The Even-Path Problem in Directed Single-Crossing-Minor-Free Graphs

Archit Chauhan
Chennai Mathematical Institute, India

Samir Datta
Chennai Mathematical Institute & UMI ReLaX, India

Chetan Gupta
Indian Institute of Technology, Roorkee, India

Vimal Raj Sharma
Indian Institute of Technology, Jodhpur, India

Abstract

Finding a simple path of even length between two designated vertices in a directed graph is a fundamental NP-complete problem [24] known as the EvenPath problem. Nedev [28] proved in 1999, that for directed planar graphs, the problem can be solved in polynomial time. More than two decades since then, we make the first progress in extending the tractable classes of graphs for this problem. We give a polynomial time algorithm to solve the EvenPath problem for classes of H-minor-free directed graphs,[1] where H is a single-crossing graph.

We make two new technical contributions along the way, that might be of independent interest. The first, and perhaps our main, contribution is the construction of small, planar, *parity-mimicking networks*. These are graphs that mimic parities of all possible paths between a designated set of terminals of the original graph.

Finding vertex disjoint paths between given source-destination pairs of vertices is another fundamental problem, known to be NP-complete in directed graphs [14], though known to be tractable in planar directed graphs [34]. We encounter a natural variant of this problem, that of finding disjoint paths between given pairs of vertices, but with constraints on parity of the total length of paths. The other significant contribution of our paper is to give a polynomial time algorithm for the 3-*disjoint paths with total parity problem*, in directed planar graphs with some restrictions (and also in directed graphs of bounded treewidth).

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Theory of computation → Graph algorithms analysis

Keywords and phrases Graph Algorithms, EvenPath, Polynomial-time Algorithms, Reachability

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.43

Related Version *Full Version*: `http://arxiv.org/abs/2407.00237`

Funding *Archit Chauhan*: Partially supported by a grant from Infosys foundation and TCS PhD fellowship.
Samir Datta: Partially supported by a grant from Infosys foundation.

Acknowledgements We would like to thank anonymous reviewers for their useful comments and corrections in past submissions. We would also like to thank Martin Grohe for pointing out some references. We are also grateful to Geevarghese Philip and Vishwa Prakash H .V. for their help in reading some sections and improving the presentation of this paper.

[1] Throughout this paper, when referring to concepts like treewidth or minors of directed graphs, we intend them to apply to the underlying undirected graph.

1 Introduction

Given a directed graph G, and two vertices s and t in it, checking for the existence of a a simple directed path from s to t is a fundamental problem in graph theory, known as the Reachability problem. The EvenPath problem is a variant of Reachability, where given a directed graph G and two vertices s and t we need to answer whether there exists a simple path of even length from s to t. EvenPath was shown to be NP-complete by LaPaugh and Papadimitriou [24] via a reduction from an NP-complete problem, the Path-Via-A-Vertex problem. On the other hand, they also show in [24] that its undirected counterpart is solvable in linear time. Several researchers have recently studied both the space, and simultaneous time-space complexity of EvenPath for special classes of graphs [5, 10]. A similar problem, that of finding a simple directed cycle of even length, called EvenCycle (which easily reduces to EvenPath), has also received significant attention. While polynomial-time algorithms have been known since long for the undirected version ([24, 39]), the question of tractablility of the directed version was open for over two decades before polynomial-time algorithms were given by McCuiag, and by Robertson, Seymour and Thomas [27]. More recently, Björklund, Husfeldt and Kaski [3] gave a randomized polynomial-time algorithm for finding a *shortest* even directed cycle in directed graphs.

Although EvenPath is NP-complete for general directed graphs, it is natural and interesting to investigate the classes of graphs for which it can be solved efficiently. In 1994, before the algorithm of [27], Gallucio and Loebl [15] gave a polynomial-time algorithm for EvenCycle in planar directed graphs. They did so by developing a routine for a restricted variant of EvenPath (when s, t lie on a common face, and there are no even directed cycles left on removal of that face). Following that, Nedev in 1999, showed that EvenPath in planar graphs is polynomial-time solvable [28]. Planar graphs are an example of a *minor-closed* family, which are families of graphs that are closed under edge contraction and deletion. Minor-closed families include many more natural classes of graphs, like graphs of bounded genus, graphs of bounded treewidth, apex graphs. A theorem of Robertson-Seymour [33] shows that every minor-closed family can be characterized by a set of finite forbidden minors. Planar graphs, for example, are exactly graphs with $K_{3,3,}, K_5$ as forbidden minors [38]. In this paper, we consider the family of H-minor-free graphs, where H is any fixed single-crossing graph, i.e., H can be drawn on the plane with at most one crossing. Such families are called single-crossing-minor-free graphs. They include well-studied classes of graphs like K_5-minor-free graphs, $K_{3,3}$-minor-free graphs.[2] Robertson and Seymour showed that single-crossing-minor free graphs admit a decomposition by (upto) 3-clique-sums, into pieces that are either of bounded treewidth, or planar [30]. This is a simpler version of their more general theorem regarding decomposition of H-minor free graphs, (where H is any fixed graph) by clique sums, into more complex pieces, involving apices and vortices [32]. Solving EvenPath on single-crossing-minor free graphs would therefore be a natural step to build an attack on more general minor closed familes.

Many results on problems like reachability, matching, coloring, isomorphism, for planar graphs have been extended to $K_{3,3}$-minor-free graphs and K_5-minor-free graphs as a next step (see [36, 21, 22, 37, 35, 11, 2]). Chambers and Eppstein showed in [6] that using the results of [4, 17] for maximum flows in planar and bounded treewidth graphs, respectively, maximum flows in single-crossing-minor-free graphs can be computed efficiently. Following the result of [1] which showed that perfect matching in planar graphs can be found in NC, Eppstein and Vazirani in [13], extended the result to single-crossing-minor-free graphs.

[2] Both $K_5, K_{3,3}$ have crossing number one. Also, note that both the families, K_5-minor-free, and $K_{3,3}$-minor-free, have graphs of $O(n)$ genus.

1.1 Our Contributions

From here onwards, we will drop the term "directed" and assume by default that the graphs we are referring to are directed, unless otherwise stated. Operations like clique sums, decomposing the graphs along separating triples, pairs, etc., will be applied on the underlying undirected graphs. The following is the main theorem we prove in this paper:

▶ **Theorem 1.** *Given an H-minor-free graph G for any fixed single-crossing graph H, the* EvenPath *problem in G can be solved in polynomial time.*

We first apply the theorem of Robertson-Seymour (theorem 3), and decompose G using 3-clique sums into pieces that are either planar or of bounded treewidth. Though EvenPath is tractable in planar graphs, and can also be solved in bounded treewidth graphs by Courcelle's theorem, straightforward dynamic programming does not yield a polynomial-time algorithm for the problem, as we will explain in subsequent sections. One of the technical ingredients that we develop to overcome the issues is that of *parity-mimicking networks*, which are graphs that preserve the parities of various paths between designated terminal vertices of the graph it mimics. We construct them for upto three terminal vertices. The idea of mimicking networks has been used in the past in other problems, like flow computation [6, 7, 17, 20, 23], and in perfect matching [13]. The ideas we use for constructing parity mimicking networks however, do not rely on any existing work that we know of. For technical reasons, we require our parity mimicking networks to be of bounded treewidth *and* planar, with all terminals lying on a common face. These requirements make it more challenging to construct them (or even to check their existence), than might seem at a first glance. One of our main contributions is to show (in lemma 8) the construction of such networks, for upto three terminals. It might be of independent interest to see if a more simpler construction exists (perhaps a constructive argument to route paths, that has eluded us so far), that avoids the hefty case analysis we do, and also if they can be constructed for more than three terminals.

