Node Max-Cut and Computing Equilibria
in Linear Weighted Congestion Games.

Dimitris Fotakis[1], Vardis Kandiros[2], Thanasis Lianeas[1], Nikos Mouzakis[1], Panagiotis Patsilinakos[1], and Stratis Skoulakis[3]

[1] National Technical University of Athens
{*fotakis,lianeas,nmouzakis,patsilinak*}*@corelab.ntua.gr*
[2]Massachussets Institute of Technology
kandiros@mit.edu
[3]Singapore Institute of Technology and Design
stratis.skoulakis@gmail.com

Abstract

In this work, we seek a more refined understanding of the complexity of local optimum computation for Max-Cut and pure Nash equilibrium (PNE) computation for congestion games with weighted players and linear latency functions. We show that computing a PNE of linear weighted congestion games is PLS-complete either for very restricted strategy spaces, namely when player strategies are paths on a series-parallel network with a single origin and destination, or for very restricted latency functions, namely when the latency on each resource is equal to the congestion. Our results reveal a remarkable gap regarding the complexity of PNE in congestion games with weighted and unweighted players, since in case of unweighted players, a PNE can be easily computed by either a simple greedy algorithm (for series-parallel networks) or any better response dynamics (when the latency is equal to the congestion). For the latter of the results above, we need to show first that computing a local optimum of a natural restriction of Max-Cut, which we call *Node-Max-Cut*, is PLS-complete. In Node-Max-Cut, the input graph is vertex-weighted and the weight of each edge is equal to the product of the weights of its endpoints. Due to the very restricted nature of Node-Max-Cut, the reduction requires a careful combination of new gadgets with ideas and techniques from previous work. We also show how to compute efficiently a $(1 + \varepsilon)$-approximate equilibrium for Node-Max-Cut, if the number of different vertex weights is constant.

1 Introduction

Motivated by the remarkable success of local search in combinatorial optimization, Johnson et al. introduced [25] the complexity class Polynomial Local Search (PLS), consisting of local search problems with polynomially verifiable local optimality. PLS includes many natural complete problems (see e.g., [28, App. C]), with CIRCUIT-FLIP [25] and MAX-CUT [33] among the best known ones, and lays the foundation for a principled study of the complexity of local optima computation. In the last 15 years, a significant volume of research on PLS-completeness was motivated by the problem of computing a pure Nash equilibrium of potential games (see e.g., [1, 34, 20] and the

references therein), where any improving deviation by a single player decreases a potential function and its local optima correspond to pure Nash equilibria [29].

Computing a local optimum of Max-Cut under the flip neighborhood (a.k.a. Local-Max-Cut) has been one of the most widely studied problems in PLS. Given an edge-weighed graph, a cut is locally optimal if we cannot increase its weight by moving a vertex from one side of the cut to the other. Since its PLS-completeness proof by Schäffer and Yannakakis [33], researchers have shown that Local-Max-Cut remains PLS-complete for graphs with maximum degree five [9], is polynomially solvable for cubic graphs [31], and its smoothed complexity is either polynomial in complete [2] and sparse [9] graphs, or almost polynomial in general graphs [7, 10]. Moreover, due to its simplicity and versatility, Max-Cut has been widely used in PLS reductions (see e.g., [1, 20, 34]). Local-Max-Cut can also be cast as a game, where each vertex aims to maximize the total weight of its incident edges that cross the cut. Cut games are potential games (the value of the cut is the potential function), which has motivated research on efficient computation of approximate equilibria for Local-Max-Cut [3, 6]. To the best of our knowledge, apart from the work on the smoothed complexity of Local-Max-Cut (and may be that Local-Max-Cut is P-complete for unweighted graphs [33, Theorem 4.5]), there has not been any research on whether (and to which extent) additional structure on edge weights affects hardness of Local-Max-Cut.

A closely related research direction deals with the complexity of computing a pure Nash equilibrium (equilibrium or PNE, for brevity) of *congestion games* [32], a typical example of potential games [29] and among the most widely studied classes of games in Algorithmic Game Theory (see e.g., [15] for a brief account of previous work). In congestion games (or CGs, for brevity), a finite set of players compete over a finite set of resources. Strategies are resource subsets and players aim to minimize the total cost of the resources in their strategies. Each resource e is associated with a (non-negative and non-decreasing) latency function, which determines the cost of using e as a function of e's *congestion* (i.e., the number of players including e in their strategy). Researchers have extensively studied the properties of special cases and variants of CGs. Most relevant to this work are *symmetric* (resp. *asymmetric*) CGs, where players share the same strategy set (resp. have different strategy sets), *network* CGs, where strategies correspond to paths in an underlying network, and *weighted* CGs, where player contribute to the congestion with a different weight.

Fabrikant et al. [12] proved that computing a PNE of asymmetric network CGs or symmetric CGs is PLS-complete, and that it reduces to min-cost-flow for symmetric network CGs. About the same time [17, 30] proved that weighted congestion games admit a (weighted) potential function, and thus a PNE, if the latency functions are either affine or exponential (and [23, 24] proved that in a certain sense, this restriction is necessary). Subsequently, Ackermann et al. [1] characterized the strategy sets of CGs that guarantee efficient equilibrium computation. They also used a variant of Local-Max-Cut, called *threshold games*, to simplify the PLS-completeness proof of [12] and to show that computing a PNE of asymmetric network CGs with (exponentially steep) linear latencies is PLS-complete.

On the other hand, the complexity of equilibrium computation for weighted CGs is not well understood. All the hardness results above carry over to weighted CGs, since they generalize standard CGs (where the players have unit weight). But on the positive side, we only know how to efficiently compute a PNE for weighted CGs on parallel links with general latencies [16] and for weighted CGs on parallel links with identity latency functions and asymmetric strategies [19]. Despite the significant interest in (exact or approximate) equilibrium computation for CGs (see e.g., [5, 6, 27] and the references therein), we do not understand how (and to which extent)

1

the complexity of equilibrium computation is affected by player weights. This is especially true for weighted CGs with linear latencies, which admit a potential function and their equilibrium computation is in PLS.

Contributions. We contribute to both research directions outlined above. In a nutshell, we show that equilibrium computation in linear weighted CGs is significantly harder than for standard CGs, in the sense that it is PLS-complete either for very restricted strategy spaces, namely when player strategies are paths on a series-parallel network with a single origin and destination, or for very restricted latency functions, namely when resource costs are equal to the congestion. Our main step towards proving the latter result is to show that computing a local optimum of NODE-MAX-CUT, a natural and interesting restriction of MAX-CUT where the weight of each edge is the product of the weights of its endpoints, is PLS-complete.

More specifically, using a tight reduction from LOCAL-MAX-CUT, we first show, in Section 3.1, that equilibrium computation for linear weighted CGs on series-parallel networks with a single origin and destination is PLS-complete (Theorem 1). The reduction results in games where both the player weights and the latency slopes are exponential. Our result reveals a remarkable gap between weighted and standard CGs regarding the complexity of equilibrium computation, since for standard CGs on series-parallel networks with a single origin and destination, a PNE can be computed by a simple greedy algorithm [18].

Aiming at a deeper understanding of how different player weights affect the complexity of equilibrium computation in CGs, we show, in Section 3.2, that computing a PNE of weighted network CGs with asymmetric player strategies and identity latency functions is PLS-complete (Theorem 2). Again the gap to standard CGs is remarkable, since for standard CGs with identity latency functions, any better response dynamics converges to a PNE in polynomial time. In the reduction of Theorem 2, NODE-MAX-CUT plays a role similar to that of threshold games in [1, Sec. 4]. The choice of NODE-MAX-CUT seems necessary, in the sense that known PLS reductions, starting from NOT-ALL-EQUAL SATISFIABILITY [12] or LOCAL-MAX-CUT [1], show that equilibrium computation is hard due to the interaction of players on different resources (where latencies simulate the edge / clause weights), while in our setting, equilibrium computation is hard due to the player weights, which are the same for all resources in a player's strategy.

