Tight Bounds for Planar Strongly Connected Steiner Subgraph with
Fixed Number of Terminals (and Extensions)

Rajesh Chitnis∗ MohammadTaghi Hajiaghayi† Dániel Marx‡

Abstract
Given a vertex-weighted directed graph \( G = (V, E) \) and a
set \( T = \{ t_1, t_2, \ldots, t_k \} \) of \( k \) terminals, the objective of the
STRONGLY CONNECTED STEINER SUBGRAPH (SCSS)
problem is to find a vertex set \( H \subseteq V \) of minimum weight
such that \( G[H] \) contains a \( t_i \rightarrow t_j \) path for each \( i \neq j \). The
problem is NP-hard, but Feldman and Ruhl (FOCS ’99;
SICOMP ’06) gave a novel \( n^{O(k)} \) algorithm for the SCSS
problem, where \( n \) is the number of vertices in the graph
and \( k \) is the number of terminals. We explore how much
easier the problem becomes on planar directed graphs.

• Our main algorithmic result is a \( 2^{O(k \log k)} \cdot n^{O(\sqrt{k})} \)
algorithm for planar SCSS, which is an improvement of a
factor of \( O(\sqrt{k}) \) in the exponent over the algo-
rithm of Feldman and Ruhl.
• Our main hardness result is a matching lower bound
for our algorithm: we show that planar SCSS does
not have an \( f(k) \cdot n^{o(\sqrt{k})} \) algorithm for any com-
putable function \( f \), unless the Exponential Time Hy-
pothesis (ETH) fails.

The algorithm eventually relies on the excluded grid
theorem for planar graphs, but we stress that it is not
simply a straightforward application of treewidth-based
techniques: we need several layers of abstraction to
arrive to a problem formulation where the speedup due
to planarity can be exploited. To obtain the lower bound
matching the algorithm, we need a delicate construction
of gadgets arranged in a grid-like fashion to tightly control
the number of terminals in the created instance.

∗Department of Computer Science , University of Maryland at
College Park, USA. Supported in part by NSF CAREER award
1053605, NSF grant CCF-1161626, ONR YIP award N000141110662,
DARPA/AFOSR grant FA9550-12-1-0423 and Simons Award for
Graduate Students in Theoretical Computer Science. Email: 
rchitnis@cs.umd.edu

†Department of Computer Science , University of Maryland at
College Park, USA. Supported in part by NSF CAREER award
1053605, NSF grant CCF-1161626, ONR YIP award N000141110662,
and DARPA/AFOSR grant FA9550-12-1-0423. Email: 
hajiagha@cs.umd.edu

‡Computer and Automation Research Institute, Hungarian Academy
of Sciences (MTA SZTAKI), Budapest, Hungary. Supported by
ERC Starting Grant PARAMTIGHT (No. 280152) and OTKA grant
NK105645. Email: dmarx@cs.bme.hu

The following additional results put our upper and
lower bounds in context:

• Our \( 2^{O(k \log k)} \cdot n^{O(\sqrt{k})} \) algorithm for planar directed
graphs can be generalized to graphs excluding a fixed
minor.
• In general graphs, we cannot hope for such a dra-
matic improvement over the \( n^{O(k)} \) algorithm of Feld-
man and Ruhl: assuming ETH, SCSS in general
graphs does not have an \( f(k) \cdot n^{o(\sqrt{k})} \) algorithm
for any computable function \( f \).
• Feldman and Ruhl generalized their \( n^{O(k)} \) algorithm
to the more general DIRECTED STEINER FOREST
(DSF) problem; here the task is to find a subgraph of
minimum weight such that for every source \( s \), there
is a path to the corresponding terminal \( t \). We show
that that, assuming ETH, there is no \( f(k) \cdot n^{o(k)} \) time
algorithm for DSF on acyclic planar graphs.

1 Introduction

The STEINER TREE (ST) problem is one of the earliest
and most fundamental problems in combinatorial opti-

mization: given an undirected graph \( G = (V, E) \) and a set
\( T \subseteq V \) of terminals, the objective is to find a tree of mini-

mum size which connects all the terminals. The STEINER
TREE problem is believed to have been first formally de-
defined by Gauss in a letter in 1836. The first combinatorial
formulation of the ST problem is attributed independently
is known be to NP-complete, and was in fact was one of
Karp’s original list [27] of 21 NP-complete problems. In
the directed version of the ST problem, called DIRECTED
STEINER TREE (DST), we are also given a root vertex \( r \) and the objective is to find a minimum size arbores-
cence which connects the root \( r \) to each terminal from \( T \).
An easy reduction from SET COVER shows that the DST
problem is also NP-complete.

Steiner-type of problems arise in the design of net-
works. Since many networks are symmetric, the directed
versions of Steiner type of problems were mostly of the-
oretical interest. However in recent years, it has been ob-
served [37, 38] that the connection cost in various net-
works such as satellite or radio networks are not sym-
metric. Therefore, directed graphs form the most suitable
model for such networks. In addition, Ramanathan [37] also used the DST problem to find low-cost multicast trees, which have applications in point-to-multipoint communication in high bandwidth networks. We refer the interested reader to Winter [39] for a survey on applications of Steiner problems in networks. In this paper we consider two generalizations of the DST problem, namely the Strongly Connected Steiner Subgraph and the Directed Steiner Forest problems.

In the Strongly Connected Steiner Subgraph (SCSS) problem, given a directed graph \(G = (V, E)\) and a set \(T = \{t_1, t_2, \ldots, t_k\}\) of \(k\) terminals the objective is to find a set \(S \subseteq V\) such that \(G[S]\) contains a \(t_i \rightarrow t_j\) path for each \(1 \leq i \neq j \leq k\). In the Directed Steiner Forest (DSF) problem, given a directed graph \(G = (V, E)\) and a set \(T = \{(s_1, t_1), (s_2, t_2), \ldots, (s_k, t_k)\}\) of \(k\) pairs of terminals, the objective is to find a set \(S \subseteq V\) such that \(G[S]\) contains a \(s_i \rightarrow t_i\) path for each \(1 \leq i \leq k\). The following reduction shows that SCSS is a special case of DSF: an instance of SCSS with \(k\) terminals can be viewed as an instance of DSF with \(k(k-1)\) pairs by listing all ordered two-tuples of the terminals. We first describe the known results for both SCSS and DSF before stating our results and techniques.

**Previous Work.** Since both DSF and SCSS are NP-complete, one can try to design polynomial-time approximation algorithms for these problems. An \(\alpha\)-approximation for DST implies a \(2\alpha\)-approximation for SCSS as follows: fix a terminal \(t \in T\) and take the union of the solutions of the DST instances \((G, t, T \setminus t)\) and \((G_{\text{rev}}, t, T \setminus t)\), where \(G_{\text{rev}}\) is the graph obtained from \(G\) by reversing the orientations of all edges. The best known approximation ratio in polynomial time for SCSS is \(k^\epsilon\) for any \(\epsilon > 0\) [10]. A result of Halperin and Krauthgamer [24] implies SCSS has no \(\Omega(\log^2 k n)\)-approximation for any \(\epsilon > 0\), unless NP has quasi-polynomial Las Vegas algorithms. For the more general DSF problem, the best known fixed parameter ratio is \(n^{2/3+\epsilon}\) for any \(\epsilon > 0\) [3].

Berman et al. [3] showed that DSF has no \(\Omega(2^{\log(1-\epsilon) n})\)-approximation for any \(0 < \epsilon < 1\), unless NP has a quasi-polynomial time algorithm.

Rather than finding approximate solutions in polynomial time, one can look for exact solutions in time that is still better than the running time obtained by brute force solutions. For both SCSS and DSF problems, brute force can be used to check in time \(n^{O(k)}\) if a solution of size at most \(p\) exists: one can go through all sets of size at most \(p\). Recall that a problem is fixed-parameter tractable (FPT) with a particular parameter \(p\) if it can be solved in time \(f(p)n^{O(1)}\), where \(f\) is an arbitrary function depending only on \(p\); see [15, 20, 35] for more background. One can also consider parameterization by the number \(k\) of terminals (terminal pairs); with this parameterization, it is not even clear if there is a polynomial-time algorithm for every fixed \(k\), much less if the problem is FPT. It is known that Steiner Tree on undirected graphs is FPT: the classical algorithm of Dreyfus and Wagner [16] solves the problem in time \(3^k \cdot n^{O(1)}\), where \(k\) is the number of terminals. The running time was recently improved to \(2^k \cdot n^{O(1)}\) by Björklund et al. [4]. The same algorithms work for Directed Steiner Tree as well.

For the SCSS and DSF problems, we cannot expect fixed-parameter tractability: Guo et al. [22] showed that SCSS is \(W[1]\)-hard parameterized by the number of terminals \(k\), and DSF is \(W[1]\)-hard parameterized by the number of terminal pairs \(k\). In fact, it is not even clear how to solve these problems in polynomial time for small fixed values of the number \(k\) of terminals/pairs. The case of \(k = 1\) in DSF is the well-known shortest path problem in directed graphs, which is known to be polynomial time solvable. For the case \(k = 2\) in DSF, an \(O(n^2)\) algorithm was given by Li et al. [30] which was later improved to \(O(mn + n^2 \log n)\) by Nata and Fang [34]. The question regarding the existence of a polynomial algorithm for DSF when \(k = 3\) was open. Feldman and Ruhl [19] solved this question by giving an \(n^{O(k)}\) algorithm for DSF, where \(k\) is the number of terminal pairs. They first designed an \(n^{O(k)}\) algorithm for SCSS, where \(k\) is the number of terminals, and used it as a subroutine in the algorithm for the more general DSF problem.

**Our Results and Techniques.** Given the amount of attention the planar version of Steiner-type problems received from the viewpoint of approximation (see, e.g., [1, 2, 8, 13, 17]) and the availability of techniques for parameterized algorithms on planar graphs (see, e.g., [6, 12, 21]), it is natural to explore SCSS and DSF restricted to planar graphs. In general, one can have the expectation that the problems restricted to planar graphs become easier, but sophisticated techniques might be needed to exploit planarity. Our main algorithmic result is the following:

**Theorem 1.1.** An instance \((G, T)\) of the Strongly Connected Steiner Subgraph problem with \(|G| = n\) and \(|T| = k\) can be solved in \(2^{O(k \log k)} \cdot n^{O(\sqrt{k})}\) time, when the underlying undirected graph of \(G\) is planar (or more generally, \(H\)-minor-free for any fixed graph \(H\)).

