its vertices are crossings, hence its edges are alternatingly from C_1 and C_2. It follows that the total length of bad faces of length 4 is $O(k)$. We can conclude that the size of B is indeed $O(k)$. ⌟

Next we show that a good face together with 8 other

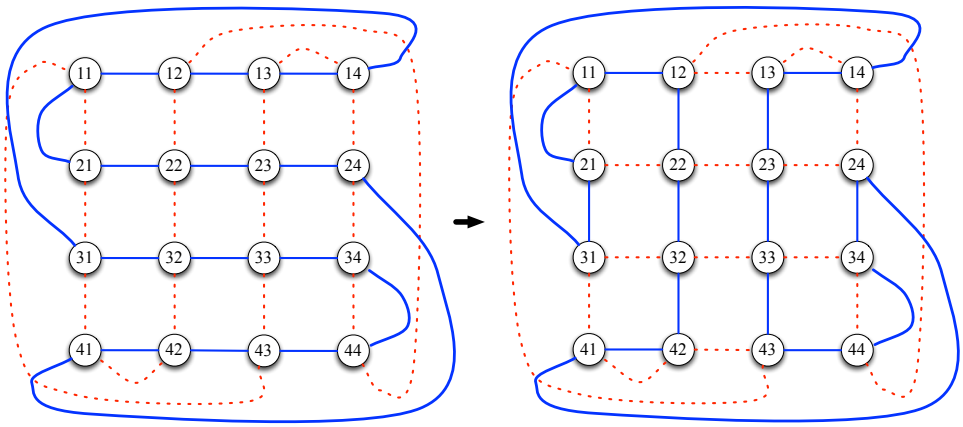


Figure 12: Modification for S+S types

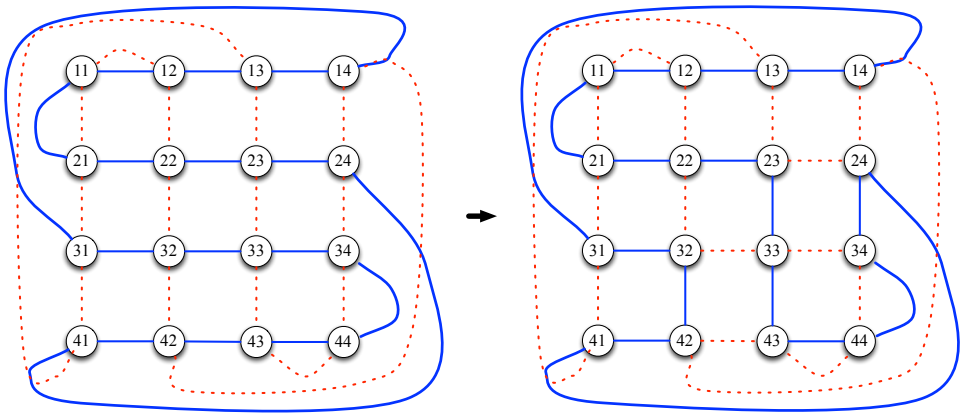


Figure 13: Modification for S+(mirror image of S) types

good faces surrounding it form a grid. While this sounds obvious, it requires a tedious case analysis to show that the vertices of the grid are distinct, and we need the technical condition that no 2-separator is present in the face.

CLAIM 4.9. *Let F be a good face whose vertices do not contain a 2-separator and are at distance at least 3 from B. Then there is a grid.*

Proof. Let the vertices of F be $(2,2)$, $(2,3)$, $(3,3)$, $(3,2)$ as in Figure 8a: the edge between $(2,2)$ and $(2,3)$ is from C_1, etc. We define $(2,1)$ as the neighbor of $(2,2)$ in C_1 different from $(2,3)$, and define the vertices $(1,2)$, $(1,3)$, $(2,4)$, $(3,1)$, $(3,4)$, $(4,2)$, $(4,3)$ in a similar way. Note that $(2,2)$ is a crossing by Proposition, thus the edges incident to $(2,2)$ are indeed ordered as in Figure 8a. Furthermore, these vertices are not in B, hence are incident only to good faces. It follows that there is an edge of C_2 between $(2,1)$ and $(3,1)$, an edge of C_1 between $(1,2)$ and $(1,3)$, and so on. The vertices $(1,2)$, $(2,2)$, $(2,1)$ are part of a good face; we define $(1,1)$ as the fourth vertex of that face. The vertices $(1,4)$, $(4,1)$, $(4,4)$ are defined similarly. By the way we defined the vertices, any two vertices adjacent in Figure 8a are adjacent in R and each of the nine faces in Figure 8a is a face in R. However, we have to prove that all the defined vertices are distinct.

