A Subpolynomial Approximation Algorithm for Graph Crossing Number in Low-Degree Graphs*

Julia Chuzhoy†
Toyota Technological Institute at Chicago
Chicago, USA
cjulia@ttic.edu

Zihan Tan‡
University of Chicago
Chicago, USA
zihantan@uchicago.edu

ABSTRACT

We consider the classical Minimum Crossing Number problem: given an n-vertex graph G, compute a drawing of G in the plane, while minimizing the number of crossings between the images of its edges. This is a fundamental and extensively studied problem, whose approximability status is widely open. In all currently known approximation algorithms, the approximation factor depends polynomially on Δ – the maximum vertex degree in G. The best current approximation algorithm achieves an $O(n^{1/2-\epsilon} \cdot \mathrm{poly}(\Delta \cdot \log n))$-approximation, for a small fixed constant ϵ, while the best negative result is APX-hardness, leaving a large gap in our understanding of this basic problem. In this paper we design a randomized $O\left(2^{O((\log n)^{7/8} \log\log n)} \cdot \mathrm{poly}(\Delta)\right)$-approximation algorithm for Minimum Crossing Number. This is the first approximation algorithm for the problem that achieves a subpolynomial in n approximation factor (albeit only in graphs whose maximum vertex degree is subpolynomial in n).

In order to achieve this approximation factor, we design a new algorithm for a closely related problem called Crossing Number with Rotation System, in which, for every vertex $v \in V(G)$, the circular ordering, in which the images of the edges incident to v must enter the image of v in the drawing is fixed as part of input. Combining this result with the recent reduction of [Chuzhoy, Mahabadi, Tan '20] immediately yields the improved approximation algorithm for Minimum Crossing Number.

CCS CONCEPTS

• **Theory of computation → Design and analysis of algorithms**.

KEYWORDS

Crossing Number, Approximation Algorithm

ACM Reference Format:
Julia Chuzhoy and Zihan Tan. 2022. A Subpolynomial Approximation Algorithm for Graph Crossing Number in Low-Degree Graphs. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing (STOC '22), June 20–24, 2022, Rome, Italy.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3519935.3519984

*A full version available at: https://arxiv.org/abs/2202.06827

†Supported in part by NSF grant CCF-2006464
‡Supported in part by NSF grant CCF-2006464

STOC '22, June 20–24, 2022, Rome, Italy

ACM ISBN 978-1-4503-9264-8/22/06…$15.00
https://doi.org/10.1145/3519935.3519984

1 INTRODUCTION

We study the classical Minimum Crossing Number (MCN) problem: given an n-vertex graph G, compute a drawing of G in the plane while minimizing the number of its crossings. Here, a drawing φ of a graph G is a mapping, that maps every vertex $v \in V(G)$ to some point $\varphi(v)$ in the plane, and every edge $e = (u, v) \in E(G)$ to a continuous simple curve $\varphi(e)$, whose endpoints are $\varphi(u)$ and $\varphi(v)$. For a vertex $v \in V(G)$ and an edge $e \in E(G)$, we refer to $\varphi(v)$ and to $\varphi(e)$ as the *images* of v and of e, respectively. We require that, for every vertex v and edge e, $\varphi(v) \in \varphi(e)$ only if v is an endpoint of e. We also require that, if some point p belongs to the images of three or more edges, then it must be the image of a shared endpoint of these edges. A *crossing* in a drawing φ of G is a point that belongs to the images of two edges of G, and is not their common endpoint. The *crossing number* of a graph G, denoted by $\mathrm{OPT}_{\mathrm{cr}}(G)$, is the minimum number of crossings in any drawing of G in the plane.

The MCN problem was initially introduced by Turán [28] in 1944, and has been extensively studied since then (see, e.g., [5–7, 9, 10, 16, 17], and also [20, 21, 23, 26] for excellent surveys). The problem is of interest to several communities, including, for example, graph theory and algorithms, and graph drawing. As such, much effort was invested into studying it from different angles. But despite all this work, most aspects of the problem are still poorly understood.

In this paper we focus on the algorithmic aspect of MCN. Since the problem is NP-hard [13], and it remains NP-hard even in cubic graphs [4, 14], it is natural to consider approximation algorithms for it. Unfortunately, the approximation ratios of all currently known algorithms depend polynomially on Δ, the maximum vertex degree of the input graph. To the best of our knowledge, no non-trivial approximation algorithms are known for the general setting, where Δ may be arbitrarily large. One of the most famous results in this area, the Crossing Number Inequality, by Ajtai, Chvátal, Newborn and Szemerédi [1] and by Leighton [18], shows that, for every graph G with $|E(G)| \geq 4|V(G)|$, the crossing number of G is $\Omega(|E(G)|^3/|V(G)|^2)$. Since the problem is most interesting when the crossing number of the input graph is low, it is reasonable to focus on low-degree graphs, where the maximum vertex degree Δ is bounded by either a constant, or a slowly-growing (e.g. subpolynomial) function of n. While we do not make such an assumption explicitly, like in all previous work, the approximation factor that we achieve also depends polynomially on Δ.

Even in this setting, there is still a large gap in our understanding of the problem's approximability, and the progress in closing this gap has been slow. On the negative side, only APX-hardness is known [2, 4], that holds even in cubic graphs. On the positive side, the first non-trivial approximation algorithm for MCN was obtained by Leighton and Rao in their seminal paper [19]. Given as input an n-vertex graph G, the algorithm computes a drawing of G with at most $O((n+\mathsf{OPT}_{\mathsf{cr}}(G))\cdot\Delta^{O(1)}\log^4 n)$ crossings. This bound was later improved to $O((n+\mathsf{OPT}_{\mathsf{cr}}(G))\cdot\Delta^{O(1)}\log^3 n)$ by [12], and then to $O((n+\mathsf{OPT}_{\mathsf{cr}}(G))\cdot\Delta^{O(1)}\log^2 n)$ following the improved approximation algorithm of [3] for Sparsest Cut. Note that all these algorithms only achieve an $O(n\,\mathrm{poly}(\Delta\log n)))$-approximation factor. However, their performance improves significantly when the crossing number of the input graph is large. A sequence of papers [7, 10] provided an improved $\tilde{O}(n^{0.9}\cdot\Delta^{O(1)})$-approximation algorithm for MCN, followed by a more recent sequence of papers by Kawarabayashi and Sidiropoulos [16, 17], who obtained an $\tilde{O}\left(\sqrt{n}\cdot\Delta^{O(1)}\right)$-approximation algorithm. All of the above results follow the same high-level algorithmic framework, and it was shown by Chuzhoy, Madan and Mahabadi [8] (see [9] for an exposition) that this framework is unlikely to yield a better than $O(\sqrt{n})$-approximation. The most recent result, by Chuzhoy, Mahabadi and Tan [9], obtained an $\tilde{O}(n^{1/2-\epsilon}\cdot\mathrm{poly}(\Delta))$-approximation algorithm for some small fixed constant $\epsilon>0$. This result was achieved by proposing a new algorithmic framework for the problem, that departs from the previous approach. Specifically, [9] reduced the MCN problem to another problem, called Minimum Crossing Number with Rotation System (MCNwRS) that we discuss below, which appears somewhat easier than the MCN problem, and then provided an algorithm for approximately solving the MCNwRS problem.

Our main result is a randomized $O\left(2^{O((\log n)^{7/8}\log\log n)}\cdot\Delta^{O(1)}\right)$-approximation algorithm for MCN. In order to achieve this result, we design a new algorithm for the MCNwRS problem that achieves significantly stronger guarantees than those of [9]. This algorithm, combined with the reduction of [9], immediately implies the improved approximation for the MCN problem. We also design several new technical tools that we hope will eventually lead to further improvements. We now turn to discuss the MCNwRS problem.

In the MCNwRS problem, the input consists of a multigraph G, and, for every vertex $v\in V(G)$, a circular ordering $\mathcal{O}_v$ of the edges that are incident to v, that we call a *rotation* for vertex v. The set $\Sigma=\{\mathcal{O}_v\}_{v\in V(G)}$ of all such rotations is called a *rotation system* for graph G. We say that a drawing φ of G *obeys* the rotation system Σ, if, for every vertex $v\in V(G)$, the images of the edges in $\delta_G(v)$ enter the image of v in the order $\mathcal{O}_v$ (but the *orientation* of the ordering can be either clock-wise or counter-clock-wise). In the MCNwRS problem, given a graph G and a rotation system Σ for G, the goal is to compute a drawing φ of G that obeys the rotation system Σ and minimizes the number of edge crossings. For an instance $I=(G,\Sigma)$ of the MCNwRS problem, we denote by $\mathsf{OPT}_{\mathsf{cnwrs}}(I)$ the value of the optimal solution for I, that is, the smallest number of crossings in any drawing of G that obeys Σ. The results of [9] show the following reduction from MCN to MCNwRS: suppose there is an efficient (possibly randomized) algorithm for the MCNwRS problem, that, for every instance $I=(G,\Sigma)$, produces a solution whose expected cost is at most $\alpha(m)\cdot(\mathsf{OPT}_{\mathsf{cnwrs}}(I)+m)$, where $m=|E(G)|$. Then there is a randomized $O(\alpha(n)\cdot\mathrm{poly}(\Delta\cdot\log n))$-approximation algorithm for the MCN problem. Our main technical result is a randomized algorithm, that, given an instance $I=(G,\Sigma)$ of MCNwRS, with high probability produces a solution to instance I with at most $2^{O((\log m)^{7/8}\log\log m)}\cdot(\mathsf{OPT}_{\mathsf{cnwrs}}(I)+m)$ crossings, where $m=|E(G)|$. Combining this with the result of [9], we immediately obtain a randomized $O\left(2^{O((\log n)^{7/8}\log\log n)}\cdot\mathrm{poly}(\Delta)\right)$-approximation algorithm for the MCN problem.

The best previous algorithm for the MCNwRS problem, due to [9], is a randomized algorithm, that, given an instance $I=(G,\Sigma)$ of the problem, with high probability produces a solution with at most $\tilde{O}\left((\mathsf{OPT}_{\mathsf{cnwrs}}(I)+m)^{2-\epsilon}\right)$ crossings, where $\epsilon=1/20$. A variant of MCNwRS was previously studied by Pelsmajer et al. [22], where for each vertex v of the input graph G, both the rotation $\mathcal{O}_v$ of its incident edges, and the orientation of this rotation (say clock-wise) are fixed. They showed that this variant of the problem is also NP-hard, and provided an $O(n^4)$-approximation algorithm with running time $O(m^n\log m)$, where $n=|V(G)|$ and $m=|E(G)|$. They also obtained approximation algorithms with improved guarantees for some special families of graphs.

We introduce a number of new technical tools, that we discuss in more detail in Section 1.2. Some of these tools require long and technically involved proofs. We view these tools as laying a pathway towards obtaining better algorithms for the Minimum Crossing Number problem, and it is our hope that these tools will eventually be streamlined and that their proofs will be simplified, leading to a better understanding of the problem and cleaner and simpler algorithms. We believe that some of these tools are interesting in their own right.

1.1 Our Results

Throughout this paper, we allow graphs to have parallel edges (but not self-loops); graphs with no parallel edges are explicitly called simple graphs. For convenience, we will assume that the input to the MCN problem is a simple graph, while graphs serving as inputs to the MCNwRS problem may have parallel edges. The latter is necessary in order to use the reduction of [9] between the two problems. Note that the number of edges in a graph with parallel edges may be much higher than the number of vertices. Our main technical contribution is an algorithm for the MCNwRS problem, that is summarized in the following theorem.

Theorem 1.1. *There is an efficient randomized algorithm, that, given an instance $I=(G,\Sigma)$ of* MCNwRS *with $|E(G)|=m$, computes a drawing of G that obeys the rotation system Σ. The number of crossings in the drawing is w.h.p. bounded by $2^{O((\log m)^{7/8}\log\log m)}\cdot(\mathsf{OPT}_{\mathsf{cnwrs}}(I)+m)$.*

We rely on the following theorem from [9] in order to obtain an approximation algorithm for the MCN problem.

Theorem 1.2 (Theorem 1.3 in [9]). *There is an efficient algorithm, that, given an n-vertex graph G with maximum vertex degree Δ, computes an instance $I=(G',\Sigma)$ of the* MCNwRS *problem, with $|E(G')|\le O(\mathsf{OPT}_{\mathsf{cr}}(G)\cdot\mathrm{poly}(\Delta\cdot\log n))$, and $\mathsf{OPT}_{\mathsf{cnwrs}}(I)\le O(\mathsf{OPT}_{\mathsf{cr}}(G)\cdot\mathrm{poly}(\Delta\cdot\log n))$. Moreover, there is an efficient algorithm that, given any solution of value X to instance I of* MCNwRS,

computes a drawing of G with the number of crossings bounded by $O((X + \mathsf{OPT}_{\mathsf{cr}}(G)) \cdot \mathrm{poly}(\Delta \cdot \log n))$.

Combining Theorem 1.1 and Theorem 1.2, we immediately obtain the following corollary, whose proof is deferred to the full version.

Corollary 1.3. *There is an efficient randomized algorithm, that, given a simple n-vertex graph G with maximum vertex degree Δ, computes a drawing of G, such that, w.h.p., the number of crossings in the drawing is at most $O\left(2^{O((\log n)^{7/8}\log\log n)} \cdot \mathrm{poly}(\Delta) \cdot \mathsf{OPT}_{\mathsf{cr}}(G)\right)$.*

1.2 Our Techniques

In this subsection we provide an overview of the techniques used in the proof of our main technical result, Theorem 1.1. For the sake of clarity of exposition, some of the discussion here is somewhat imprecise. Our algorithm relies on the divide-and-conquer technique. Given an instance $I = (G, \Sigma)$ of the MCNwRS problem, we compute a collection $\mathcal{I}$ of new instances, whose corresponding graphs are significantly smaller than G, and then solve each of the resulting new instances separately. Collection $\mathcal{I}$ of instances is called a *decomposition of I*. We require that the decomposition has several useful properties that will allow us to use it in order to obtain the guarantees from Theorem 1.1, by solving the instances in $\mathcal{I}$ recursively. Before we define the notion of decomposition of an instance, we need the notion of a *contracted graph*, that we use throughout the paper. Suppose G is a graph, and let $\mathcal{R} = \{R_1, \ldots, R_q\}$ be a collection of disjoint subsets of vertices of G. The contracted graph of G with respect to $\mathcal{R}$, that we denote by $G_{|\mathcal{R}}$, is a graph that is obtained from G, by contracting, for all $1 \le i \le q$, the vertices of R_i into a supernode u_i. Note that every edge of the resulting graph $G_{|\mathcal{R}}$ corresponds to some edge of G, and we do not distinguish between them. The vertices in set $V(G_{|\mathcal{R}}) \setminus \{u_1, \ldots, u_q\}$ are called *regular vertices*. Each such vertex v also lies in G, and moreover, $\delta_{G_{|\mathcal{R}}}(v) = \delta_G(v)$. Abusing the notation, given a collection $\mathcal{C} = \{C_1, \ldots, C_r\}$ of disjoint subgraphs of G, we denote by $G_{|\mathcal{C}}$ the contracted graph of G with respect to the collection $\{V(C_1), \ldots, V(C_r)\}$ of subsets of vertices of G. Given a graph G and its drawing φ, we denote by $\mathsf{cr}(\varphi)$ the number of crossings in φ.