We also come across a natural variant of another famous problem. Suppose we are given a graph G and vertices $s_1, t_1, s_2, t_2 \dots s_k, t_k$ (we may call them terminals) in it. The problem of finding pairwise vertex disjoint paths, from each s_i to t_i is a well-studied problem called the disjoint paths problem. In undirected graphs, the problem is in P when k is fixed [31, 29], but NP-complete otherwise [26]. For (directed) graphs, the problem is NP-complete even for $k = 2$ [14]. In planar graphs, it is known to be in P for fixed k [34, 9, 25]. We consider this problem, with the additional constraint that the sum of lengths of the s_i-t_i paths must be of specified parity. We hereafter refer to the parity of the sum of lengths as total parity, and refer to the problem as DisjPathsTotalParity. In the undirected setting, a stricter version of this problem has been studied, where each s_i-t_i path must have parity p_i that is specified in input. This problem was shown to be in P for fixed k, by Kawarabayashi et al. [18]. However much less is known in directed setting. While DisjPathsTotalParity can be solved for fixed k in bounded treewidth graphs using Courcelle's theorem [8], we do not yet know if it is tractable in planar graphs, even for $k = 2$. The other main technical contribution of our paper is in lemma 10, where we show that in some special cases, i.e., when there are four terminals, three of which lie on a common face of a planar graph, DisjPathsTotalParity can be solved in polynomial time for $k = 3$. We do this by showing that under the extra constraints, the machinery developed by [28] can be further generalized and applied to find a solution in polynomial time. The question of tractability of DisjPathsTotalParity in planar graphs, without any constraint of some terminals lying on a common face is open, and would be interesting to resolve. A polynomial-time algorithm for it (for fixed k), would yield a polynomial-time algorithm for EvenPath in graphs with upto k crossings, which is currently unknown.

Though the proofs of lemmas 8,10 form the meat of technical contributions of the paper, we give a proof idea, deferring the proofs to the full version of the paper.

2 Preliminaries

From now onwards we will refer to simple, directed paths as just paths. For a path P, and a pair of vertices u and v on P, such that u occurs before v in P, $P[u, v]$ denotes the subpath of P from u to v. If P_1 and P_2 are two paths that are vertex disjoint, except possibly sharing starting or ending vertices, then we say that P_1, P_2 are *internally disjoint* paths. If P_1's ending vertex is same as the starting vertex of P_2, then we denote the concatenation of P_1 and P_2 by $P_1.P_2$. We will use the numbers $0, 1$ to refer to *parities*, 0 for even parity and 1 for odd parity. We say a path P is of *parity* p ($p \in \{0, 1\}$), if its length modulo 2 is p. We will use a well-known structural decomposition of H-minor-free graphs due to [30]. We recall the definition of clique sums:

▶ **Definition 2.** *A k-clique-sum of two graphs G_1, G_2 can be obtained from the disjoint union of G_1, G_2 by identifying a clique in G_1 of at most k vertices with a clique of the same number of vertices in G_2, and then possibly deleting some of the edges of the merged clique.*

Thus when separating G along a separating pair/triplet, we can add virtual edges if needed, to make the separating pair/triplet a clique. The virtual edges will not be used in computation of path parities, they are only used to compute the decomposition. We can keep track of which edges in the graph are virtual edges and which are the real edges throughout the algorithm. We can repeatedly apply this procedure to decompose any graph G into smaller pieces. The following is a theorem from [30].

▶ **Theorem 3** (Robertson-Seymour [30]). *For any single-crossing graph H, there is an integer τ_H such that every graph with no minor isomorphic to H is either*
1. *the proper* 0-, 1-, 2- *or* 3-*clique-sum of two graphs, or*
2. *planar*
3. *of treewidth $\leq \tau_H$.*

Thus, every H-minor-free graph, where H is a single-crossing graph, can be decomposed by 3-clique sums into graphs that are either planar or have treewidth at most τ_H. Polynomial time algorithms are known to compute this decomposition [12, 19, 16] (and also NC algorithms [13]). The decomposition can be thought of as a two colored tree (see [12, 6, 13] for further details on the decomposition), where the blue colored nodes represent *pieces* (subgraphs that are either planar or have bounded treewidth), and the red nodes represent cliques at which two or more pieces are attached. We call these nodes of the tree decomposition as *piece nodes* and *clique nodes*, respectively. The edges of the tree describe the incidence relation between pieces and cliques (see Figure 2). We will denote this decomposition tree by T_G. We will sometimes abuse notation slightly and refer to a piece of T_G (and also phrases like *leaf* piece, *child* piece), when it is clear from the context that we mean the piece represented by the corresponding node of T_G. Note that the bounded treewidth and planarity condition on the pieces we get in the decomposition, is along with their virtual edges. As explained in [6, 13], we can assume that in any planar piece of the decomposition, the vertices of a separating pair or triplet lie on a common face (Else we could decompose the graph further).

Suppose G decomposes via a 3-clique sum at clique c into G_1 and G_2. Then we write G as $G_1 \oplus_c G_2$. More generally, if $G_1, G_2, \ldots, G_\ell$ all share a common clique c, then we use $G_1 \oplus_c G_2 \oplus_c \ldots \oplus_c G_\ell$ to mean $G_1, G_2, \ldots, G_\ell$ are glued together at the shared clique. If it is clear from the context which clique we are referring to, we will sometimes drop the subscript

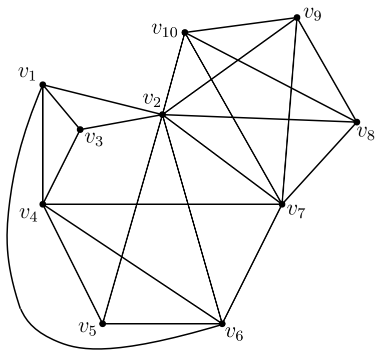

Figure 1 An example of a graph G. We ignore directions here.

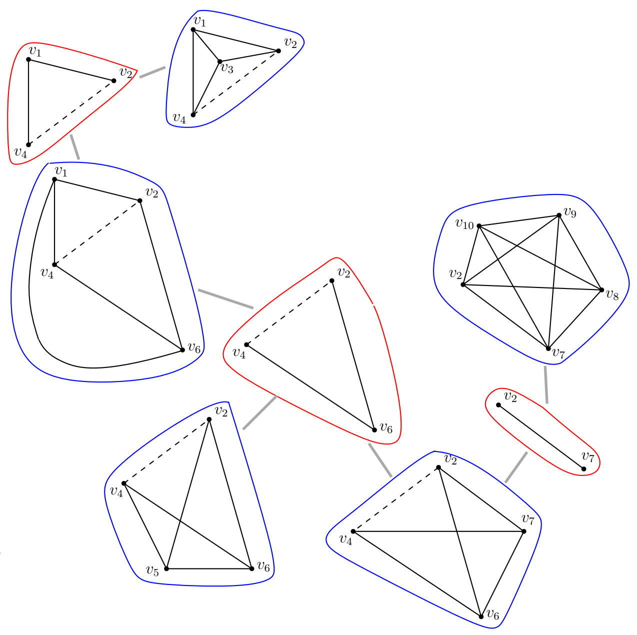

Figure 2 A clique sum decomposition of G. Red nodes are the clique nodes and blue node the piece nodes. Dashed edges denote virtual edges.

and simply use $G_1 \oplus G_2 \oplus \ldots \oplus G_\ell$ instead. Suppose G_2' is a graph that contains the vertices of the clique c shared by G_1 and G_2. We denote by $G[G_2 \to G_2']$, the graph $G_1 \oplus_c G_2'$, i.e., replacing the subgraph G_2 of G, by G_2', keeping the clique vertices intact. We will also use the notion of *snapshot* of a path in a subgraph. If G can be decomposed into G_1 and G_2 as above, and P is an s-t path in G, its snapshot in G_1 is the set of maximal subpaths of P, restricted to vertices of G_1. Within a piece, we will sometimes refer to the vertices of separating cliques, and s and t, as *terminals*.