We will be repeatedly using the following argument. Let C be any cycle of the representation and let I be the set of vertices strictly enclosed by C. Let J_1 and J_2 be the subset of edges of C_1 and C_2, respectively, that are between I and C. Then both $|J_1|$ and $|J_2|$ are even.

Let us define

$$X_1 = \{(2,2),(2,3),(3,2),(3,3)\},$$
$$X_2 = \{(1,2),(1,3),(2,1),(2,4),(3,1),(3,4),(4,2),(4,3)\},$$
$$X_3 = \{(1,1),(1,4),(4,1),(4,4)\}.$$

It is clear that the vertices of X_1 are distinct. Let us show that every vertex of X_2 is distinct from the vertices in X_1; by symmetry, it is sufficient to show this for $(2,1)$. We consider the following cases:

- $(2,1) = (2,2)$: they are adjacent by definition, a contradiction.
- $(2,1) = (2,3)$: by definition, $(2,1)$ is a neighbor of $(2,2)$ different from $(2,3)$, a contradiction.
- $(2,1) = (3,2)$: then they form a cycle of length 2, contradicting Claim 4.7.
- $(2,1) = (3,3)$: then $(2,1) = (3,3)$, $(2,2)$, and $(2,3)$ form a cycle of length 3, contradicting Claim 4.7.

Let us show that every vertex $x \in X_2$ is distinct from every other vertex of X_2; by symmetry, it is sufficient to show this for $x = (2,1)$. Let e be the edge between $(2,1)$ and $(2,2)$. If we know that there is another edge between x and one of $(2,2)$, $(2,3)$, or $(3,2)$, then $(2,1)$ is part of a cycle of length at most 3, a contradiction. Therefore, we have to consider only the two cases where there is an edge between $(2,1)$ and $(3,3)$, that is, $(2,1)$ is the same as $(3,4)$ or $(4,3)$.

- $(2,1) = (3,4)$: Let C be the cycle formed by $(2,1) = (3,4)$, $(2,2)$, $(2,3)$, $(3,3)$ (see Figure 15). Assume without loss of generality that C encloses F and define I and J_1 as above. No edge of J_1 is incident to $(2,1)$ or $(2,2)$, as all edges of C_1 incident to these vertices are in C. As F is enclosed by C and $(3,2)$ is not on C, we have that $(3,2) \in I$ and hence the edge between $(3,2)$ and $(3,3)$ is in J_1. The face $(2,3)$, $(2,4)$, $(3,4)$, $(3,3)$ is not enclosed by C (it is adjacent to F via an edge of C), hence $(2,4) \notin I$ and the edge between $(2,3)$ and $(2,4)$ is not in J_1. Therefore, we have shown that J_1 contains only a single edge, a contradiction.
- $(2,1) = (4,3)$: we show that $\{(2,2),(3,3)\}$ is a 2-separator. Consider the cycle C formed by $(2,1) = (4,3)$, $(2,2)$, $(2,3)$, $(3,3)$ (see Figure 15). Assume without loss of generality that C encloses F. There is a vertex strictly enclosed by C (e.g., $(3,2)$) and there is a vertex not enclosed by C (e.g., $(2,4)$, which is distinct from any of the vertices of C, otherwise they would form a cycle of length at most 3). Therefore, C separates the vertices strictly enclosed by C and the vertices not enclosed by C. We claim that $\{(2,2),(3,3)\}$ already separates those vertices. Let us observe that both neighbors of $(2,1) = (4,3)$ that are not on C, namely $(3,1)$ and $(4,2)$, are strictly enclosed by C, as they are incident to faces that are enclosed by C (note also that we already know that $(3,1)$ and $(4,2)$ are distinct). Similarly, both neighbors of $(2,3)$ not on C, namely $(1,3)$ and $(2,4)$, are not enclosed by C. Therefore, we cannot use $(2,1)$ or $(2,3)$ to go between the inside and the outside of C, hence $\{(2,2),(3,3)\}$ is a 2-separator, a contradiction.