NODE-MAX-CUT is a natural restriction of MAX-CUT and settling the complexity of its local optima computation may be of independent interest, both conceptually and technically. NODE-MAX-CUT coincides with the restriction of MAX-CUT shown (weakly) NP-complete on complete graphs in the seminal paper of Karp [26], while a significant generalization of NODE-MAX-CUT with polynomial weights was shown P-complete in [33].

A major part of our technical effort concerns reducing CIRCUIT-FLIP to NODE-MAX-CUT, thus showing that computing a local optimum of NODE-MAX-CUT is PLS-complete (Section 5). Since NODE-MAX-CUT is a very restricted special case of MAX-CUT we have to start from a PLS-complete problem lying before LOCAL-MAX-CUT on the "reduction paths" of PLS. The reduction is technically involved, due to the very restricted nature of the problem. In NODE-MAX-CUT, every vertex contributes to the cut value of its neighbors with the same weight, and differentiation comes only as a result of the different total weight in the neighborhood of each vertex. To deal with this restriction, we combine some new carefully contstructed gadgets with the gadgets used by Schäffer and Yannakakis [33], Elsässer and Tscheuschner [9]. and Gairing and Savani [20]. In general, as a very resctricted special case of MAX-CUT, NODE-MAX-CUT is a natural and convenient starting point for future PLS reductions, especially when one wants to show hardness

of equilibrium computation for restricted classes of games that admit weighted potential functions (e.g., as that in [17]). So, our results may serve as a first step towards a better understanding of the complexity of (exact or approximate) equilibrium computation for weighted potential games.

We also show that a $(1 + \varepsilon)$-approximate equilibrium for NODE-MAX-CUT, where no vertex can switch sides and increase the weight of its neighbors across the cut by a factor larger than $1 + \varepsilon$, can be computed in time exponential in the number of different weights (see Theorem 3 for a precise statement). Thus, we can efficiently compute a $(1 + \varepsilon)$-approximate equilibrium for NODE-MAX-CUT, for any $\varepsilon > 0$, if the number of different vertex weights is constant. Since similar results are not known for MAX-CUT, we believe that Theorem 3 may indicate that approximate equilibrium computation for NODE-MAX-CUT may not be as hard as for MAX-CUT. An interesting direction for further research is to investigate (i) the quality of efficiently computable approximate equilibria for NODE-MAX-CUT; and (ii) the smoothed complexity of its local optima.

Related Work. Existence and efficient computation of (exact or approximate) equilibria for weighted congestion games have received significant research attention. We briefly discuss here some of the most relevant previous work. There has been significant research interest in the convergence rate of best response dynamics for weighted congestion games (see e.g., [8, 4, 11, 13, 22]). Gairing et al. [19] presented a polynomial algorithm for computing a PNE for load balancing games on restricted parallel links. Caragiannis et al. [6] established existence and presented efficient algorithms for computing approximate PNE in weighted CGs with polynomial latencies (see also [14, 21]).

Bhalgat et al. [3] presented an efficient algorithm for computing a $(3 + \varepsilon)$-approximate equilibrium in MAX-CUT games, for any $\varepsilon > 0$. The approximation guarantee was improved to $2 + \varepsilon$ in [6]. We highlight that the notion of approximate equilibrium in cut games is much stronger than the notion of approximate local optimum of MAX-CUT, since the former requires that no vertex can significantly improve the total weight of its incidence edges that cross the cut (as e.g., in [3, 6]), while the latter simply requires that the total weight of the cut cannot be significantly improved (as e.g., in [6]).

Johnson et al. [25] introduced the complexity class PLS and proved that CIRCUIT-FLIP is PLS-complete. Subsequently, Schäffer and Yannakakis [33] proved that MAX-CUT is PLS-complete. From a technical viewpoint, our work is close to previous work by Elsässer and Tscheuschner [9] and Gairing and Savani [20], where they show that LOCAL-MAX-CUT in graphs of maximum degree five [9] and computing a PNE for hedonic games [20] are PLS-complete, and by Ackermann et al. [1], where they reduce LOCAL-MAX-CUT to computing a PNE in network congestion games.

2 Basic Definitions and Notation

Polynomial-Time Local Search (PLS). A *polynomial-time local search* (PLS) problem L [25, Sec. 2] is specified by a (polynomially recognizable) set of instances I_L, a set $S_L(x)$ of feasible solutions for each instance $x \in I_L$, with $|s| = O(\text{poly}(|x|))$ for every solution $s \in S_L(x)$, an objective function $f_L(s, x)$ that maps each solution $s \in S_L(x)$ to its value in instance x, and a *neighborhood* $N_L(s, x) \subseteq S_L(x)$ of feasible solutions for each $s \in S_L(x)$. Moreover, there are three polynomial-time algorithms that for any given instance $x \in I_L$: (i) the first generates an initial solution $s_0 \in S_L(x)$; (ii) the second determines whether a given s is a feasible solution and (if $s \in S_L(x)$) computes its objective value $f_L(s, x)$; and (iii) the third returns either that s is *locally optimal* or a feasible solution $s' \in N_L(s, x)$ with better objective value than s. If L is a maximization (resp. minimization) problem, a

solution s is locally optimal if for all $s' \in N_L(s, x)$, $f_L(s, x) \geq f_L(s', x)$ (resp. $f_L(s, x) \leq f_L(s', x)$). If s is not locally optimal, the third algorithm returns a solution $s' \in N_L(s, x)$ with $f(s, x) < f(s', x)$ (resp. $f(s, x) > f(s', x)$). The complexity class PLS consists of all polynomial-time local search problems. By abusing the terminology, we always refer to polynomial-time local search problem simply as local search problems.

PLS Reductions and Completeness. A local search problem L is PLS-*reducible* to a local search problem L', if there are polynomial-time algorithms ϕ_1 and ϕ_2 such that (i) ϕ_1 maps any instance $x \in I_L$ of L to an instance $\phi_1(x) \in I_{L'}$ of L'; (ii) ϕ_2 maps any (solution s' of instance $\phi_1(x)$, instance x) pair, with $s' \in S_{L'}(\phi_1(x))$, to a solution $s \in S_L(x)$; and (iii) for every instance $x \in I_L$, if s' is locally optimal for $\phi_1(x)$, then $\phi_2(s', x)$ is locally optimal for x.

By definition, if a local search problem L is PLS-reducible to a local search problem L', a polynomial-time algorithm that computes a local optimum of L' implies a polynomial time algorithm that computes a local optimum of L. Moreover, a PLS-reduction is transitive. As usual, a local search problem Q is PLS-*complete*, if $Q \in$ PLS and any local search problem $L \in$ PLS is PLS-reducible to Q.

Max-Cut and Node-Max-Cut. An instance of MAX-CUT consists of an undirected edge-weighted graph $G(V, E)$, where V is the set of vertices and E is the set of edges. Each edge e is associated with a positive weight w_e. A cut of G is a vertex partition $(S, V \setminus S)$, with $\emptyset \neq S \neq V$. We usually identify a cut with one of its sides (e.g., S). We denote $\delta(S) = \{\{u, v\} \in E : u \in S \wedge v \notin S\}$ the set of edges that cross the cut S. The weight (or the value) of a cut S, denoted $w(S)$, is $w(S) = \sum_{e \in \delta(S)} w_e$. In MAX-CUT, the goal is to compute an optimal cut S^* of maximum value $w(S^*)$.

In NODE-MAX-CUT, each vertex v is associated with a positive weight w_v and the weight of each edge $e = \{u, v\}$ is $w_e = w_u w_v$, i.e. equal to the product of the weights of e's endpoints. Again the goal is to compute a cut S^* of maximum value $w(S^*)$. As optimization problems, both MAX-CUT and NODE-MAX-CUT are NP-complete [26].