Decomposition of an Instance. Given an instance $I = (G, \Sigma)$ of the MCNwRS problem, we will informally refer to $|E(G)|$ as the *size of the instance*. Assume that we are given an instance $I = (G, \Sigma)$ of MCNwRS with $|E(G)| = m$, and some parameter η (we will generally use $\eta = 2^{O((\log m)^{3/4}\log\log m)}$). Assume further that we are given another collection $\mathcal{I}$ of instances of MCNwRS. We say that $\mathcal{I}$ is an *η-decomposition of I*, if $\sum_{I'=(G',\Sigma')\in\mathcal{I}} |E(G')| \le m \,\mathrm{poly}\log m$, and $\sum_{I'\in\mathcal{I}} \mathsf{OPT}_{\mathsf{cnwrs}}(I') \le (\mathsf{OPT}_{\mathsf{cnwrs}}(I) + |E(G)|)\cdot\eta$. Additionally, we require that there is an efficient algorithm $\mathsf{Alg}(I)$, that, given a feasible solution $\varphi(I')$ to every instance $I' \in \mathcal{I}$, computes a feasible solution φ for instance I, with at most $O\left(\sum_{I'\in\mathcal{I}} \mathsf{cr}(\varphi(I'))\right)$ crossings.

At a high level, our algorithm starts with the input instance $I^* = (G^*, \Sigma^*)$ of the MCNwRS problem. Throughout the algorithm, we denote $m^* = |E(G^*)|$, and we use a parameter $\mu = 2^{O((\log m^*)^{7/8}\log\log m^*)}$. Over the course of the algorithm, we consider various other instances I of MCNwRS, but parameters m^* and μ remain unchanged, and they are defined with respect to the original input instance I^*. The main subroutine of the algorithm, that we call AlgDecompose, receives as input an instance $I = (G, \Sigma)$ of MCNwRS, and computes an η-decomposition $\mathcal{I}$ of I, for $\eta = 2^{O((\log m)^{3/4}\log\log m)}$, where $m = |E(G)|$. The subroutine additionally ensures that every instance in the decomposition is sufficiently small compared to I, that is, for each instance $I' = (G', \Sigma') \in \mathcal{I}$, $|E(G')| \le |E(G)|/\mu$. We note that this subroutine is in fact randomized, and, instead of ensuring that $\sum_{I'\in\mathcal{I}} \mathsf{OPT}_{\mathsf{cnwrs}}(I') \le (\mathsf{OPT}_{\mathsf{cnwrs}}(I) + |E(G)|)\cdot\eta$, it only ensures this in expectation. We will ignore this minor technicality in this high-level exposition.

It is now easy to complete the proof of Theorem 1.1 using Algorithm AlgDecompose: we simply apply Algorithm AlgDecompose to the input instance I^*, obtaining a collection $\mathcal{I}$ of new instances. We recursively solve each instance in $\mathcal{I}$, and then combine the resulting solutions using Algorithm $\mathsf{Alg}(I^*)$, in order to obtain the final solution to instance I^*. Since the sizes of the instances decrease by the factor of at least μ with each application of the algorithm, the depth of the recursion is bounded by $O\left((\log m^*)^{1/8}\right)$. At each recursive level, the sum of the optimal solution costs and of the number of edges in all instances at that recursive level increases by at most factor $2^{O((\log m^*)^{3/4}\log\log m^*)}$, leading to the final bound of $2^{O((\log m^*)^{7/8}\log\log m^*)} \cdot (\mathsf{OPT}_{\mathsf{cnwrs}}(I^*) + m^*)$ on the solution cost.

From now on we focus on the description of AlgDecompose. We start by describing several technical tools that this algorithm builds on. Throughout, given a graph G, we refer to connected vertex-induced subgraphs of G as *clusters*. Given a collection $\mathcal{C}$ of disjoint clusters of G, we denote by $E^{\mathsf{out}}_G(\mathcal{C})$ the set of all edges $e \in E(G)$, such that the endpoints of e do not lie in the same cluster. We will also use the notion of *subinstances* that we define next.

Subinstances. Suppose we are given two instances $I = (G, \Sigma)$ and $I' = (G', \Sigma')$ of MCNwRS. We say that I' is a *subinstance* of instance I, if the following hold. First, graph G' must be a graph that is obtained from a subgraph of G by contracting some subsets of its vertices into supernodes. Formally[1], there must be a graph $G'' \subseteq G$, and a collection $\mathcal{R} = \{R_1, \ldots, R_q\}$ of disjoint subsets of vertices of G'', such that $G' = G''_{|\mathcal{R}}$. For every regular vertex v of G', the rotation $\mathcal{O}_v \in \Sigma'$ must be consistent with the rotation $\mathcal{O}_v \in \Sigma$ (recall that $\delta_{G'}(v) \subseteq \delta_G(v)$). For every supernode u_i of G', its rotation $\mathcal{O}_{u_i} \in \Sigma'$ can be chosen arbitrarily. Note that the notion of subinstances is transitive: if I' is a subinstance of I and I'' is a subinstance of I', then I'' is a subinstance of I.

The main tool that we use is *disengagement of clusters*. Intuitively, given an instance $I = (G, \Sigma)$ of MCNwRS, and a collection $\mathcal{C}$ of disjoint clusters of G, the goal is to compute an η-decomposition $\mathcal{I}$ of I, such that every instance $I' = (G', \Sigma') \in \mathcal{I}$ is a subinstance of I, and moreover, there is at most one cluster $C \in \mathcal{C}$ that is contained in G', and all edges of G' that do not lie in C must belong to $E^{\mathsf{out}}_G(\mathcal{C})$. Assume for now that we can design an efficient algorithm for computing such a decomposition. In this case, the high-level plan for implementing Algorithm AlgDecompose would be as follows. First, we compute a collection $\mathcal{C}$ of disjoint clusters of graph G, such that, for each cluster $C \in \mathcal{C}$, $|E(C)| \le |E(G)|/(2\mu)$, and

[1] We note that this definition closely resembles the notion of graph minors, but, in contrast to the definition of minors, we do not require that the induced subgraphs $\{G[R_i]\}_{1\le i\le q}$ are connected.

$|E^{\text{out}}_G(C)| \leq |E(G)|/(2\mu)$. Then we perform disengagement of clusters in $\mathcal{C}$, obtaining an η-decomposition of the input instance I. We are then guaranteed that each resulting instance in $\mathcal{I}$ is sufficiently small. We note that it is not immediately clear how to compute the desired collection $\mathcal{C}$ of disjoint clusters of G; we discuss this later. For now we focus on algorithms for computing disengagement of clusters. We do not currently have an algorithm to compute the disengagement of clusters in the most general setting described above. In this paper, we design a number of algorithms for computing disengagement of clusters, under some conditions. We start with the simplest algorithm that only works in some restricted settings, and then generalize it to more advanced algorithms that work in more and more general settings. In order to describe the disengagement algorithm for the most basic setting, we need the notion of congestion, and of internal and external routers, that we use throughout the paper, and describe next.

Congestion, Internal Routers, and External Routers. Given a graph G and a set $\mathcal{P}$ of paths in G, the *congestion* that the set $\mathcal{P}$ of paths causes on an edge $e \in E(G)$, that we denote by $\text{cong}_G(\mathcal{P}, e)$, is the number of paths in $\mathcal{P}$ containing e. The total congestion caused by the set $\mathcal{P}$ of paths in G is $\text{cong}_G(\mathcal{P}) = \max_{e \in E(G)} \left\{\text{cong}_G(\mathcal{P}, e)\right\}$.

Consider now a graph G and a cluster $C \subseteq G$. We denote by $\delta_G(C)$ the set of all edges $e \in E(G)$, such that exactly one endpoint of e lies in C. An *internal C-router* is a collection $\mathcal{Q}(C) = \{Q(e) \mid e \in \delta_G(C)\}$ of paths, such that, for each edge $e \in \delta_G(C)$, path $Q(e)$ has e as its first edge, and all its inner vertices lie in C. We additionally require that all paths in $\mathcal{Q}(C)$ terminate at a single vertex of C, that we call the *center vertex of the router*. Similarly, an *external C-router* is a collection $\mathcal{Q}'(C) = \{Q'(e) \mid e \in \delta_G(C)\}$ of paths, such that, for each edge $e \in \delta_G(C)$, path $Q'(e)$ has e as its first edge, and all its inner vertices lie in $V(G) \setminus V(C)$. We additionally require that all paths in $\mathcal{Q}'(C)$ terminate at a single vertex of $V(G) \setminus V(C)$, called the *center vertex of the router*. For a cluster $C \subseteq G$, we denote by $\Lambda_G(C)$ and $\Lambda'_G(C)$ the sets of internal and external C-routers, respectively.

Basic Cluster Disengagement. In the most basic setting for cluster disengagement, we are given an instance $I = (G, \Sigma)$ of the MCNwRS problem, and a collection $\mathcal{C}$ of disjoint clusters of G. Additionally, for each cluster $C \in \mathcal{C}$, we are given an internal C-router $Q(C)$, whose center vertex we denote by $u(C)$, and an external C-router $Q'(C)$, whose center vertex we denote by $u'(C)$. The output of the disengagement procedure is a collection $\mathcal{I}$ of subinstances of I, that consists of a single global instance $\hat{I} = (\hat{G}, \hat{\Sigma})$, and, for every cluster $C \in \mathcal{C}$, an instance $I_C = (G_C, \Sigma_C)$ associated with it. Graph $\hat{G}$ is the contracted graph of G with respect to $\mathcal{C}$; that is, it is obtained from G by contracting every cluster $C \in \mathcal{C}$ into a supernode v_C. For each cluster $C \in \mathcal{C}$, graph G_C is obtained from G by contracting the vertices of $V(G) \setminus V(C)$ into a supernode v^*_C. For every cluster $C \in \mathcal{C}$, the rotation $\mathcal{O}_{v_C} \in \hat{\Sigma}$ of the supernode v_C in instance $\hat{I}$ and the rotation $\mathcal{O}_{v^*_C} \in \Sigma_C$ of the supernode v^*_C in instance I_C need to be defined carefully, in order to ensure that the sum of the optimal solution costs of all resulting instances is low, and that we can combine the solutions to these instances to obtain a solution to I. Observe that the set of edges incident to vertex v_C in $\hat{G}$ and the set of edges incident to vertex v^*_C in G_C are both equal to $\delta_G(C)$. We define a single ordering $\mathcal{O}^C$ of the edge set $\delta_G(C)$, that will serve both as the rotation $\mathcal{O}_{v_C} \in \hat{\Sigma}$, and as the rotation $\mathcal{O}_{v^*_C} \in \Sigma_C$. The ordering $\mathcal{O}^C$ is defined via the internal C-router $\mathcal{Q}(C)$, as the order in which the images of the paths of $\mathcal{Q}(C)$ enter the image of vertex $u(C)$. On the one hand, letting $\mathcal{O}_{v_C} = \mathcal{O}_{v^*_C}$ for every cluster $C \in \mathcal{C}$ allows us to easily combine solutions $\varphi(I')$ to instances $I' \in \mathcal{I}$, in order to obtain a solution to instance I, whose cost is at most $O\left(\sum_{I' \in \mathcal{I}} \text{cr}(\varphi(I'))\right)$. On the other hand, defining $\mathcal{O}^C$ via the set $\mathcal{Q}(C)$ of paths, for each cluster $C \in \mathcal{C}$, allows us to bound $\sum_{I' \in \mathcal{I}} \text{OPT}_{\text{cnwrs}}(I')$.

We now briefly describe how this latter bound is established, since it will motivate the remainder of the algorithm and clarify the bottlenecks of this approach. We consider an optimal solution φ^* to instance I, and we use it in order to construct, for each instance $I' \in \mathcal{I}$, a solution $\psi(I')$, such that $\sum_{I' \in \mathcal{I}} \text{cr}(\psi(I'))$ is relatively small compared to $\text{cr}(\varphi^*) + |E(G)|$. In order to construct a solution $\psi(\hat{I})$ to the global instance $\hat{I}$, we start with solution φ^* to instance I. We erase from this solution all edges and vertices that lie in the clusters of $\mathcal{C}$. For each cluster $C \in \mathcal{C}$, we let the image of the supernode v_C coincide with the original image of the vertex $u(C)$ – the center of the internal C-router $Q(C)$. In order to draw the edges that are incident to the supernode v_C in $\hat{G}$ (that is, the edges of $\delta_G(C)$), we utilize the images of the paths of the internal C-router $\mathcal{Q}(C)$ in φ^*, that connect, for each edge $e \in \delta_G(C)$, the original image of edge e to the original image of vertex $u(C)$.

Consider now some cluster $C \in \mathcal{C}$. In order to construct a solution $\psi(I_C)$ to instance I_C, we start again with the solution φ^* to instance I. We erase from it all edges and vertices except for those lying in C. We let the image of the supernode v^*_C be the original image of vertex $u'(C)$ – the center of the external C-router $\mathcal{Q}'(C)$. In order to draw the edges that are incident to the supernode v^*_C in G_C (that is, the edges of $\delta_G(C)$), we utilize the images of the paths of the external C-router $\mathcal{Q}'(C)$, that connect, for each edge $e \in \delta_G(C)$, the original image of edge e to the original image of vertex $u'(C)$.

Observe that the only increase in $\sum_{I' \in \mathcal{I}} \text{cr}(\psi(I'))$, relatively to $\text{cr}(\varphi^*)$, is due to the crossings incurred by drawing the edges incident to the supernodes in $\{v_C\}_{C \in \mathcal{C}}$ in instance $\hat{I}$, and for each subinstance I_C, drawing the edges incident to supernode v^*_C. All such edges are drawn along the images of the paths in $\bigcup_{C \in \mathcal{C}}(\mathcal{Q}(C) \cup \mathcal{Q}'(C))$ in φ^*. However, an edge may belong to a number of such paths. With careful accounting we can bound this number of new crossings as follows. Assume that, for every $C \in \mathcal{C}$, $\text{cong}_G(\mathcal{Q}'(C)) \leq \beta$. Assume further that, for each $C \in \mathcal{C}$, and for each edge $e \in E(C)$, $\text{cong}_G(\mathcal{Q}(C), e)^2 \leq \beta$. Then $\sum_{I' \in \mathcal{I}} \text{cr}(\psi(I')) \leq O(\beta^2(\text{OPT}_{\text{cnwrs}}(I) + |E(G)|))$. Therefore, in order to ensure that the collection $\mathcal{I}$ of subinstances of I that we have obtained via the cluster disengagement procedure is an η-decomposition of I, we need to ensure that, for every cluster $C \in \mathcal{C}$, $\text{cong}_G(\mathcal{Q}'(C)) \leq \beta$, and, for every edge $e \in E(C)$, $(\text{cong}_G(\mathcal{Q}(C), e))^2 \leq \beta$, for $\beta = O(\eta^{1/2})$. This requirement seems impossible to achieve. For example, if maximum vertex degree in graph G is small (say a constant), then some edges incident to the center vertices $\{u(C), u'(C)\}_{C \in \mathcal{C}}$ must incur very high congestion. In order to overcome this obstacle, we slightly weaken our requirements. Instead of providing, for every cluster $C \in \mathcal{C}$, a single internal C-router $Q(C)$, and a single external C-router $Q'(C)$, it is sufficient for us to obtain, for each cluster $C \in \mathcal{C}$, a *distribution* $\mathcal{D}(C)$ over the collection $\Lambda_G(C)$ of internal C-routers, such that, for

every edge $e \in E(C)$, $\mathbf{E}_{\mathcal{Q}(C)\sim\mathcal{D}(C)}\left[(\mathrm{cong}_G(\mathcal{Q}(C),e))^2\right] \leq \beta$, and a distribution $\mathcal{D}'(C)$ over the collection $\Lambda'_G(C)$ of external C-routers, such that for every edge e, $\mathbf{E}_{\mathcal{Q}'(C)\sim\mathcal{D}'(C)}\left[\mathrm{cong}_G(\mathcal{Q}'(C),e)\right] \leq \beta$.