In figures, we will generally use the convention that a single arrow denotes a path segment of odd parity and double arrow denotes a path segment of even parity, unless there is an explicit expression for the parity mentioned beside the segment.

3 Overview and Technical Ingredients

We first compute the 3-clique sum decomposition tree of G, T_G. We can assume that s, t, each occur in only one piece of T_G, S and T, respectively.[3] We call the pieces S and T, along with the pieces corresponding to nodes that lie in the unique path in T_G joining S and T, as the *main* pieces of T_G, and the remaining pieces are called the *branch* pieces of T_G. We will assume throughout that T_G is rooted at S.

The high level strategy of our algorithm follows that of [6]. The algorithm has two phases. In the first phase, we simplify the branch pieces of the decomposition tree. Any s-t even path P must start and end inside the main pieces S and T, respectively. However, it may

[3] If they are part of a separating vertex/pair/triplet then they may occur in multiple pieces of T_G. Say s is a part of many pieces in T_G. To handle that case, we can introduce a dummy s' and add an edge from s' to s and reduce the problem to finding an odd length path from s' to t. The vertex s' now will occur in a unique piece in T_G. Vertex t can be handled similarly.

take a detour into the branch pieces. Suppose L is a leaf branch piece of T_G, attached to its parent piece, say G_i, via a 3-clique c. Using Nedev's algorithm or Courcelle's theorem, we can find paths of various parities between vertices of c in L, which constitutes the *parity configuration* of L with respect to c (formally defined in next subsection). We will replace L by a parity mimicking network of L with respect to vertices of c, L'. L' will mimic the parity configuration of L and hence preserve the parities of all s-t paths of original graph. The parity mimicking networks we construct are small and planar, with the terminals (vertices of c) all lying on a common face, as decribed in lemma 8. Therefore, if G_i is of bounded treewidth, then $G_i \oplus L'$ will be of bounded treewidth. And if G_i is planar, then we can plug L' in the face of G_i that is common to vertices of c, and $G_i \oplus L'$ will be planar. This allows us compute the parity configurations of the merged piece, and repeat this step until a single branch, i.e. a path, remains in the decomposition tree, consisting only of the main pieces (connected by cliques), including S and T.

In the second phase, we start simplifying the main pieces, starting with the leaf piece T. Instead of a single mimicking network for T, we will store a set of small networks, each of them mimicking a particular snapshot of a solution. We call them *projection networks*. Since a snapshot of an s-t even path in T can possibly be a set of disjoint paths between the (upto) four terminals in T, we require the DisjPathsTotalParity routine of lemma 10 to compute these projection networks. We combine the parent piece with each possible projection network. The merged piece will again be either planar or of bounded treewidth, allowing us to continue this operation towards the root node until a single piece containing both s and t remains. We query for an s-t even path in this piece and output yes iff there exists one. At each step, the number of projection networks used to replace the leaf piece, and their combinations with its parent piece will remain bounded by a constant number.

Once we have the decision version of EvenPath, we show a poly-time self-reduction using the decision oracle of EvenPath to construct a solution, in the full version of the paper.

Necessity of a two phased approach

We mention why we have two phases and different technical ingredients for each.

- Instead of a single parity mimicking network, we need a set of projection networks for the leaf piece in the second phase because it can have upto four terminals (three vertices of the separating clique and the vertex t), and we do not yet know how to find (or even the existence of) parity mimicking networks with the constraints we desire, for graphs with four terminals.
- We cannot however use a set of networks for each piece in phase I because of the unbounded degree of T_G. Suppose a branch piece G_i is connected to its parent piece by clique c, and suppose G_i has child pieces $L_1, L_2, \ldots, L_\ell$, attached to G_i via disjoint cliques $c_1, c_2, \ldots, c_\ell$, respectively. An even s-t path can enter G_i via a vertex of c, then visit any of $L_1, L_2, \ldots, L_\ell$ in any order and go back to the parent of G_i via another vertex of c. If we store information regarding parity configurations of $L_1, L_2, \ldots, L_\ell$ as sets of projection networks, we could have to try exponentially many combinations to compute information of parity configurations between vertices of c in the subtree rooted at G_i (note that ℓ could be $O(n)$). Therefore, we compress the information related to parity configurations of L_i into a single parity mimicking network L_i', while preserving solutions, so that the combined graph $(((G_i \oplus_{c_1} L_1') \oplus_{c_2} L_2' \ldots) \oplus_{c_\ell} L_\ell')$ is either planar or of bounded treewidth.

We will now describe these ingredients formally in the remaining part of this section.

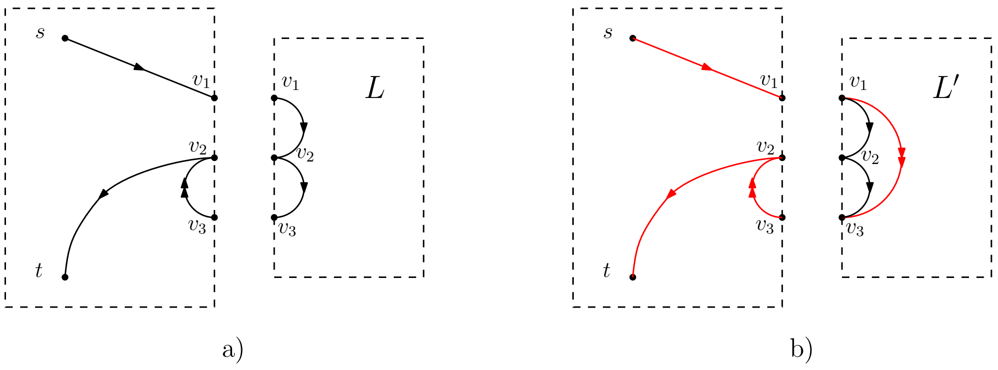

Figure 3 Figure a) shows the input graph and b) shows the graph with L replaced by an erroneous mimicking network L'. Suppose the original graph in a) has no s-t even path but does have an s-t even walk as shown in the figure, using vertex v_2 twice. If we query for a path from v_1 to v_3 in L, and add a direct v_1 to v_3 path of that parity in L', we end up creating a false solution since v_2 is freed up to be used outside L'. Hence there must be equality between corresponding direct sets.

3.1 Parity Mimicking Networks

We first define the parity configuration of a graph, which consists of subsets of $\{0, 1\}$ for each pair, triplet of terminals, depending on whether there exists "direct" or "via" paths of parity even, odd, or both (we use 0 for even parity and 1 for odd). We formalise this below.