In this work, we study MAX-CUT and NODE-MAX-CUT as local search problems under the FLIP neighborhood. Then, they are referred to as LOCAL-MAX-CUT and LOCAL-NODE-MAX-CUT. The neighborhood $N(S)$ of a cut $(S, V \setminus S)$ consists of all cuts $(S', V \setminus S')$ where S and S' differ by a single vertex. Namely, the cut S' is obtained from S by moving a vertex from one side of the cut to the other. A cut S is locally optimal if for all $S' \in N(S)$, $w(S) \geq w(S')$. In LOCAL-MAX-CUT (resp. LOCAL-NODE-MAX-CUT), given an edge-weighted (resp. vertex-weighted) graph, the goal is to compute a locally optimal cut. Clearly, both MAX-CUT and NODE-MAX-CUT belong to PLS. In the following, we abuse the terminology and refer to LOCAL-MAX-CUT and LOCAL-NODE-MAX-CUT as MAX-CUT and NODE-MAX-CUT, for brevity, unless we need to distinguish between the optimization and the local search problem.

Weighted Congestion Games. A *weighted congestion game* $\mathcal{G}$ consists of n players, where each player i is associated with a positive weight w_i, a set of resources E, where each resource e is associated with a non-decreasing latency function $\ell_e : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$, and a non-empty strategy set $\Sigma_i \subseteq 2^E$ for each player i. A game is linear if $\ell_e(x) = a_e x + b_e$, for some $a_e, b_e \geq 0$, for all $e \in E$. The identity latency function is $\ell(x) = x$. The player strategies are *symmetric*, if all players share the same strategy set Σ, and *asymmetric*, otherwise.

We focus on *network* weighted congestion games, where the resources E correspond to the edges of an underlying network $G(V, E)$ and the player strategies are paths on G. A network game is *single-commodity*, if G has an origin o and a destination d and the player strategies are all (simple)

$o - d$ paths. A network game is *multi-commodity*, if G has an origin o_i and a destination d_i for each player i, and i's strategy set Σ_i consists of all (simple) $o_i - d_i$ paths. A single-commodity network $G(V, E)$ is *series-parallel*, if it either consists of a single edge (o, d) or can be obtained from two series-parallel networks composed either in series or in parallel (see e.g., [35] for details on composition and recognition of series-parallel networks).

A configuration $\vec{s} = (s_1, \ldots, s_n)$ consists of a strategy $s_i \in \Sigma_i$ for each player i. The congestion s_e of resource e in configuration $\vec{s}$ is $s_e = \sum_{i: e \in s_i} w_i$. The cost of resource e in $\vec{s}$ is $\ell_e(s_e)$. The *individual cost* (or *cost*) $c_i(\vec{s})$ of player i in configuration $\vec{s}$ is the total cost for the resources in her strategy s_i, i.e., $c_i(\vec{s}) = \sum_{e \in s_i} \ell_e(s_e)$. A configuration $\vec{s}$ is a *pure Nash equilibrium* (equilibrium or PNE, for brevity), if for every player i and every strategy $s' \in \Sigma_i$, $c_i(\vec{s}) \leq c_i(\vec{s}_{-i}, s')$ (where $(\vec{s}_{-i}, s')$ denotes the configuration obtained from $\vec{s}$ by replacing s_i with s'). Namely, no player can improve her cost by unilaterally switching her strategy.

Equilibrium Computation and Local Search. [17] shows that for linear weighted congestion games, with latencies $\ell_e(x) = a_e x + b_e$, $\Phi(\vec{s}) = \sum_{e \in E}(a_e s_e^2 + b_e s_e) + \sum_i w_i \sum_{e \in s_i}(a_e w_i + b_e)$ changes by $2w_i(c_i(\vec{s}) - c_i(\vec{s}_{-i}, s'))$, when a player i switches from strategy s_i to strategy s' in $\vec{s}$. Hence, Φ is a weighted potential function, whose local optimal (wrt. single player deviations) correspond to PNE of the underlying game. Hence, equilibrium computation for linear weighted congestion games is in PLS. Specifically, configurations corresponds to solutions, the neighborhood $N(\vec{s})$ of a configuration $\vec{s}$ consists of all configurations $(\vec{s}_{-i}, s')$ with $s' \in \Sigma_i$, for some player i, and local optimality is defined wrt. the potential function Φ.

Max-Cut and Node-Max-Cut as Games. Local-Max-Cut and Local-Node-Max-Cut can be cast as cut games, where players correspond to vertices of $G(V, E)$, strategies $\Sigma = \{0, 1\}$ are symmetric, and configurations $\vec{s} \in \{0, 1\}^{|V|}$ correspond to cuts, e.g., $S(\vec{s}) = \{v \in V : s_v = 0\}$. Each player v aims to maximize $w_v(\vec{s}) = \sum_{e = \{u, v\} \in E: s_u \neq s_v} w_e$, that is the total weight of her incident edges that cross the cut. For Node-Max-Cut, this becomes $w_v(\vec{s}) = \sum_{u: \{u, v\} \in E \wedge s_u \neq s_v} w_u$, i.e., v aims to maximize the total weight of her neighbors across the cut. A cut $\vec{s}$ is a PNE if for all players v, $w_v(\vec{s}) \geq w_v(\vec{s}_{-i}, 1 - s_v)$. Equilibrium computation for cut games is equivalent to local optimum computation, and thus, is in PLS.

A cut $\vec{s}$ is a $(1 + \varepsilon)$-approximate equilibrium, for some $\varepsilon > 0$, if for all players v, $(1 + \varepsilon) w_v(\vec{s}) \geq w_v(\vec{s}_{-i}, 1 - s_v)$. Note that the notion of $(1 + \varepsilon)$-approximate equilibrium is stronger than the notion of $(1 + \varepsilon)$-approximate local optimum, i.e., a cut S such that for all $S' \in N(S)$, $(1 + \varepsilon) w(S) \geq w(S')$ (see also the discussion in [6]).

3 Hardness of Computing Equilibria in Weighted Congestion Games

We next show that computing a PNE in weighted congestion games with linear latencies is PLS-complete either for single-commodity series-parallel networks or for multi-commodity networks with identity latency functions. Missing technical details can be found in Appendix A.

3.1 Weighted Congestion Games on Series-Parallel Networks

Theorem 1. *Computing a pure Nash equilibrium in weighted congestion games on single-commodity series-parallel networks with linear latency functions is PLS-complete.*