To recap, in order to use Basic Cluster Disengagement procedure described above to compute an η-decomposition of the input instance I of MCNwRS into sufficiently small instances, it is now enough to design a procedure that, given an instance $I = (G, \Sigma)$ of MCNwRS, computes a collection $\mathcal{C}$ of disjoint clusters of G, and, for every cluster $C \in \mathcal{C}$, a distribution $\mathcal{D}(C)$ over the collection $\Lambda_G(C)$ of internal C-routers, such that, for every edge $e \in E(C)$, $\mathbf{E}_{\mathcal{Q}(C)\sim\mathcal{D}(C)}\left[(\mathrm{cong}_G(\mathcal{Q}(C),e))^2\right] \leq \beta$, together with a distribution $\mathcal{D}'(C)$ over the collection $\Lambda'_G(C)$ of external C-routers, such that, for every edge e, $\mathbf{E}_{\mathcal{Q}'(C)\sim\mathcal{D}'(C)}\left[\mathrm{cong}_G(\mathcal{Q}'(C),e)\right] \leq \beta$, for $\beta = O(\sqrt{\eta})$. Additionally, we need to ensure that, for every cluster $C \in \mathcal{C}$, $|E(C)| \leq |E(G)|/(2\mu)$, and that $|E^{\mathrm{out}}_G(C)| \leq |E(G)|/(2\mu)$. While computing a collection $\mathcal{C}$ of clusters with the latter two properties seems possible (at least when the maximum vertex degree in G is small), computing the distributions over the internal and the external routers for each cluster C with the required properties seems quite challenging. As a first step towards this goal, we employ the standard notions of well-linkedness and bandwidth property of clusters as a proxy to constructing internal C-routers with the required properties. Before we turn to discuss these notions, we note that the Basic Cluster Disengagement procedure that we have just described can be easily generalized to the more general setting, where the set $\mathcal{C}$ of clusters is laminar (instead of only containing disjoint clusters). This generalization will be useful for us later.

Assume that we are given a laminar family $\mathcal{C}$ of clusters (that is, for every pair $C, C' \in \mathcal{C}$ of clusters, either $C \subseteq C'$, or $C' \subseteq C$, or $C \cap C' = \emptyset$ holds), with $G \in \mathcal{C}$. Assume further that we are given, for each cluster $C \in \mathcal{C}$, a distribution $\mathcal{D}(C)$ over the collection $\Lambda_G(C)$ of internal C-routers, in which, for every edge $e \in E(C)$, $\mathbf{E}_{\mathcal{Q}(C)\sim\mathcal{D}(C)}\left[(\mathrm{cong}_G(\mathcal{Q}(C),e))^2\right] \leq \beta$, together with a distribution $\mathcal{D}'(C)$ over the collection $\Lambda'_G(C)$ of external C-routers, where for every edge e, $\mathbf{E}_{\mathcal{Q}'(C)\sim\mathcal{D}'(C)}\left[\mathrm{cong}_G(\mathcal{Q}'(C),e)\right] \leq \beta$, for some parameter β. The Basic Cluster Disengagement procedure, when applied to $\mathcal{C}$, produces a collection $\mathcal{I} = \{I_C = (G_C, \Sigma_C) \mid C \in \mathcal{C}\}$ of instances. For every cluster $C \in \mathcal{C}$, graph G_C associated with instance I_C is obtained from graph G, by first contracting all vertices of $V(G) \setminus V(C)$ into a supernode v^*_C, and then contracting, for each child-cluster $C' \in \mathcal{C}$ of C, the vertices of $V(C')$ into a supernode $v_{C'}$. We define, for every cluster C, an ordering of the set $\delta_G(C)$ of edges via an internal C-router that is selected from the distribution $\mathcal{D}(C)$, and we let the rotation $\mathcal{O}_{v^*_C}$ in the rotation system Σ_C, and the rotation $\mathcal{O}_{v_C}$ in the rotation system $\Sigma_{C'}$, where C' is the parent-cluster of C, to be identical to this ordering. Using the same reasoning as in the case where $\mathcal{C}$ is a set of disjoint clusters, we show that $\mathbf{E}\left[\sum_{I'\in\mathcal{I}} \mathsf{OPT}_{\mathsf{cnwrs}}(I')\right] \leq O\left(\beta^2 \cdot \mathsf{dep}(\mathcal{C}) \cdot (\mathsf{OPT}_{\mathsf{cnwrs}}(I) + |E(G)|)\right)$, where $\mathsf{dep}(\mathcal{C})$ is the depth of the laminar family $\mathcal{C}$ of clusters. We then show that $\mathcal{I}$ is an η'-decomposition of instance I, where $\eta' = O(\beta^2 \cdot \mathsf{dep}(\mathcal{C}))$.

As noted already, one of the difficulties in exploiting the Basic Cluster Disengagement procedure in order to compute an η-decomposition of the input instance $\mathcal{I}$ is the need to compute distributions over the sets of internal and the external C-routers for every cluster $C \in \mathcal{C}$, with the required properties. We turn instead to the notions of well-linkedness and bandwidth properties of clusters. These notions are extensively studied, and there are many known algorithms for computing a collection $\mathcal{C}$ of clusters that have bandwidth property in a graph. We will use this property as a proxy, that will eventually allow us to construct a distribution over the sets of internal C-routers for each cluster $C \in \mathcal{C}$, with the required properties.

Well-Linkedness, Bandwidth Property, and Cluster Classification. We use the standard notion of well-linkedness. Let G be a graph, let T be a subset of the vertices of G, and let $0 < \alpha < 1$ be a parameter. We say that the set T of vertices is *α-well-linked* in G if for every partition (A, B) of vertices of G into two subsets, $|E_G(A, B)| \geq \alpha \cdot \min\{|A \cap T|, |B \cap T|\}$.

We also use a closely related notion of *bandwidth property* of clusters. Suppose we are given a graph G and a cluster $C \subseteq G$. Intuitively, cluster C has the α-bandwidth property (for a parameter $0 < \alpha < 1$), if the edges of $\delta_G(C)$ are α-well-linked in C. More formally, we consider the augmentation C^+ of cluster C, that is defined as follows. We start with the graph G, and subdivide every edge $e \in \delta_G(C)$ with a vertex t_e, denoting $T = \{t_e \mid e \in \delta_G(C)\}$. The augmentation C^+ of C is the subgraph of the resulting graph induced by $V(C) \cup T$. We say that cluster C has the α-bandwidth property if set T of vertices is α-well-linked in C^+.

We note that, if a cluster C has the α-bandwidth property, then, using known techniques, we can efficiently construct a distribution $\mathcal{D}$ over the set $\Lambda_G(C)$ of internal C-routers, such that, for every edge $e \in E(C)$, $\mathbf{E}_{\mathcal{Q}(C)\sim\mathcal{D}(C)}[\mathrm{cong}(\mathcal{Q}(C),e)] \leq O(1/\alpha)$. However, in order to use the Basic Cluster Disengagement procedure, we need a stronger property: namely, for every edge $e \in E(C)$, we require that $\mathbf{E}_{\mathcal{Q}(C)\sim\mathcal{D}(C)}\left[(\mathrm{cong}(\mathcal{Q}(C),e))^2\right] \leq \beta$, for some parameter β. If we are given a distribution $\mathcal{D}(C)$ over the set $\Lambda_G(C)$ of internal C-routers with this latter property, then we say that cluster C is *η-light* with respect to $\mathcal{D}(C)$. Computing a distribution $\mathcal{D}(C)$ for which cluster C is η-light is a much more challenging task. We come close to achieving it in our Cluster Classification Theorem. Before we describe the theorem, we need one more definition. Let C be a cluster of a graph G, and let η' be some parameter. Assume that we are given some rotation system Σ for graph G, and let Σ^C be the rotation system for cluster C that is induced by Σ. Let $I^C = (C, \Sigma^C)$ be the resulting instance of MCNwRS. We say that cluster C is *η'-bad* if $\mathsf{OPT}_{\mathsf{cnwrs}}(I^C) \geq |\delta_G(C)|^2/\eta'$.

In the Cluster Classification Theorem, we provide an efficient algorithm, that, given an instance $I = (G, \Sigma)$ of MCNwRS with $|E(G)| = m$, and a cluster $C \subseteq G$ that has the α-bandwidth property (where $\alpha = \Omega(1/\mathrm{poly}\log m)$), either correctly establishes that cluster C is η'-bad, for $\eta' = 2^{O((\log m)^{3/4}\log\log m)}$, or produces a distribution $\mathcal{D}(C)$ over the set $\Lambda_G(C)$ of internal C-routers, such that cluster C is β-light with respect to $\mathcal{D}(C)$, for $\beta = 2^{O(\sqrt{\log m}\cdot\log\log m)}$. In fact, the algorithm is randomized, and, with a small probability, it may erroneously classify cluster C as being η'-bad, when this is not the case. This small technicality is immaterial to this high-level exposition, and we will ignore it here. The proof of the Cluster Classification Theorem is long and technically involved, and is partially responsible for the high approximation factor that we eventually obtain. It is our hope that a simpler and a cleaner proof of the theorem with better parameters will be discovered in

the future. We believe that the theorem is a graph-theoretic result that is interesting in its own right. We now provide a high-level summary of the main challenges in its proof.

At the heart of the proof is an algorithm called AlgFindGuiding. Suppose we are given an instance $I = (H, \Sigma)$ of MCNwRS, and a set T of k vertices of H called terminals, that are α-well-linked in H, for some parameter $0 < \alpha < 1$. Denote $C = H \setminus T$ and $|V(H)| = n$. The goal of the algorithm is to either establish that $\mathsf{OPT}_{\mathsf{cnwrs}}(H) + |E(H)| \geq k^2 \operatorname{poly}(\alpha/\log n)$; or to compute a distribution $\mathcal{D}(C)$ over internal C-routers, such that cluster C is $\eta' = \operatorname{poly}(\log n/\alpha)$-light with respect to $\mathcal{D}(C)$.

Consider first a much simpler setting, where H is a grid graph, and T is the set of vertices on the first row of the grid. For this special case, the algorithm of [27] (see also Lemma D.10 in the full version of [7]) provides the construction of a distribution $\mathcal{D}(C)$ over internal C-routers with the required properties. This result can be easily generalized to the case where H is a bounded-degree planar graph, since such a graph must contain a large grid minor. If H is a planar graph, but its maximum vertex degree is no longer bounded, we can still compute a grid-like structure in it, and apply the same arguments as in [27] in order to compute the desired distribution $\mathcal{D}(C)$. The difficulty in our case is that the graph H may be far from being planar, and, even though, from the Excluded Grid theorem of Robertson and Seymour [24, 25], it must contain a large grid-like structure, without having a drawing of H in the plane with a small number of crossings, we do not know how to compute such a structure[2]. We provide an algorithm that either establishes that $\mathsf{OPT}_{\mathsf{cnwrs}}(H)$ is large compared to k^2, or computes a grid-like structure in graph H, even if it is not a planar graph. Unfortunately, this algorithm only works in the setting where $|E(H)|$ is not too large comparable to k. Specifically, if we ensure that $|E(H)| \leq k \cdot \hat{\eta}$ for some parameter $\hat{\eta}$, then the algorithm either computes a distribution $\mathcal{D}(C)$ over internal C-routers that is η'-light (with $\eta' = \operatorname{poly}(\log n/\alpha)$ as before), or it establishes that $\mathsf{OPT}_{\mathsf{cnwrs}}(H) + |E(H)| \geq k^2 \operatorname{poly}(\alpha/(\hat{\eta} \log n))$.

Typically, this algorithm would be used in the following setting: we are given a cluster C of a graph G, that has the α-bandwidth property. We then let $H = C^+$ be the augmentation of C, and we let T be the set of vertices of C^+ corresponding to the edges of $\delta_H(C)$. In order for this result to be meaningful, we need to ensure that $|E(C)|$ is not too large compared to $|\delta_H(C)|$. Unfortunately, we may need to apply the classification theorem to clusters C for which $|E(C)| \gg |\delta_H(C)|$ holds. In order to overcome this difficulty, given such a cluster C, we construct a recursive decomposition of C into smaller and smaller clusters. Let $\mathcal{L}$ denote the resulting family of clusters, which is a laminar family of subgraphs of C. We ensure that every cluster $C' \in \mathcal{L}$ has $\alpha = \Omega(1/\operatorname{poly}\log m)$-bandwidth property, and, additionally, if we let $\hat{C}'$ be the graph obtained from C' by contracting every child-cluster of C' into a supernode, then the number of edges in $\hat{C}'$ is comparable to $|\delta_H(C')|$. We consider the clusters of $\mathcal{L}$ from smallest to largest. For each such cluster C', we carefully apply Algorithm AlgFindGuiding to the corresponding contracted graph $\hat{C}'$, in order to either classify cluster $\hat{C}'$ as $\eta(C')$-bad, or to compute a distribution $\mathcal{D}(C')$ over internal C'-routers, such that C' is $\beta(C')$-light with respect to $\mathcal{D}(C')$. Parameters $\eta(C')$ and $\beta(C')$ depend on the height of the cluster C' in the decomposition tree that is associated with the laminar family $\mathcal{L}$ of clusters. This recursive algorithm is eventually used to either establish that cluster C is $\eta(C)$-bad, or to compute a distribution $\mathcal{D}(C)$ over the set $\Lambda_G(C)$ of internal C-routers, such that cluster C is $\beta(C)$-light with respect to $\mathcal{D}(C)$. The final parameters $\eta(C)$ and $\beta(C)$ depend exponentially on the height of the decomposition tree associated with $\mathcal{L}$. This strong dependence on $\operatorname{dep}(\mathcal{L})$ is one of the reasons for the high approximation factor that our algorithm achieves.

Obstacles to Using Basic Cluster Disengagement. Let us now revisit the Basic Cluster Disengagement routine. We start with an instance $I = (G, \Sigma)$ of MCNwRS, and denote $|E(G)| = m$. Throughout, we use a parameter $\eta = 2^{O((\log m)^{3/4} \log\log m)}$, and $\beta = \eta^{1/8}$. Recall that the input to the procedure is a collection $\mathcal{C}$ of disjoint clusters of G. For every cluster $C \in \mathcal{C}$, we are also given a distribution $\mathcal{D}'(C)$ over the set of external C-routers, such that, for every edge e, $\mathbf{E}_{Q'(C)\sim\mathcal{D}'(C)}\left[\operatorname{cong}_G(Q'(C), e)\right] \leq \beta$, and a distribution $\mathcal{D}(C)$ over the set of internal C-routers, such that cluster C is β-light with respect to $\mathcal{D}(C)$. We are then guaranteed that the collection $\mathcal{I}$ of subinstances of I that is constructed via Basic Cluster Disengagement is an η-decomposition of I. We can slightly generalize this procedure to handle bad clusters as well. Specifically, suppose we are given a partition $(\mathcal{C}^{\text{light}}, \mathcal{C}^{\text{bad}})$ of the clusters in $\mathcal{C}$, and, for each cluster $C \in \mathcal{C}^{\text{light}}$, a distribution $\mathcal{D}(C)$ over internal C-routers, such that cluster C is β-light with respect to $\mathcal{D}(C)$. Assume further that each cluster $C \in \mathcal{C}^{\text{bad}}$ is β-bad. Additionally, assume that we are given, for every cluster $C \in \mathcal{C}$, a distribution $\mathcal{D}'(C)$ over external C-routers, such that, for every edge e, $\mathbf{E}_{Q'(C)\sim\mathcal{D}'(C)}\left[\operatorname{cong}_G(Q'(C), e)\right] \leq \beta$, and that every cluster $C \in \mathcal{C}$ has the α-bandwidth property, for some $\alpha = \Omega(1/\operatorname{poly}\log m)$. We can then generalize the Basic Cluster Disengagement procedure to provide the same guarantees as before in this setting, to obtain an η-decomposition of instance I.