This algorithm presents a major improvement over Feldman-Ruhl algorithm for SCSS in general graphs which runs in \(n^{O(k)}\) time. Let us give a very high-level intuition of our algorithm. The algorithm of Feldman-Ruhl for SCSS is based on defining a game with 2\(k\) tokens and costs associated with the moves of the tokens such that the minimum cost of the game is equivalent to the minimum cost of a solution of the SCSS problem; then the minimum cost of the game can be computed by exploring a state space of size \(n^{O(k)}\). We slightly generalize this game by introducing supermoves, which are sequences of
certain types of moves. The generalized game still has a state space of $n^{O(k)}$, but it has the advantage that we can now give a bound of $O(k)$ on the number of supermoves required for the game (such a bound is not possible for the original version of the game). We define a description of Feldman-Ruhl game: it is essentially a running commentary of the moves that occur in Feldman-Ruhl game in the sense that we report each move as it happens. As we can bound the length of the description by $O(k)$, we can guess the description (types and order of moves etc.), with the exception of the actual location of the vertices appearing in the description. We then need to map each of the $O(k)$ vertices appearing in the description to an actual vertex of the planar graph; trying all possibilities by brute force would still need $n^{O(k)}$ time. This is the point where planarity comes into play. With each description $\Gamma$, we associate a graph $D_\Gamma$ where the edges are added according to the moves in the description. Since the number of supermoves was bounded by $O(k)$, we are able to conclude that there is a description of Feldman-Ruhl game whose associated graph is also planar and has $O(k)$ non-isolated vertices. It is well-known that a planar graph with $O(k)$ vertices has treewidth $O(\sqrt{k})$, hence the treewidth of the graph $D_\Gamma$ associated with the description is $O(\sqrt{k})$. Therefore, we can use an embedding theorem given in Marx and Klein [28] to find in time $n^{O(\sqrt{k})}$ a minimum cost mapping of the vertices in the description and obtain a minimum-cost subgraph corresponding to the given description of the Feldman-Ruhl game. Our algorithm uses planarity in a very robust way: the only result on planarity we need is the planar grid minor theorem; we argue that $D_\Gamma$ is planar by showing that it is a minor of the input graph. This allows transparent generalization to the case when the underlying undirected graph is $H$-minor-free. All we need to do is to use the grid minor theorem for $H$-minor-free graphs due to Demaine and Hajiaghayi [14], which implies for any fixed graph $H$, every $H$-minor-free graph $G$ has treewidth $O(\sqrt{|V(G)|})$.

Can we get a better speedup in planar graphs than the improvement from $O(k)$ to $O(\sqrt{k})$ in the exponent of $n$? Our main hardness result matches our algorithm: it shows that $O(\sqrt{k})$ is best possible.

**Theorem 1.2.** The Strongly Connected Steiner Subgraph problem restricted to the case when the underlying undirected graph is planar is W[1]-hard parameterized by the number of terminals, and cannot be solved in time $f(k)n^{o(\sqrt{k})}$ unless ETH fails, where $f$ is any computable function, $k$ is the number of terminals, and $n$ is the number of vertices in the instance.

This also answers the question of Guo et al. [22], who showed the W[1]-hardness of these problems on general graphs and left the fixed-parameter tractability status on planar graphs as an open question. Recall that ETH can be stated as the assumption that $n$-variable 3SAT cannot be solved in time $2^{o(n)}$ [25]. There are relatively few parameterized problems that are W[1]-hard on planar graphs [7, 9, 18, 33]. The reason for the scarcity of such hardness results is mainly because for most problems, the fixed-parameter tractability of finding a solution of size $k$ in a planar graph can be reduced to a bounded-treewidth problem by standard layering techniques. However, in our case the parameter $k$ is the number of terminals, hence such a simple reduction to the bounded-treewidth case does not seem to be possible. Our reduction is from the GRID TILING problem formulated by Marx [31, 33], which is a convenient starting point for parameterized reductions for planar problems. Two types of gadgets, namely the connector gadget and main gadget, need to be constructed and then they are arranged in a grid-like structure (see Figure 2). The main technical part of the reduction is the structural results the existence and construction of particular types of connector gadgets and main gadgets (Lemma 7.1 and Lemma 7.2). Interestingly, the construction of the connector gadget poses a greater challenge: here we exploit in a fairly delicate way the fact that the $t_i \rightarrow t_j$ and the $t_j \rightarrow t_i$ paths appearing in the solution subgraph might need to share edges to reduce the weight.

We present additional results that put our algorithm and lower bound for SCSS in a wider context. Given our speedup for SCSS in planar graphs, one may ask if it is possible to get any similar speedup in general graphs. Our next result shows that the $n^{O(k)}$ algorithm of Feldman-Ruhl is almost optimal in general graphs:

**Theorem 1.3.** The Strongly Connected Steiner Subgraph problem cannot be solved in time $f(k)n^{o(k/\log k)}$ where $f$ is an arbitrary function, $k$ is the number of terminals and $n$ is the number of vertices in the instance, unless ETH fails.

Our proof is similar to the W[1]-hardness proof of Guo et al. [22]. They showed the W[1]-hardness of SCSS on general graphs parameterized by the number $k$ of terminals by giving a reduction from $k$-CLIQUE. However, this reduction uses “edge selection gadgets” and since a $k$-clique has $\Theta(k^2)$ edges, the parameter is increased at least to $\Theta(k^2)$. Combining with the result of Chen et al. [11] regarding the non-existence of an $f(k) \cdot n^{o(k)}$ algorithm for $k$-CLIQUE under ETH, this gives a lower bound of $f(k) \cdot n^{o(\sqrt{k})}$ for SCSS on general graphs. To avoid the quadratic blowup in the parameter and thereby get a stronger lower bound, we use the Colored Subgraph Isomorphism (CSI) problem as the source problem of our reduction. For this problem, Marx [32] gave a $f(k) \cdot n^{O(k/\log k)}$ lower bound under ETH, where $k = |E(G)|$ is the number of edges of the subgraph $G$ to be found in graph $H$. The reduction of Guo et al. [22] from CLIQUE can be turned into
a reduction from CSI which uses only $|E(G)|$ edge selection gadgets, and hence the parameter is $\Theta(|E(G)|)$. Then the lower bound of $f(k) \cdot n^{O(k/\log k)}$ transfers from CSI to SCSS.

Even though Feldman and Ruhl were able to generalize their $n^{O(k)}$ time algorithm from SCSS to DSF, we show that, surprisingly, such a generalization is not possible for our $2^{O(\log k)} \cdot n^{O(\sqrt{k})}$ time algorithm for planar SCSS.

**Theorem 1.4.** The Directed Steiner Forest problem on planar directed acyclic graphs (DAGs) is W[1]-hard parameterized by the number $k$ of terminal pairs and there is no $f(k)n^{o(k)}$ algorithm for any function $f$, unless the ETH fails.

This implies that the Feldman-Ruhl algorithm for DSF is optimal, even on planar directed acyclic graphs. As in out lower bound for planar SCSS, the proof is by reduction from an instance of $k \times k$ Grid Tiling problem. However, unlike in the reduction to SCSS where we needed $O(k^2)$ terminals, the reduction to DSF needs only $O(k)$ pairs of terminals (see Figure 4). Since the parameter blowup is linear, the $f(k) \cdot n^{O(k)}$ lower bound for Grid Tiling from [31] transfers to DSF. All our hardness results are shown for edge versions with integer weights. A simple reduction shows that the unweighted vertex version is more general than the integer weighted edge version, and hence all our results also hold for the unweighted vertex versions.

Finally, instead of parameterizing by the number of terminals, we can consider parameterization by the number of edges/vertices. Let us briefly and informally discuss this parameterization. Note that the number of terminals is a lower bound on the number of edges/vertices of the solution (up to a factor of 2 in the case of DSF parameterized by the number of edges), thus fixed-parameter tractability could be easier to obtain by parameterizing with the number of edges/vertices. However, our lower bound for SCSS or general graphs (as well as the W[1]-hardness of Guo et al. [22]) actually proves hardness also with these parameterizations, making fixed-parameter tractability unlikely. On the other hand, it follows from standard techniques that both SCSS and DSF are FPT on planar graphs when parameterizing by the number $k$ of edges/vertices in the solution. The main argument here is that the solution is fully contained in the $k$-neighborhood of the terminals, whose number is $O(k)$. It is known that the $k$-neighborhood of $O(k)$ vertices in a planar graph has treewidth $O(k)$, thus one can use standard techniques on bounded-treewidth graphs (dynamic programming or Courcelle’s Theorem). Alternatively, at least in the unweighted case, one can formulate the problem as a first order formula of size depending only on $k$ and then invoke the result of Frick and Grohe [21] stating that such problems are FPT. Therefore, as fixed-parameter tractability is easy to establish on planar graphs, the challenge here is to obtain optimal dependence on $k$. One would expect a subexponential dependence on $k$ (e.g., $2^{O(\sqrt{k})}$ or $k^{O(\sqrt{k})}$) at least for SCSS, but this is not yet fully understood even for undirected Steiner Tree [36]. A slightly different parameterization is to consider the number $k$ of nonterminal vertices in the solution, which can be much smaller than the number of terminals. This leads to problems of somewhat different flavor, see, e.g., [26].

2 Feldman-Ruhl Algorithm for SCSS

In this section we give a self-contained description of Feldman-Ruhl algorithm for SCSS [19]. They consider the two connectivity problems of SCSS and DSF. We first define the two problems below:

**k-Strongly Connected Steiner Subgraph (k-SCSS)**

**Input:** A directed graph $G = (V,E)$ and a set of terminals $T = \{t_1, t_2, \ldots, t_k\}$.

**Question:** Find the smallest $H \subseteq V(G)$ such that $T \subseteq H$ and $G[T]$ is strongly connected.

Feldman and Ruhl [19] give an algorithm for $k$-SCSS which runs in $O(mn^{2k-3} + n^{2k-2}\log n)$ time, where $|V(G)| = n$ and $|E(G)| = m$. The SCSS problem is more well-known as a special case of the DSF problem:

**k-Directed Steiner Forest (k-DSF)**

**Input:** A directed graph $G = (V,E)$ and a set of terminal pairs $T = \{(s_1, t_1), (s_2, t_2), \ldots, (s_k, t_k)\}$.

**Question:** Find the smallest $H \subseteq V(G)$ such that $G[H]$ has a $s_i \rightarrow t_i$ path for each $i \in [k]$.

The following reduction shows that SCSS is a special case of DSF: an instance of SCSS with $k$ terminals can be viewed as an instance of DSF with $k(k-1)$ pairs by listing all ordered two-tuples of the terminals. Feldman and Ruhl [19] give an algorithm for $k$-SCSS which runs in $O(mn^{2k-3} + n^{2k-2}\log n)$ time. Their algorithm also works for vertex weighted and edge weighted versions of the problem. For the sake of simplicity we only describe the algorithm for the unweighted vertex version.

2.1 Legal Token Moves Let the set of terminals for SCSS be $T = \{t_1, t_2, \ldots, t_k\}$. For ease of notation, we set $q := k - 1$ and $r = t_k$. Any solution $H$ for SCSS contains paths from each of $t_1, t_2, \ldots, t_{k-1}$ to $r$. These paths together can be chosen to form an in-tree $T_{in}$ rooted at $r$. Similarly $H$ must also contain paths from $r$ to each of $t_1, t_2, \ldots, t_{k-1}$; these paths together can be chosen to form an out-tree $T_{out}$ rooted at $r$. Furthermore, any subgraph $H$ which is the union of such an in-tree and an out-tree rooted at $r$ is a solution for SCSS. However, a
crucial feature of the problem is that these two trees can share edges/vertices, thus taking the union of an optimum in-tree and an optimum out-tree is not necessarily an optimum solution.

The algorithm can be visualized as follows: we have two types of tokens, namely $F$-tokens and $B$-tokens. Place a “$F$-token” and a “$B$-token” at each $t_i$ for $i \in [q]$. The $F$-tokens move forward along edges of the in-tree $T_{in}$ towards $r$. The $B$-tokens move backward along edges of the out-tree $T_{out}$ towards $r$. The set of tokens left at any stage are called “alive” tokens. Since tokens of the same type trace out a tree, as soon as two tokens of the same type arrive at a common vertex we can merge them into one token. This can also be viewed as one token “eating” the other token, which then becomes dead. Therefore one token. This can also be viewed as one token “eating” type arrive at a common vertex we can merge them into one token. This can also be viewed as one token “eating” the other token, which then becomes dead. Therefore it is enough to describe the pair of sets $(F,B)$ which denote the set of nodes occupied by the $F$-tokens and $B$-tokens, respectively. Since there are at most $q$ tokens of each type, the sets $F,B$ have size at most $q$. Let $\left(\binom{V}{q}\right)$ denote the set of subsets of $V(G)$ of size at most $q$. We now define the set of “legal” token moves in Table 1, and show that a minimum solution corresponds to a solution for SCSS.