Proof sketch. Membership in PLS follows from the potential function argument of [17]. To show hardness, we present a reduction from Max-Cut (see also Appendix A.1).
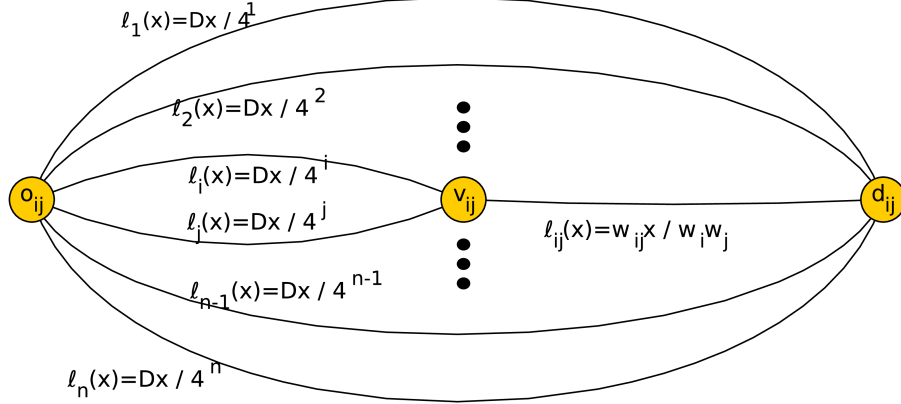
Figure 1: The series-parallel network F_{ij} that corresponds to edge $\{i, j\} \in A$.
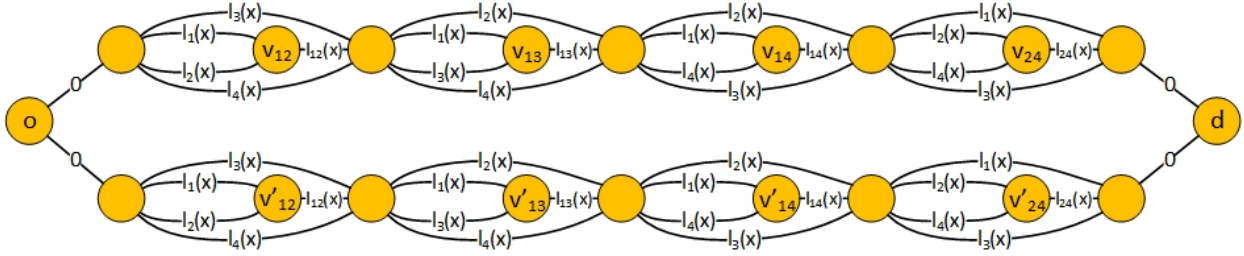

Figure 2: An example of the network G constructed in the proof of Theorem 1 for graph $H(V, A)$, with $V = \{1, 2, 3, 4\}$ and $A = \{\{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 4\}\}$. G is a parallel composition of two parts, each consisting of the smaller networks F_{12}, F_{13}, F_{14} and F_{24} (see also Figure 1) connected in series.

Let $H(V, A)$ be an instance of LOCAL-MAX-CUT with n vertices and m edges. Based on H, we construct a weighted congestion game on a single-commodity series-parallel network G with $3n$ players, where for every $i \in [n]$, there are three players with weight $w_i = 16^i$. Network G is a parallel composition of two identical copies of a simpler series-parallel network. We refer to these copies as G_1 and G_2. Each of G_1 and G_2 is a series composition of m simple series-parallel networks F_{ij}, each corresponding to an edge $\{i, j\} \in A$. Network F_{ij} is depicted in Figure 1, where D is assumed to be a constant chosen (polynomially) large enough. An example of the entire network G is shown in Figure 2.

In each of G_1 and G_2, there is a unique path that contains all edges with latency functions $\ell_i(x) = Dx/4^i$, for each $i \in [n]$. We refer to these paths as p_i^u for G_1 and p_i^l for G_2. In addition to the edges with latency $\ell_i(x)$, p_i^u and p_i^l include all edges with latencies $\ell_{ij}(x) = \frac{w_{ij}x}{w_i w_j} = \frac{w_{ij}x}{16^{i+j}}$, which correspond to the edges incident to vertex i in H.

Due to the choice of the player weights and the latency slopes, a player with weight w_i must choose either p_i^u or p_i^l in any PNE (see also propositions 5 and 6 in Appendix A.1). We can prove this claim by induction on the player weights. The players with weight $w_n = 16^n$ have a dominant strategy to choose either p_n^u or p_n^l, since the slope of $\ell_n(x)$ is significantly smaller than the slope of any other latency $\ell_i(x)$. In fact, the slope of ℓ_n is so small that even if all other $3n - 1$ players choose one of p_n^u or p_n^l, a player with weight w_n would prefer either p_n^u or p_n^l over all other paths.

6

Therefore, we can assume that each of p_n^u and p_n^l are used by at least one player with weight w_n in any PNE, which would increase their latency so much that no player with smaller weight would prefer them any more. The inductive argument applies the same reasoning for players with weights w_{n-1}, who should choose either p_{n-1}^u or p_{n-1}^l in any PNE, and subsequently, for players with weights $w_{n-2}, \ldots, w_1$. Hence, we conclude that for all $i \in [n]$, each of p_i^u and p_i^l is used by at least one player with weight w_i.

Moreover, we note that two players with different weights, say w_i and w_j, go through the same edge with latency $\ell_{ij}(x) = \frac{w_{ij}x}{w_i w_j}$ in G only if the corresponding edge $\{i, j\}$ is present in H. The correctness of the reduction follows the fact that a player with weight w_i aims to minimize her cost through edges with latencies ℓ_{ij} in G in the same way that in the MAX-CUT instance, we want to minimize the weight of the edges incident to a vertex i and do not cross the cut. Formally, we next show that a cut S is locally optimal for the MAX-CUT instance if and only if the configuration where for every $k \in S$, two players with weight w_k use p_k^u and for every $k \notin S$, two players with weight w_k use p_k^l is a PNE of the weighted congestion game on G.

Assume an equilibrium configuration and consider a player a of weight w_k that uses p_k^u together with another player of weight w_k (if this is not the case, vertex k is not included in S and we apply the symmetric argument for p_k^l). By the equilibrium condition, the cost of player a on p_k^u is at most her cost on p_k^l, which implies that

$$\sum_{k=1}^{m} \frac{2D16^k}{4^k} + \sum_{j:\{k,j\}\in A} \frac{w_{kj}(2 \cdot 16^k + x_j^u 16^j)}{16^{k+j}} \leq \sum_{k=1}^{m} \frac{2D16^k}{4^k} + \sum_{j:\{k,j\}\in A} \frac{w_{kj}(2 \cdot 16^k + x_j^l 16^j)}{16^{k+j}},$$

where x_j^u (resp. x_j^l) is either 1 or 2 (resp. 2 or 1) depending on whether, for each vertex j connected to vertex k in H, one or two players (of weight w_j) use p_j^u. Simplifying the inequality above, we obtain that:

$$\sum_{j:\{k,j\}\in A} w_{kj}(x_j^u - 1) \leq \sum_{j:\{k,j\}\in A} w_{kj}(x_j^l - 1) \tag{1}$$

Let $S = \{i \in V : x_i^u = 2\}$. By hypothesis, $k \in S$ and the left-hand side of (1) corresponds to the total weight of the edges in H that are incident to k and do not cross the cut S. Similarly, the right-hand side of (1) corresponds to the total weights of the edges in H that are incident to k and cross the cut S. Therefore, (1) implies that we cannot increase the value of the cut S by moving vertex k from S to $V \setminus S$. Since this or its symmetric condition holds for any vertex k of H, the cut $(S, V \setminus S)$ is locally optimal. To conclude the proof, we argue along the same lines that any locally optimal cut of H corresponds to a PNE in the weighted congestion game on G. $\square$

3.2 Weighted Congestion Games with Identity Latency Functions

We next prove that computing a PNE in weighted congestion games on multi-commodity networks with identity latency functions is PLS-complete. Compared to Theorem 1, we allow for a significantly more general strategy space, but we significantly restrict the latency functions, only allowing for the player weights to be exponentially large.

Theorem 2. *Computing a pure Nash equilibrium in weighted congestion games on multi-commodity networks with identity latency functions is PLS-complete.*

Proof sketch. We use a reduction from NODE-MAX-CUT, which as we show in Theorem 4, is PLS-complete. Our construction draws ideas from [1]. Missing technical details can be found in Appendix A.2.

Let $H(V, A)$ be an instance of NODE-MAX-CUT. We construct a weighted congestion game on a multi-commodity network G with identity latency functions $\ell_e(x) = x$ such that equilibria of the congestion game correspond to locally optimal cuts of H.