Assume now that we are given an instance $I = (G, \Sigma)$ of MCNwRS, with $|E(G)| = m$. For simplicity, assume for now that the maximum vertex degree in G is quite small (it is sufficient, for example, that it is significantly smaller than m.) Using known techniques, we can compute a collection $\mathcal{C}$ of disjoint clusters of G, such that, for every cluster $C \in \mathcal{C}$, $|E(C)| \leq m/(2\mu)$; $|E^{\text{out}}_G(C)| \leq m/(2\mu)$; and every cluster $C \in \mathcal{C}$ has α-bandwidth property. If we could, additionally, compute, for each cluster $C \in \mathcal{C}$, a distribution $\mathcal{D}'(C)$ over external C-routers, such that, for every edge e, $\mathbf{E}_{Q'(C)\sim\mathcal{D}'(C)}\left[\operatorname{cong}_G(Q'(C), e)\right] \leq \beta$, then we could use the Cluster Classification Theorem to partition the set $\mathcal{C}$ of clusters into subsets $\mathcal{C}^{\text{light}}$ and $\mathcal{C}^{\text{bad}}$, and to compute, for every cluster $C \in \mathcal{C}^{\text{light}}$, a distribution $\mathcal{D}(C)$ over the set of its internal routers, such that every cluster in $\mathcal{C}^{\text{bad}}$ is η'-bad, and every cluster $C \in \mathcal{C}^{\text{light}}$ is η'-light with respect to $\mathcal{D}(C)$, for some parameter η'. We could then apply the Basic Cluster Disengagement procedure in order to compute the desired η-decomposition of the input instance I. Unfortunately, we currently do not have an algorithm that computes both the collection $\mathcal{C}$ of clusters of G with the above properties, and the required distributions over the external C-routers for each such cluster C. In order to overcome

[2] We note that we need the grid-like structure to have dimensions $(k' \times k')$, where k' is almost linear in k. Therefore, we cannot use the known bounds for the Excluded Minor Theorem (e.g. from [11]) for general graphs, and instead we need to use an analogue of the stronger version of the theorem for planar graphs.

this difficulty, we design Advanced Cluster Disengagement procedure, that generalizes Basic Cluster Disengagement, and no longer requires the distribution over external C-routers for each $C \in \mathcal{C}$.

Advanced Cluster Disengagement. The input to the Advanced Cluster Disengagement procedure is an instance $I = (G, \Sigma)$ of MCNwRS, and a set $\mathcal{C}$ of disjoint clusters of G, that we refer to as *basic clusters*, each of which has α-bandwidth property. Let $m = |E(G)|$, and $\eta = 2^{O((\log m)^{3/4} \log\log m)}$ as before. The output is an η-decomposition $\mathcal{I}$ of I, such that every instance $I' = (G', \Sigma') \in \mathcal{I}$ is a subinstance of I, and moreover, there is at most one basic cluster $C \in \mathcal{C}$ with $E(C) \subseteq E(G')$, with all other edges of G' lying in $E^{\text{out}}_G(\mathcal{C})$. The algorithm for the Advanced Cluster Disengagement and its analysis are significantly more involved than Basic Cluster Disengagement. We start with some intuition.

Consider the contracted graph $H = G_{|\mathcal{C}}$, and its Gomory-Hu tree T. This tree naturally defines a laminar family $\mathcal{L}$ of clusters of H: for every vertex $v \in V(H)$, we add to $\mathcal{L}$ the cluster U_v, that is the subgraph of H induced by vertex set $V(T_v)$, where T_v is the subtree of T rooted at v. From the properties of Gomory-Hu trees, if v' is the parent-vertex of vertex v in T, there is an external U_v-router $\mathcal{Q}'(U_v)$ in graph H with $\text{cong}_H(\mathcal{Q}'(U_v)) = 1$. Laminar family $\mathcal{L}$ of clusters of H naturally defines a laminar family $\mathcal{L}'$ of clusters of the original graph G, where for each cluster $U_v \in \mathcal{L}$, set $\mathcal{L}'$ contains a corresponding cluster U'_v, that is obtained from U_v, by un-contracting all supernodes that correspond to clusters of $\mathcal{C}$. For each such cluster $U'_v \in \mathcal{L}'$, we can use the external U_v-router $\mathcal{Q}'(U_v)$ in graph H in order to construct a distribution $\mathcal{D}'(U'_v)$ over external U'_v-routers in graph G, where for every edge e, $\mathbf{E}_{\mathcal{Q}'(U'_v) \sim \mathcal{D}'(U'_v)} \left[\text{cong}_G(\mathcal{Q}'(U'_v), e)\right] \leq O(1/\alpha)$. We can then apply the Basic Cluster Disengagement procedure to the laminar family $\mathcal{L}'$ and the distributions $\left\{\mathcal{D}'(U'_v)\right\}_{U'_v \in \mathcal{L}'}$ in order to compute an η^*-decomposition $\mathcal{I}$ of instance I, where every instance in $\mathcal{I}$ is a subinstance of I. Recall that the parameter η^* depends on the depth of the laminar family $\mathcal{L}'$, which is equal to the depth of the laminar family $\mathcal{L}$. Therefore, if $\text{dep}(\mathcal{L})$ is not too large (for example, it is at most $2^{O((\log m)^{3/4} \log\log m)}$), then we will obtain the desired η-decomposition of I. But unfortunately we have no control over the depth of the laminar family $\mathcal{L}$, and in particular the tools described so far do not work when the Gomory-Hu tree T is a path.

Roughly speaking, we would like to design a different disengagement procedure for the case where the tree T is a path, and then reduce the general problem (by exploiting Basic Cluster Disengagement) to this special case. In fact we follow a similar plan. We define a special type of instances (that we call *nice instances*), that resemble the case where the Gomory-Hu tree of the contracted graph $H = G_{|\mathcal{C}}$ is a path. While the motivation behind the definition of nice instances is indeed this special case, the specifics of the definition are somewhat different, in that it is more general in some of its aspects, and more restrictive and well-structured in others. We provide an algorithm for Cluster Disengagment of nice instances, that ensures that, for each resulting subinstance $I' = (G', \Sigma')$, there is at most one cluster $C \in \mathcal{C}$ with $C \subseteq G'$, and all other edges of G' lie in $E^{\text{out}}_G(\mathcal{C})$. We also provide another algorithm, that, given an instance $I = (G, \Sigma)$ of MCNwRS and a collection $\mathcal{C}$ of disjoint basic clusters of graph G, computes a decomposition $\mathcal{I}'$ of instance I, such that each resulting instance $I' \in \mathcal{I}'$ is a subinstance of I and a nice instance, with respect to the subset $\mathcal{C}(I') \subseteq \mathcal{C}$ of clusters, that contains every cluster $C \in \mathcal{C}$ with $C \subseteq G'$. Combining these two algorithms allows us to compute Advanced Cluster Disengagement.

Algorithm AlgDecompose**.** Recall that Algorithm AlgDecompose, given an instance $I = (G, \Sigma)$ of MCNwRS, needs to compute an η-decomposition $\mathcal{I}$ of I, where $\eta = 2^{O((\log m)^{3/4} \log\log m)}$ and $m = |E(G)|$, such that, for each instance $I' = (G', \Sigma') \in \mathcal{I}$, $|E(G')| \leq |E(G)|/\mu$ (here, $\mu = 2^{O((\log m^*)^{7/8} \log\log m^*)}$, and m^* is the number of edges in the original input instance I^* of MCNwRS). We say that a vertex v of G is a *high-degree vertex* if $|\delta_G(v)| \geq m/\text{poly}(\mu)$.

Consider first the special case where no vertex of G is a high-degree vertex. For this case, it is not hard to generalize known well-linked decomposition techniques to obtain a collection $\mathcal{C}$ of disjoint clusters of G, such that each $C \in \mathcal{C}$ has $\alpha = (1/\text{poly}\log m)$-bandwidth property, with $|E(C)| < O(m/\mu)$, and $|E^{\text{out}}_G(\mathcal{C})| \leq O(m/\mu)$. We can now apply the Advanced Cluster Disengagement procedure to the set $\mathcal{C}$ of clusters, in order to obtain the desired η-decomposition of I. Recall that we are guaranteed that each resulting instance $I' = (G', \Sigma') \in \mathcal{I}$ is a subinstance of I, and there is at most one cluster $C \in \mathcal{C}$ with $C \subseteq G'$, with all other edges of G' lying in $E^{\text{out}}_G(\mathcal{C})$. This ensures that $|E(G')| \leq m/\mu$, as required.

In general, however, it is possible that the input instance $I = (G, \Sigma)$ contains high-degree vertices. We then consider two cases. We say that instance I is *wide* if there is a vertex $v \in V(G)$, a partition (E_1, E_2) of the edges of $\delta_G(v)$, such that the edges of E_1 appear consecutively in the rotation $\mathcal{O}_v \in \Sigma$, and a collection $\mathcal{P}$ of at least $m/\text{poly}(\mu)$ simple edge-disjoint cycles in G, such that every cycle $P \in \mathcal{P}$ contains one edge of E_1 and one edge of E_2. An instance that is not wide is called *narrow*. We provide separate algorithms for dealing with narrow and wide instances.

Narrow Instances. The algorithm for decomposing narrow instances relies on and generalizes the algorithm for the special case where G has no high-degree vertices. As a first step, we compute a collection $\mathcal{C}$ of disjoint clusters of G, such that each cluster $C \in \mathcal{C}$ has $\alpha = \Omega(1/\text{poly}\log m)$-bandwidth property, and $|E^{\text{out}}_G(\mathcal{C})| < O(E(G)/\mu)$. The set $\mathcal{C}$ of clusters is partitioned into two subsets: set $\mathcal{C}^s$ of small clusters, and set $\mathcal{C}^f$ of *flower clusters*. For each cluster $C \in \mathcal{C}^s$, $|E(C)| < O(|E(G)|/\mu)$ holds. If C is a cluster of $\mathcal{C}^f$, then we no longer guarantee that $|E(C)|$ is small. Instead, we guarantee that cluster C has a special structure. Specifically, C must contain a single high-degree vertex $u(C)$, that we call the *flower center*, and all other vertices of C must be low-degree vertices. Additionally, there must be a set $\mathcal{X}(C) = \{X_1, \ldots, X_k\}$ of subgraphs of C, that we call *petals*, such that, for all $1 \leq i < j \leq k$, $V(X_i) \cap V(X_j) = \{u(C)\}$. We also require that, for all $1 \leq i \leq k$, there is a set E_i of $\Theta(m/\text{poly}(\mu))$ edges of $\delta_G(u(C))$ that are contiguous in the rotation $\mathcal{O}_{u(C)} \in \Sigma$, and lie in X_i. Lastly, we require that, for all $1 \leq i \leq k$, there is a set $\mathcal{Q}_i$ of edge-disjoint paths, connecting every edge of $\delta_G(X_i) \setminus \delta_G(u(C))$ to vertex $u(C)$, with all inner vertices on the paths lying in X_i.

We apply Advanced Cluster Disengagement to the set $\mathcal{C}$ of clusters, in order to compute an initial decomposition $\mathcal{I}_1$ of the input instance I, such that every instance in $\mathcal{I}_1$ is a subinstance of I. Unfortunately, it is possible that, for some instances $I' = (G', \Sigma') \in \mathcal{I}_1$, $|E(G')| > m/\mu$. For each such instance I', there must be some

flower cluster $C \in \mathcal{C}^f$ that is contained in G', and all other edges of G' must lie in $E_G^{\text{out}}(\mathcal{C})$.

We now consider each instance $I' = (G', \Sigma') \in \mathcal{I}_1$ with $|E(G')| > m/\mu$ one by one. Assume that $C \in \mathcal{C}^f$ is the flower cluster that is contained in G', and $\mathcal{X}(C) = \{X_1, \ldots, X_k\}$ is the set of its petals. We further decompose instance I' into a collection $\mathcal{I}(C)$ of subinstances, that consists of a single global instance $\hat{I}(C)$, and k additional instances $I_1(C), \ldots, I_k(C)$. We ensure that the graph $\hat{G}(C)$ associated with the global instance $\hat{I}(C)$ only contains edges of $E_G^{\text{out}}(\mathcal{C})$, so $|E(\hat{G}(C))| < m/\mu$ holds. For all $1 \le j \le k$, graph $G_j(C)$ associated with instance $I_j(C) \in \mathcal{I}(C)$ contains the petal X_j, and all other edges of $G_j(C)$ lie in $E_G^{\text{out}}(\mathcal{C})$. We note that unfortunately it is still possible that, for some $1 \le j \le k$, graph $G_j(C)$ contains too many edges (this can only happen if $|E(X_j)|$ is large). However, our construction ensures that, for each such instance $I_j(C)$, no high-degree vertices lie in graph $G_j(C)$. We can then further decompose instance $I_j(C)$ into subinstances using the algorithm that we designed for the case where no vertex of the input graph is a high-degree vertex. After this final decomposition, we are guaranteed that each of the resulting subinstances of instance I that we obtain contains fewer than m/μ edges, as required.

Wide Instances. Suppose we are given a wide instance $I = (G, \Sigma)$ of MCNwRS. In this case, we compute an η-decomposition $\mathcal{I}$ of instance I, such that, for each resulting instance $I' = (G', \Sigma') \in \mathcal{I}$, either $|E(G')| < m/\mu$ (in which case we say that I' is a *small instance*), or I' is a narrow instance. We will then further decompose each resulting narrow instance using the algorithm described above.

In order to obtain the decomposition $\mathcal{I}$ of I, we start with $\mathcal{I} = \{I\}$. As long as set $\mathcal{I}$ contains at least one wide instance $I' = (G', \Sigma')$ with $|E(G')| \ge m/\mu$, we perform a procedure that "splits" instance I' into two smaller subinstances. We now turn to describe this procedure at a high level.