▶ **Definition 4.** *Let L be a directed graph and $T(L) = \{v_1, v_2, v_3\}$ be the set of terminal vertices of L. Then, $\forall i, j, k \in \{1, 2, 3\}$, such that i, j, k are distinct, we define the sets* $\mathrm{Dir}_L(v_i, v_j)$, *and* $\mathrm{Via}_L(v_i, v_k, v_j)$ *as:*

- $\mathrm{Dir}_L(v_i, v_j) =$*{p | there exists a path of parity p from v_i to v_j in $L - v_k$ }*
- $\mathrm{Via}_L(v_i, v_k, v_j) =$*{p | there exists a path of parity p from v_i to v_j via v_k* and *there does not exist a path of parity p from v_i to v_j in $L - v_k$}*

We say that the $\mathrm{Dir}_L, \mathrm{Via}_L$ *sets constitute the* parity configuration *of the graph L with respect to $T(L)$. We call the paths corresponding to elements in* $\mathrm{Dir}_L, \mathrm{Via}_L$ *sets as* Direct paths *and* Via paths*, respectively.*

The parity configuration of a graph can be visualised as a table. We have defined it for three terminals, it can be defined in a similar way for two terminals. It is natural to ask the question that given a parity configuration $\mathcal{P}$ independently with respect to some terminal vertices, does there exist a graph with those terminal vertices, realising that parity configuration. If not, we say that $\mathcal{P}$ is unrealisable. It is easy to see that the number of parity configurations for a set of three terminals is bounded by 4^{12}, many of which are unrealizable. We now define parity mimicking networks.

▶ **Definition 5.** *A graph L' is a parity mimicking network of a another graph L (and vica versa), if they share a common set of terminals, and have the same parity configuration, $\mathcal{P}$, w.r.t. the terminals. We also call them parity mimicking networks of parity configuration $\mathcal{P}$.*

The reason we differentiate between direct paths and via paths while defining parity configurations is to ensure that no false solutions are introduced on replacing a leaf piece of T_G by its mimicking network (see Figure 3). Note that in our definition of *Via* sets, we exclude parity entries of via paths between two terminals if that parity is already present in *Dir* set between the same terminals. We do so because this makes the parity configurations easier to enumerate in our construction of parity mimicking networks. In Figure 4, we describe why doing this will still preserve solutions.

We also need to consider the case where multiple leaf pieces, in T_G are attached to a common parent piece via a shared clique (as seen in Figure 2). In this case, we will replace the entire subgraph corresponding to the clique sum of the sibling leaf pieces by one parity mimicking network. To compute the parity configuration of the combined subgraph of leaf pieces, we make the following observation:

▶ **Observation 6.** *Let $L_1, L_2, \ldots, L_\ell$ be leaf branch pieces that are pairwise disjoint except for a common set of terminal vertices, say $\{v_1, v_2, v_3\}$. Let $L = L_1 \oplus L_2 \oplus \ldots \oplus L_\ell$. Then the parity configuration of L with respect to $\{v_1, v_2, v_3\}$ can be computed by:*

$$\mathrm{Dir}_L(v_i, v_j) = \bigcup_{a=1}^{\ell} \mathrm{Dir}_{L_a}(v_i, v_j) \tag{1}$$

$$\mathrm{Via}_L(v_i, v_k, v_j) = \left(\bigcup_{a=1}^{\ell} \mathrm{Via}_{L_a}(v_i, v_k, v_j) \cup \bigcup_{a,b=1}^{\ell} \left(\mathrm{Dir}_{L_a}(v_i, v_k) \boxplus \mathrm{Dir}_{L_b}(v_k, v_j)\right) \right) \backslash \mathrm{Dir}_L(v_i, v_j) \tag{2}$$

where $A \boxplus B$ denotes the set formed by addition modulo 2 between all pairs of elements in sets A, B, and $i, j, k \in \{1, 2, 3\}$ are distinct.

The intuition behind the observation is simple. Any direct path in L from v_i to v_j must occur as a direct path in one of $L_1, L_2 \ldots L_\ell$ since they are disjoint except for terminal vertices. Any via path in L from v_i to v_j via v_k can occur in two ways, either as a v_i-v_k-v_j via path in one of $L_1, L_2 \ldots L_\ell$, or as a concatenation of two direct paths, one from v_i to v_k in some piece L_i, and another from v_k to v_j in another piece L_j. Note that although the observation is for the case when all $L_1, L_2, \ldots, L_\ell$ share a common 3-clique $\{v_1, v_2, v_3\}$, it is easy to see it can be tweaked easily to handle the cases when some of the $L_i's$ are attached via a 2-clique that is a subset of the 3-clique.

The next lemma states that replacing leaf piece nodes in T_G by parity mimicking networks obeying some planarity conditions, will preserve the existence of s-t paths of any particular parity, and also preserve conditions on treewidth and planarity for the combined piece.

▶ **Lemma 7.** *Let G be a graph with clique sum decomposition tree T_G, and let $L_1, L_2 \ldots, L_\ell$ be set of leaf branch pieces of T_G, attached to their parent piece G_1 via a common clique c. Let L' be a parity mimicking network of $L_1 \oplus L_2 \oplus \ldots L_\ell$ with respect to c, such that L' is planar, and vertices of c lie on a common face in L'. Then:*

1. *There is a path of parity p from s to t in G iff there is a path of parity p from s to t in $G[L_1 \oplus L_2 \oplus \ldots L_\ell \to L']$.*
2. *If G_1 is planar, then $G_1 \oplus L'$ is also planar.*
3. *If G_1 has treewidth τ_H, and L' has treewidth $\tau_{L'}$, then $G_1 \oplus L'$ has treewidth max$(\tau_H, \tau_{L'})$*

Proof.

1. The proof essentially follows from the definition of parity mimicking networks and observation 6, since we can replace the snapshot of any s-t path P in L_i by a path of corresponding parity in L_i' and vice-versa.
2. This follows since in the decomposition, vertices of separting cliques in every piece lie on the same face, and so is the case for L' by assumption. Therefore we can embed L' inside the face in G_1, on the boundary of which v_1, v_2, v_3 lie.
3. This follows since we can merge tree decompositons of G_1, L' along bags consisting of the common clique. ◀

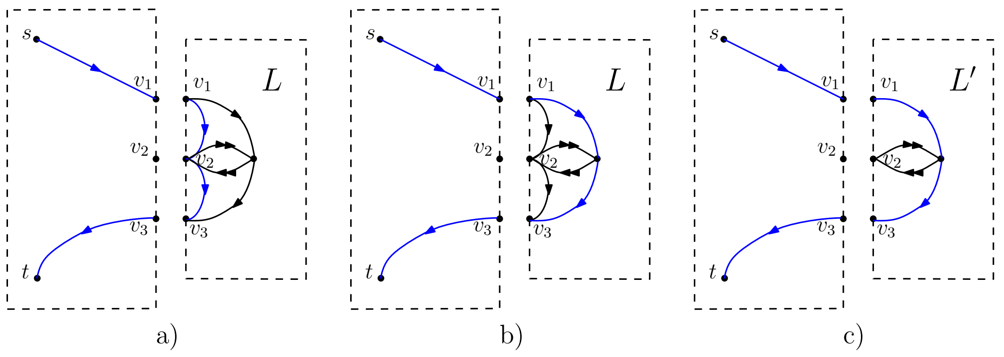

Figure 4 Figure a) denotes the original graph which has both a direct path, as well as a via path of even parity from v_1 to v_3. Suppose the via path is part of an even s-t path solution, as marked by blue. Then in L itself, we could replace the via path by the direct path and it would still be a valid even s-t path, as marked in blue in b). Hence in the mimicking network L', too (shown in c)), we could use the direct v_1 to v_3 path of the same parity. Therefore we do not need to put the parity of the v_1-v_2-v_3 path in $Via_L(v_1, v_2, v_3)$, since the same parity is already present in $Dir_L(v_1, v_3)$, and $Dir_L(v_1, v_3) = Dir_{L'}(v_1, v_3)$.