**Cost of flips.** There is a technical issue about the cost of a flipping move that is not explained in detail in [19]. Initially, [19] defines the cost $c$ of the move as the size of the set $M$ of vertices of a shortest walk from $f$ to $b$ in $G$ going through all vertices in $F' \cup B'$, excluding $f,b$ and vertices in $F' \cup B'$. The problem is that it is not clear how to find a walk minimizing this definition of cost. However, one can try all possible ordering of the tokens in $F' \cup B'$, and find a shortest walk that visits the tokens in this order. Then we can define the cost of the walk as its lengths plus one (i.e., the number of visited vertices, possibly with repetitions), minus the size of the set $\{f,b\} \cup F' \cup B'$. We will denote the cost as $c_1$-cost and $c_2$-cost if the cost of a flip is interpreted these two ways, respectively. Clearly, the $c_1$-cost is at most the $c_2$-cost. It turns out that these two costs are the same in optimum solutions (see Lemmas 2.1 and 2.2 below).

![Image](image_url)

**Figure 1:** Flipping move between $f$ and $b$: the black nodes form the set $M$ and tokens $F' \cup B'$ need to be “picked up”.

**Intuition about the legal moves.** A single move for an $F$-token corresponds to that $F$-token moving forward along an edge. Similarly, a single move for a $B$-token corresponds to that $B$-token moving backward along an edge. We charge only if the new vertex (where the token has now moved to) does not already have a token on it. The flipping move allows $F$-tokens and $B$-tokens to pass each other. The two outer tokens are an $F$-token $f$ and a $B$-token $b$ (see Figure 1). In between the outer tokens $f$ and $b$, there are other $F$-tokens moving forward along the edges and trying to pass $b$, and $B$-tokens moving backward along edges and trying to pass $f$. These tokens, which occupy the vertex sets $F'$ and $B'$ respectively, are picked up during the flipping move.

**Building the game graph $\tilde{G}$.** Let $\tilde{V} = \left(\binom{V}{q}\right) \times \left(\binom{V}{q}\right)$. Build a game graph $\tilde{G} = (\tilde{V}, \tilde{E})$, where $\tilde{E}$ is the set of all legal token moves. We assign weights to the edges of $\tilde{G}$ according to the costs of the corresponding legal moves. Consider a single move for an $F$-token given by $(F,B) \xrightarrow{c} (\left(F \setminus \{v\}\right) \cup \{v\}, B)$. Its cost can be computed easily: it is 1 if $v \notin F \cup B$, and 0 otherwise. Similarly, the cost of a single move for a $B$-token can be computed easily. On the other hand, to compute the cost of a flipping move between $f$ and $b$, we need to find the size of the shortest $f \sim b$ walk in $G$ that passes through all vertices in $F' \cup B'$. The main observation is the following: if we know the order in which the vertices from $F' \cup B'$ appear on the $f \sim b$ walk, then the shortest walk is just the concatenation of shortest paths between two consecutive nodes in the ordering. The number of tokens is at most $2q$ and hence $|F' \cup B'| \leq 2q - 2$. We try all the at most $(2q-2)!$ permutations of vertices from $F' \cup B'$, and select the one that gives the shortest walk. In this way, we can build the game graph $\tilde{G}$ and assign weights to its edges.

### 2.2 Algorithm for SCSS

Recall that initially each of the vertices $t_1,t_2,\ldots,t_q$ has an $F$-token and a $B$-token as well. Finally we want all the $F$-tokens and all the $B$-tokens to reach the vertex $r$ via legal moves. This suggests the following algorithm for SCSS.

**Algorithm 1** Feldman-Ruhl Algorithm for SCSS

1: Construct the game graph $\tilde{G} = (\tilde{V}, \tilde{E})$, where $\tilde{E}$ is the set of all legal token moves.
2: Find a minimum weight path $P$ in $\tilde{G}$ from \(\langle \{t_1,t_2,\ldots,t_q\}, \{t_1,t_2,\ldots,t_q\} \rangle\) to \(\langle r,r \rangle\).
3: Let $H$ be the union of $\{t_1,t_2,\ldots,t_q,r\}$ and all nodes given by $P$ (including those in sets $M$ for flipping moves).

4: return $H$

To show the correctness, the main idea is that when we move the tokens, we only pay a cost when a new vertex is encountered. The following two lemmas demonstrate the correctness of Algorithm 1:

**Lemma 2.1.** (Lemma 3.1 from [19]) If there is a move sequence from \(\langle \{t_1,t_2,\ldots,t_q\}, \{t_1,t_2,\ldots,t_q\} \rangle\) to \(\langle r,r \rangle\) of
Legal Token Moves for SCSS

1. Single Moves for \( F \)-tokens: For each edge \( (u,v) \in E \) and all token sets \( F,B \in (\mathcal{V},E) \) such that \( u \in F \), we have the move
\[
\langle F,B \rangle \rightarrow \langle (F \setminus u) \cup \{v\}, B \rangle.
\]
The cost \( c \) of this move is 1 if \( v \notin F \cup B \) and 0 otherwise.

2. Single Moves for \( B \)-tokens: For each edge \( (u,v) \in E \) and all token sets \( F,B \in (\mathcal{V},E) \) such that \( v \in B \) we have the move
\[
\langle F,B \rangle \rightarrow \langle (F \setminus v) \cup \{u\}, B \rangle.
\]
The cost \( c \) of this move is 1 if \( u \notin F \cup B \) and 0 otherwise.

3. Flipping: For every pair \( f,b \) and vertex sets \( F,B,F' \subset F,B' \subset B \) such that
   
   - \( f \in F \) and \( F' \in (\mathcal{V},E) \),
   - \( b \in B \) and \( B' \in (\mathcal{V},E) \), and
   - there is an \( f \leadsto b \) walk in \( G \) going through all vertices in \( F' \cup B' \),

   we have the move
   \[
   \langle F,B \rangle \rightarrow \langle (F \setminus \{f\}) \cup \{b\}, (B \setminus \{b\} \cup \{f\}) \rangle.
   \]
   The cost \( c \) of this move is discussed below.

\[
\begin{array}{|c|c|}
\hline
\text{Cost} & \text{Solution} \\
\hline
\leq c & \text{H can be easily constructed.} \\
\hline
\end{array}
\]

The crucial point in the proof of Lemma 2.2 is that when moving the tokens, we “pay” each time we encounter a new vertex. However, it can happen that we pay twice for a vertex if a token enters the vertex, then leaves it, then later some token visits the vertex again. Feldman and Ruhl are able to avoid this situation by enforcing the following rule:

Once a token moves off a vertex, no other token will ever move to that vertex again. (\*)

They say that a vertex becomes “dead” once a token moves from it, so that tokens are allowed to only move to vertices in \( H' \) that are “alive.” We need to clarify what we mean by “moving off” in a flipping move. We imagine that tokens \( f \) and \( b \) change places by following the walk, hence we consider all vertices of \( M \) becoming dead. However, Feldman and Ruhl do not state explicitly whether or not the original locations of \( f \) and \( b \) become dead in a flipping move. Observation of [19, Claim 3.4] shows that we may make \( f \) and \( b \) dead (the proof of Claim 3.4 works even in the case when some token \( f' \) requires \( b \) itself; in fact, the first step of the proof is to conclude that \( f' \) requires \( b \)). Therefore, we interpret Property (\*) in such a way that the locations of \( f \) and \( b \) also become dead in a flipping move. An important consequence is that a vertex \( v \) can participate in at most one flip, as it becomes dead after the first flip and then no other token can move to it with a flip.

For the analysis of the running time of Algorithm 1, the interested reader is referred to Section 6.1 of [19].
3 Another Look at Moves of the Feldman-Ruhl game

In this section, we introduce notation describing the moves of the Feldman-Ruhl game in more detail. This will allow us to prove our $2^\Omega(k^2) \cdot n^{O(\sqrt{k})}$ algorithm for SCSS on planar (and more generally $H$-minor-free) graphs. Recall that the legal moves for SCSS are defined in Section 2.1.

3.1 The Flipping Move For ease of notation, we call the Flipping move as $\text{Flip}(f, b, u, v, f', b')$, which is used to denote that the forward token $f$ located at $v$ flips with the backward token $b$ located at $v$, and the sets of vertices $F'$ and $B'$ that denote the locations of the forward and backward tokens, respectively, are picked up. Let $P$ be the shortest $u \rightarrow v$ walk in $G$ which goes through all the vertices where the tokens from $F' \cup B'$ are located. Then the cost of this move is the number of vertices on $P$ which do not have a token from the set $F' \cup B' \cup \{f, b\}$.

We have two cases: either the set $F' \cup B'$ is empty, or not.

- If $F' \cup B' = \emptyset$, then we call this move an $\text{EmptyFlip}(f, b, u, v)$ move.
- Otherwise $F' \cup B' \neq \emptyset$, and we call this move a $\text{NonEmptyFlip}(f, b, u, v, F', B')$ move. In particular, we use $\text{NonEmptyFlip}(f, b, u, v, g_1, g_2, \ldots, g_\ell, w_1, w_2, \ldots, w_\ell)$ to denote the NonEmptyFlip that picks up the tokens $g_i$ at vertex $w_i$ for each $1 \leq i \leq \ell$.

3.2 Single Moves for $F$ and $B$ tokens We define various types of possible moves of the type Single Move for an $F$-token. The discussion for $B$ tokens is similar, and we do not repeat it again. For ease of notation, we call the “Single Move for $F$-token” as $\text{Forward}(f, u, v)$ if the forward token $f$ located at $u$ moves forward to the vertex $v$ along the edge $(u, v)$ in this move. Similarly we call the “Single Move for $B$-token” as $\text{Backward}(b, u, v)$ if the backward token $b$ located at vertex $u$ moves to vertex $v$ backward along the edge $(v, u)$ in this move.

For the $\text{Forward}(f, u, v)$ move, the cost of this move is 1 if there is a token from $F \cup B$ present on $v$, and 0 otherwise. We have three cases:

- If there was no token at $v$, then the cost of this move is 1. We call this move a $\text{SingleForwardAlone}(f, u, v)$ move since at the end of this move the token originally located at $f$ does not encounter any token.
- If there was no forward token at $v$, but there was a backward token $b$, then again the cost of this move is 0. We call this move a $\text{SingleForwardMeet}(f, u, v, b)$ move since after this move the forward token $f$ meets the backward token $b$ at the vertex $v$. We follow the convention that every SingleForwardMeet is followed by an EmptyFlip (of length 0) at vertex $v$; as this does not move the tokens at all, it does not influence the solution. This convention simplifies some of the arguments later in the proof of Theorem 5.1.

- If there was a forward token $f'$ and a backward token $b$ at $v$, then the cost of this move is 0. We call this move a $\text{SingleForwardAbsorbAndMeet}(f, u, v, f', b)$ move since after this move the forward token $f$ absorbs the forward token $f'$. However, in this case we do not require an EmptyFlip of length 0 to occur.

Similarly, we can also define the SingleBackwardAlone move, the SingleBackwardAbsorb move, and the SingleBackwardMeet move.