Let $I' = (G', \Sigma') \in \mathcal{I}$ be a wide instance with $|E(G')| \ge m/\mu$. Recall that from the definition of a wide instance, there is a vertex $v \in V(G')$, a partition (E_1, E_2) of the edges of $\delta_{G'}(v)$, such that the edges of E_1 appear consecutively in the rotation $\mathcal{O}_v \in \Sigma'$, and a collection $\mathcal{P}$ of at least $m/\text{poly}(\mu)$ simple edge-disjoint cycles in G', such that every cycle in $\mathcal{P}$ contains one edge of E_1 and one edge of E_2. Consider the experiment, in which we choose a cycle $W \in \mathcal{P}$ uniformly at random. Since $|\mathcal{P}|$ is very large, with reasonably high probability, the edges of the cycle W participate in relatively few crossings in the optimal solution to instance I' of MCNwRS. We show that with high enough probability, there is a near-optimal solution to I', in which cycle W is drawn in the natural way. We use the cycle W in order to partition instance I' into two subinstances I_1, I_2 (intuitively, one subinstance corresponds to edges and vertices of G' that are drawn "inside" the cycle W in the near-optimal solution to I', and the other subinstance contains all edges and vertices that are drawn "outside" W). Each of the resulting two instances contains the cycle W, and, in order to be able to combine the solutions to the two subinstances to obtain a solution to I', we contract all vertices and edges of W, in each of the two instances, into a supernode. Let I_1', I_2' denote these two resulting instances. The main difficulty in the analysis is to show that there is a solution to each resulting instance of MCNwRS, such that the sum of the costs of two solutions is close to $\text{OPT}_{\text{cnwrs}}(I')$. The difficulty arises from the fact that we do not know what the optimal solution to instance I' looks like, and so our partition of G' into two subgraphs that are drawn on different sides of the cycle W in that solution may be imprecise. Instead, we need to "fix" the solutions to instances I_1, I_2 (that are induced by the optimal solution to I') in order to move all edges and vertices of each subinstance to lie on one side of the cycle W. In fact we are unable to do so directly. Instead, we show that we can compute a relatively small collection E' of edges, such that, if we remove the edges of E' from the graphs corresponding to instances I_1, I_2, then each of the resulting subinstances has the desired structure: namely, it can be drawn completely inside or completely outside the cycle W with relatively few crossings compared to $\text{OPT}_{\text{cnwrs}}(I')$. After we solve the two resulting subinstances recursively, we combine the resulting solutions, and add the images of the edges of E' back, in order to obtain a solution to instance I'.

2 PRELIMINARIES

By default, all logarithms in this paper are to the base of 2. All graphs are undirected and finite. Graphs may contain parallel edges but they may not contain self loops. Graphs without parallel edges are explicitly referred to as simple graphs.

2.1 Curves in General Position, Graph Drawings, Faces, and Crossings

Let γ be an open curve in the plane, and let P be a set of points in the plane. We say that γ is *internally disjoint* from P if no inner point of γ lies in P. In other words, $P \cap \gamma$ may only contain the endpoints of γ. Given a set Γ of open curves in the plane, we say that the curves in Γ are *internally disjoint* if, for every pair $\gamma, \gamma' \in \Gamma$ of distinct curves, every point $p \in \gamma \cap \gamma'$ is an endpoint of both curves. We use the following definition of curves in general position.

Definition 2.1 (Curves in general position). Let Γ be a finite set of open curves in the plane. We say that the curves of Γ are *in general position*, if the following conditions hold:

- for every pair $\gamma, \gamma' \in \Gamma$ of distinct curves, there is a finite number of points p with $p \in \gamma \cap \gamma'$;
- for every pair $\gamma, \gamma' \in \Gamma$ of distinct curves, an endpoint of γ may not serve as an inner point of γ' or of γ; and
- for every triple $\gamma, \gamma', \gamma'' \in \Gamma$ of distinct curves, if some point p lies on all three curves, then it must be an endpoint of each of these three curves.

Let Γ be a set of curves in general position, and let $\gamma, \gamma' \in \Gamma$ be a pair of curves. Let p be any point that lies on both γ and γ', but is not an endpoint of either curve. We then say that point p is a *crossing* between γ and γ', or that curves γ and γ' *cross* at point p. We are now ready to formally define graph drawings.

Definition 2.2 (Graph Drawings). A *drawing* φ of a graph G in the plane is a map φ, that maps every vertex v of G to a point $\varphi(v)$ in the plane (called the *image of* v), and every edge $e = (u, v)$ of G to a simple curve $\varphi(e)$ in the plane whose endpoints are $\varphi(u)$ and $\varphi(v)$ (called the *image of* e), such that all points in set $\{\varphi(v) \mid v \in V(G)\}$ are distinct, and the set $\{\varphi(e) \mid e \in E(G)\}$ of curves is in general position. Additionally, for every vertex $v \in V(G)$ and edge $e \in E(G)$, $\varphi(v) \in \varphi(e)$ only if v is an endpoint of e.

Assume now that we are given some drawing φ of graph G in the plane, and assume that for some pair e, e' of edges, their images $\varphi(e), \varphi(e')$ cross at point p. Then we say that $(e, e')_p$ is a *crossing* in the drawing φ (we may sometimes omit the subscript p if the images of the two edges only cross at one point). We also say that p is a *crossing point* of drawing φ. We denote by $\mathrm{cr}(\varphi)$ the total number of crossings in the drawing φ.

Planar Graphs and Planar Drawings. A graph G is *planar* if there is a drawing of G in the plane with no crossings. A drawing φ of a graph G in the plane with $\mathrm{cr}(\varphi) = 0$ is called a *planar drawing* of G. We use the following result by Hopcroft and Tarjan.

Theorem 2.3 ([15]). *There is an algorithm, that, given a graph G, correctly establishes whether G is planar, and if so, computes a planar drawing of G. The running time of the algorithm in $O(|V(G)|)$.*

2.2 Circular Orderings, Orientations, and Rotation Systems

Suppose we are given a collection $U = \{u_1, \ldots, u_r\}$ of elements. Let D be any disc in the plane. Assume further that we are given, for every element $u_i \in U$, a point p_i on the boundary of D, so that all resulting points in $\{p_1, \ldots, p_r\}$ are distinct. As we traverse the boundary of the disc D in the clock-wise direction, the order in which we encounter the points $p_1, \ldots, p_r$ defines a *circular ordering $\mathcal{O}$ of the elements of U*. If we traverse the boundary of the disc D in the counter-clock-wise direction, we obtain a circular ordering $\mathcal{O}'$ of the elements of U, which is the mirror image of the ordering $\mathcal{O}$. We say that the orderings $\mathcal{O}$ and $\mathcal{O}'$ are *identical* but their *orientations* are different, or opposite: $\mathcal{O}$ has a negative and $\mathcal{O}'$ has a positive orientation. Whenever we refer to an ordering $\mathcal{O}$ of elements, we view it as *unoriented* (that is, the orientation can be chosen arbitrarily). When the orientation of the ordering is fixed, we call it an *oriented ordering*, and denote it by $(\mathcal{O}, b)$, where $\mathcal{O}$ is the associated (unoriented) ordering of elements of U, and $b \in \{-1, 1\}$ is the orientation, with $b = -1$ indicating a negative (that is, clockwise), orientation. Given a graph G and a vertex $v \in V(G)$, a circular ordering $\mathcal{O}_v$ of the edges of $\delta_G(v)$ is called a *rotation*. A collection of circular orderings $\mathcal{O}_v$ for all vertices $v \in V(G)$ is called a *rotation system* for graph G.

2.3 Tiny v-Discs and Drawings that Obey Rotations

Given a graph G, its drawing φ, and a vertex $v \in V(G)$, we will sometimes utilize the notion of a *tiny v-disc*, that we define next.

Definition 2.4 (Tiny v-Disc). Let G be a graph and let φ be a drawing of G on the sphere or in the plane. For each vertex $v \in V(G)$, we denote by $D_\varphi(v)$ a very small disc containing the image of v in its interior, and we refer to $D_\varphi(v)$ as *tiny v-disc*. Disc $D_\varphi(v)$ must be small enough to ensure that, for every edge $e \in \delta_G(v)$, the image $\varphi(e)$ of e intersects the boundary of $D_\varphi(v)$ at a single point, and $\varphi(e) \cap D_\varphi(v)$ is a contiguous curve. Additionally, we require that for every vertex $u \in V(G) \setminus \{v\}$, $\varphi(u) \notin D_\varphi(v)$; for every edge $e' \in E(G) \setminus \delta_G(v)$, $\varphi(e') \cap D_\varphi(v) = \emptyset$; and that no crossing point of drawing φ is contained in $D_\varphi(v)$. Lastly, we require that all discs in $\left\{D_\varphi(v) \mid v \in V(G)\right\}$ are mutually disjoint.

Consider now a graph G, a vertex $v \in V(G)$, and a drawing φ of G. Consider the tiny v-disc $D = D_\varphi(v)$. For every edge $e \in \delta_G(v)$, let p_e be the (unique) intersection point of the image $\varphi(e)$ of e and the boundary of the disc D. Let $\mathcal{O}$ be the (unoriented) circular ordering in which the points of $\{p_e\}_{e \in \delta_G(v)}$ appear on the boundary of D. Then $\mathcal{O}$ naturally defines a circular ordering $\mathcal{O}^*_v$ of the edges of $\delta_G(v)$: ordering $\mathcal{O}^*_v$ is obtained from $\mathcal{O}$ by replacing, for each edge $e \in \delta_G(v)$, point p_e with the edge e. We say that *the images of the edges of $\delta_G(v)$ enter the image of v in the order $\mathcal{O}^*_v$* in the drawing φ. For brevity, we may sometimes say that the edges of $\delta_G(v)$ enter v in the order $\mathcal{O}^*_v$ in φ. While we view the ordering $\mathcal{O}^*_v$ as unoriented, drawing φ also defines an orientation for this ordering. If the points in set $\{p_e \mid e \in \delta_G(v)\}$ are encountered in the order $\mathcal{O}^*_v$ when traversing the boundary of D in the counter-clock-wise direction, then the orientation is 1; otherwise it is -1.

Assume now that we are given a graph G and a rotation system Σ for G. Let φ be a drawing of G. Consider any vertex $v \in V(G)$, and its rotation $\mathcal{O}_v \in \Sigma$. We say that the drawing φ *obeys the rotation* $\mathcal{O}_v \in \Sigma$, if the order in which the edges of $\delta_G(v)$ enter v in φ is precisely $\mathcal{O}_v$ (note that both orderings are unoriented). We say that the *orientation of v is* -1, or *negative*, in the drawing φ if the orientation of the ordering $\mathcal{O}_v$ of the edges of $\delta_G(v)$ as they enter v is -1, and otherwise, the orientation of v in φ is 1, or positive. We say that drawing φ of G *obeys the rotation system* Σ, if it obeys the rotation $\mathcal{O}_v \in \Sigma$ for every vertex $v \in V(G)$.

Assume now that we are given a set Γ of curves in general position, where each curve $\gamma \in \Gamma$ is an open curve. Let p be any point that serves as an endpoint of at least one curve in Γ, and let $\Gamma' \subseteq \Gamma$ be the set of curves for which p serves as an endpoint. We then define a *tiny p-disc $D(p)$* to be a small disc that contains the point p in its interior; does not contain any other point that serves as an endpoint of a curve in Γ; and does not contain any crossing point of curves in Γ. Additionally, we ensure that, for every curve $\gamma \in \Gamma$, if $\gamma \in \Gamma'$, then $\gamma \cap D(p)$ is a simple curve, and otherwise $\gamma \cap D(p) = \emptyset$. For every curve $\gamma \in \Gamma'$, let $q(\gamma)$ be the unique point of γ lying on the boundary of the disc $D(p)$. Note that all points in $\{q(\gamma) \mid \gamma \in \Gamma'\}$ are distinct. Let $\mathcal{O}$ be the circular order in which these points are encountered when we traverse the boundary of $D(p)$. As before, this ordering naturally defines a circular ordering $\mathcal{O}'$ of the curves in Γ'. We then say that the curves of Γ' *enter the point p in the order $\mathcal{O}'$*.

2.4 Problem Definitions and Trivial Algorithms

In the Minimum Crossing Number problem, the input is an n-vertex graph G, and the goal is to compute a drawing of G in the plane with minimum number of crossings. The value of the optimal solution, also called the *crossing number* of G, is denoted by $\mathrm{OPT}_{\mathrm{cr}}(G)$.

We also consider a closely related problem called Minimum Crossing Number with Rotation System (MCNwRS). In this problem, the input is a graph G, and a rotation system Σ for G. Given an instance $I = (G, \Sigma)$ of the MCNwRS problem, we say that a drawing φ of G is a *feasible solution* for I if φ obeys the rotation system Σ. The *cost* of the solution is the number of crossings in φ. The goal in the MCNwRS problem is to compute a feasible solution to the given input instance I of smallest possible cost. We denote the cost of the optimal solution of the MCNwRS instance I by $\mathrm{OPT}_{\mathrm{cnwrs}}(I)$.

We use the following two simple theorems about the MCNwRS problem, whose proofs are deferred to the full version.

Theorem 2.5. *There is an efficient algorithm, that, given an instance $I = (G, \Sigma)$ of* MCNwRS*, correctly determines whether or not $\mathsf{OPT}_{\mathsf{cnwrs}}(I) = 0$, and, if so, computes a feasible solution to instance I of cost 0.*

Theorem 2.6. *There is an efficient algorithm, that given an instance $I = (G, \Sigma)$ of* MCNwRS*, computes a feasible solution to I, of cost at most $|E(G)|^2$.*

We refer to the solution computed by the algorithm from Theorem 2.6 as a *trivial solution.*

2.5 A ν-Decomposition of an Instance

A central tool that we use in our divide-and-conquer algorithm is a ν-decomposition of instances.

Definition 2.7 (ν-Decomposition of Instances). Let $I = (G, \Sigma)$ be an instance of MCNwRS with $|E(G)| = m$, and let $\nu \geq 1$ be a parameter. We say that a collection $\mathcal{I}$ of instances of MCNwRS is a *ν-decomposition of I*, if the following hold:

- (D1) $\sum_{I'=(G',\Sigma')\in\mathcal{I}} |E(G')| \leq m \cdot (\log m)^{O(1)}$;
- (D2) $\sum_{I'\in\mathcal{I}} \mathsf{OPT}_{\mathsf{cnwrs}}(I') \leq (\mathsf{OPT}_{\mathsf{cnwrs}}(I) + m) \cdot \nu$; and
- (D3) there is an efficient algorithm $\mathsf{Alg}(\mathcal{I})$, that, given, a feasible solution $\varphi(I')$ to every instance $I' \in \mathcal{I}$, computes a feasible solution φ to instance I, of cost $\mathsf{cr}(\varphi) \leq O\left(\sum_{I'\in\mathcal{I}} \mathsf{cr}(\varphi(I'))\right)$.

We say that a randomized algorithm Alg is a *ν-decomposition algorithm for a family $\mathcal{I}^*$ of instances of* MCNwRS if Alg is an efficient algorithm, that, given an instance $I = (G, \Sigma) \in \mathcal{I}^*$, produces a collection $\mathcal{I}$ of instances that has properties D1 and D3, and ensures the following additional property (that replaces Property D2):

- (D'2) $\mathbf{E}\left[\sum_{I'\in\mathcal{I}} \mathsf{OPT}_{\mathsf{cnwrs}}(I')\right] \leq (\mathsf{OPT}_{\mathsf{cnwrs}}(I) + |E(G)|) \cdot \nu$.

2.6 Subinstances

We use the following definition of subinstances.