At the conceptual level, each player i of the congestion game corresponds to vertex $i \in V$ and has weight w_i (i.e., equal to the weight of vertex i in H). The key step is to construct a network G (see also Figure 4) such that for every player $i \in [n]$, there are two paths, say p_i^u and p_i^l, whose cost dominate the cost of any other path for player i. Therefore, in any equilibrium, player i selects either p_i^u or p_i^l (which corresponds to vertex i selecting one of the two sides of a cut). For every edge $\{i, j\} \in A$, paths p_i^u and p_j^u (resp. paths p_i^l and p_j^l) have an edge e_{ij}^u (resp. e_{ij}^l) in common. Intuitively, the cost of p_i^u (resp. p_i^l) for player i is determined by the set of players j, with j connected to i in H, that select path p_j^u (resp. p_j^l).

Let us consider any equilibrium configuration $\vec{s}$ of the weighted congestion game. By the discussion above, each player $i \in [n]$ selects either p_i^u or p_i^l in $\vec{s}$. Let $S = \{i \in [n] : \text{player } i \text{ selects } p_i^u \text{ in } \vec{s}\}$. Applying the equilibrium condition, we next show that S is a locally optimal cut.

We let $V_i = \{j : \{i, j\} \in A\}$ be the neighborhood of vertex i in H. By the construction of G, the individual cost of a player i on path p_i^u (resp. p_i^l) in $\vec{s}$ is equal to $K + |V_i| w_i + \sum_{j \in S \cap V_i} w_j$ (resp. $K + |V_i| w_i + \sum_{j \in V_i \setminus S} w_j$), where K is a large constant that depends on the network G only. Therefore, for any player $i \in S$, equilibrium condition for $\vec{s}$ implies that

$$K + |V_i| w_i + \sum_{j \in S \cap V_i} w_j \leq K + |V_i| w_i + \sum_{j \in V_i \setminus S} w_j \Rightarrow \sum_{j \in S \cap V_i} w_j \leq \sum_{j \in V_i \setminus S} w_j$$

Multiplying both sides by w_i, we get that the total weight of the edges that are incident to i and cross the cut S is no less than the total weight of the edges that are incident to i and do not cross the cut. By the same reasoning, we reach the same conclusion for any player $i \notin S$. Therefore, the cut $(S, V \setminus S)$ is locally optimal for the NODE-MAX-CUT instance $H(V, A)$.

To conclude the proof, we argue along the same lines that any locally optimal cut S for the NODE-MAX-CUT instance $H(V, A)$ corresponds to an equilibrium in the network G, by letting a player i select path p_i^u if and only if $i \in S$. □

4 Computing Approximate Equilibria for Node-Max-Cut

We complement our PLS-completeness proof for NODE-MAX-CUT, in Section 5, with an efficient algorithm computing $(1 + \varepsilon)$-approximate equilibria for NODE-MAX-CUT, when the number of different vertex weights is a constant. We note that similar results are not known (and a similar approach fails) for MAX-CUT. Investigating if stronger approximation guarantees are possible for efficiently computable approximate equilibria for NODE-MAX-CUT is beyond the scope of this work and an intriguing direction for further research.

Given a vertex-weighted graph $G(V, E)$ with n vertices and m edges, our algorithm, called BRIDGEGAPS (Algorithm 1), computes a $(1 + \varepsilon)^3$-approximate equilibrium for a NODE-MAX-CUT, for any $\varepsilon > 0$, in $(m/\varepsilon)(n/\varepsilon)^{O(D_\varepsilon)}$ time, where D_ε is the number of different vertex weights in G, when the weights are rounded down to powers of $1 + \varepsilon$. We next sketch the algorithm and the proof of Theorem 3. Missing technical details can be found in Appendix B.

For simplicity, we assume that n/ε is an integer (we use $\lceil n/\varepsilon \rceil$ in Appendix B) and that vertices are indexed in nondecreasing order of weight, i.e., $w_1 \leq w_2 \leq \cdots \leq w_n$. BRIDGEGAPS first rounds down vertex weights to the closest power of $(1 + \varepsilon)$. Namely, each weight w_i is replaced by weight $w_i' = (1 + \varepsilon)^{\lfloor \log_{1+\varepsilon} w_i \rfloor}$. Clearly, an $(1 + \varepsilon)^2$-approximate equilibrium for the new instance G' is an $(1 + \varepsilon)^3$-approximate equilibrium for the original instance G. The number of different weights D_ε, used in the analysis, is defined wrt. the new instance G'.

Then, BRIDGEGAPS partitions the vertices of G' into groups $g_1, g_2, \ldots$, so that the vertex weights in each group increase with the index of the group and the ratio of the maximum weight in group g_j to the minimum weight in group g_{j+1} is no less than n/ε. This can be performed by going through the vertices, in nondecreasing order of their weights, and assign vertex $i + 1$ to the same group as vertex i, if $w_{i+1}'/w_i' \leq n/\varepsilon$. Otherwise, vertex $i + 1$ starts a new group. The idea is that for an $(1+\varepsilon)^2$-approximate equilibrium in G', we only need to enforce the $(1+\varepsilon)$-approximate equilibrium condition for each vertex i only for i's neighbors in the highest-indexed group (that includes some neighbor of i). To see this, let g_j be the highest-indexed group that includes some neighbor of i and let ℓ be the lowest indexed neighbor of i in g_j. Then, the total weight of i's neighbors in groups $g_1, \ldots, g_{j-1}$ is less than $\varepsilon w_\ell'$. This holds because i has at most $n - 2$ neighbors in these groups and by definition, $w_q' \leq (\varepsilon/n)w_\ell'$, for any i's neighbor q in groups $g_1, \ldots, g_{j-1}$. Therefore, we can ignore all neighbors of i in groups $g_1, \ldots, g_{j-1}$, at the expense of one more $1+\varepsilon$ factor in the approximate equilibrium condition (see also the proof of Lemma 2).

Since for every vertex i, we need to enforce its (approximate) equilibrium condition only for i's neighbors in a single group, we can scale down vertex weights in the same group uniformly (i.e., dividing all the weights in each group by the same factor), as long as we maintain the key property in the definition of groups (i.e., that the ratio of the maximum weight in group g_j to the minimum weight in group g_{j+1} is no less than n/ε). Hence, we uniformly scale down the weights in each group so that (i) the minimum weight in group g_1 becomes 1; and (ii) for each $j \geq 2$, the ratio of the maximum weight in group g_{j-1} to the minimum weight in group g_j becomes exactly n/ε (see Appendix B for the details). This results in a new instance G'' where the minimum weight is 1 and the maximum weight is $(n/\varepsilon)^{D_\varepsilon}$. Therefore, a $(1 + \varepsilon)$-approximate equilibrium in G'' can be computed, in a standard way, after at most $(m\varepsilon)(n/\varepsilon)^{2D_\varepsilon}$ ε-best response moves (see the proof of Lemma 1).

Putting everything together and using $\varepsilon' = \varepsilon/7$, so that $(1 + \varepsilon')^3 \leq 1 + \varepsilon$, for all $\varepsilon \in (0, 1]$, we obtain the following (see Appendix B for the formal proof). We note that the running time of BRIDGEGAPS is polynomial, if $D_\varepsilon = O(1)$ (and quasipolynomial if $D_\varepsilon = \text{poly}(\log n)$).

Theorem 3. *For any vertex-weighted graph G with n vertices and m edges and any $\varepsilon > 0$, BRIDGEGAPS computes a $(1+\varepsilon)$-approximate pure Nash equilibrium for NODE-MAX-CUT on G in $(m/\varepsilon)(n/\varepsilon)^{O(D_\varepsilon)}$ time, where D_ε denotes the number of different vertex weights in G, after rounding them down to the nearest power of $1 + \varepsilon$.*

5   PLS-Completeness of Node-Max-Cut

We outline the proof of Theorem 4 and the main differences of our reduction from known PLS reductions to MAX-CUT [9, 20, 33]. The technical details can be found in Appendix C.