Now we will show how to compute parity mimicking networks that are small in size (and hence of bounded treewidth), and also planar, with terminal vertices lying on the same face, for a given parity configuration of a graph L.

▶ **Lemma 8.** *Suppose L is a graph with terminals $T(L) = \{v_1, v_2, v_3\}$, and suppose we know the parity configuration of L with respect to $\{v_1, v_2, v_3\}$. We can in polynomial-time, find a parity mimicking network L' of L, with respect to $\{v_1, v_2, v_3\}$ which consists of at most* 18 *vertices, and is also planar, with v_1, v_2, v_3 lying on a common face.*

Proof. We give a brief idea of the proof and defer the full proof to the full version of this paper.As noted above, the number of possible parity configurations are finite (bounded by 4^{12} for three terminals), but the number is too large to enumerate over all of them and individually construct the mimicking networks. We use some observations to make the case analysis tractable. We refer to elements of sets $Dir_L(v_i, v_j)$ as *entries*. But we abuse notation slightly and distinguish them from the boolean values $0, 1$. For example, we always distinguish between an entry of $Dir_L(v_1, v_2)$, and an entry of $Dir_L(v_2, v_3)$, even if they have the same *value* (0 or 1). A natural constructive approach would be to iteratively do the following step for all i, j, p : add a path of length $2 - p$ from v_i to v_j in L', disjoint from existing paths of L', if there is an entry of parity p present in $\mathrm{Dir}_{L'}(v_i, v_j)$. Its easy to check that this will result in a planar L' with terminals on a common face. However, this could lead to wrong parity configurations in L'. For example, L could have a direct paths of parity 1 from v_1 to v_2, and a direct path of parity 0 from v_2 to v_3, but no path of parity 1 from v_1 to v_3, either direct or via v_2 (see Figure 5). We will call pairs of such entries as *bad pairs.* The entries that are part of *any* bad pair are called *bad entries.* Though the example in Figure 5 has a simple fix for the bad pair, it becomes more complicated to maintain the planarity conditions as the number of bad pairs increase. Let $\mathcal{P}_L$ be the parity configuration of L. The idea of the proof is to define a *bad kernel* of $\mathcal{P}_L$, as the *sub-configuration* consisting of all the *bad entries* of $\mathcal{P}_L$. The *closure* of the bad kernel is defined as the parity configuration obtained from it by adding "minimal" number of entiries to make it realizable. We observe that the closure remains a sub-configuration of $\mathcal{P}_L$. Suppose that we can somehow construct a planar mimicking network for the *closure* of the *bad kernel* of $\mathcal{P}_L$, with terminals lying on a

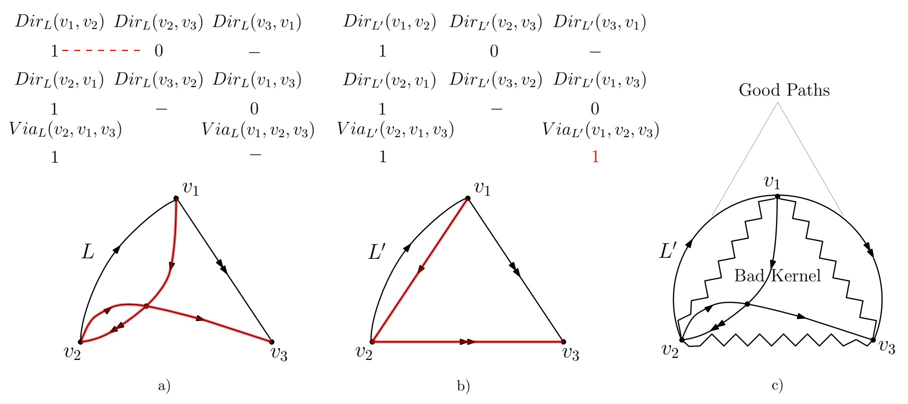
$Dir_L(v_1,v_2)$ $Dir_L(v_2,v_3)$ $Dir_L(v_3,v_1)$
1 0 –
$Dir_L(v_2,v_1)$ $Dir_L(v_3,v_2)$ $Dir_L(v_1,v_3)$
1 – 0
$Via_L(v_2,v_1,v_3)$ $Via_L(v_1,v_2,v_3)$
1 –
$Dir_{L'}(v_1,v_2)$ $Dir_{L'}(v_2,v_3)$ $Dir_{L'}(v_3,v_1)$
1 0 –
$Dir_{L'}(v_2,v_1)$ $Dir_{L'}(v_3,v_2)$ $Dir_{L'}(v_1,v_3)$
1 – 0
$Via_{L'}(v_2,v_1,v_3)$ $Via_{L'}(v_1,v_2,v_3)$
1 1
Good Paths
Bad Kernel
L
L'
v_1
v_2
v_3
a)
b)
c)

Figure 5 Fig a) denotes a graph L with its parity configuration table (only relevant sets). Fig b) denotes a "parity mimicking network", if for each pair of terminals, we just independently put paths of correct parity, disjoint from each other. It leads to an extra path (highlighted in red) from v_1 to v_3 via v_2 in L', of odd parity. Pairs of such entries, for which we cannot add disjoint paths are called bad entries as marked by the dashed red line in the parity configuration table in a). Fig c) outlines the approach used to construct the correct mimicking network. The two paths corresponding to bad pair entries, form the bad kernel, for which we construct a mimicking network by enumerating cases. The remaining paths can be added iteratively, disjoint from all existing paths, on the outer face.

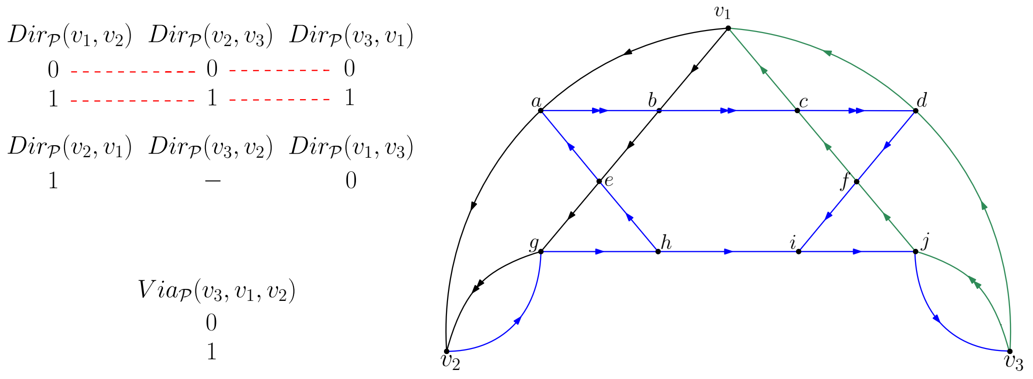
$Dir_{\mathcal{P}}(v_1,v_2)$ $Dir_{\mathcal{P}}(v_2,v_3)$ $Dir_{\mathcal{P}}(v_3,v_1)$
0 0 0
1 1 1
$Dir_{\mathcal{P}}(v_2,v_1)$ $Dir_{\mathcal{P}}(v_3,v_2)$ $Dir_{\mathcal{P}}(v_1,v_3)$
1 – 0
$Via_{\mathcal{P}}(v_3,v_1,v_2)$
0
1
v_1
a
b
c
d
e
f
g
h
i
j
v_2
v_3