4 A Bird’s-eye View of the Feldman-Ruhl game

In this section, we take a bird’s-eye view of the Feldman-Ruhl game for SCSS. More formally, we introduce new “supermoves” for their game. The Feldman-Ruhl game takes place in a sequence of moves: each “supermove” is nothing but a collection of contiguous moves from the Feldman-Ruhl game. The advantage is that we are able to show that there is a solution for the Feldman-Ruhl game that can be partitioned into $O(k)$ supermoves, where $k$ is the number of terminals in the SCSS instance. We now define the supermoves. Let $H$ be an optimum solution of the Feldman-Ruhl game satisfying (*), and let the moves in $H$ be $M_1, M_2, \ldots, M_p$.

4.1 Forward, Backward and Flip Supermoves First, we define the supermoves associated with forward tokens. Let $f$ be a forward token. Consider a contiguous sequence of moves $H_{i_1}, H_{i_2}, \ldots, H_{i_{j-1}}, H_{i_j}$ (with $i_1 < i_2 < \ldots < i_j$) such that

- $H_{i_s}$ is the move $\text{SingleForwardAlone}(f, v_s, v_{s+1})$ for each $1 \leq s \leq i_{j-1}$.
- $H_{i_s}$ is a single move involving $f$ which takes $f$ from $v_{i_j}$ to $v_{i_{j+1}}$
- The only moves between $H_{i_1}$ and $H_{i_j}$ involving $f$ are $H_{i_1}, H_{i_2}, \ldots, H_{i_{j-1}}, H_{i_j}$

Then we can delete the moves $H_{i_1}, H_{i_2}, \ldots, H_{i_{j-1}}$. Depending on the type of the move $H_{i_j}$, we replace it with a “supermove” $M$ as follows:

- If there was no token at $v_{i_{j+1}}$, then the cost of $M$
is \(i_j\). We call \(M\) as the \textbf{Forward Alone}(\(f, v_{i_1}, v_{i_{j+1}}\)) supermove, and set \(\text{involved}(M) = \{f\}\).

- If there was a backward token \(b\) (but no forward token) at the vertex \(v_{i_{j+1}}\), then the cost of \(M\) is \(i_j - 1\). We call this the \textbf{Forward Meet}(\(f, v_{i_1}, v_{i_{j+1}}, b\)) supermove, and set \(\text{involved}(M) = \{f, b\}\). For ease of notation, we say that an EmptyFlip (of length 0) occurs at this point, freeing the tokens \(f, b\) to move along their subsequent paths.

- If there was both a forward token \(f'\) and a backward token \(b\) at the vertex \(v_{i_{j+1}}\), then the cost of \(M\) is \(i_j - 1\) and \(f\) absorbs the token \(f'\). We call this the \textbf{Forward Absorb And Meet}(\(f, v_{i_1}, v_{i_{j+1}}, f', b\)) supermove, and set \(\text{involved}(M) = \{f, f', b\}\). However, in this case we do not require an EmptyFlip of length 0 to occur.

- If there was a forward token \(f'\) (but no backward token) at the vertex \(v_{i_{j+1}}\), then the cost of \(M\) is \(i_j - 1\) and \(f\) absorbs the token \(f'\). We call this the \textbf{Forward Absorb}(\(f, v_{i_1}, v_{i_{j+1}}, f'\)) supermove, and set \(\text{involved}(M) = \{f, f'\}\).

We also define \(\text{corner}(M) = \{v_{i_1}, v_{i_{j+1}}\}\) and \(\text{internal}(M) = \{v_{i_2}, v_{i_3}, \ldots, v_{i_j}\}\).

Similarly, we can also define the \textbf{Backward Alone} supermoves, the \textbf{Backward Absorb} supermoves, the \textbf{Backward Absorb And Meet} and the \textbf{Backward Meet} supermoves. By Alone supermoves, we refer to the union of Backward Alone supermoves and the Forward Alone supermoves. The Absorb supermoves, Absorb And Meet and Meet supermoves are also defined similarly.

The \textbf{NonEmpty Flip} moves defined in Section 3.1 are also included in the set of supermoves.

If \(M = \text{NonEmptyFlip}(f, b, u, v, g_1, g_2, \ldots, g_t, w_1, w_2, \ldots, w_t)\), then we define

- \(\text{involved}(M) = \{f, b, g_1, g_2, \ldots, g_t\}\),
- \(\text{corners}(M) = \{u, v\}\), and
- \(\text{internal}(M) = P \setminus \{u, v, w_1, w_2, \ldots, w_t\}\), where \(P\) is a shortest \(u \to v\) walk in \(G\) passing through each \(w_i\).

### 4.2 Multiple Flips

Our final supermove is called \textbf{MultipleFlip}. Let \(H\) be an optimum solution of the Feldman-Ruhl game satisfying (*), and let the moves in \(H\) be \(H_1, H_2, \ldots, H_t\).

**Definition 4.1.** Let \(f\) and \(b\) be forward and backward tokens, respectively. Consider a consecutive sequence of moves \(H_i, H_{i+1}, \ldots, H_{j-1}, H_j\) such that

- There exists \(i \leq s \leq j\) such that \(H_s\) is an \textbf{Empty Flip} involving \(f\) and \(b\) (potentially of length 0)
- For each \(i \leq r \leq j\), the move \(H_r\) is of one of the following types:
  - \textbf{Empty Flip} move involving \(f\) and \(b\).
  - \textbf{Single Forward Alone} move involving \(f\).
  - \textbf{Single Backward Alone} move involving \(b\).
  - \textbf{Single Forward Meet} or \textbf{Single Backward Meet} move involving both \(f\) and \(b\).\(^2\)

Let \(v_1, v_2\) be the initial locations of \(f, b\) before \(H_i\) occurs are \(v_1, v_2\) respectively. Similarly, let the final locations of \(f, b\) after \(H_i\) occurs are \(v_2, v_1\) respectively. Then we define \(M = \text{MultipleFlip}(f, b, v_1, v_2, w_1, w_2)\) as the supermove which is given by the sequence of consecutive moves \(H_i, H_{i+1}, \ldots, H_j\). We say that the \(H_i, H_{i+1}, \ldots, H_j\) are the components of \(M\).

Note that an EmptyFlip is a special case of a MultipleFlip with just one component which is an EmptyFlip, and also \(v_1 = w_2\) and \(v_2 = w_1\). For the supermove \(M = \text{MultipleFlip}(f, b, v_1, v_2, w_1, w_2)\), we define the following sets:

- \(\text{involved}(M) = \{f, b\}\)
- \(\text{corners}(M) = \{v_1, v_2, w_1, w_2\}\)
- \(\text{internal}(M) = \{v_1, v_2, w_1, w_2, v_1 \cup \{H \in M \mid \text{corner}(H) \cup \text{internal}(H) \setminus \{v_1, v_2, w_1, w_2\}\}, \text{where } H \in M \text{ means that } H \text{ is a component of the MultipleFlip } M\).

The following property of a MultipleFlip will be helpful for our algorithm:

**Definition 4.2.** Let \(M\) be given by \text{MultipleFlip}(f, b, v_1, v_2, w_1, w_2). Then \(\text{corners}(M)\) is given by \(\{v_1, v_2, w_1, w_2\}\). We say that \(M\) is a \textbf{clean MultipleFlip} if either

- \(|\text{corners}(M)| = 2\), or
- \(|\text{corners}(M)| \geq 3\) and \(\text{internal}(M)\) is connected (in the undirected sense), and adjacent to every vertex of \(\text{corner}(M)\).

Note that if \(M\) is an EmptyFlip, then \(|\text{corners}(M)| = 2\) and it is clean by definition.

### 4.3 List of all supermoves

The final set of supermoves that we consider are the following:

<table>
<thead>
<tr>
<th>Final Set of Supermoves</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Alone, Absorb, Absorb And Meet and Meet</td>
</tr>
<tr>
<td>- NonEmptyFlip</td>
</tr>
<tr>
<td>- MultipleFlip</td>
</tr>
</tbody>
</table>

\(^2\)Recall from Section 4.1 that every Single Forward Meet or Single Backward Meet move must be followed by an EmptyFlip of length 0.
5 Description Associated with a Partition of a Solution to the Feldman-Ruhl Game

Consider a solution $H$ for the Feldman-Ruhl game and a partition $P(H)$ of $H$ into supermoves. Then the description $\Gamma_{P(H)}$ associated with $P(H)$ is essentially a running commentary of the game as it happens, i.e., we list all the supermoves which form the partition $P(H)$.

First there are $k$ entries of the form Location$(f, b, v_i)$ for $1 \leq i \leq k$ which tell that the initial location of the tokens $f, b$ is vertex $v_i$ of $G$. Note that the vertices $v_1, v_2, \ldots, v_k$ are given in the input instance. Then there is a sequence of entries where each entry has one of the following types:

1. ForwardAlone$(f, w_1, w_2)$: The forward token $f$ went from vertex $w_1$ to $w_2$ in $G$ and then did not meet any other token at $w_2$.
2. BackwardAlone$(b, w_1, w_2)$: The backward token $b$ went from vertex $w_2$ to $w_1$ in $G$ and then did not meet any other token at $w_1$.
3. ForwardAbsorb$(f_1, w_1, w_2, f_2)$: The forward token $f_1$ went from vertex $w_1$ to $w_2$ in $G$ and then absorbed another forward token $f_2$.
4. BackwardAbsorb$(b_1, w_2, w_1, b_2)$: The backward token $b_1$ went from vertex $w_2$ to $w_1$ in $G$ and then absorbed another backward token $b_2$.
5. ForwardMeet$(f, w_1, w_2, b)$: The forward token $f$ went from $w_1$ to $w_2$ in $G$, and then performed an EmptyFlip (of length $0$) with a backward token $b$ at $w_2$.
6. BackwardMeet$(b, w_2, w_1, f)$: The backward token $b$ went from $w_2$ to $w_1$ in $G$, and then performed an EmptyFlip (of length $0$) with a forward token $f$ at $w_1$.
7. ForwardAbsorbAndMeet$(f_1, w_1, w_2, f_2, b)$: The forward token $f_1$ went from vertex $w_1$ to $w_2$ in $G$ and then absorbed another forward token $f_2$ at $w_2$, where a backward token $b$ was also present.
8. BackwardAbsorbAndMeet$(b_1, w_2, w_1, b_2, f)$: The backward token $b_1$ went from vertex $w_2$ to $w_1$ in $G$ and then absorbed another backward token $b_2$ at $w_1$, where a forward token $f$ was also present.
9. NonEmptyFlip$(f, b, v_1, v_2, e_1, e_2, \ldots, e_t, w_1, w_2, \ldots, w_j)$: The tokens $f$ and $b$ were initially located at vertices $v_1$ and $v_2$ respectively in $G$. They then made a NonEmpty flip picking up the tokens $e_i$ which was located at vertex $w_i$ in $G$ along the way, in that order.
10. MultipleFlip$(f, b, v_1, v_2, w_1, w_2)$: The tokens $f, b$ were located initially at vertices $v_1, w_1$ in $G$ respectively. They then participated in a MultipleFlip and finally were located at vertices $v_2, w_2$ respectively.

The next theorem is the main combinatorial result that we use in the algorithm. It justifies introducing the supermoves: it shows that there is a solution and a partition of this solution into $O(k)$ supermoves.

**Theorem 5.1.** \[^{[\ast]}\] There is an optimum solution $H^\ast$ of the Feldman-Ruhl game and a partition $P(H^\ast)$ of this solution into supermoves such that the total number of entries (i.e., the number of supermoves) in the description of $P(H^\ast)$, say $X_{\text{label}}^\ast$ are $O(k)$. Furthermore, every MultipleFlip supermove is clean.