Let us show next that every vertex of X_3 is distinct from every vertex of X_1; by symmetry, it is sufficient to show this for $(1,1)$. Vertex $(1,2)$ is a neighbor of $(1,1)$ in C_1. For $x \in \{(2,2),(2,3)\}$, both neighbors of x in C_1 are of the form $(2,i)$ for some $1 \leq i \leq 4$. For $x \in \{(3,2),(3,3)\}$, both of its neighbors in C_1 are of the form $(3,i)$ for some $1 \leq i \leq 4$. Therefore, if $(1,1)$ is in X_1, then $(1,2)$ coincides with a vertex of the form $(2,i)$ or $(3,i)$, which we have ruled out earlier.

Finally, let us show that every vertex of X_3 is distinct from every vertex $x \in X_2 \cup X_3$; by symmetry, it is sufficient to show this for $(1,1) \in X_3$. If x is one of $(1,2)$, $(1,3)$, $(1,4)$, $(2,1)$, $(3,1)$, $(4,1)$, then there is a cycle of length at most 3 going through x, a contradiction. Therefore, we have to consider only the following cases:

- $(1,1) = (2,4)$. Consider the cycle C formed by $(1,1) = (2,4)$, $(1,2)$, $(1,3)$, $(1,4)$. Assume without loss of generality that C encloses F and let I and J_1 be defined as above. No edge of J_1 is incident to $(1,2)$ or $(1,3)$, as all edges of C_1 incident to these vertices are in C. As F is enclosed by C, we have that $(2,3) \in I$ and hence the edge between $(2,3)$ and $(2,4)$ is in J_1. The face $(1,3)$, $(1,4)$, $(2,4)$, $(2,3)$ is enclosed by C (it is incident to $(2,3) \in I$) and in fact it is the only face incident to $(1,4)$ that is enclosed by C. It follows that no edge of J_1 is incident to $(1,4)$. Therefore, we have shown that J_1 contains only a single edge, a contradiction.
- $(1,1) = (4,2)$. Symmetric to the previous case.
- $(1,1) = (3,4)$. Consider the cycle C formed by $(1,1) = (3,4)$, $(1,2)$, $(1,3)$, $(1,4)$, $(2,4)$ (see Figure 15). Assume without loss of generality that C encloses F and let I and J_2 as defined above. One can observe that J_2 contains exactly three edges: the edge between $(1,2)$ and $(2,2)$, the edge between $(1,3)$ and $(2,3)$, and the edge between $(1,1)$ and $(2,1)$. Thus $|J_2|$ is odd, a contradiction.
- $(1,1) = (4,3)$. Symmetric to the previous case.
- $(1,1) = (4,4)$. Consider the cycle formed by $(1,1) = (4,4)$, $(1,2)$, $(1,3)$, $(1,4)$, $(2,4)$, $(3,4)$ (see Figure 15). Assume without loss of generality that C encloses F and let I and J_1 be defined as above. One can observe that J_1 contains exactly three edges: the one between $(2,3)$ and $(2,4)$, the one between $(3,3)$ and $(3,4)$, and the one between $(4,3)$ and $(4,4)$. Thus $|J_1|$ is odd, a contradiction.

Therefore, we have shown that all the 16 defined vertices are distinct and they indeed form a grid. ⌟

By Theorem 2.1, a planar graph with $O(k)$ vertices has treewidth $O(\sqrt{k})$. Unfortunately, we are not able to give an upper bound of $O(k)$ on the number of vertices of