Figure 6 An example of a more non-trivial bad kernel, and a mimicking network realising its closure. This is a subcase of case $(4, 0)$ described in the full proof. We give a list of paths along with their lengths, for ease of reader to check that the network obeys the parity configuration.

common face. Then we show that paths corresponding to leftover parity entries of L can be safely added using the constructive approach described above. Hence it suffices to construct parity mimicking networks for closures of all possible bad kernels. We use some observations to show that the number of possible types of bad kernels cannot be too large, and enumerate over each type, explicitly constructing the parity mimicking networks of their closures. ◀

3.2 Disjoint Paths with Parity Problem

In this section, we will define and solve the DisjPathsTotalParity problem for some special cases and types of graphs. We define the problem for three paths between four terminals.

▶ **Definition 9.** *Given a graph G and four distinct terminals $v_1, v_2, v_3,$ and v_4 in $V(G)$, the* DisjPathsTotalParity *problem is to find a set of three pairwise disjoint paths, from v_1 to v_2, v_2 to v_3, and from v_3 to v_4, such that the total parity is even, if such a set of paths exist, and output no otherwise.*

The problem where total parity must be odd can be easily reduced to this by adding a dummy neighbour to v_4. The problem is NP-hard in general graphs since the even path problem trivially reduces to this. We show that the above problem can be solved in polynomial time in following two cases:

▶ **Lemma 10.** *Let G be a graph, and v_1, v_2, v_3, v_4 be four vertices of G. Both decision as well as search versions of* DisjPathsTotalParity *for these vertices as defined above can be solved in polynomial time in the following cases:*

1. *If G has constant treewidth.*
2. *If G is planar and v_1, v_2, v_3 lie on a common face of G.*

Proof. Proofs of both parts can be found in the full version of the paper. We give a high level idea of the proof of the second part. The argument of Nedev for EvenPath uses two main lemmas. One lemma states that if there are two paths P_1, P_2, of different parities from s to t, then their union forms a (at least one) structure, which they call an *odd list superface*. It (roughly) consists of two internally disjoint paths of different parities, with a common starting vertex, say b and a common ending vertex, say e. Let F denote such a superface. They show that there exist two disjoint paths in $P_1 \cup P_2 - F$, one from s to b, and one from e to t. This provides a "switch" in $P_1 \cup P_2$, and if we can find this switch efficiently, then we can use existing 2-disjoint path algorithms to connect s and t via this switch. But the number of odd list superfaces in a graph can be exponential. The second lemma of Nedev says that we can exploit the structure of planarity and show that each of the odd list superfaces formed by $P_1 \cup P_2$, "contain" a "minimal" odd list superface, which they call a *simple* odd list superface, that obeys the same conditions. The set of simple odd list superfaces is small and can be enumerated in polynomial time. In our setting, we start from the case that two instances of three disjoint paths between the specified terminals exist, such that they have different total parity. Say the instances are P_1, P_2, P_3, and P'_1, P'_2, P'_3. At least one of P_i, P'_i must be of different parity. We show that using the constraints of three terminals on a face, and using ideas of *leftmost* (and rightmost) paths of $P_i \cup P'_i$, for each case of $i \in \{1, 2, 3\}$, there does exist an analogous structure: a simple odd list super face, and four disjoint path segments connecting the required vertices. A point to note is that in Nedev's argument, *any* odd list superface formed by P_1, P_2 could be trimmed to a simple odd list superface that would give a valid solution. That does not hold true here. We generalise their lemma, and argue that there does exist *at least one* odd list superface between P_i, P'_i that will work in our setting. ◀

4 Main Algorithm

We now explain the two phases of the algorithm.

4.1 Phase 1

1. Find the 3-clique sum decomposition tree T_G. Mark the piece that contains the vertex s as the root of T_G.
2. Pick any maximal set of leaf branch pieces of T_G, say $L_1, L_2, \ldots, L_\ell$, which are attached to a parent piece G_i via a common clique. Compute their parity configurations using Nedev's algorithm, or using Courcelle's theorem.Then compute the parity configuration of $L_1 \oplus L_2 \oplus \ldots \oplus L_\ell$ using observation 6.
3. Compute the parity mimicking network, L', of $L_1 \oplus L_2 \oplus \ldots \oplus L_\ell$ using lemma 8. Replace $L_1 \oplus L_2 \oplus \ldots \oplus L_\ell$ by L' and merge it with G_i.
4. Since $G_i \oplus L'$ is either of bounded treewidth or is planar by lemma 7, we can repeat this step until no branch pieces remain.

4.2 Phase II

Let G' denote the graph after phase I. After phase I, the modified tree $T_{G'}$ looks like a path of pieces, $G_1, G_2, \ldots, G_m$, joined at cliques $c_1, c_2 \ldots c_{m-1}$.[4] The vertex s is in root piece G_1, and t in leaf piece G_m (we use G_1, G_m instead of S, T here for notational convenience). We can write $G' = G_1 \oplus_{c_1} G_2 \oplus_{c_2} \ldots \oplus_{c_{m-1}} G_m$. Since it is clear in this phase that c_i is the clique joining G_i, G_{i+1}, we will omit the subscript for notational convenience and just write $G_1 + G_2 + \ldots + G_m$ instead. Let $c_{m-1} = \{v_1, v_2, v_3\}$ and let $i, j, k \in \{1, 2, 3\}$ be distinct. The snapshot of any even s-t path P in G_m, can be one of the following four types (see figure 7):

- Type 1 : A path from v_i to t without using v_j, v_k.
- Type 2 : A path from v_i to t via v_j, without using v_k.
- Type 3 : A path from v_i to t via v_j, v_k.
- Type 4 : A path from v_i to v_j and a path from v_k to t, both disjoint from each other.

We call any path/set of paths in G_m of one of the above types as a *potential snapshot* of G_m. We now construct the *projection networks* of potential snapshots of G_m.

▶ **Definition 11.** *Let G_m be the leaf piece as described above with clique $c_{m-1} = \{v_1, v_2, v_3\}$, and vertex t present in G_m.*

- *For each of the types described above, for all $i \in \{1, 2, 3\}$, and for all $p \in \{0, 1\}$, find a potential snapshot (if it exists) in G_m from v_i to t, of total parity p, using lemma 10.*
- *Let J be a potential snapshot found in the previous step, Its* projection network*, is defined as the* graph *obtained from J by keeping terminal vertices intact, and replacing every terminal to terminal path in J by a path of length $2 - p$.*

The type of the projection network is the type of the corresponding potential snapshot.
The set of projection networks *of G_m, denoted by $\mathcal{N}(G_m)$, is the set of all projection networks obtained for G_m by the above procedure.*

See Figure 7 for an example. Since the total number of terminals is at most 4 (with one fixed as t), it is easy to see that the number of possible projections networks for G_m is bounded. Therefore $\mathcal{N}(G_m)$ can be computed in polynomial time. Note that $\mathcal{N}(G_m)$ is not uniquely defined. But it is sufficient for our purpose, to compute any one of the various possible choices of the set $\mathcal{N}(G_m)$ as explained in Figure 7. The next lemma shows that the projection networks of G_m preserve solutions, and also maintain invariants on planarity and treewidth, when merged with the parent piece.