As the proof of Theorem 5.1 requires a deep analysis of the game, we defer it to the full version of the paper due to lack of space. We observe here the following simple property of an optimum solution:

**Lemma 5.1.** Let $M, M'$ be any two supermoves of $P(H^\ast)$, then $\text{internal}(M) \cap \text{internal}(M') = \emptyset$.

**Proof.** By definition, each vertex in the set $\text{internal}(M)$ is visited by some token. Hence Property (*) implies that if $M$ and $M'$ are any two supermoves then $\text{internal}(M) \cap \text{internal}(M') \neq \emptyset$. ■

5.1 Unlabeled Descriptions In this section, we consider unlabeled descriptions, i.e., descriptions where we replace the vertices in the descriptions by variables (which will always be denoted by greek letters). Recall that we have $2k$ tokens. We now show that it is enough to consider $O(k)$ variables to represent the unlabeled descriptions.

**Corollary 5.1.** The number of vertices of $G$ (with multiplicities) listed over all entries of the description $X_{\text{label}}^\ast$ is $O(k)$.

**Proof.** By Theorem 5.1, we know that the description has $O(k)$ entries. We now refer to Section 5. For the Alone, Absorb, Meet and MultipleFlip type of entries in the description we use a $O(1)$ number of vertices of $G$ per entry. For the NonEmptyFlip case we might add some more vertices in the description (like the $w_1, w_2, \ldots, w_j$) but their total number is bounded by $2k$, as each such vertex is picked up in the NonEmptyFlip and hence can occur in only one such entry. Therefore, the total number of vertices of $G$ (with multiplicities) listed over all entries of the description $X_{\text{label}}^\ast$ is $O(k)$. ■

Our goal is to guess the description $X_{\text{label}}^\ast$. We will do it as follows: first guess an unlabeled description, and then guess a labeling of the vertices of $G$ to the variables of the unlabeled description. The next lemma bounds the number of distinct unlabeled descriptions having $O(k)$ entries.

\[^{[\ast]}\]The proofs of results with $[\ast]$ have been deferred to the full version due to lack of space.
LEMMA 5.2. The number of distinct unlabeled descriptions having \(O(k)\) entries is \(2^{O(k \log k)}\)

Proof. For an unlabeled description, we call each of the following as a bit of the description: the names of the supermoves, the listed variables or the listed tokens.

Referring to Section 5, Each supermove (except NonEmptyFlip) contains \(O(1)\) variable bits. In addition to two variables corresponding to the endpoints of the flip, each NonEmptyFlip also lists several internal variables, each of which corresponds to a token that gets picked up in the NonEmptyFlip. Hence the total number of internal variable bits listed by the NonEmptyFlip is at most 2\(k\). Since the number of NonEmptyFlips is upper bounded by the total number of supermoves, which is \(O(k)\), the number of non-internal variable bits listed by NonEmptyFlips is also upper bounded by \(2 \times O(k) = O(k)\). Hence the total number of variable bits is \(O(k) + 2k + O(k) = O(k)\). By Corollary 5.1, it is enough to consider only \(O(k)\) variables in our unlabeled descriptions. Hence, the total number of guesses for the variable bits is \(k^{O(k)}\).

We have \(O(1) = 10\) choices for the type of the supermove. Since we want to enumerate only unlabeled descriptions with \(O(k)\) entries, the number of choices for this is \(10^{O(k)}\). Each supermove (except NonEmptyFlip) lists at most 3 tokens. Any non-internal token listed in a NonEmptyFlip does not appear in any other supermove, and hence their number is upper bounded by the total number of tokens which is \(2k\). Also we consider only unlabeled descriptions with \(O(k)\) entries. Hence the total number of token bits is \(O(k)\). Since there are \(2k\) tokens, the number of choices for the token bits is \((2k)^{O(k)}\).

Therefore, the total number of distinct unlabeled descriptions with \(O(k)\) entries is \(10^{O(k)} \times (2k)^{O(k)} \times k^{O(k)} = 2^{O(k \log k)}\).

Let the set of all unlabeled descriptions be \(X\). It is easy to see that we can enumerate all elements of \(X\) in time \(|X| = 2^{O(k \log k)}\). We now show how to check if an unlabeled description \(X \in X\) is valid or not:

DEFINITION 5.1. Consider an unlabeled description \(X \in X\). We say that \(X\) is valid if the following holds:

- The first \(k\) entries of \(X\) are given by Location\((f_i, b_i, \alpha_i)\) for \(i \in [k]\) such that \(\alpha_i \neq \alpha_j\) for each \(i \neq j\).
- For every token \(f\), the variables assigned to \(f\) in its current supermove is the same variable that ended up being assigned to \(f\) at the end of the last supermove in \(X\) involving \(f\) (if it exists).
- Any token which is absorbed (in an Absorb or AbsorbAndMeet supermove) or picked up (in a NonEmptyFlip) supermove cannot be involved in any subsequent move in \(X\).
- At the end of all the supermoves in \(X\), all the alive tokens are assigned to the variable \(\alpha_i\).

Given an unlabeled description \(X \in X\), it is easy to see that we can check whether \(X\) is valid in \(O(k)\) time by simply keeping a list of alive tokens and their currently assigned variables. Hence, in \(2^{O(k \log k)}\) time, we can build the set \(X'\) of valid unlabeled descriptions.

5.2 Directed Graphs Associated with Descriptions

With each valid unlabeled description \(X \in X'\), we can associate a directed graph \(D_X = (V_X, E_X)\). The vertex set \(V_X\) contains all the variables listed in \(X\), plus at most one additional variable for each MultipleFlip. By Theorem 5.1 and Corollary 5.1, we have that \(|V_X| = O(k)\). The edge set \(E_X\) is defined in Table 2.

By the way we defined this directed graph, if the description corresponds to a solution in a graph \(G\), then the graph of the description is a minor of \(G\) (and in particular, it is planar if \(G\) is planar).

THEOREM 5.2. Let \(X'_{\text{label}}\) be as in Theorem 5.1 and let \(X^*\) be the corresponding unlabeled description. The underlying undirected graph of the directed graph \(D_{X^*}\) is a minor of the underlying undirected graph of \(G\).

Proof. We construct an undirected graph \(G'\) from the underlying undirected graph of \(G\) the following way. For every supermove, we do the following:

- In the first 8 cases above, there are two corner vertices. Either the two corner vertices are adjacent in \(G\), or the internal vertices of the supermove give a path between them. In the latter case, we contract this path to make the two corner vertices adjacent.
- In the case of a NonEmptyFlip, by Proposition 2.1, there is a simple \(v_1 \rightarrow w_1 \rightarrow \ldots \rightarrow w_j \rightarrow v_2\) path on the internal vertices of the supermove. Then we contract subpaths of this path to make \(v_1w_1\ldots w_jv_2\) a path (i.e., to make these vertices adjacent).
- In the case of a MultipleFlip with two corner vertices, there is a path on the internal vertices between the two corner vertices (note that the case \(v_1 = v_2\) and \(w_1 = w_2\) need not be considered, since then the two tokens do not move at all). As in the first case, we contract this path to make the two corners adjacent.
- In the case of a MultipleFlip with at least three corner vertices, Theorem 5.2 implies that this MultipleFlip is clean, that is, the internal vertices induce a connected graph that is adjacent to all corners. Then we contract the internal vertices to a single vertex.

By Lemma 5.1, no two supermoves of \(P'(H^*)\) share any internal vertex, thus these contractions are independent. It is easy to see now that the underlying undirected
The edge set $E_X$ for the digraph $D_X$ corresponding to a valid unlabeled description $X$

1. ForwardAlone($f, \alpha, \alpha'$): Add the edge $(\alpha, \alpha')$.
2. BackwardAlone($b, \alpha, \alpha'$): Add the edge $(\alpha', \alpha)$.
3. ForwardAbsorb($f_1, \alpha, \alpha', f_2$): Add the edge $(\alpha, \alpha')$.
4. BackwardAbsorb($b_1, \alpha, \alpha', b_2$): Add the edge $(\alpha', \alpha)$.
5. ForwardMeet($f, \alpha, \alpha', b$): Add the edge $(\alpha', \alpha)$.
6. BackwardMeet($b, \alpha, \alpha', f$): Add the edge $(\alpha', \alpha)$.
7. ForwardAbsorbAndMeet($f, \alpha, \alpha', f'$): Add the edge $(\alpha, \alpha')$.

- If $|L| = 2$ then we know that $\alpha = \alpha'$ and $\gamma = \gamma'$ cannot occur since in this case both tokens do not move at all. So the only two cases are:
  - If $\alpha = \gamma$ and $\alpha' = \gamma'$, then add the edge $(\alpha, \alpha')$ and color it red.
  - If $\alpha = \gamma'$ and $\alpha' = \gamma$, then add the edge $(\alpha, \alpha')$ and color it blue.
- If $|L| \geq 3$ then introduce a new vertex $\delta$ and add the edges $E(\delta, 1) = (\delta, \alpha), E(\delta, 2) = (\delta, \alpha'), E(\delta, 3) = (\delta, \gamma)$ and $E(\delta, 4) = (\delta, \gamma')$.

8. MultipleFlip($b, \alpha, \alpha', \gamma, \gamma'$): Let $L$ be the (multi)set $\{\alpha, \alpha', \gamma, \gamma'\}$.

9. NonEmptyFlip($b, \alpha, \alpha', e_1, e_2, \ldots, e_r, \gamma_2, \gamma_3, \ldots, \gamma_l$): Add the path $\alpha \to \gamma_1 \to \gamma_2 \to \ldots \to \gamma_l \to \alpha'$.

10. MultipleFlip($b, \alpha, \alpha', \gamma, \gamma'$):

- If $|L| = 2$ then we know that $\alpha = \alpha'$ and $\gamma = \gamma'$ cannot occur since in this case both tokens do not move at all. So the only two cases are:
  - If $\alpha = \gamma$ and $\alpha' = \gamma'$, then add the edge $(\alpha, \alpha')$ and color it red.
  - If $\alpha = \gamma'$ and $\alpha' = \gamma$, then add the edge $(\alpha, \alpha')$ and color it blue.
- If $|L| \geq 3$ then introduce a new vertex $\delta$ and add the edges $E(\delta, 1) = (\delta, \alpha), E(\delta, 2) = (\delta, \alpha'), E(\delta, 3) = (\delta, \gamma)$ and $E(\delta, 4) = (\delta, \gamma')$.

11. MultipleFlip($b, \alpha, \alpha', \gamma, \gamma'$):

- If $|L| = 2$ then we know that $\alpha = \alpha'$ and $\gamma = \gamma'$ cannot occur since in this case both tokens do not move at all. So the only two cases are:
  - If $\alpha = \gamma$ and $\alpha' = \gamma'$, then add the edge $(\alpha, \alpha')$ and color it red.
  - If $\alpha = \gamma'$ and $\alpha' = \gamma$, then add the edge $(\alpha, \alpha')$ and color it blue.
- If $|L| \geq 3$ then introduce a new vertex $\delta$ and add the edges $E(\delta, 1) = (\delta, \alpha), E(\delta, 2) = (\delta, \alpha'), E(\delta, 3) = (\delta, \gamma)$ and $E(\delta, 4) = (\delta, \gamma')$.