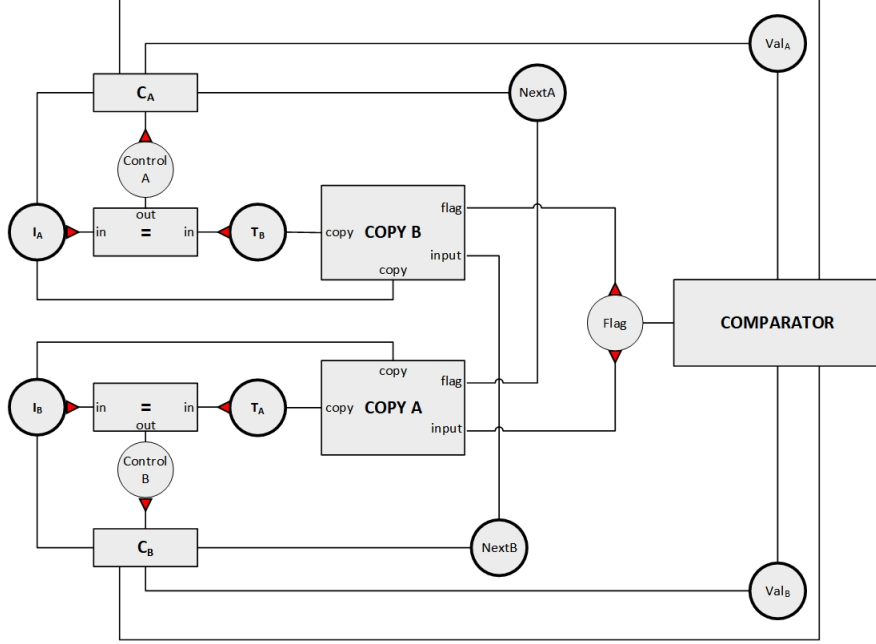**Theorem 4.** LOCAL-NODE-MAX-CUT *is PLS-complete.*

9

Figure 3: The general structure of the NODE-MAX-CUT instance constructed in the proof of Theorem 4. Rectangles denote the main gadgets, which are defined and discussed in Appendix C, circles denote vertices that participate in multiple gadgets, and circles with bold border denote groups of n such vertices. The small red triangles are used to indicate the "information flow".

As discussed in Section 2, the local search version of NODE-MAX-CUT is in PLS. To establish PLS-hardness of NODE-MAX-CUT, we present a reduction from CIRCUIT-FLIP.

An instance of CIRCUIT-FLIP consists of a Boolean circuit C with n inputs and m outputs (and wlog. only NOR gates). The value $C(s)$ of an n-bit input string s is the integer corresponding to the m-bit output string. The neighborhood $N(s)$ of s consists of all n-bit strings s' at Hamming distance 1 to s (i.e., s' is obtained from s by flipping one bit of s). The goal is to find a locally optimal input string s, i.e., an s with $C(s) \geq C(s')$, for all $s' \in N(s)$. CIRCUIT-FLIP was the first problem shown to be PLS-complete in [25].

Given an instance C of CIRCUIT-FLIP, we construct below a vertex-weighted undirected graph $G(V, E)$ so that from any locally optimum cut of G, we can recover, in polynomial time, a locally optimal input of C. The graph G consists of different gadgets (see Figure 3), which themselves might be regarded as smaller instances of NODE-MAX-CUT. Intuitively, each of these gadgets receives information from its "input" vertices, process this information, while carrying it through its internal part, and outputs the results through its "output" vertices. Different gadgets are glued together through their "input" and "output" vertices.

Our construction follows the *flip-flop* architecture (Figure 3), previously used e.g., in [9, 20, 33], but requires more sophisticated implementations of several gadgets, so as to conform with the very restricted weight structure of NODE-MAX-CUT. Next, we outline the functionality of the main gadgets and how the entire construction works (see Appendix C.1).

Given a circuit C, we construct two *Circuit Computing* gadgets C_ℓ (ℓ always stands for either A or B), which are instances of NODE-MAX-CUT that simulate circuit C in the following sense:

Each C_ℓ has a set I_ℓ of n "input" vertices, whose (cut) values correspond to the input string of circuit C, and a set Val_ℓ of m "output" vertices, whose values correspond to the output string of C on input I_ℓ. There is also a set $Next\ell$ of n vertices whose values correspond to a n-bit string in the neighborhood of I_ℓ of circuit value larger than that of I_ℓ (if the values of $Next\ell$ coincide with the values of I_ℓ, I_ℓ is locally optimal).

The *Circuit Computing* gadgets operate in two different modes, determined by $Control_\ell$: the *write mode*, when $Control_\ell = 0$, and the *compute mode*, when $Control_\ell = 1$. If C_ℓ operates in write mode, the input values of I_ℓ can be updated to the values of the complementary $Next$ set (i.e., I_A is updated to $NextB$, and vice versa). When, C_ℓ operates in the compute mode, C_ℓ simulates the computation of circuit C and the values of $Next\ell$ and Val_ℓ are updated to the corresponding values produced by C. Throughout the proof, we let $Real\text{-}Val(I_\ell)$ denote the output string of circuit C on input I_ℓ (i.e., the input of C takes the cut values of the vertices in I_ℓ), and let $Real\text{-}Next(I_\ell)$ denote a neighbor of I_ℓ with circuit value larger than the circuit value of I_ℓ. If I_ℓ is locally optimal, $Real\text{-}Next(I_\ell) = I_\ell$.

Our *Circuit Computing* gadgets C_A and C_B are based on the gadgets of Schäffer and Yannakakis [33] (see also Figure 5 for an abstract description of them). Their detailed construction is described in Section C.3 and their properties are summarized in Theorem 7.

The *Comparator* gadget compares Val_A with Val_B, which are intended to be $Real\text{-}Val(I_A)$ and $Real\text{-}Val(I_B)$, respectively, and outputs 1, if $Val_A \leq Val_B$, and 0, otherwise. The result of the *Comparator* is stored in the value of the *Flag* vertex. If $Flag = 1$, the *Circuit Computing* gadget C_A enters its write mode and the input values in I_A are updated to the neighboring solution of I_B, currently stored in $NextB$ (everything operates symmetrically, if $Flag = 0$). Then, in the next "cycle", the input in I_A leads C_A to a $Val_A > Val_B$ (and to a better neighboring solution at $NextA$), $Flag$ becomes 0, and the values of I_B are updated to $NextA$. When we reach a local optimum, I_A and I_B are stabilized to the same values.

The workflow above is implemented by the *Copy* and the *Equality* gadgets. The *CopyB* (resp. *CopyA*) gadget updates the values of I_A (resp. I_B) to the values in $NextB$ (resp. $NextA$), if $Val_A \leq Val_B$ and $Flag = 1$ (resp. $Val_A > Val_B$ and $Flag = 0$). When $Flag = 1$, the vertices in T_B take the values of the vertices in $NextB$. If the values of I_A and $NextB$ are different, the *Equality* gadget sets the value of $Control_A$ to 0. Hence, the *Circuit Computing* gadget C_A enters its write mode and the vertices in I_A take the values of the vertices in $NextB$. Next, $Control_A$ becomes 1, because the values of I_A and $NextB$ are now identical, and C_A enters its compute mode. As a result, the vertices in $ValA$ and $NextA$ take the values of $Real\text{-}Val(I_A)$ and $Real\text{-}Next(I_A)$, and we proceed to the next cycle.

A key notion used throughout the reduction is the *bias* that a vertex i experiences wrt. a vertex subset. The bias of vertex i wrt. (or from) $V' \subseteq V$ is $\left| \sum_{j \in V_i^1 \cap V'} w_j - \sum_{j \in V_i^0 \cap V'} w_j \right|$, where V_i^1 (resp. V_i^0) denotes the set of i's neighbors on the 1 (resp. 0) side of the cut.