▶ **Lemma 12.** *Given $G' = G_1 + \ldots + G_m$ as described above.*

1. *Given a $\mathcal{N}(G_m)$, there is an s-t path in G' of parity p iff $\exists N \in \mathcal{N}(G_m)$ such that $G'[G_m \to N]$ has an s-t path of parity p.*
2. *If G_{m-1} is planar/of bounded treewidth, then for any projection network $N \in \mathcal{N}(G_m)$, $G_{m-1} + N$ is planar/of bounded treewidth, respectively.*

Proof.

1. This follows from the definition of projection networks. The only minor technical point to note is that $\mathcal{N}(G_m)$ is not unique. For example, suppose there are two potential snaphots in G_m of type 4. One is J_1, consisting of a path P_1 from v_1 to v_2 of even parity, and a path P_2 from v_3 to t of odd parity. The other is J_2, consisting of a path P_1' from v_1 to v_2

[4] Note that the vertices of a clique, say c_i need no longer lie on the same face of G_i after phase I, since we might have merged the parity mimicking network of the branch pieces incident at c_i into the face corresponding to c_i.

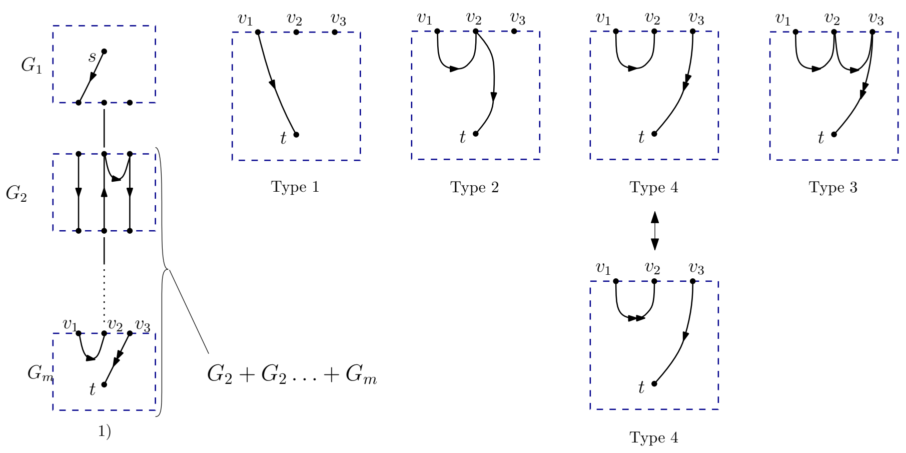

Figure 7 Fig 1) Denotes the decomposition tree after phase 1, with $G_1, G_2 \ldots, G_m$ denoting the pieces. We skip drawing clique nodes here. On the right are examples of projection networks of different types. In fig 1), the snapshot of the s-t path in G_m is of type 4. The two projection networks of type 4 drawn on the right have same total parity, but different parities of individual segments. It is sufficient for our purpose to find any one of them since they are interchangeable.

of odd parity, and a path P_2' from v_3 to t, of even parity. Since the total parity of J_1 and J_2 is same, Lemma 10 could output either one of them. We don't have control over it to find both. But finding any one of them is sufficient for us, since if J_1 is a snapshot of an actual solution, then replacing J_1 by J_2 would also give a valid solution and vice versa.(See Figure 7)

2. Suppose $c_{m_1} = \{v_1, v_2, v_3\}$ is the clique where G_{m-1}, G_m are attached. The argument of treewidth bound is same as that of Lemma 7 in previous phase, when we attached mimicking networks to parent pieces. However if G_{m-1} is planar, there could have been a parity mimicking network L' attached to G_{m-1} via c_{m-1} during phase I. Hence v_1, v_2, v_3 might not lie on a common face in G_{m-1} after phase I. We observe however, since L' was attached at a 3-clique, c_{m-1}, every pair v_i, v_j of vertices of c_{m-1}, must share a common face in G_{m-1}. Now, the projection networks consist of at most three paths, two between v_1, v_2, v_3, and one from them to t. For any v_i, v_j, we can embed the path between v_i, v_j in N, in the face in G_{m-1} shared by v_i, v_j, and finally just add the path leading to t. Therefore if G_{m-1} is planar, all projection networks of G_m can be embedded in their parent nodes. ◀

We make the following observation to compute a $\mathcal{N}(G_i + \ldots G_m)$ recursively:

$$\mathcal{N}(G_i + \ldots G_m) = \bigcup_{N \in \mathcal{N}(G_{i+1} + \ldots G_m)} \mathcal{N}(G_i + N) \tag{3}$$

Thus we can proceed as follows:

1. Compute $\mathcal{N}(G_m)$ using lemma 10
2. For all $N \in \mathcal{N}(G_m)$, compute $G_{m-1} + N$, and hence compute $\mathcal{N}(G_m + G_{m-1})$ using the observation above.
3. For all $N \in \mathcal{N}(G_m + G_{m-1})$, compute $G_{m-2} + N$, and hence compute $\mathcal{N}(G_m + G_{m-1} + G_{m-2})$. Repeat until we reach G_1.

References

1 Nima Anari and Vijay V. Vazirani. Planar graph perfect matching is in nc. *J. ACM*, 67(4), May 2020. doi:10.1145/3397504.

2 Rahul Arora, Ashu Gupta, Rohit Gurjar, and Raghunath Tewari. Derandomizing isolation lemma for $K_{3,3}$-free and K_5-free bipartite graphs. In *Symposium on Theoretical Aspects of Computer Science*, 2014. URL: https://api.semanticscholar.org/CorpusID:1484963.

3 Andreas Björklund, Thore Husfeldt, and Petteri Kaski. The shortest even cycle problem is tractable. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2022, pages 117–130, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3519935.3520030.

4 Glencora Borradaile and Philip Klein. An o(n log n) algorithm for maximum st-flow in a directed planar graph. *J. ACM*, 56(2), April 2009. doi:10.1145/1502793.1502798.

5 Diptarka Chakraborty and Raghunath Tewari. Simultaneous time-space upper bounds for red-blue path problem in planar dags. In M. Sohel Rahman and Etsuji Tomita, editors, *WALCOM: Algorithms and Computation - 9th International Workshop, WALCOM 2015, Dhaka, Bangladesh, February 26-28, 2015. Proceedings*, volume 8973 of *Lecture Notes in Computer Science*, pages 258–269. Springer, 2015. doi:10.1007/978-3-319-15612-5_23.

6 Erin Wolf Chambers and David Eppstein. Flows in one-crossing-minor-free graphs. *Journal of Graph Algorithms and Applications*, 17(3):201–220, 2013. doi:10.7155/jgaa.00291.

7 Shiva Chaudhuri, K. Subrahmanyam, Frank Wagner, and Christos Zaroliagis. Computing mimicking networks. *Algorithmica*, 26:31–49, January 2000. doi:10.1007/s004539910003.

8 Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.

9 Marek Cygan, Daniel Marx, Marcin Pilipczuk, and Michal Pilipczuk. The planar directed k-vertex-disjoint paths problem is fixed-parameter tractable. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 197–206, 2013. doi:10.1109/FOCS.2013.29.