Definition 2.8 (Subinstances). Let $I = (G, \Sigma)$ and $I' = (G', \Sigma')$ be two instances of MCNwRS. We say that instance I' is a *subinstance* of instance I, if there is a subgraph $\tilde{G} \subseteq G$, and a collection $S_1, \ldots, S_r$ of mutually disjoint subsets of vertices of $\tilde{G}$, such that graph G' can be obtained from $\tilde{G}$ by contracting, for all $1 \leq i \leq r$, every vertex set S_i into a supernode u_i; we keep parallel edges but remove self-loops[3]. We do not distinguish between the edges incident to the supernodes in graph G' and their counterparts in graph G. For every vertex $v \in V(G') \cap V(G)$, its rotation $\mathcal{O}'_v$ in Σ' must be consistent with the rotation $\mathcal{O}_v \in \Sigma$, while for every supernode u_i, its rotation $\mathcal{O}'_{u_i}$ in Σ' can be defined arbitrarily.

Observe that, if instance $I' = (G', \Sigma')$ is a subinstance of $I = (G, \Sigma)$, then $|E(G')| \leq |E(G)|$. Also notice that the subinstance relation is transitive: if instance I_1 is a subinstance of instance I_0, and instance I_2 is a subinstance of I_1, then I_2 is a subinstance of I_0.

[3]Note that this definition is similar to the definition of a minor, except that we do not require that the induced subgraphs $G[S_i]$ of G are connected.

3 AN ALGORITHM FOR MCNwRS

In this section we provide the proof of Theorem 1.1, with most of the details deferred to the full version. Throughout the paper, we denote by $I^* = (G^*, \Sigma^*)$ the input instance of the MCNwRS problem, and we denote $m^* = |E(G^*)|$. We also use the following parameter that is central to our algorithm: $\mu = 2^{c^*(\log m^*)^{7/8} \log\log m^*}$, where c^* is a large enough constant.

As mentioned already, our algorithm for solving the MCNwRS problem is recursive, and, over the course of the recursion, we will consider various other instances I of MCNwRS. Throughout the algorithm, parameters m^* and μ remain unchanged, and are defined with respect to the original input instance I^*. The main technical ingredient of the proof is the following theorem.

Theorem 3.1. *There is a constant c'', and an efficient randomized algorithm, that, given an instance $I = (G, \Sigma)$ of* MCNwRS *with $|E(G)| = m$, such that $\mu^{c''} \leq m \leq m^*$, either returns FAIL, or computes a collection $\mathcal{I}$ of instances of* MCNwRS *with the following properties:*

- *for every instance $I' = (G', \Sigma') \in \mathcal{I}$, $|E(G')| \leq m/\mu$;*
- *$\sum_{I'=(G',\Sigma')\in\mathcal{I}} |E(G')| \leq m \cdot (\log m)^{O(1)}$;*
- *there is an efficient algorithm called* AlgCombineDrawings*, that, given a solution $\varphi(I')$ to every instance $I' \in \mathcal{I}$, computes a solution φ to instance I; and*
- *if $\mathsf{OPT}_{\mathsf{cnwrs}}(I) \leq |E(G)|^2/\mu^{c''}$, then with probability at least 15/16, all of the following hold: (i) the algorithm does not return FAIL; (ii) $\mathcal{I} \neq \emptyset$; (iii) $\sum_{I'\in\mathcal{I}} \mathsf{OPT}_{\mathsf{cnwrs}}(I') \leq (\mathsf{OPT}_{\mathsf{cnwrs}}(I) + m) \cdot 2^{O((\log m)^{3/4} \log\log m)}$; and (iv) if* AlgCombineDrawings *is given as input a solution $\varphi(I')$ to every instance $I' \in \mathcal{I}$, then the resulting solution φ to instance I that it computes has cost at most: $O\left(\sum_{I'\in\mathcal{I}} \mathsf{cr}(\varphi(I'))\right) + (\mathsf{OPT}_{\mathsf{cnwrs}}(I) + m) \cdot \mu^{O(1)}$.*

In the next subsection, we prove Theorem 1.1 using Theorem 3.1.

3.1 Proof of Theorem 1.1

Throughout the proof, we assume that m^* is larger than a sufficiently large constant, since otherwise we can return a trivial solution to instance I^*, from Theorem 2.6.

We let $c_g > 100$ be a large enough constant, so that, for example, when the algorithm from Theorem 3.1 is applied to an instance $I = (G, \Sigma)$ with $m = |E(G)|$, such that $\mu^{c''} \leq m \leq m^*$ holds, it is guaranteed to return a family $\mathcal{I}$ of instances of MCNwRS, with $\sum_{I'=(G',\Sigma')\in\mathcal{I}} |E(G')| \leq m \cdot (\log m)^{c_g}$. We say that the algorithm from Theorem 3.1 is *successful* if all of the following hold: (i) the algorithm does not return FAIL; (ii) if $\mathcal{I}$ is the collection of instances returned by the algorithm, then $\mathcal{I} \neq \emptyset$; (iii) $\sum_{I'\in\mathcal{I}} \mathsf{OPT}_{\mathsf{cnwrs}}(I') \leq (\mathsf{OPT}_{\mathsf{cnwrs}}(I) + m) \cdot 2^{c_g((\log m)^{3/4} \log\log m)}$; and (iv) if algorithm AlgCombineDrawings is given a solution $\varphi(I')$ to every instance $I' \in \mathcal{I}$, then it computes a solution φ to instance I, of cost at most $c_g \cdot (\sum_{I'\in\mathcal{I}} \mathsf{cr}(\varphi(I')) + (\mathsf{OPT}_{\mathsf{cnwrs}}(I) + m) \cdot \mu^{c_g}$.

By letting c_g be a large enough constant, Theorem 3.1 guarantees that, if $\mathsf{OPT}_{\mathsf{cnwrs}}(I) \leq |E(G)|^2/\mu^{c''}$, then with probability at least 15/16 the algorithm is successful. We assume that the parameter c^* in the definition of μ is sufficiently large, so that, e.g., $c^* > 2c_g$.

We use a simple recursive algorithm called AlgRecursiveCNwRS, that appears in Figure 1.

AlgRecursiveCNwRS

Input: an instance $I = (G, \Sigma)$ of MCNwRS with $|E(G)| \leq m^*$.
Output: a feasible solution to instance I.

(1) Use the algorithm from Theorem 2.5 to determine whether $\mathsf{OPT}_{\mathsf{cnwrs}}(I) = 0$. If so, use the algorithm from Theorem 2.5 to compute a solution to I of cost 0. Return this solution, and terminate the algorithm.
(2) Use the algorithm from Theorem 2.6 to compute a trivial solution φ' to instance I.
(3) If $|E(G)| \leq \mu^{c''}$, return the trivial solution φ' and terminate.
(4) For $1 \leq j \leq \lceil \log m^* \rceil$:
  (a) Apply the algorithm from Theorem 3.1 to instance I.
  (b) If the algorithm returns FAIL, let $\varphi_j = \varphi'$ be the trivial solution to instance I, and set $\mathcal{I}_j(I) = \emptyset$.
  (c) Otherwise:
    (i) Let $\mathcal{I}_j(I)$ be the instances computed by the algorithm.
    (ii) For every $I' \in \mathcal{I}_j(I)$, apply AlgRecursiveCNwRS to instance I', to obtain a solution $\varphi(I')$ to this instance.
    (iii) Apply AlgCombineDrawings from Theorem 3.1 to $\{\varphi(I')\}_{I' \in \mathcal{I}_j(I)}$, to obtain a solution φ_j to instance I.

Return the solution from $\left\{\varphi', \varphi_1, \ldots, \varphi_{\lceil \log m^* \rceil}\right\}$ that has fewest crossings.

Figure 1: AlgRecursiveCNwRS

In order to analyze the algorithm, it is convenient to associate a *partitioning tree* T with it, whose vertices correspond to all instances of MCNwRS considered over the course of the algorithm. Let $L = \lceil \log m^* \rceil$. We start with the tree T containing a single root vertex $v(I^*)$, representing the input instance I^*. Consider now some vertex $v(I)$ of the tree, representing some instance $I = (G, \Sigma)$. When Algorithm AlgRecursiveCNwRS was applied to instance I, if it did not terminate after the first three steps, it constructed L collections $\mathcal{I}_1(I), \ldots, \mathcal{I}_L(I)$ of instances (some of which may be empty, in case the algorithm from Theorem 3.1 returned FAIL in the corresponding iteration). For each such instance $I' \in \bigcup_{j=1}^{L} \mathcal{I}_j(I)$, we add a vertex $v(I')$ representing instance I' to T, that becomes a child vertex of $v(I)$. This concludes the description of the partitioning tree T.

We denote by $\mathcal{I}^* = \{I \mid v(I) \in V(T)\}$ the set of all instances of MCNwRS, whose corresponding vertex appears in the tree T. For each such instance $I \in \mathcal{I}^*$, its *recursive level* is the distance from vertex $v(I)$ to the root vertex $v(I^*)$ in the tree T (so the recursive level of $v(I^*)$ is 0). For $j \geq 0$, we denote by $\hat{\mathcal{I}}_j \subseteq \mathcal{I}^*$ the set of all instances $I \in \mathcal{I}^*$, whose recursive level is j. Lastly, the *depth* of the tree T, denoted by $\mathsf{dep}(T)$, is the largest recursive level of any instance in $\mathcal{I}^*$. In order to analyze the algorithm, we start with the following two simple observations, whose proofs are deferred to the full version.

Observation 3.1. $\mathsf{dep}(T) \leq \frac{(\log m^*)^{1/8}}{c^* \log \log m^*}$.

Observation 3.2. $\sum_{I=(G,\Sigma) \in \mathcal{I}^*} |E(G)| \leq m^* \cdot 2^{(\log m^*)^{1/8}}$.

We use the following immediate corollary of Observation 3.2.

Corollary 3.2. *The number of instances $I = (G, \Sigma) \in \mathcal{I}^*$ with $|E(G)| \geq \mu^{c''}$ is at most m^*.*

We say that an instance $I \in \mathcal{I}^*$ is a *leaf instance*, if vertex $v(I)$ is a leaf vertex of the tree T, and we say that it is a non-leaf instance otherwise. Consider now a non-leaf instance $I = (G, \Sigma) \in \mathcal{I}^*$. We say that a bad event $\mathcal{E}(I)$ happens, if $0 < \mathsf{OPT}_{\mathsf{cnwrs}}(I) \leq |E(G)|^2/\mu^{c''}$, and, for all $1 \leq j \leq L$, the jth application of the algorithm from Theorem 3.1 to instance I was unsuccessful. Clearly, from Theorem 3.1, $\mathbf{Pr}[\mathcal{E}(I)] \leq (1/16)^L \leq 1/(m^*)^4$. Let $\mathcal{E}$ be the bad event that event $\mathcal{E}(I)$ happened for any instance $I \in \mathcal{I}^*$. From the Union Bound and Corollary 3.2, we get that $\mathbf{Pr}[\mathcal{E}] \leq 1/(m^*)^2$. We use the following immediate observation.

Observation 3.3. *If Event $\mathcal{E}$ does not happen, then for every leaf vertex $v(I)$ of T with $I = (G, \Sigma)$, either $|E(G)| \leq \mu^{c''}$; or $\mathsf{OPT}_{\mathsf{cnwrs}}(I) = 0$; or $\mathsf{OPT}_{\mathsf{cnwrs}}(I) > |E(G)|^2/\mu^{c''}$.*

We use the next lemma to complete the proof of Theorem 1.1.

Lemma 3.3. *If Event $\mathcal{E}$ does not happen, then* AlgRecursiveCNwRS *computes a solution for instance $I^* = (G^*, \Sigma^*)$ of cost at most $2^{O((\log m^*)^{7/8} \log \log m^*)} \cdot (\mathsf{OPT}_{\mathsf{cnwrs}}(I^*) + |E(G^*)|)$.*

Proof. Consider a non-leaf instance $I = (G, \Sigma)$, and let $\mathcal{I}_1(I), \ldots, \mathcal{I}_L(I)$ be families of instances of MCNwRS that AlgRecursiveCNwRS computed, when applied to instance I. Recall that, for each $1 \leq j \leq L$ with $\mathcal{I}_j(I) \neq \emptyset$, the algorithm computes a solution φ_j to instance I, by first solving each of the instances in $\mathcal{I}_j(I)$ recursively, and then combining the resulting solutions using Algorithm AlgCombineDrawings. Eventually, the algorithm returns the best solution of $\{\varphi', \varphi_1, \ldots, \varphi_L\}$, where φ' is the trivial solution, whose cost is at most $|E(G)|^2$. We fix an arbitrary index $1 \leq j \leq L$, such that the jth application of the algorithm from Theorem 3.1 to instance I was successful. Note that the cost of the solution to instance I that the algorithm returns is at most $\mathsf{cr}(\varphi_j)$. We then *mark* the vertices of $\left\{v(I') \mid I' \in \mathcal{I}_j(I)\right\}$ and the root in the tree T. Let T^* be the subgraph of T induced by all marked vertices. It is easy to verify that T^* is a tree, and moreover, if Event $\mathcal{E}$ did not happen, every leaf vertex of T^* is also a leaf vertex of T. For a vertex $v(I) \in V(T^*)$, we denote by $h(I)$ the length of the longest path in tree T^*, connecting vertex $v(I)$ to any of its descendants in the tree. We use the following claim, whose proof is deferred to the full version.

Claim 3.1. *Assume that Event $\mathcal{E}$ did not happen. Then there is a fixed constant $\tilde{c} \geq \max\left\{c'', c_g, c^*\right\}$, such that, for every vertex $v(I) \in V(T^*)$, whose corresponding instance is denoted by $I = (G, \Sigma)$, the cost of the solution that the algorithm computes for I is at most:*

$$2^{\tilde{c} h(I) (\log m^*)^{3/4} \log \log m^*} \mu^{c'' c_g} \cdot \mathsf{OPT}_{\mathsf{cnwrs}}(I) + (\log m^*)^{4 c_g h(I)} \mu^{2c'' \tilde{c}} |E(G)|.$$

We are now ready to complete the proof of Lemma 3.3. Recall that $h(I^*) = \mathsf{dep}(T^*) \leq \mathsf{dep}(T) \leq \frac{(\log m^*)^{1/8}}{c^* \log \log m^*}$ from Observation 3.1. Therefore, from Claim 3.1, the cost of the solution that the algorithm

computes for instance I^* is bounded by:

$$\begin{aligned}
&2^{O(\mathrm{dep}(T))\cdot(\log m^*)^{3/4}\log\log m^*}\cdot\mu^{O(1)}\cdot\mathsf{OPT}_{\mathsf{cnwrs}}(I^*)\\
&+(\log m^*)^{O(\mathrm{dep}(T))}\cdot\mu^{O(1)}\cdot m^*\\
&\leq 2^{O((\log m^*)^{7/8})}\cdot\mu^{O(1)}\cdot\mathsf{OPT}_{\mathsf{cnwrs}}(I^*)\\
&+(\log m^*)^{O((\log m^*)^{1/8}/\log\log m^*)}\cdot\mu^{O(1)}\cdot m^*\\
&\leq 2^{O((\log m^*)^{7/8}\log\log m^*)}\cdot\left(\mathsf{OPT}_{\mathsf{cnwrs}}(I^*)+|E(G^*)|\right).
\end{aligned}$$

□

In order to complete the proof of Theorem 1.1, it is now enough to prove Theorem 3.1, which we do in the next subsection.

3.2 Proof of Theorem 3.1 – Main Definitions and Theorems

We classify instances of MCNwRS into *wide* and *narrow*. Wide instances are, in turn, classified into *well-connected* and not well-connected instances. We then provide different algorithms for decomposing instances of each of the resulting three kinds. We use the following notion of a high-degree vertex.