Theorem 6.1. Let $D$ be a directed graph, $U$ a set of elements, and functions $cv : V(D) \times U \to \mathbb{Z}^+ \cup \{\infty\}$, $ce : V(D) \times V(D) \times U \times U \to \mathbb{Z}^+ \cup \infty$. In time $|U|^{O(tw(D))}$ we can find a mapping $\phi : V(D) \to U$ that minimizes

$$B_\phi = \sum_{v \in V(D)} cv(v, \phi(v)) + \sum_{(u,v) \in E(D)} ce(u, v, \phi(u), \phi(v))$$

where $tw(D)$ denotes the treewidth of the underlying undirected graph of $D$.

Recall that each $X \in \mathcal{X}''$ has treewidth $O(\sqrt{k})$. Note that $|U| = n^O(1)$, and hence for any choice of functions $ce$ and $cv$ we will be able to compute the minimum mapping $\phi$ in time $n^{O(\sqrt{k})}$. Our goal is to now apply Theorem 6.1 for the graph $D_X$ for each $X \in \mathcal{X}''$, and define the functions $ce$ and $cv$ in a way such that the objective value of Theorem 6.1 exactly captures the cost of the labeled description $X_{label}$ obtained by replacing each variable $\alpha$ by the vertex $\phi(\alpha)$.

Guessing a Labeling for an Unlabeled Description using Dynamic Programming

For each valid unlabeled description $X \in \mathcal{X}''$, the digraph $D_X$ comes with $k$ special variables, say $\alpha_1, \alpha_2, \ldots, \alpha_k$, that must be mapped to the vertices $v_1, v_2, \ldots, v_k$ where the terminals are placed in $G$. We try to map the remaining vertices of $D_X$ to elements of $U = V \cup V^4$ so that the unlabelled description coincides with $X^*_{label}$. For this purpose, we use the following theorem due to Klein and Marx [28]:

**Theorem 6.1.** Let $D$ be a directed graph, $U$ a set of elements, and functions $cv : V(D) \times U \to \mathbb{Z}^+ \cup \{\infty\}$, $ce : V(D) \times V(D) \times U \times U \to \mathbb{Z}^+ \cup \infty$. In time $|U|^{O(tw(D))}$ we can find a mapping $\phi : V(D) \to U$ that minimizes

$$B_\phi = \sum_{v \in V(D)} cv(v, \phi(v)) + \sum_{(u,v) \in E(D)} ce(u, v, \phi(u), \phi(v))$$

where $tw(D)$ denotes the treewidth of the underlying undirected graph of $D$.

Recall that each $X \in \mathcal{X}''$ has treewidth $O(\sqrt{k})$. Note that $|U| = n^O(1)$, and hence for any choice of functions $ce$ and $cv$ we will be able to compute the minimum mapping $\phi$ in time $n^{O(\sqrt{k})}$. Our goal is to now apply Theorem 6.1 for the graph $D_X$ for each $X \in \mathcal{X}''$, and define the functions $ce$ and $cv$ in a way such that the objective value of Theorem 6.1 exactly captures the cost of the labeled description $X_{label}$ obtained by replacing each variable $\alpha$ by the vertex $\phi(\alpha)$.

**6.1 Defining the Functions $ce$ and $cv$** First we see how to compute the minimum cost of a MultipleFlip, since we need it in the $cv$ function. Let $v_1, v_2, \ldots, v_k$ be the vertices of $G$ which have the $k$ terminals of the SCSS instance.

**Lemma 6.1.** The minimum cost of MultipleFlip($f_i, b_j$) can be found in polynomial time.

**Proof.** Let the initial locations of $f_i, b_j$ be $u_i, u_j$ and the final locations be $v_i, v_j$ respectively. We first build a game graph $G$ where the vertex set is $V \times V$. Then we add the weights between the edges similar to Section 6.1 of the Feldman-Ruhl paper [19]. Since all the flips in between
are empty flips, their cost is just the shortest paths in \( G \).
Then we find a shortest path in the game graph \( \tilde{G} \) from \( (u_i, v_i) \) to \( (u_j, v_j) \).

We define the \( cv \) and \( ce \) functions in Table 3 and Table 4 respectively: If \( u \in V \), then we make sure that the cost is infinity if a “marked” vertex in \( D_X \) is not mapped to the correct vertex having a terminal in \( G \). For all other vertices in \( D \) they get a cost of one to be assigned to other vertices in \( G \). If \( u \in V^4 \), then the cost of mapping \( \alpha \in D_X \) to \( u = (u_1, u_2, u_3, u_4) \) is the cost of the MultipleFlip between \((u_1, u_3)\) and \((u_2, u_4)\).

Recall that we defined a special optimal solution \( H^* \) in Theorem 5.1.

**Lemma 6.2.** \( \star \) If the cost of the solution \( H^* \) for the Feldman-Ruhl game is \( C \), then there is an unlabeled description \( X^* \in X'' \) and a mapping \( \phi : V(D_X) \rightarrow U \) such that \( B_0 = C \).

**Lemma 6.3.** \( \star \) If there is a valid unlabeled description \( X \in X'' \) and a mapping \( \phi : V(D_X) \rightarrow U \) such that \( B_\phi = R \) for some \( R < \infty \), then there is a solution \( H \) for the Feldman-Ruhl game of cost exactly \( R \).

Lemma 6.3 and Lemma 6.2 give the correctness of the following algorithm:

1. Enumerate the set \( X'' \)
2. For each \( X \in X'' \), apply Theorem 6.1 to the graph \( D_X \) to get the minimum mapping say \( \phi_X \) which gives cost \( C_X \).
3. Output \( \min_{X \in X''} \{C_X\} \)

We now analyze the running time. As seen before in Section 5.2, we can compute the set \( X'' \) in \( 2^{O(k \log k)} \) time. For each \( X \in X'' \), we can create the graph \( D_X \) in \( O(k^2) \) time, since it has \( O(k) \) vertices. Finally, applying Theorem 6.1 to \( D_X \) takes \( n^{O(\sqrt{k})} \) time. Hence, the total running time of the algorithm is \( 2^{O(k \log k)} \cdot n^{O(\sqrt{k})} \).

This concludes the proof of Theorem 1.1.

### 7 Hardness for SCSS in Planar Graphs

The goal of this section is to prove Theorem 1.2. We reduce from the Grid TILING problem:

**Grid TILING**

**Input:** Integers \( k \), \( n \), and \( k^2 \) non-empty sets \( S_{i,j} \subseteq [n] \times [n] \) where \( 1 \leq i, j \leq k \)

**Question:** For each \( 1 \leq i, j \leq k \) does there exist a value \( \gamma_{i,j} \in S_{i,j} \) such that

- If \( \gamma_{i,j} = (x, y) \) and \( \gamma_{i,j+1} = (x', y') \) then \( x = x' \).
- If \( \gamma_{i,j} = (x, y) \) and \( \gamma_{i+1,j} = (x', y') \) then \( y = y' \).

The reductions of Chen et al. [11] and Marx [31] together imply that, assuming ETH, the problem of \( k \times k \) Grid TILING cannot be solved in time \( f(k) \cdot n^{o(k)} \) for any computable function \( f \).

To prove Theorem 1.2, we give a reduction which transforms the problem of \( k \times k \) Grid TILING into an instance of SCSS with \( O(k^2) \) terminals.

We design two types of gadgets: the connector gadget and the main gadget. The reduce from Grid TILING represents each cell of the grid with a copy of the main gadget, with a connector gadget between main gadgets that are adjacent either horizontally or vertically (see Figure 2).

The proof of Theorem 1.2 is divided into the following steps: In Sections 7.1 we first introduce the connector gadget and Lemma 7.1 proves the existence of a particular type of connector gadgets. In Sections 7.2 we introduce the main gadget and Lemma 7.2 proves the existence of a particular type of main gadgets. Using Lemmas 7.1 and 7.2 as a blackbox, we prove Theorem 1.2 in Section 7.4.

The proofs of Lemmas 7.1 and Lemma 7.2 are deferred to the full version of the paper due to lack of space.

#### 7.1 Existence of connector gadgets

A connector gadget \( CG_n \) is an embedded planar graph with \( O(n^2) \) vertices and weights on its edges. It has a total of \( 2n + 2 \) distinguished vertices divided into the following 3 types:

- The vertices \( p, q \) are called internal-distinguished vertices
- The vertices \( p_1, p_2, \ldots, p_n \) are called source-distinguished vertices
- The vertices \( q_1, q_2, \ldots, q_n \) are called sink-distinguished vertices

Let \( P = \{p_1, p_2, \ldots, p_n\} \) and \( Q = \{q_1, q_2, \ldots, q_n\} \). The vertices \( P \cup Q \) appear in the order \( p_1, \ldots, p_n, q_n, \ldots, q_1 \) on the boundary of the gadget. In the connector gadget \( CG_n \), every vertex in \( P \) is a source and has exactly one outgoing edge. Also every vertex in \( Q \) is a sink and has exactly one incoming edge.

**Definition 7.1.** We say an edge set \( E' \subseteq E(CG_n) \) satisfies the connectedness property if each of the following four conditions hold for the graph \( CG_n[E'] \):

1. \( p \) can be reached from some vertex in \( P \)
2. \( q \) can be reached from some vertex in \( P \)
3. \( p \) can reach some vertex in \( Q \)
4. \( q \) can reach some vertex in \( Q \)

**Definition 7.2.** We say an edge set \( E' \) satisfying the connectedness property represents an integer \( i \in [n] \) if in \( E' \) the only outgoing edge from \( P \) is the one incident to \( p_i \) and the only incoming edge into \( Q \) is the one incident to \( q_i \).
The Function \( cv : V(D_X) \times U \to \mathbb{Z}_+ \cup \{ \infty \} \)

- For \( u \in V \) we define
  \[
  cv(\alpha, u) = \begin{cases} 
  1 & \text{if } \alpha = \alpha_i \text{ and } u = v_i \text{ for some } i \in [k] \\
  \infty & \text{if } \alpha = \alpha_i \text{ and } u \neq v_i \text{ for some } i \in [k] \\
  1 & \text{otherwise}
  \end{cases}
  \]

- For \( u = (v_1, v_2, w_1, w_2) \in V^4 \), let \( L \) be the multiset \( \{v_1, v_2, w_1, w_2\} \).
  - If \( |L| \geq 3 \), then define \( cv(\alpha, u) = (\text{cost of MultipleFlip}(v_1, v_2, w_1, w_2)) - |X| \), where \( X \) is the multiset \( \{v_1, v_2, w_1, w_2\} \setminus \{v_1, w_1\} \).
  - Otherwise if \( |L| = 2 \), then define \( cv(*, u) = \infty \) where * denotes any variable in \( D_X \).

### Table 3:

#### Definition 7.3.
We say an edge set \( E' \subseteq E(MG) \) satisfies the connectedness property if each of the following four conditions hold for the graph \( MG[E'] \):

1. There is a directed path from some vertex in \( L \) to \( R \cup B \)
2. There is a directed path from some vertex in \( T \) to \( R \cup B \)
3. Some vertex in \( R \) can be reached from \( L \cup T \)
4. Some vertex in \( B \) can be reached from \( L \cup T \)

#### Definition 7.4.
An edge set \( E' \subseteq E(MG) \) satisfying the connectedness property represents a pair \((i, j) \in [n] \times [n]\) if each of the following four conditions holds:

- The only edge of \( E' \) leaving \( L \) is the one incident to \( \ell_i \)
- The only edge of \( E' \) entering \( R \) is the one incident to \( r_i \)
- The only edge of \( E' \) leaving \( T \) is the one incident to \( t_j \)
- The only edge of \( E' \) entering \( B \) is the one incident to \( b_j \)

### 7.2 Existence of main gadgets

A main gadget \( MG \) is an embedded planar graph with \( O(n^3) \) vertices and weights on its edges. It has 4\( n \) distinguished vertices given by the following four sets:

- The set \( L = \{\ell_1, \ell_2, \ldots, \ell_n\} \) of left-distinguished vertices.
- The set \( R = \{r_1, r_2, \ldots, r_n\} \) of right-distinguished vertices.
- The set \( T = \{t_1, t_2, \ldots, t_n\} \) of top-distinguished vertices.
- The set \( B = \{b_1, b_2, \ldots, b_n\} \) of bottom-distinguished vertices.