10 Samir Datta, Arjun Gopalan, Raghav Kulkarni, and Raghunath Tewari. Improved bounds for bipartite matching on surfaces. In Christoph Dürr and Thomas Wilke, editors, *29th International Symposium on Theoretical Aspects of Computer Science, STACS 2012, February 29th - March 3rd, 2012, Paris, France*, volume 14 of *LIPIcs*, pages 254–265. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012. doi:10.4230/LIPIcs.STACS.2012.254.

11 Samir Datta, Prajakta Nimbhorkar, Thomas Thierauf, and Fabian Wagner. Graph isomorphism for k_{3, 3}-free and k_5-free graphs is in log-space. *Electron. Colloquium Comput. Complex.*, TR10, 2009. URL: https://api.semanticscholar.org/CorpusID:7978883.

12 Erik D Demaine, MohammadTaghi Hajiaghayi, Naomi Nishimura, Prabhakar Ragde, and Dimitrios M Thilikos. Approximation algorithms for classes of graphs excluding single-crossing graphs as minors. *Journal of Computer and System Sciences*, 69(2):166–195, 2004. doi:10.1016/j.jcss.2003.12.001.

13 David Eppstein and Vijay V. Vazirani. Nc algorithms for computing a perfect matching and a maximum flow in one-crossing-minor-free graphs. *SIAM Journal on Computing*, 50(3):1014–1033, 2021. doi:10.1137/19M1256221.

14 Steven Fortune, John Hopcroft, and James Wyllie. The directed subgraph homeomorphism problem. *Theoretical Computer Science*, 10(2):111–121, 1980. doi:10.1016/0304-3975(80)90009-2.

15 Anna Galluccio and Martin Loebl. Even/odd dipaths in planar digraphs. *Optimization Methods and Software*, 3(1-3):225–236, 1994. doi:10.1080/10556789408805566.

16 Martin Grohe, Ken-ichi Kawarabayashi, and Bruce Reed. A simple algorithm for the graph minor decomposition: logic meets structural graph theory. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '13, pages 414–431, USA, 2013. Society for Industrial and Applied Mathematics.

17 Torben Hagerup, Jyrki Katajainen, Naomi Nishimura, and Prabhakar Ragde. Characterizing multiterminal flow networks and computing flows in networks of small treewidth. *Journal of Computer and System Sciences*, 57(3):366–375, 1998. doi:10.1006/jcss.1998.1592.

18 Ken-ichi Kawarabayashi, Bruce Reed, and Paul Wollan. The graph minor algorithm with parity conditions. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 27–36, 2011. doi:10.1109/FOCS.2011.52.

19 Ken-ichi Kawarabayashi and Paul Wollan. A simpler algorithm and shorter proof for the graph minor decomposition. In *Proceedings of the Forty-Third Annual ACM Symposium on Theory of Computing*, STOC '11, pages 451–458, New York, NY, USA, 2011. Association for Computing Machinery. doi:10.1145/1993636.1993697.

20 Arindam Khan and Prasad Raghavendra. On mimicking networks representing minimum terminal cuts. *Information Processing Letters*, 114(7):365–371, 2014. doi:10.1016/j.ipl.2014.02.011.

21 Samir Khuller. Coloring algorithms for k5-minor free graphs. *Information Processing Letters*, 34(4):203–208, 1990. doi:10.1016/0020-0190(90)90161-P.

22 Samir Khuller. Extending planar graph algorithms to k3,3-free graphs. *Information and Computation*, 84(1):13–25, 1990. doi:10.1016/0890-5401(90)90031-C.

23 Robert Krauthgamer and Inbal Rika. *Mimicking Networks and Succinct Representations of Terminal Cuts*, pages 1789–1799. SIAM, 2013. doi:10.1137/1.9781611973105.128.

24 Andrea S. LaPaugh and Christos H. Papadimitriou. The even-path problem for graphs and digraphs. *Networks*, 14(4):507–513, 1984. doi:10.1002/net.3230140403.

25 Daniel Lokshtanov, Pranabendu Misra, Michał Pilipczuk, Saket Saurabh, and Meirav Zehavi. An exponential time parameterized algorithm for planar disjoint paths. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2020, pages 1307–1316, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3357713.3384250.

26 James F. Lynch. The equivalence of theorem proving and the interconnection problem. *SIGDA Newsl.*, 5(3):31–36, September 1975. doi:10.1145/1061425.1061430.

27 William McCuaig, Neil Robertson, P. D. Seymour, and Robin Thomas. Permanents, pfaffian orientations, and even directed circuits (extended abstract). In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, STOC '97, pages 402–405, New York, NY, USA, 1997. Association for Computing Machinery. doi:10.1145/258533.258625.

28 Zhivko Prodanov Nedev. Finding an even simple path in a directed planar graph. *SIAM J. Comput.*, 29(2):685–695, 1999. doi:10.1137/S0097539797330343.

29 Bruce Reed, Neil Robertson, Alexander Schrijver, and Paul Seymour. Finding dsjoint trees in planar graphs in linear time. In *Graph Structure Theory, Proceedings of a AMS-IMS-SIAM Joint Summer Research Conference on Graph Minors held June 22 to July 5, 1991, at the University of Washington, Seattle, USA*, volume 147 of *Contemporary Mathematics*, pages 295–302, January 1991. doi:10.1090/conm/147/01180.

30 N. Robertson and P.D. Seymour. Excluding a graph with one crossing. *Graph struc- ture theory (Seattle, WA, 1991)*, 1993. doi:10.1090/conm/147.

31 N. Robertson and P.D. Seymour. Graph minors .xiii. the disjoint paths problem. *Journal of Combinatorial Theory, Series B*, 63(1):65–110, 1995. doi:10.1006/jctb.1995.1006.

32 Neil Robertson and P.D Seymour. Graph minors. xvi. excluding a non-planar graph. *Journal of Combinatorial Theory, Series B*, 89(1):43–76, 2003. doi:10.1016/S0095-8956(03)00042-X.

33 Neil Robertson and P.D. Seymour. Graph minors. xx. wagner's conjecture. *Journal of Combinatorial Theory, Series B*, 92(2):325–357, 2004. Special Issue Dedicated to Professor W.T. Tutte. doi:10.1016/j.jctb.2004.08.001.

34 Alexander Schrijver. Finding k disjoint paths in a directed planar graph. *SIAM J. Comput.*, 23(4):780–788, 1994. doi:10.1137/S0097539792224061.

35 Simon Straub, Thomas Thierauf, and Fabian Wagner. Counting the number of perfect matchings in k5-free graphs. In *2014 IEEE 29th Conference on Computational Complexity (CCC)*, pages 66–77, 2014. `doi:10.1109/CCC.2014.15`.

36 Thomas Thierauf and Fabian Wagner. Reachability in k3,3-free graphs and k5-free graphs is in unambiguous log-space. In Mirosław Kutyłowski, Witold Charatonik, and Maciej Gębala, editors, *Fundamentals of Computation Theory*, pages 323–334, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

37 Vijay V. Vazirani. Nc algorithms for computing the number of perfect matchings in k3,3-free graphs and related problems. *Information and Computation*, 80(2):152–164, 1989. `doi:10.1016/0890-5401(89)90017-5`.

38 Klaus Von Wagner. Über eine eigenschaft der ebenen komplexe. *Mathematische Annalen*, 114:570–590, 1937. URL: `https://api.semanticscholar.org/CorpusID:123534907`.

39 Raphael Yuster and Uri Zwick. Finding even cycles even faster. *SIAM Journal on Discrete Mathematics*, 10(2):209–222, 1997.