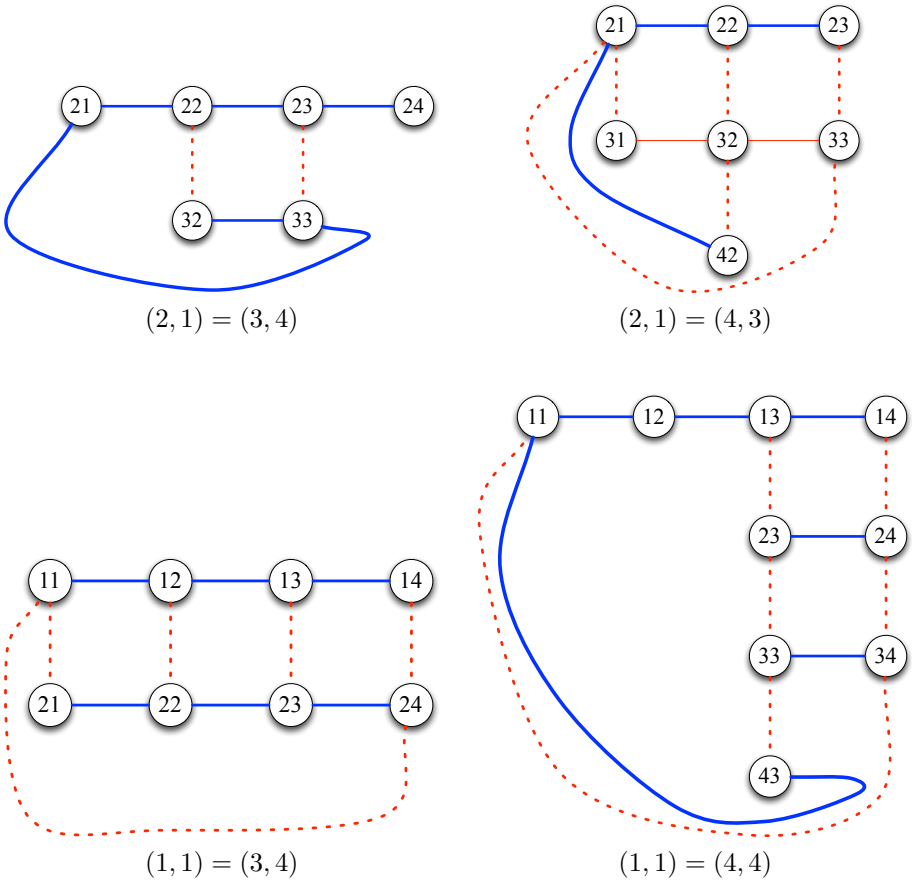


Figure 15: Cases in the proof of Claim 4.9.

R. Instead, we prove that every 3-connected component (see Theorem 2.2) has $O(k)$ vertices and then invoke Proposition 2.4 to conclude that the treewidth of R is $O(\sqrt{k})$.

Let $(T, \mathcal{B})$ be the tree decomposition of R given by Theorem 2.2. For every $t \in V(T)$, let G_t^* be the torso at t. We observe that G_t^* is also planar: whenever we add an edge xy to $G[B_t]$ to obtain G_t^*, the 2-connectivity of R implies that there is a path from x to y with all internal vertices outside B_t.

Claim 4.10. *Every bag B_t has size $O(k)$.*

Proof. For any neighbor t' of t in T, let us define $T_{t'}$ to be the component of $T \setminus t$ containing t' and let $V_{t'} = \bigcup_{t'' \in V(T_{t'})} B_{t''}$, the set of vertices appearing in the bags of $T_{t'}$. Let $Z_{t'} = B_t \cap B_{t'} = \{x, y\}$. If $Z_{t'} = V_{t'}$, then we may remove t' and the component of $T \setminus t$ containing t' from the tree decomposition.

Suppose now that $V_{t'} \setminus Z_{t'}$ is not empty. Observe that each of C_1 and C_2 has at least two edges between $V_{t'} \setminus Z_{t'}$ and $Z_{t'}$, and two edges between $Z_{t'}$ and $V(R) \setminus V_{t'}$. This is only possible if each of C_1 and C_2 has exactly one edge connecting each of x and y with $V_{t'} \setminus Z_{t'}$, that is, for $i = 1, 2$, cycle C_i has a path P_i from x to y with internal vertices in $V_{t'} \setminus Z_{t'}$, and no other edges incident to $V_{t'} \setminus Z_{t'}$.

We claim that $V_{t'} \setminus Z_{t'}$ contains at least one vertex of S. If not, then $V_{t'} \setminus Z_{t'}$ can be removed from the realization and replaced by adding, for $i = 1, 2$, an edge xy to C_i whose length is the same as the length of P_i. By the existence of the paths discussed above, the resulting realization is planar. It follows that $V_{t'} \setminus Z_{t'}$ contains at least one vertex of S, thus t can have at most k neighbors in T. Let X be the union of $B_{t'} \cap B_t$ taken over all neighbors t' of t in T; we have $|X| \leq 2k$.