Technical Novelty. Next, we briefly discuss the points where our reduction needs to deviate substantially from known PLS reductions to MAX-CUT. Our discussion is unavoidably technical and assumes familiarity with (at least one of) the reductions in [9, 20, 33].

Our *Circuit Computing* gadgets are based on similar gadgets used e.g., in [33]. A key difference is that our *Circuit Computing* gadgets are designed to operate with $w_{Control}$ (i.e., weight of the Control vertex) arbitrarily smaller than the weight of any other vertex in the *Circuit Computing* gadget. Hence, the Control vertex can achieve a very small bias wrt. the *Circuit Computing* gadget (see Case 3, in Theorem 7), which in turn, allows us to carefully balance the weights in the *Equality*

11

gadget. The latter weights should be large enough, so as to control the write and compute modes of C_ℓ, and at the same time, small enough, so as not to interfere with the values of the input vertices I_ℓ. The second important reason for setting $w_{Control}$ sufficiently small is that we need the Control vertex to follow the "output" of the *Equality* gadget, and not the other way around.

The discussion above highlights a key difference between Max-Cut and Node-Max-Cut. Previous reductions to Max-Cut (see e.g., the reduction in [33]) implement Control's functionality using different weights for its incident edges. More specifically, Control is connected with edges of appropriately large weight to the vertices of the circuit gadget, so that it can control the gadget's mode, and with edges of appropriately small weight to vertices outside the circuit gadget, so that its does not interfere with its other neighbors.

For Node-Max-Cut, we need to achieve the same desiderata with a single vertex weight. We manage to do so by introducing a *Leverage* gadget (see Section C.2 and cases 1 and 2, in Theorem 7). Our *Leverage* gadget reduces the influence of a vertex with large weight to a vertex with small weight and is used internally in the *Circuit Computing* gadget. Hence, we achieve that Control has small bias wrt. the circuit gadget and weight comparable to the weights of the circuit gadget's internal vertices.

Another important difference concerns the implementation of the *red marks* (denoting the information flow) between *Flag* and the *Copy* gadgets, in Figure 3. They indicate that the value of *Flag* should agree with the output of the *Comparator* gadget. This part of the information flow is difficult to implement, because the *Comparator* gadget and the *Copy* gadgets receive input from $NextA$, $NextB$, Val_A and Val_B, where the vertex weights are comparable to the weights of the output vertices in the *Circuit Computing* gadgets. As a result, the weights of the vertices inside the *Comparator* gadget cannot become sufficiently larger than the weights of the vertices inside the *Copy* gadgets. [33, 20, 9] connect *Flag* to the *Copy* gadgets with edges of sufficiently small weight, which makes the bias of *Flag* from the *Copy* gadgets negligible compared against its bias from *Comparator*. Again, the *Leverage* gadget comes to rescue. We use it internally in the *Copy* gadgets, in order to decrease the influence of the vertices inside the *Copy* gadgets to *Flag*. As a result, *Flag*'s bias from the *Copy* gadgets becomes much smaller than its bias from *Comparator* (see Lemma 9).

Another key technical difference concerns the design of the *Comparator* gadget. As stated in Lemma 9, the *Comparator* gadget computes the result of the comparison $Real\text{-}Val(I_A) \geq Real\text{-}Val(I_A)$, even if some "input vertices" have incorrect values. In previous work [33, 20, 9], *Comparator* guarantees correctness of the values in both $NextB$ and Val_B using appropriately chosen edge weights. With the correctness of the input values guaranteed, the comparison is not hard to implement. It is not clear if this decoupled architecture of the *Comparator* gadget can be implemented in Node-Max-Cut, due to the special structure of edge weights. Instead, we implement a new *all at once Comparator*, which ensures correctness to a subset of its input values enough to perform the comparison correctly.

6   Conclusions and Future Work

In this work, we showed that equilibrium computation in linear weighted congestion games is PLS-complete either on single-commodity series-parallel networks or on multi-commodity networks with identity latency functions, where computing an equribrium for (unweighted) congestion games is known to be easy. The key step for the latter reduction is to show that local optimum computa-

tion for NODE-MAX-CUT, a natural and significant restriction of MAX-CUT, is PLS-complete. The reductions in Section 3 are both *tight* [33], thus preserving the structure of the local search graph. In particular, for the first reduction, we have that (i) there are instances of linear weighted congestion games on single-commodity series-parallel networks such that any best response sequence has exponential length; and (ii) that the problem of computing the equilibrium reached from a given initial state is PSPACE-hard.

However, our reduction of CIRCUIT-FLIP to NODE-MAX-CUT is not tight. Specifically, our *Copy* and *Equality* gadgets allow that the *Circuit Computing* gadget might enter its *compute* mode, before the entire input has changed. Thus, we might "jump ahead" and reach an equilibrium before CIRCUIT-FLIP would allow, preventing the reduction from being tight.

Our work leaves leaves several interesting directions for further research. A natural first step is to investigate the complexity of equilibrium computation for weighted congestion games on series-parallel (or extension-parallel) networks with identity latency functions. An intriguing research direction is to investigate whether our ideas (and gadgets) in the PLS-reduction for NODE-MAX-CUT could lead to PLS-hardness results for approximate equilibrium computation for standard and weighted congestion games (similarly to the results of Skopalik and Vöcking [34], but for latency functions with non-negative coefficients). Finally, it would be interesting to understand better the quality of efficiently computable approximate equilibria for NODE-MAX-CUT and the smoothed complexity of its local optima.

References

[1] Heiner Ackermann, Heiko Röglin, and Berthold Vöcking. On the impact of combinatorial structure on congestion games. *J. ACM*, 55(6):25:1–25:22, 2008.

[2] Omer Angel, Sébastien Bubeck, Yuval Peres, and Fan Wei. Local Max-Cut in Smoothed Polynomial Time. In *Proc. of the 49th ACM SIGACT Symposium on Theory of Computing (STOC 2017)*, pages 429–437, 2017.

[3] Anand Bhalgat, Tanmoy Chakraborty, and Sanjeev Khanna. Approximating pure Nash equilibrium in cut, party affiliation, and satisfiability games. In *Proc. of the 11th ACM Conference on Electronic Commerce (EC 2010)*, pages 73–82, 2010.

[4] Ioannis Caragiannis and Angelo Fanelli. On Approximate Pure Nash Equilibria in Weighted Congestion Games with Polynomial Latencies. In *Proc. of the 46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, volume 132, pages 133:1–133:12, 2019.

[5] Ioannis Caragiannis, Angelo Fanelli, Nick Gravin, and Alexander Skopalik. Efficient Computation of Approximate Pure Nash Equilibria in Congestion Games. In *Proc. of the IEEE 52nd Symposium on Foundations of Computer Science, (FOCS 2011)*, pages 532–541, 2011.

[6] Ioannis Caragiannis, Angelo Fanelli, Nick Gravin, and Alexander Skopalik. Approximate Pure Nash Equilibria in Weighted Congestion Games: Existence, Efficient Computation, and Structure. *ACM Transactions on Economics and Computation*, 3(1):2:1–2:32, 2015.

[7] Xi Chen, Chenghao Guo, Emmanouil-Vasileios Vlatakis-Gkaragkounis, Mihalis Yannakakis, and Xinzhi Zhang. Smoothed complexity of local max-cut and binary max-csp. *CoRR*, abs/1911.10381, 2019. URL: http://arxiv.org/abs/1911.10381.

[8] Steve Chien and Alistair Sinclair. Convergence to approximate Nash equilibria in congestion games. *Games and Economic Behavior*, 71(2):315–327, 2011.

[9] Robert Elsässer and Tobias Tscheuschner. Settling the Complexity of Local Max-Cut (Almost) Completely. In *Proc. of the 38th International Colloquium on Automata, Languages and Programming (ICALP 2011)*, pages 171–182, 2011.