The distinguished vertices appear in the order \( t_1, \ldots, t_n, r_1, \ldots, r_n, b_n, \ldots, b_1, \ell_n, \ldots, \ell_1 \) on the boundary of the gadget. In the main gadget \( MG \), every vertex in \( L \cup T \) is a source and has exactly one outgoing edge. Also each vertex in \( R \cup B \) is a sink and has exactly one incoming edge.

---

\( ^4 \) We use the notation \( C_n^* \) to emphasize that \( C^* \) depends only on \( n \) and not on the set \( S \).
The Function $ce : V(D_X) \times V(D_X) \times U \times U \rightarrow \mathbb{Z}^+ \cup \infty$

If $(\alpha, \alpha') \notin E(D_X)$ define $ce(\alpha, \alpha', *, *) = \infty$, where $*$ denote any element of $U$. In the remaining cases below, we assume that $(\alpha, \alpha') \in E(D_X)$.

- For $v, w \in V$
  - If $(\alpha, \alpha')$ is a red edge in $D_X$, then define $ce(\alpha, \alpha', v, w) = \text{cost of the MultipleFlip } (v, w, v, w) \cdot cv(\alpha, v) \cdot cv(\alpha', w)$
  - If $(\alpha, \alpha')$ is a blue edge in $D_X$, then define $ce(\alpha, \alpha', v, w) = \text{cost of the MultipleFlip } (v, w, v, w) \cdot cv(\alpha, v) \cdot cv(\alpha', w)$
  - If $(\alpha, \alpha')$ is an edge in $D_X$ with no color, then define $ce(\alpha, \alpha', v, w) = (d_G(v, w) - 1) \cdot cv(\alpha, v) \cdot cv(\alpha', w)$, where $d_G(v, w)$ is the length of the shortest $v \rightarrow w$ path in $G$.
- For $y = (v_1, v_2, w_1, w_2) \in V^4$ and $x \in V$, let $L$ be the multiset \{v_1, v_2, w_1, w_2\}.
  - If $|L| = 2$, then define $ce(*, *, y, x) = \infty$, where $*$ denotes any variable from $D_X$.
  - If $|L| \geq 3$, then define
    $$ce(\alpha, \alpha', y, x) = \begin{cases} 
    0 & \text{if } x = v_1 \text{ and } (\alpha, \alpha') = E(\alpha, 1) \\
    0 & \text{if } x = v_2 \text{ and } (\alpha, \alpha') = E(\alpha, 2) \\
    0 & \text{if } x = w_1 \text{ and } (\alpha, \alpha') = E(\alpha, 3) \\
    0 & \text{if } x = w_2 \text{ and } (\alpha, \alpha') = E(\alpha, 4) \\
    \infty & \text{otherwise}
    \end{cases}$$
- All others are set to $\infty$

Table 4:

2. If there is an edge set $E' \subseteq E(MG_S)$ such that $E'$ has weight at most $M^*_n$ and satisfies the connectedness connectivity property, then $E'$ has weight exactly $M^*_n$ and represents some $(\alpha, \beta) \in S$.

7.3 Construction of the SCSS instance

In order to prove Theorem 1.2, we reduce from the Grid Tiling problem. The following assumption will be helpful in handling some of the border cases of the gadget construction. We may assume that $1 < x, y < n$ holds for every $(x, y) \in S_{i,j}$; indeed, we can increase $n$ by two and replace every $(x, y)$ by $(x + 1, y + 1)$ without changing the problem.

Given an instance of Grid Tiling, we construct an instance of SCSS the following way (see Figure 2):

- We introduce a total of $k^2$ main gadgets and $2k(k + 1)$ connector gadgets.
- For every non-empty set $S_{i,j}$ in the Grid Tiling instance, we construct a main gadget $MG_{i,j}$ using Lemma 7.2 for the subset $S_{i,j}$.
- Half of the connector gadgets have the same orientation, and we call them $HCG$ to denote horizontal connector gadgets. The other half of the connector gadgets are rotated anti-clockwise by 90 degrees with respect to the orientation of the horizontal connector gadgets, and we call them $VCG$ to denote vertical connector gadgets. The internal-distinguished vertices of the connector gadgets are shown in Figure 2.
- For each $1 \leq i, j \leq k$, the main gadget $MG_{i,j}$ is surrounded by the following four connector gadgets:
  1. The horizontal connector gadgets $HCG_{i,j}$ are on the top and $HCG_{i+1,j}$ are on the bottom. Identify (or glue together) each sink-distinguished vertex of $HCG_{i,j}$ with the top-distinguished vertex of $MCG_{i,j}$ of the same index. Similarly identify each source-distinguished vertex of $HCG_{i+1,j}$ with the bottom-distinguished vertex of $MCG_{i,j}$ of the same index.
  2. The vertical connector gadgets $VCG_{i,j}$ are on the left and $VCG_{i,j+1}$ are on the right. Identify (or glue together) each sink-distinguished vertex of $VCG_{i,j}$ with the left-distinguished vertex of $MCG_{i,j}$ of the same index. Similarly identify each source-distinguished vertex of $VCG_{i,j+1}$ with the right-distinguished vertex of $MCG_{i,j}$ of the same index.
- We introduce to special vertices $x^*, y^*$ and an edge $(x^*, y^*)$ of weight 0.
For each $1 \leq i \leq k$, consider the horizontal connector gadget $HCG_{1,i}$ and collapse all its source-distinguished vertices into $y^*$. For each $1 \leq j \leq k$, consider the vertical connector gadget $VCG_{j,1}$ and collapse all its source-distinguished vertices into $y^*$. For each $1 \leq i \leq k$, consider the horizontal connector gadget $HCG_{k+1,i}$ and collapse all its sink-distinguished vertices into $x^*$. For each $1 \leq j \leq k$, consider the vertical connector gadget $VCG_{j,k+1}$ and collapse all its sink-distinguished vertices into $x^*$. For each $i \in [k+1], j \in [k]$, denote the internal-distinguished vertices of $HCG_{i,j}$ by $\{p_{i,j}^h, q_{i,j}^h\}$. For each $i \in [k], j \in [k+1]$, denote the internal-

distinguished vertices of $VCG_{i,j}$ by $\{p_{i,j}^v, q_{i,j}^v\}$. The set of terminals $T^*$ for the SCSS instance on $G'$ is $\{x^*, y^*\} \cup \{p_{i,j}^h, q_{i,j}^h \mid 1 \leq i \leq k+1, 1 \leq j \leq k\} \cup \{p_{i,j}^v, q_{i,j}^v \mid 1 \leq i \leq k, 1 \leq j \leq k+1\}$. We note that the total number of terminals is $|T^*| = 4k(k+1) + 2 = O(k^2)$. The edge set of $G'$ is a disjoint union of edge sets of all main gadgets, vertical connector gadgets, horizontal gadgets, and the edge $(x^*, y^*)$. Define the following quantity $^6$:

\[
W_n^* = k^2 \cdot M_n^* + 2k(k+1) \cdot C_n^*
\]

$^6$ We use the notation $W_n^*$ to emphasize that $W^*$ depends only on $n$.
The next two lemmas together show that GRID TILING has a solution if and only if the SCSS instance \((G^*, T^*)\) has a solution of weight at most \(W_n^*\).

**Lemma 7.3.** \([\star]\) If the GRID TILING instance has a solution, then the SCSS instance \((G^*, T^*)\) has a solution of weight at most \(W_n^*\).

**Lemma 7.4.** \([\star]\) If the SCSS instance \((G^*, T^*)\) has a solution, then the GRID TILING instance has a solution.

### 7.4 Proof of Theorem 1.2
Recall that Marx [31] showed the \(W[1]\)-hardness of GRID TILING parameterized by \(k\).

**Proof.** We note that the number of terminals in the SCSS instance is \(2k(k + 1) + 2 = O(k^2)\). By Lemmas 7.1 and 7.2, the connector and main gadgets are constructed in polynomial time, hence their size can be bounded by a polynomial in \(n\). It follows that the constructed instance has polynomial size. It is easy to see the underlying undirected graph of \(G^*\) constructed in Figure 2 is planar, since the underlying graph of each connector gadget and each main gadget is planar. Lemma 7.3 and Lemma 7.4 together imply the \(W[1]\)-hardness of SCSS parameterized by the number of terminals, even when the underlying graph is planar.

Chen et al. [11] showed for any function \(f\) an \(f(k) n^{o(k)}\) algorithm for CLIQUE implies ETH fails. Marx [31] gave a reduction that transforms the problem of finding a \(k\)-clique into a \(k \times k\) GRID-TILING instance. Lemma 7.3 and Lemma 7.4 together give a reduction which transforms the problem of \(k \times k\) grid-tiling into an instance of SCSS with \(O(k^2)\) terminal pairs. Composing the two reductions, we obtain that, under ETH, there is no \(f(k) n^{o(k)}\) time algorithm for SCSS (even when the underlying undirected graph is planar) for any function \(f\). This shows that the \(2^{O(k^2)} n^{O(\sqrt{k})}\) algorithm for SCSS given in Theorem 1.1 is essentially optimal. 

### 8 Hardness for SCSS in general graphs
The main goal of this section is to prove Theorem 1.3. We note that the reduction of Guo et al. [22] gives a reduction from MULTICOLORED CLIQUE which builds an equivalent instance of STRONGLY CONNECTED STEINER SUBGRAPH with quadratic blowup in the number of terminals. Hence using the reduction of Guo et al. [22] only an \(n^{O(\sqrt{k})}\) algorithm for SCSS can be ruled out under ETH. We are able to improve upon this hardness by using the COLORED SUBGRAPH ISOMORPHISM problem introduced by Marx [32]. Our reduction is also slightly simpler than the one given by Guo et al.

### COLORED SUBGRAPH ISOMORPHISM

**Input:** Undirected graphs \(G = (V_G = \{g_1, g_2, \ldots, g_t\}, E_G)\) and \(H = (V_H, E_H)\), and a partition of \(V_H\) into disjoint subsets \(H_1, H_2, \ldots, H_r\)

**Question:** Is there an injection \(\phi : V_G \rightarrow V_H\) such that

1. For every \(i \in [r]\) we have \(\phi(g_i) \in H_i\).
2. For every edge \(\{g_i, g_j\} \in E_G\) we have \(\{\phi(g_i), \phi(g_j)\} \in E_H\).

Marx [32] showed the following hardness result:

**Theorem 8.1.** COLORED SUBGRAPH ISOMORPHISM cannot be solved in time \(f(r)n^{o((r^{1/2})}\) where \(f\) is an arbitrary function, \(r\) is the number of edges in \(G\) and \(n\) is the number of vertices in \(H\), unless ETH fails.