Let Y be the set of vertices of B_t that are at distance at most 4 from $B \cup X$ (distance is measured in R and *not* in the torso). As R is 4-regular, $|Y| \leq 4^5 |B \cup X| = O(k)$. Thus if $B_t = Y$, then $|B_t| = O(k)$, and we are done. Otherwise, let v be a vertex that is at distance at least 5 from $B \cup X$ and let F be a face incident to v (the face F is good, as v is not in B). Note that F cannot contain

a 2-separator: by the remark after Theorem 2.2, the 2-separators are of the form $B_{t'} \cap B_{t''}$ for two distinct nodes t' and t'', i.e., they have to contain vertices from bags other than B_t and all such vertices are in X. Therefore, the conditions of Claim 4.9 hold, and there exists a grid, a contradiction. ⌟

Finally, we can complete the proof of Lemma 4.5. By Claim 4.10, every bag has size $O(k)$. As every torso G_t^* is planar, Theorem 2.1 implies that every torso G_t^* has treewidth $O(\sqrt{k})$. By Proposition 2.4, it follows that R has treewidth $O(\sqrt{k})$. □

Now we are ready to prove the main combinatorial result, Theorem 1.2.

Proof. of Theorem 1.2. Let us invoke Lemma 4.4 on the metric d and the 4-opt solution T_4; let T_{opt} be the resulting optimal solution and (G', C_1', C_2') be the minimal representation of (T_4, T_{opt}) containing no grid. By Lemma 4.5, the treewidth of G' is $O(\sqrt{k})$.

We construct a graph G'' from G' by replacing each vertex $v \in V(G')$ with two adjacent vertices v^1, v^2, and for every edge $xy \in E(G')$, we make every vertex in $\{x^1, x^2\}$ adjacent to every vertex in $\{y^1, y^2\}$. If G' has a tree decomposition of width $w-1$ (that is, maximum bag size w), then it is clear that G'' has a tree decomposition of width $2w-1$ (that is, maximum bag size $2w$). Therefore, the treewidth of G'' is also $O(\sqrt{k})$.

We claim that $T_4 \cup T_{\text{opt}}$ is a minor of G'', hence the treewidth bound of $O(\sqrt{k})$ follows also for $T_4 \cup T_{\text{opt}}$. We define the cycle C_4'' by replacing every $v \in V(C_4)$ with v^1. We define cycle C_{opt}'' by replacing every $v \in V(C_{\text{opt}}) \setminus S$ with v^2 and replacing every $v \in S$ with v^1. Now it is easy to see that $C_4'' \cup C_{\text{opt}}''$ is a subdivision of $T_4 \cup T_{\text{opt}}$, hence $T_4 \cup T_{\text{opt}}$ is a minor of G''. □

References

[1] S. Arora. Polynomial-time approximation schemes for Euclidean TSP and other geometric problems. *Journal of the ACM*, 45(5):753–782, 1998.

[2] S. Arora, M. Grigni, D. R. Karger, P. N. Klein, and A. Woloszyn. A polynomial-time approximation scheme for weighted planar graph TSP. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 33–41, 1998.

[3] A. Atserias, A. A. Bulatov, and V. Dalmau. On the power of k-consistency. In *ICALP*, pages 279–290, 2007.

[4] R. Bellman. Dynamic programming treatment of the travelling salesman problem. *J. ACM*, 9(1):61–63, Jan. 1962.

[5] A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto. The travelling salesman problem in bounded degree graphs. In L. Aceto, I. Damgrd, L. Goldberg, M. Halldrsson, A. Inglfsdttir, and I. Walukiewicz, editors, *Automata, Languages and Programming*, volume 5125 of *Lecture Notes in Computer Science*, pages 198–209. Springer Berlin Heidelberg, 2008.

[6] H. Chen and V. Dalmau. Beyond hypertree width: Decomposition methods without decompositions. In *CP*, pages 167–181, 2005.

[7] N. Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical Report 388, Graduate School of Industrial Administration, Carnegie Mellon University, 1976.