Definition 3.4 (High-degree vertex). Let G be any graph. A vertex $v\in V(G)$ is a *high-degree* vertex, if $\deg_G(v)\geq|E(G)|/\mu^4$.

We are now ready to define wide and narrow instances.

Definition 3.5 (Wide and Narrow Instances). Let $I=(G,\Sigma)$ be an instance of MCNwRS with $|E(G)|=m$. We say that I is a *wide* instance, if there is a high-degree vertex $v\in V(G)$, a partition (E_1,E_2) of the edges of $\delta_G(v)$, such that the edges of E_1 appear consecutively in the rotation $\mathcal{O}_v\in\Sigma$, and so do the edges of E_2, and there is a collection $\mathcal{P}$ of at least $\left\lfloor m/\mu^{50}\right\rfloor$ simple edge-disjoint cycles in G, such that every cycle $P\in\mathcal{P}$ contains one edge of E_1 and one edge of E_2. An instance that is not wide is called *narrow*.

Note that there is an efficient algorithm to check whether a given instance I of MCNwRS is wide, and, if so, to compute the corresponding set $\mathcal{P}$ of cycles, via standard algorithms for maximum flow. (For every vertex $v\in V(G)$, we try all possible partitions (E_1,E_2) of $\delta_G(v)$ with the required properties, as the number of such partitions is bounded by $|\delta_G(v)|^2$.) We will use the following simple observation regarding narrow instances, whose proof is deferred to the full version.

Observation 3.4. *If an instance $I=(G,\Sigma)$ of* MCNwRS *is narrow, then for every pair u,v of distinct high-degree vertices of G, and any set $\mathcal{P}$ of edge-disjoint paths connecting u to v in G, $|\mathcal{P}|\leq 2\left\lceil|E(G)|/\mu^{50}\right\rceil$ must hold.*

Next, we define well-connected wide instances.

Definition 3.6 (Well-Connected Wide Instances). Let $I=(G,\Sigma)$ be a wide instance of MCNwRS with $|E(G)|=m$. We say that it is a *well-connected* instance iff for every pair u,v of distinct vertices of G with $\deg_G(v),\deg_G(u)\geq m/\mu^5$, there is a collection of at least $8m/\mu^{50}$ edge-disjoint paths connecting u to v in G.

The proof of Theorem 3.1 relies on the following three theorems, whose proofs are deferred to the full version. The first theorem deals with wide instances that are not necessarily well-connected.

Theorem 3.7. *There is an efficient randomized algorithm, whose input is a wide instance $I=(G,\Sigma)$ of* MCNwRS*, with $m=|E(G)|$, such that $\mu^{20}\leq m\leq m^*$. The algorithm computes a ν-decomposition $\mathcal{I}$ of I, for $\nu=2^{O((\log m)^{3/4}\log\log m)}$, where every instance $I'=(G',\Sigma')\in\mathcal{I}$ is a subinstance of I, such that either $|E(G')|\leq m/\mu$; or I' is a narrow instance; or I' is a wide and well-connected instance.*

The second theorem deals with wide well-connected instances.

Theorem 3.8. *There is an efficient randomized algorithm, whose input is a wide and well-connected instance $I=(G,\Sigma)$ of* MCNwRS*, with $m=|E(G)|$, such that $\mu^{c'}\leq m\leq m^*$ holds, for some large enough constant c'. The algorithm either returns FAIL, or computes a non-empty collection $\mathcal{I}$ of instances of* MCNwRS*, such that the following hold:*

- *$\sum_{I'=(G',\Sigma')\in\mathcal{I}}|E(G')|\leq 2|E(G)|$;*
- *for every instance $I'=(G',\Sigma')\in\mathcal{I}$, either $|E(G')|\leq m/\mu$, or instance I' narrow;*
- *there is an efficient algorithm called* AlgCombineDrawings′*, that, given a solution $\varphi(I')$ to every instance $I'\in\mathcal{I}$, computes a solution φ to instance I; and*
- *if $\mathsf{OPT}_{\mathsf{cnwrs}}(I)\leq m^2/\mu^{c'}$ then, with probability at least $1-1/\mu^2$, all of the following hold: (i) the algorithm does not return FAIL; (ii) $\sum_{I'\in\mathcal{I}}\mathsf{OPT}_{\mathsf{cnwrs}}(I')\leq\mathsf{OPT}_{\mathsf{cnwrs}}(I)\cdot(\log m)^{O(1)}$; and (iii) if algorithm* AlgCombineDrawings′ *is given as input a solution $\varphi(I')$ to every instance $I'\in\mathcal{I}$, then the resulting solution φ to instance I that it computes has cost at most: $\mathsf{cr}(\varphi)\leq\sum_{I'\in\mathcal{I}}\mathsf{cr}(\varphi(I'))+\mathsf{OPT}_{\mathsf{cnwrs}}(I)\cdot\mu^{O(1)}$.*

The third theorem deals with narrow instances.

Theorem 3.9. *There is an efficient randomized algorithm, whose input is a narrow instance $I=(G,\Sigma)$ of* MCNwRS*, with $m=|E(G)|$, such that $\mu^{50}\leq|E(G)|\leq 2m^*$. The algorithm either returns FAIL, or computes a ν-decomposition $\mathcal{I}$ of I, for $\nu=2^{O((\log m)^{3/4}\log\log m)}$, such that, for every instance $I'=(G',\Sigma')\in\mathcal{I}$, $|E(G')|\leq m/(2\mu)$. Moreover, if $\mathsf{OPT}_{\mathsf{cnwrs}}(I)<m^2/\mu^{21}$, then the probability that the algorithm returns FAIL is at most $O(1/\mu^2)$.*

We now complete the proof of Theorem 3.1 using Theorems 3.7, 3.8, and 3.9. Recall that we are given an instance $I=(G,\Sigma)$ of MCNwRS, with $\mu^{c''}\leq|E(G)|\leq m^*$, for some large enough constant c''. We assume that $c''>100c'$, where c' is the constant in Theorem 3.8. We use another large constant c'_g, and we assume that $c^*>c'_g>c''$, where c^* is the constant in the definition of the parameter μ. Throughout, we denote $m=|E(G)|$. We compute the desired collection $\mathcal{I}^*$ of instances in three steps.

Step 1. Assume first that the input instance I is a wide instance. We apply the algorithm from Theorem 3.7 to I. Let $\hat{\mathcal{I}}$ be the resulting collection of instances. We partition the set $\hat{\mathcal{I}}$ of instances into three subsets. The first set, denoted by $\hat{\mathcal{I}}_{\mathsf{small}}$, contains all instances $I'=(G',\Sigma')\in\hat{\mathcal{I}}$ with $|E(G')|\leq m/\mu$. The second set, denoted by $\hat{\mathcal{I}}^{(n)}_{\mathsf{large}}$, contains all narrow instances in $\hat{\mathcal{I}}\setminus\hat{\mathcal{I}}_{\mathsf{small}}$. The third set, denoted by $\hat{\mathcal{I}}^{(w)}_{\mathsf{large}}$, contains all remaining instances of $\hat{\mathcal{I}}$. From Theorem 3.7, every instance in $\hat{\mathcal{I}}^{(w)}_{\mathsf{large}}$ is wide and well-connected. Since every instance $I'=(G',\Sigma')\in\hat{\mathcal{I}}$ is a subinstance of

I, $|E(G')| \leq |E(G)| \leq m^*$ must hold. Recall that, from Theorem 3.7, $\hat{\mathcal{I}}$ is a ν_1-decomposition for I, for $\nu_1 = 2^{O((\log m)^{3/4}\log\log m)}$, so

$$\sum_{I'=(G',\Sigma')\in\hat{\mathcal{I}}} |E(G')| \leq m\cdot(\log m)^{c'_g}, \tag{1}$$

and $\mathbf{E}\left[\sum_{I'\in\hat{\mathcal{I}}}\mathsf{OPT}_{\mathsf{cnwrs}}(I')\right] \leq (\mathsf{OPT}_{\mathsf{cnwrs}}(I)+m)\cdot\nu_1$.

Bad Event $\mathcal{E}_1$. We say the bad event $\mathcal{E}_1$ happens, iff

$$\sum_{I'\in\hat{\mathcal{I}}}\mathsf{OPT}_{\mathsf{cnwrs}}(I') > 100(\mathsf{OPT}_{\mathsf{cnwrs}}(I)+m)\nu_1.$$

From the Markov Bound, $\mathbf{Pr}[\mathcal{E}_1] \leq 1/100$. Note that, if event $\mathcal{E}_1$ did not happen, then for each instance $I'\in\hat{\mathcal{I}}$, $\mathsf{OPT}_{\mathsf{cnwrs}}(I') \leq 100(\mathsf{OPT}_{\mathsf{cnwrs}}(I)+m)\nu_1$. We need the following simple observation, whose proof is deferred to the full version.

Observation 3.5. *Assume that $\mathsf{OPT}_{\mathsf{cnwrs}}(I) \leq m^2/\mu^{c''}$, and that Event $\mathcal{E}_1$ did not happen. Then for every instance $I'=(G',\Sigma')\in\hat{\mathcal{I}}^{(n)}_{large}\cup\hat{\mathcal{I}}^{(w)}_{large}$, $\mathsf{OPT}_{\mathsf{cnwrs}}(I') \leq |E(G')|^2/\mu^{c'}$.*

Assume now that instance I is a narrow instance. Then we simply set $\hat{\mathcal{I}}=\hat{\mathcal{I}}^{(n)}_{\mathsf{large}}=\{I\}$ and $\hat{\mathcal{I}}_{\mathsf{small}}=\hat{\mathcal{I}}^{(w)}_{\mathsf{large}}=\emptyset$. This completes the description of the first step.

Step 2. In the second step, we apply the algorithm from Theorem 3.8 to every instance $I'\in\hat{\mathcal{I}}^{(w)}_{\mathsf{large}}$. If the algorithm returns FAIL, then we terminate our algorithm and return FAIL as well. Assume now that the algorithm from Theorem 3.8, when applied to instance I', did not return FAIL. We let $\tilde{\mathcal{I}}(I')$ be the collection of instances that the algorithm computes. Recall that we are guaranteed that, for each instance $\tilde{I}=(\tilde{G},\tilde{\Sigma})\in\tilde{\mathcal{I}}(I')$, either $\tilde{I}$ is a narrow instance, or $|E(\tilde{G})| \leq \frac{|E(G')|}{\mu} \leq \frac{m}{\mu}$ (we have used the fact that $|E(G')|\leq m$, since $I'=(G',\Sigma')$ is a subinstance of I). Additionally, we are guaranteed that:

$$\sum_{\tilde{I}=(\tilde{G},\tilde{\Sigma})\in\tilde{\mathcal{I}}(I')}|E(\tilde{G})| \leq 2|E(G')|. \tag{2}$$

In particular, for every instance $\tilde{I}=(\tilde{G},\tilde{\Sigma})\in\tilde{\mathcal{I}}(I')$, $|E(\tilde{G})|\leq 2|E(G')|\leq 2m\leq 2m^*$.

We say that the application of the algorithm from Theorem 3.8 to an instance $I'=(G',\Sigma')\in\hat{\mathcal{I}}^{(w)}_{\mathsf{large}}$ is *successful*, if (i) the algorithm does not return FAIL; (ii) $\sum_{\tilde{I}\in\tilde{\mathcal{I}}(I')}\mathsf{OPT}_{\mathsf{cnwrs}}(\tilde{I}) \leq \mathsf{OPT}_{\mathsf{cnwrs}}(I')\cdot(\log m)^{c'_g}$; and (iii) there is an efficient algorithm, that we call $\mathsf{AlgCombineDrawings}'$, that, given a solution $\varphi(\tilde{I})$ to every instance $\tilde{I}\in\tilde{\mathcal{I}}(I')$, computes a solution $\varphi(I')$ to instance I' with $\mathsf{cr}(\varphi(I')) \leq \sum_{\tilde{I}\in\tilde{\mathcal{I}}(I')}\mathsf{cr}(\varphi(\tilde{I})) + \mathsf{OPT}_{\mathsf{cnwrs}}(I')\cdot\mu^{c'_g}$.

Bad Event $\mathcal{E}_2$. For an instance $I'=(G',\Sigma')\in\hat{\mathcal{I}}^{(w)}_{\mathsf{large}}$, we say that a bad event $\mathcal{E}_2(I')$ happens if the algorithm from Theorem 3.8, when applied to instance I', was not successful. From Theorem 3.8 and Observation 3.5, if $\mathsf{OPT}_{\mathsf{cnwrs}}(I)\leq m^2/\mu^{c''}$, then $\mathbf{Pr}[\mathcal{E}_2(I')\mid\neg\mathcal{E}_1]\leq 1/\mu^2$ (since we can assume that c'_g is a large enough constant). We let $\mathcal{E}_2$ be the bad event that at least one of the events in $\left\{\mathcal{E}_2(I')\mid I'\in\hat{\mathcal{I}}^{(w)}_{\mathsf{large}}\right\}$ happened. Recall that, from the definition of the set $\hat{\mathcal{I}}^{(w)}_{\mathsf{large}}$ of instances, for every instance $I'=(G',\Sigma')\in\hat{\mathcal{I}}^{(w)}_{\mathsf{large}}$, $|E(G')|\geq\frac{m}{\mu}$ holds. On the other hand, from Equation 1, $\sum_{I'=(G',\Sigma')\in\hat{\mathcal{I}}^{(w)}_{\mathsf{large}}}|E(G')| \leq \sum_{I'=(G',\Sigma')\in\hat{\mathcal{I}}}|E(G')| \leq m\cdot(\log m)^{c'_g}$. Therefore, we get that $|\hat{\mathcal{I}}^{(w)}_{\mathsf{large}}|\leq\mu\cdot(\log m)^{c'_g}$. From the Union Bound, if $\mathsf{OPT}_{\mathsf{cnwrs}}(I)\leq m^2/\mu^{c''}$, then $\mathbf{Pr}[\mathcal{E}_2\mid\neg\mathcal{E}_1]\leq\frac{\mu\cdot(\log m)^{c'_g}}{\mu^2}\leq\frac{1}{100}$. Let $\tilde{\mathcal{I}}=\bigcup_{I'\in\hat{\mathcal{I}}^{(w)}_{\mathsf{large}}}\tilde{\mathcal{I}}(I')$. Note that, from Inequalities 1 and 2, we get that:

$$\sum_{\tilde{I}=(\tilde{G},\tilde{\Sigma})\in\tilde{\mathcal{I}}}|E(\tilde{G})| \leq 2m\cdot(\log m)^{c'_g}. \tag{3}$$

We partition the instances in set $\tilde{\mathcal{I}}$ into two subsets: set $\tilde{\mathcal{I}}_{\mathsf{small}}$, containing all instances $\tilde{I}=(\tilde{G},\tilde{\Sigma})$ in $\tilde{\mathcal{I}}$ with $|E(\tilde{G})|\leq m/\mu$, and set $\tilde{\mathcal{I}}^{(n)}_{\mathsf{large}}$, containing all remaining instances. From Theorem 3.8, every instance $\tilde{I}\in\tilde{\mathcal{I}}^{(n)}_{\mathsf{large}}$ is narrow. This completes the description of the second step.