By giving a reduction from COLORED SUBGRAPH ISOMORPHISM to STRONGLY CONNECTED STEINER SUBGRAPH where \(k = O(|E_G|)\) we will get a \(n^{O(k^{1/2})}\) hardness for SCSS under the ETH, where \(k\) is the number of terminals. Consider an instance \((G, H)\) of COLORED SUBGRAPH ISOMORPHISM. We now build an instance of STRONGLY CONNECTED STEINER SUBGRAPH as follows:

- \(B = \{b_i \mid i \in [\ell]\}\)
- \(C = \{c_v \mid v \in V_H\}\)
- \(C' = \{c'_v \mid v \in V_H\}\)
- \(D = \{d_{uv} \cup d'_{uv} \mid \{u, v\} \in E_H\}\)
- \(A = \{a_{uv} \cup a'_{uv} \mid \{u, v\} \in E_H\}\)
- \(F = \{f_{ij} \mid 1 \leq i,j \leq \ell \} \cup \{g_{ij} \in E_G\}\)
- \(V^* = B \cup C \cup C' \cup D \cup A \cup F\)
- \(E_1 = \{(c_v, b_i) \mid v \in H_i, 1 \leq i \leq \ell\}\)
- \(E_2 = \{(b_i, c_v) \mid v \in H_i, 1 \leq i \leq \ell\}\)
- \(E_3 = \{(c'_v, c_v) \mid v \in V_H\}\)
- \(E_4 = \{(c_v, d_{uv}) \mid \{u, v\} \in E_H\}\)
- \(E_5 = \{(a_{uv}, c'_v) \mid \{u, v\} \in E_H\}\)
- \(E_6 = \{(d_{uv}, d'_{uv}) \mid \{u, v\} \in E_H\}\)
- \(E_7 = \{(f_{ij}, d_{uv}) \cup (a_{uv}, f_{ij}) \mid \{u, v\} \in E_H; v \in H_i; u \in H_j; 1 \leq i, j \leq \ell\}\)
- \(E^* = E_1 \cup E_2 \cup E_3 \cup E_4 \cup E_5 \cup E_6 \cup E_7\)

An illustration of the construction for a small graph is given in Figure 3. We imagine that the edges of \(E_4\) are colored red to help us argue the proof. Let the terminals be \(T = B \cup F\). In the instance of COLORED SUBGRAPH ISOMORPHISM we can assume the graph \(G\) is connected, otherwise we can solve the problem for each connected component. Therefore \(k = |T| = \ell + 2|E_G| = O(|E_G|)\).

**Theorem 8.2.** \([\star]\) The instance \((G, H)\) of COLORED SUBGRAPH ISOMORPHISM answers YES if and only if there is a solution for the STRONGLY CONNECTED STEINER SUBGRAPH instance \((V^*, E^*, T)\) of size \(3\ell + 10|E_G|\).
Similarly we call the canonical paths from $c$ bottom. They are named using magenta color in Figure 4. An edge going into $d$ downward path of black edges and finally ends with a blue edge going into $b$. Similarly an edge going into $d$ downward path of black edges and finally ends with a blue edge coming out of $c$. Similarly an edge going into $d$ downward path of black edges and finally ends with a blue edge going into $d_j$.

There are $n$ edge-disjoint $a_i \sim b_i$ canonical paths: let us call them $P_1, P_2, \ldots, P_n$ as viewed from top to bottom. They are named using magenta color in Figure 4. Similarly we call the canonical paths from $c$ to $d_j$ as $O_1, O_2, \ldots, O_{k^2}$ when viewed from left to right. For each $i \in [k]$ and $\ell \in [n]$ we assign a weight of $\Delta(n + 1 - \ell), \Delta \ell$ to the first, last blue edges of $P_i$ respectively. Similarly for each $j \in [k]$ and $\ell \in [n]$ we assign a weight of $\Delta(n + 1 - \ell), \Delta \ell$ to the first, last blue edges of $O_j$ respectively.

Thus the total weight of first and last blue edges on any canonical path is exactly $\Delta(n + 1)$. The idea is to choose $\Delta$ large enough such that in any optimum solution the paths between the terminals will be exactly the canonical paths. We will see $\Delta = 4n^2$ will suffice for our purposes. Any canonical path uses two blue edges (which sum up to $\Delta(n + 1)$) black edges not inside the gadgets and $(n - 1)$ black edges inside each gadget. Since the number of gadgets each canonical path visits is $k$ and the weight of each black edge is 2, we have the total weight of any canonical path is $\Delta(n + 1) + 2(k + 1) + 2k(n - 1)$. Intuitively the $k^2$ gadgets correspond to the $k^2$ sets in the GRID TILING instance. Let us denote the gadget which is the intersection of the $a_i \sim b_i$ path and $c_j \sim d_j$ path by $G_i^j$. If $(x, y) = s_{i,j} \in S_{i,j}$ then we color green the vertex in the gadget $G_{i,j}$ which is the unique intersection of the canonical paths $P_i$ and $Q_j$. Then we add a shortcut as shown in Figure 5. The idea is if both the $a_i \sim b_i$ path and $c_j \sim d_j$ path pass through the green vertex then the $a_i \sim b_i$ path can save a weight of 1 by using the green edge and a vertical edge to reach the green vertex, instead of paying a weight of 2 to use the horizontal edge reaching the green vertex. It is easy to see there is an easy solution (without using green edges) for the DSF instance of weight $\hat{\beta} = 2k \left( \Delta(n + 1) + 2(k + 1) + 2k(n - 1) \right)$: each terminal pair just uses a canonical path and these canonical paths are pairwise edge-disjoint.
Figure 5: Let $u, v$ be two consecutive vertices on the canonical path say $P_i$. Let $v$ be on the canonical path $Q^o_j$ and let $y$ be the vertex preceding it on this path. If $v$ is a green vertex then we subdivide the edge $y, v$ by introducing a new vertex $x$ and adding two edges $(y, x)$ and $(x, v)$ of weight 1. We also add an edge $(u, x)$ of weight 1. The idea is if both the edges $(y, v)$ and $(u, v)$ were being used initially then now we can save a weight of 1 by making the horizontal path choose $(u, x)$ and then we get $(x, v)$ for free, as it is already being used by the vertical canonical path.

We need a small technical modification: we add one dummy row and column to the Grid Tiling instance. Essentially we now have a dummy index 1. So neither the first row nor the first column of any $S_{i,j}$ has any elements in the Grid Tiling instance. That is, no green vertex can be in the first row or first column of any gadget. Combining this fact with the orientation of the edges we get the only gadgets which can intersect any $a_i \sim b_1$ path are $G_{i,1}, G_{i,2}, \ldots, G_{i,k}$. Similarly the only gadgets which can intersect any $c_j \sim d_j$ path are $G_{1,j}, G_{2,j}, \ldots, G_{k,j}$.

We now prove two theorems which together give a reduction from Grid Tiling to DSF.

**Theorem 9.1.** Grid Tiling has a solution implies OPT for DSF is at most $\beta - k^2$.

**Proof.** For each $1 \leq i, j \leq k$ let $s_{i,j} \in S_{i,j}$ be the vertex in the solution of the Grid Tiling instance. Therefore for every $i \in [k]$ we know that each of the $k$ vertices $s_{i,1}, s_{i,2}, \ldots, s_{i,k}$ have the same $x$-coordinate, say $\alpha_i$. Similarly for every $j \in [k]$ each of the $k$ vertices $s_{1,j}, s_{2,j}, \ldots, s_{k,j}$ has the same $x$-coordinate, say $\gamma_j$. We now use the canonical path $P^o_i$ for $(a_i, b_1)$ and the canonical path $Q^o_j$ for $(c_j, d_j)$. Each of the $c_j \sim d_j$ paths will pay the full weight of a canonical path, which is $\Delta(n + 1) + 2(k + 1) + 2k(n - 1)$. However each $a_i \rightarrow b_1$
path will encounter a green vertex in each of the $k$ gadgets along its way, and hence will save one in each gadget (as shown in Figure 5) for a total saving of $k$. Hence over all the terminals we save a weight of $k^2$, and this is a solution for DSF instance of weight $\beta - k^2$. ■

We now prove the other direction which is more involved. First we show some preliminary lemmas:

**Lemma 9.1.** [*] In any optimum solution for DSF there is a $c_j \leadsto d_j$ canonical path for some $j \in [k]$.

Note the shortcut described in Figure 5 again brings the $a_i \leadsto b_i$ path back to the same horizontal canonical path.

**Definition 9.2.** We call an $a_i \leadsto b_i$ path as an almost canonical path if it is basically a canonical path, but can additionally take the small detour given by the green edge in Figure 5. An almost canonical path must however end on the same horizontal level on which it began.

**Lemma 9.2.** [*] In any optimum solution for DSF there is an $a_i \leadsto b_i$ almost canonical path for every $i \in [k]$.

**Theorem 9.2.** OPT for DSF is at most $\beta - k^2$ implies the GRID TILING instance has a solution.

**Proof.** Consider any optimum solution say $\mathcal{X}$. By Lemma 9.1 and Lemma 9.2 we know that $\mathcal{X}$ has a $a_i \leadsto b_i$ almost canonical path and a $c_j \leadsto d_j$ canonical path for every $1 \leq i, j \leq k$. Moreover these set of $2k$ paths form a solution for DSF. Since any optimum solution is minimal $\mathcal{X}$ is the union of these $2k$ paths: one for each terminal pair. For the moment let us forget the modifications we did in Figure 5. So the $a_i \leadsto b_i$ path and $c_j \leadsto d_j$ path in $\mathcal{X}$ intersect in a unique point (in the gadget $G_{i,j}$). The weight of $\mathcal{X}$ is exactly $\beta$. However we know that there is a solution of weight at most $\beta - k^2$. It is easy to see any $a_i \leadsto b_i$ almost canonical path and a $c_j \leadsto d_j$ canonical path can have at most one edge in common: the edge which comes vertically downwards into the green vertex (see Figure 5). There are $k^2$ gadgets, and there is at most one edge per gadget which is double counted in $\mathcal{X}$. Hence for each gadget $G_{i,j}$ there is exactly one edge which is used by both the $a_i \leadsto b_i$ almost canonical path and the $c_j \leadsto d_j$ canonical path in $\mathcal{X}$. So the endpoint of each of these common edges must be green vertices, and at each such point we save a weight of one as described in Figure 5. Since each $a_i \leadsto b_i$ path is an almost canonical path and each $c_j \leadsto d_j$ path is a canonical path, the green vertices form a solution for the GRID TILING instance. ■

Recall that Marx [31] showed the W[1]-hardness of GRID TILING: in fact he gave a reduction which transforms the problem of finding a $k$-clique into a $k \times k$ GRID-TILING instance. We are now ready the prove Theorem 1.4 which essentially says the $n^{O(k)}$ algorithm of Feldman-Ruhl [19] for DSF is optimal.

**Proof.** Theorem 9.1 and Theorem 9.2 together imply the W[1]-hardness. It is not hard to see the graph we constructed in Figure 4 is a planar DAG.

Chen et al. [11] showed for any function $f$ an $f(k)n^{o(k)}$ algorithm for CLIQUE implies ETH fails. Theorem 1.4 gives a reduction which transforms the problem of $k \times k$ grid-tiling into an instance of DSF with $2k$ terminal pairs. Composing the reduction from [31] from CLIQUE to GRID TILING to DSF, we obtain under ETH there is no $f(k)n^{o(k)}$ algorithm for DSF (even on planar DAGs) for any function $f$. This shows the $n^{O(k)}$ algorithm for DSF due to Feldman and Ruhl [19] is optimal. ■

**References**