[8] W. Cook and P. D. Seymour. Tour merging via branch-decomposition. *INFORMS Journal on Computing*, 15(3):233–248, 2003.

[9] E. D. Demaine, F. V. Fomin, M. T. Hajiaghayi, and D. M. Thilikos. Subexponential parameterized algorithms on bounded-genus graphs and h-minor-free graphs. *J. ACM*, 52(6):866–893, 2005.

[10] E. D. Demaine and M. Hajiaghayi. The bidimensionality theory and its algorithmic applications. *Comput. J.*, 51(3):292–302, 2008.

[11] R. Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

[12] F. Dorn, E. Penninkx, H. L. Bodlaender, and F. V. Fomin. Efficient exact algorithms on planar graphs: Exploiting sphere cut decompositions. *Algorithmica*, 58(3):790–810, 2010.

[13] D. Eppstein. The traveling salesman problem for cubic graphs. In F. Dehne, J.-R. Sack, and M. Smid, editors, *Algorithms and Data Structures*, volume 2748 of *Lecture Notes in Computer Science*, pages 307–318. Springer Berlin Heidelberg, 2003.

[14] H. Gebauer. Enumerating all Hamilton cycles and bounding the number of Hamilton cycles in 3-regular graphs. *Electr. J. Comb.*, 18(1), 2011.

[15] M. Grigni, E. Koutsoupias, and C. H. Papadimitriou. An approximation scheme for planar graph TSP. In *FOCS*, pages 640–645, 1995.

[16] M. Grohe. Fixed-point definability and polynomial time on graphs with excluded minors. *J. ACM*, 59(5):27:1–27:64, Nov. 2012.

[17] M. Held and R. Karp. The traveling-salesman problem and minimum spanning trees: Part II. *Mathematical Programming*, 1(1):6–25, 1971.

[18] K. Iwama and T. Nakashima. An improved exact algorithm for cubic graph TSP. In G. Lin, editor, *Computing and Combinatorics*, volume 4598 of *Lecture Notes in Computer Science*, pages 108–117. Springer Berlin Heidelberg, 2007.

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

[20] D. Karapetyan and G. Gutin. Lin-Kernighan heuris-

tic adaptations for the generalized traveling salesman problem. *European Journal of Operational Research*, 208(3):221–232, 2011.

[21] D. Karapetyan and G. Gutin. Efficient local search algorithms for known and new neighborhoods for the generalized traveling salesman problem. *European Journal of Operational Research*, 219(2):234–251, 2012.

[22] P. N. Klein. A linear-time approximation scheme for planar weighted TSP. In *FOCS*, pages 647–657, 2005.

[23] P. N. Klein. A subset spanner for planar graphs, with application to subset TSP. In *STOC*, pages 749–756, 2006.

[24] P. N. Klein. A linear-time approximation scheme for TSP in undirected planar graphs with edge-weights. *SIAM Journal on Computing*, 37(6):1926–1952, 2008.

[25] T. Kloks. *Treewidth*, volume 842 of *Lecture Notes in Computer Science*. Springer, Berlin, 1994.

[26] S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21(2):pp. 498–516, 1973.

[27] D. Lokshtanov, D. Marx, and S. Saurabh. Lower bounds based on the Exponential Time Hypothesis. *Bulletin of the EATCS*, 105:41–72, 2011.

[28] D. Marx. Searching the k-change neighborhood for TSP is W[1]-hard. *Oper. Res. Lett.*, 36(1):31–36, 2008.

[29] J. Mitchell. Guillotine subdivisions approximate polygonal subdivisions: Part ii – a simple PTAS for geometric k-MST, TSP, and related problems. *SIAM Journal on Computing*, 28:298–1309, 1999.

[30] S. Rao and W. Smith. Approximating geometrical graphs via "spanners" and "banyans". In *Proceedings of the 30th Annual ACM Symposium on the Theory of Computing*, pages 540–550, 1998.

[31] N. Robertson, P. D. Seymour, and R. Thomas. Quickly excluding a planar graph. *J. Comb. Theory, Ser. B*, 62(2):323–348, 1994.

[32] W. D. Smith and N. C. Wormald. Geometric separator theorems & applications. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, pages 232–243, 1998.