Step 3. We focus on four sets of instances that we have constructed so far: $\hat{\mathcal{I}}_{\mathsf{small}}, \hat{\mathcal{I}}^{(n)}_{\mathsf{large}}, \tilde{\mathcal{I}}_{\mathsf{small}}, \tilde{\mathcal{I}}^{(n)}_{\mathsf{large}}$. Recall that, if instance $I'=(G',\Sigma')$ belongs to set $\hat{\mathcal{I}}_{\mathsf{small}}\cup\tilde{\mathcal{I}}_{\mathsf{small}}$, then $|E(G')|\leq m/\mu$. If instance $I'=(G',\Sigma')$ belongs to set $\hat{\mathcal{I}}^{(n)}_{\mathsf{large}}\cup\tilde{\mathcal{I}}^{(n)}_{\mathsf{large}}$, then $m/\mu<|E(G')|\leq 2m$, and instance I' is narrow. We use the following simple observation, whose proof is deferred to the full version.

Observation 3.6. *If $\mathsf{OPT}_{\mathsf{cnwrs}}(I)\leq m^2/\mu^{c''}$, and neither of the events $\mathcal{E}_1,\mathcal{E}_2$ happened, then for every instance $I'=(G',\Sigma')\in\hat{\mathcal{I}}^{(n)}_{large}\cup\tilde{\mathcal{I}}^{(n)}_{large}$, $\mathsf{OPT}_{\mathsf{cnwrs}}(I')<|E(G')|^2/\mu^{21}$.*

Next, we process every instance $I'\in\hat{\mathcal{I}}^{(n)}_{\mathsf{large}}\cup\tilde{\mathcal{I}}^{(n)}_{\mathsf{large}}$ one by one. Notice that for each such instance $I'=(G',\Sigma')$, $|E(G')|\geq m/\mu\geq\mu^{50}$ must hold, since $m\geq\mu^{c''}$. Additionally, as observed already, $|E(G')|\leq 2m\leq 2m^*$. When instance $I'=(G',\Sigma')$ is processed, we apply the algorithm from Theorem 3.9 to it. If the algorithm returns FAIL, then we terminate the algorithm and return FAIL as well. Otherwise, we obtain a collection $\overline{\mathcal{I}}(I')$ of instances of MCNwRS. From Theorem 3.9, for every instance $I''=(G'',\Sigma'')\in\overline{\mathcal{I}}(I')$, $|E(G'')|\leq\frac{|E(G')|}{2\mu}\leq\frac{m}{\mu}$. Moreover, from the definition of a ν-decomposition of an instance, and from the fact that $|E(G')|\leq 2m$,

$$\sum_{I''=(G'',\Sigma'')\in\overline{\mathcal{I}}(I')}|E(G'')| \leq |E(G')|\cdot(\log m)^{c'_g}. \tag{4}$$

Bad Events $\mathcal{E}_3$ and $\mathcal{E}$. For an instance $I'=(G',\Sigma')\in\hat{\mathcal{I}}^{(n)}_{\mathsf{large}}\cup\tilde{\mathcal{I}}^{(n)}_{\mathsf{large}}$, we say that the event $\mathcal{E}_3(I')$ happens if the algorithm from Theorem 3.9, when applied to instance I', returns FAIL. From Theorem 3.9, if $\mathsf{OPT}_{\mathsf{cnwrs}}(I')<|E(G')|^2/\mu^{21}$, then the algorithm returns FAIL with probability $O(1/\mu^2)$. Therefore, from Observation 3.6, if $\mathsf{OPT}_{\mathsf{cnwrs}}(I)\leq m^2/\mu^{c''}$, then $\mathbf{Pr}[\mathcal{E}_3(I')\mid\neg\mathcal{E}_1\wedge\neg\mathcal{E}_2]\leq O(1/\mu^2)$. We let $\mathcal{E}_3$ to be the bad event that $\mathcal{E}_3(I')$ happened for any instance $I'\in\hat{\mathcal{I}}^{(n)}_{\mathsf{large}}\cup\tilde{\mathcal{I}}^{(n)}_{\mathsf{large}}$. Recall that, for every instance $I'=(G',\Sigma')\in\hat{\mathcal{I}}^{(n)}_{\mathsf{large}}\cup\tilde{\mathcal{I}}^{(n)}_{\mathsf{large}}$, $|E(G')|\geq\frac{m}{\mu}$. On the other hand, from Inequality 1, $\sum_{I'=(G',\Sigma')\in\hat{\mathcal{I}}^{(n)}_{\mathsf{large}}}|E(G')| \leq m\cdot(\log m)^{c'_g}$, and from

Inequality 3, $\sum_{I'=(G',\Sigma')\in\tilde{\mathcal{I}}^{(n)}_{\text{large}}}|E(G')|\leq 2m\cdot(\log m)^{c'_g}$. Therefore, $|\hat{\mathcal{I}}^{(n)}_{\text{large}}\cup\tilde{\mathcal{I}}^{(n)}_{\text{large}}|\leq 3\mu\cdot(\log m)^{c'_g}$. From the Union Bound, assuming that the constant c^* in the definition of the parameter μ is large enough, if $\mathsf{OPT}_{\mathsf{cnwrs}}(I)\leq m^2/\mu^{c''}$, then $\mathbf{Pr}\left[\mathcal{E}_3\mid\neg\mathcal{E}_1\wedge\neg\mathcal{E}_2\right]\leq O\left(\frac{\mu\cdot(\log m)^{c'_g}}{\mu^2}\right)\leq\frac{1}{100}$. Lastly, we define bad event $\mathcal{E}$ to be the event that at least one of the events $\mathcal{E}_1,\mathcal{E}_2,\mathcal{E}_3$ happened. Note that $\mathbf{Pr}\left[\mathcal{E}\right]\leq\mathbf{Pr}\left[\mathcal{E}_1\right]+\mathbf{Pr}\left[\mathcal{E}_2\mid\neg\mathcal{E}_1\right]+\mathbf{Pr}\left[\mathcal{E}_3\mid\neg\mathcal{E}_1\wedge\neg\mathcal{E}_2\right]$. Therefore, altogether, if $\mathsf{OPT}_{\mathsf{cnwrs}}(I)\leq m^2/\mu^{c''}$, then $\mathbf{Pr}\left[\mathcal{E}\right]\leq\frac{3}{100}\leq\frac{1}{30}$. Note that, if bad event $\mathcal{E}$ does not happen, then the algorithm does not return FAIL.

If the third step of the algorithm did not terminate with a FAIL, we let $\overline{\mathcal{I}}_{\text{small}}=\bigcup_{I'\in\hat{\mathcal{I}}^{(n)}_{\text{large}}\cup\tilde{\mathcal{I}}^{(n)}_{\text{large}}}\overline{\mathcal{I}}(I')$. By combining Inequalities 1, 3 and 4, we get that:

$$\sum_{I''=(G'',\Sigma'')\in\overline{\mathcal{I}}_{\text{small}}}|E(G'')|\leq 3m\cdot(\log m)^{2c'_g}. \tag{5}$$

The output of the algorithm is the collection $\mathcal{I}^*=\hat{\mathcal{I}}_{\text{small}}\cup\tilde{\mathcal{I}}_{\text{small}}\cup\overline{\mathcal{I}}_{\text{small}}$ of instances of MCNwRS. From the above discussion, for every instance $I''=(G'',\Sigma'')\in\mathcal{I}^*$, $|E(G'')|\leq m/\mu$. As discussed already, if bad event $\mathcal{E}$ does not happen, then the algorithm does not return FAIL.

From now on we assume that the algorithm did not return FAIL. From Inequalities 1, 3 and 5, $\sum_{I''=(G'',\Sigma'')\in\mathcal{I}^*}|E(G'')|\leq 6m\cdot(\log m)^{2c'_g}$.

Next, we provide Algorithm AlgCombineDrawings in the following claim, whose proof is conceptually straightforward but somewhat technical, and is deferred to the full version.

Claim 3.2. *There is an efficient algorithm* AlgCombineDrawings, *that, given a solution $\varphi(I'')$ to every instance $I''\in\mathcal{I}^*$, computes a solution $\varphi(I)$ to instance I. Moreover, if $\mathsf{OPT}_{\mathsf{cnwrs}}(I)\leq m^2/\mu^{c''}$, and event $\mathcal{E}$ did not happen, then $\mathsf{cr}(\varphi(I))\leq O(\sum_{I''\in\mathcal{I}^*}\mathsf{cr}(\varphi(I'')))+(\mathsf{OPT}_{\mathsf{cnwrs}}(I)+m)\cdot\mu^{O(1)}$.*

The following observation, whose proof is deferred to the full version, will complete the proof of Theorem 3.1.

Observation 3.7. *If $\mathsf{OPT}_{\mathsf{cnwrs}}(I)\leq|E(G)|^2/\mu^{c'}$ and bad event $\mathcal{E}$ did not happen, then for some constant c, with probability at least 0.99: $\sum_{I''\in\mathcal{I}^*}\mathsf{OPT}_{\mathsf{cnwrs}}(I'')\leq(\mathsf{OPT}_{\mathsf{cnwrs}}(I)+m)\cdot 2^{c(\log m)^{3/4}\log\log m}$.*

We define $\mathcal{E}'$ to be the bad event that $\sum_{I''\in\mathcal{I}^*}\mathsf{OPT}_{\mathsf{cnwrs}}(I'')>(\mathsf{OPT}_{\mathsf{cnwrs}}(I)+m)\cdot 2^{c(\log m)^{3/4}\log\log m}$. Clearly, if $\mathsf{OPT}_{\mathsf{cnwrs}}(I)\leq m^2/\mu^{c''}$, then the probability that either of the events $\mathcal{E}$ or $\mathcal{E}'$ happens is at most $\mathbf{Pr}\left[\mathcal{E}\right]+\mathbf{Pr}\left[\mathcal{E}'\mid\neg\mathcal{E}\right]\leq 1/16$. Therefore, we conclude that, if $\mathsf{OPT}_{\mathsf{cnwrs}}(I)\leq m^2/\mu^{c''}$, then with probability at least 15/16, all of the following hold: (i) the algorithm does not return FAIL; (ii) $\mathcal{I}^*\neq\emptyset$; (iii) $\sum_{I''\in\mathcal{I}^*}\mathsf{OPT}_{\mathsf{cnwrs}}(I'')\leq(\mathsf{OPT}_{\mathsf{cnwrs}}(I)+m)\cdot 2^{O((\log m)^{3/4}\log\log m)}$; and (iv) if algorithm AlgCombineDrawings is given as input a solution $\varphi(I'')$ to every instance $I''\in\mathcal{I}^*$, then the resulting solution φ to instance I that it computes has cost at most: $O\left(\sum_{I''\in\mathcal{I}^*}\mathsf{cr}(\varphi(I''))\right)+(\mathsf{OPT}_{\mathsf{cnwrs}}(I)+m)\cdot\mu^{O(1)}$. This concludes the proof of Theorem 3.1 from Theorems 3.7, 3.8, and 3.9.

REFERENCES

[1] M. Ajtai, V. Chvátal, M. Newborn, and E. Szemerédi. 1982. Crossing-free subgraphs. *Theory and Practice of Combinatorics* (1982), 9–12.

[2] Christoph Ambuhl, Monaldo Mastrolilli, and Ola Svensson. 2007. Inapproximability results for sparsest cut, optimal linear arrangement, and precedence constrained scheduling. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS'07)*. IEEE, 329–337.

[3] Sanjeev Arora, Satish Rao, and Umesh V. Vazirani. 2009. Expander flows, geometric embeddings and graph partitioning. *J. ACM* 56, 2 (2009).

[4] Sergio Cabello. 2013. Hardness of approximation for crossing number. *Discrete & Computational Geometry* 49, 2 (2013), 348–358.

[5] Chandra Chekuri and Anastasios Sidiropoulos. 2013. Approximation algorithms for Euler genus and related problems. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*. IEEE, 167–176.

[6] Markus Chimani and Petr Hliněný. 2011. A tighter insertion-based approximation of the crossing number. In *International Colloquium on Automata, Languages, and Programming*. Springer, 122–134.

[7] Julia Chuzhoy. 2011. An algorithm for the graph crossing number problem. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*. ACM, 303–312.

[8] Julia Chuzhoy, Vivek Madan, and Sepideh Mahabadi. 2016. In *Private Communication*.

[9] Julia Chuzhoy, Sepideh Mahabadi, and Zihan Tan. 2020. Towards Better Approximation of Graph Crossing Number. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 73–84.

[10] Julia Chuzhoy, Yury Makarychev, and Anastasios Sidiropoulos. 2011. On graph crossing number and edge planarization. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete algorithms*. SIAM, 1050–1069.

[11] Julia Chuzhoy and Zihan Tan. 2019. Towards tight (er) bounds for the excluded grid theorem. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 1445–1464.

[12] Guy Even, Sudipto Guha, and Baruch Schieber. 2002. Improved approximations of crossings in graph drawings and VLSI layout areas. *SIAM J. Comput.* 32, 1 (2002), 231–252.

[13] M. R. Garey and D. S. Johnson. 1983. Crossing number is NP-complete. *SIAM J. Algebraic Discrete Methods* 4, 3 (1983), 312–316.

[14] P. Hlinený. 2006. Crossing number is hard for cubic graphs. *J. Comb. Theory, Ser. B* 96, 4 (2006), 455–471.

[15] John Hopcroft and Robert Tarjan. 1974. Efficient planarity testing. *Journal of the ACM (JACM)* 21, 4 (1974), 549–568.

[16] Ken-ichi Kawarabayashi and Anastasios Sidiropoulos. 2017. Polylogarithmic Approximation for Minimum Planarization (Almost). In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*. 779–788. https://doi.org/10.1109/FOCS.2017.77

[17] Ken-ichi Kawarabayashi and Anastasios Sidiropoulos. 2019. Polylogarithmic approximation for Euler genus on bounded degree graphs. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*. ACM, 164–175.

[18] F. T. Leighton. 1983. *Complexity issues in VLSI: optimal layouts for the shuffle-exchange graph and other networks*. MIT Press.

[19] Tom Leighton and Satish Rao. 1999. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *Journal of the ACM (JACM)* 46, 6 (1999), 787–832.

[20] J. Matoušek. 2002. *Lectures on discrete geometry*. Springer-Verlag.

[21] J. Pach and G. Tóth. 2000. Thirteen problems on crossing numbers. *Geombinatorics* 9, 4 (2000), 194–207.

[22] Michael J Pelsmajer, Marcus Schaefer, and Daniel Štefankovič. 2011. Crossing numbers of graphs with rotation systems. *Algorithmica* 60, 3 (2011), 679–702.

[23] R. B. Richter and G. Salazar. 2009. Crossing numbers. In *Topics in Topological Graph Theory*, L. W. Beineke and R. J. Wilson (Eds.). Cambridge University Press, Chapter 7, 133–150.

[24] Neil Robertson, Paul Seymour, and Robin Thomas. 1994. Quickly excluding a planar graph. *Journal of Combinatorial Theory, Series B* 62, 2 (1994), 323–348.

[25] Neil Robertson and Paul D Seymour. 1986. Graph minors. V. Excluding a planar graph. *Journal of Combinatorial Theory, Series B* 41, 1 (1986), 92–114.

[26] Marcus Schaefer. 2012. The graph crossing number and its variants: A survey. *The electronic journal of combinatorics* (2012), DS21–Sep.

[27] Anastasios Sidiropoulos. 2010. Personal communication. (2010).

[28] P. Turán. 1977. A note of welcome. *J. Graph Theory* 1 (1977), 1–5.