[10] Michael Etscheid and Heiko Röglin. Smoothed analysis of local search for the maximum-cut problem. *ACM Transactions on Algorithms*, 13(2):25:1–25:12, 2017.

[11] Eyal Even-Dar, Alexander Kesselman, and Yishay Mansour. Convergence time to nash equilibria. In *Proc. of the 30th International Colloquium on Automata, Languages and Programming (ICALP 2003)*, pages 502–513, 2003.

[12] Alex Fabrikant, Christos H. Papadimitriou, and Kunal Talwar. The Complexity of Pure Nash Equilibria. In *Proc. of the 36th Annual ACM Symposium on Theory of Computing (STOC 2004)*, pages 604–612, 2004.

[13] Angelo Fanelli and Luca Moscardelli. On best response dynamics in weighted congestion games with polynomial delays. In *Proc. of the 5th Workshop on Internet and Network Economics (WINE 2009)*, pages 55–66, 2009.

[14] Matthias Feldotto, Martin Gairing, Grammateia Kotsialou, and Alexander Skopalik. Computing approximate pure nash equilibria in shapley value weighted congestion games. In *Proc. of the 13th International Conference on Web and Internet Economics (WINE 2017)*, pages 191–204, 2017.

[15] Dimitris Fotakis. A selective tour through congestion games. In *Algorithms, Probability, Networks, and Games: Scientific Papers and Essays Dedicated to Paul G. Spirakis on the Occasion of His 60th Birthday*, pages 223–241, 2015.

[16] Dimitris Fotakis, Spyros C. Kontogiannis, Elias Koutsoupias, Marios Mavronicolas, and Paul G. Spirakis. The structure and complexity of Nash equilibria for a selfish routing game. *Theoretical Computer Science*, 410(36):3305–3326, 2009.

[17] Dimitris Fotakis, Spyros C. Kontogiannis, and Paul G. Spirakis. Selfish unsplittable flows. *Theoretical Computer Science*, 348(2-3):226–239, 2005.

[18] Dimitris Fotakis, Spyros C. Kontogiannis, and Paul G. Spirakis. Symmetry in Network Congestion Games: Pure Equilibria and Anarchy Cost. In *Proc. of the 3rd Workshop on Approximation and Online Algorithms, Revised Papers (WAOA 2005)*, pages 161–175, 2005.

[19] Martin Gairing, Thomas Lücking, Marios Mavronicolas, and Burkhard Monien. Computing Nash equilibria for scheduling on restricted parallel links. In *Proc. of the 36th Annual ACM Symposium on Theory of Computing (STOC 2004)*, pages 613–622, 2004.

[20] Martin Gairing and Rahul Savani. Computing Stable Outcomes in Hedonic Games. In *Proc. of the 3rd Symposium on Algorithmic Game Theory (SAGT 2010)*, pages 174–185, 2010.

[21] Yiannis Giannakopoulos, Georgy Noarov, and Andreas S. Schulz. An improved algorithm for computing approximate equilibria in weighted congestion games. *CoRR*, abs/1810.12806, 2018.

[22] Paul W. Goldberg. Bounds for the convergence rate of randomized local search in a multiplayer load-balancing game. In *Proc. of the 23rd Annual ACM Symposium on Principles of Distributed Computing (PODC 2004)*, pages 131–140, 2004.

[23] Tobias Harks and Max Klimm. On the Existence of Pure Nash Equilibria in Weighted Congestion Games. In *Proc. of the 37th International Colloquium on Automata, Languages and Programming (ICALP 2010)*, pages 79–89, 2010.

[24] Tobias Harks, Max Klimm, and Rolf H. Möhring. Characterizing the existence of potential functions in weighted congestion games. *Theory Computing Systems*, 49(1):46–70, 2011.

[25] David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. How easy is local search? *Journal of Computer and System Sciences*, 37(1):79–100, 1988.

[26] Richard M. Karp. Reducibility among combinatorial problems. In *Proc. of Symposium on the Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103, 1972.

[27] Pieter Kleer and Guido Schäfer. Potential Function Minimizers of Combinatorial Congestion Games: Efficiency and Computation. In *Proc. of the 2017 ACM Conference on Economics and Computation (EC 2017)*, pages 223–240, 2017.

[28] Wil Michiels, Emile Aarts, and Jan Korst. *Theoretical Aspects of Local Search*. EATCS Monographs in Theoretical Computer Science. Springer, 2007.

[29] Dov Monderer and Lloyd S. Shapley. Potential Games. *Games and economic behavior*, 14(1):124–143, 1996.

[30] Panagiota N. Panagopoulou and Paul G. Spirakis. Algorithms for pure Nash equilibria in weighted congestion games. *ACM Journal of Experimental Algorithmics*, 11, 2006.

[31] Svatopluk Poljak. Integer Linear Programs and Local Search for Max-Cut. *SIAM Journal on Computing*, 21(3):450–465, 1995.

[32] R.W. Rosenthal. A Class of Games Possessing Pure-Strategy Nash Equilibria. *International Journal of Game Theory*, 2:65–67, 1973.

[33] Alejandro A. Schäffer and Mihalis Yannakakis. Simple Local Search Problems That are Hard to Solve. *SIAM Journal on Computing*, 20(1):56–87, 1991.

[34] Alexander Skopalik and Berthold Vöcking. Inapproximability of Pure Nash Equilibria. In *Proc. of the 40th Annual ACM Symposium on Theory of Computing (STOC 2008)*, pages 355–364, 2008.

[35] J. Valdez, R.E. Tarjan, and E.L. Lawler. The Recognition of Series-Parallel Digraphs. *SIAM Journal on Computing*, 11(2):298–313, 1982.

A   The Proofs of the Theorems of Section 3

A.1   The Proof of Theorem 1

We will reduce from the PLS-complete problem MAX-CUT and given an instance of MAX-CUT we will construct a network weighted network Congestion Game for which the Nash equilibria will correspond to maximal solutions of MAX-CUT and vice versa. First we give the construction and then we prove the theorem. For the formal PLS-reduction, which needs functions ϕ_1 and ϕ_2, ϕ_1 returns the (polynomially) constructed instance described below and ϕ_2 will be revealed later in the proof.

Let $H(V, E)$ be an edge-weighted graph of a MAX-CUT instance and let $n = |V|$ and $m = |E|$. In the constructed network weighted CG instance there will be $3n$ players which will share n different weights inside the set $\{16^i : i \in [n]\}$ so that for every $i \in [n]$ there are exactly 3 players having weight $w_i = 16^i$. All players share a common origin-destination pair $o - d$ and choose $o - d$ paths on a series-parallel graph G. Graph G is a parallel composition of two identical copies of a series-parallel graph. Call these copies G_1 and G_2. In turn, each of G_1 and G_2 is a series composition of m different series-parallel graphs, each of which corresponds to the m edges of H. For every $\{i, j\} \in E$ let F_{ij} be the series-parallel graph that corresponds to $\{i, j\}$. Next we describe the construction of F_{ij}, also shown in Fig. 1.

F_{ij} has 3 vertices, namely o_{ij}, v_{ij} and d_{ij} and $n+1$ edges. For any $k \in [n]$ other than i, j there is an $o_{ij} - d_{ij}$ edge with latency function $\ell_k(x) = \frac{Dx}{4^k}$, where D serves as a big constant to be defined later. There are also two $o_{ij} - v_{ij}$ edges, one with latency function $\ell_i(x) = \frac{Dx}{4^i}$ and one with latency function $\ell_j(x) = \frac{Dx}{4^j}$. Last, there is a $v_{ij} - d_{ij}$ edge with latency function $\ell_{ij}(x) = \frac{w_{ij}x}{w_i w_j}$, where w_{ij} is the weight of edge $\{i, j\} \in E$ and w_i and w_j are the weights of players i and j, respectively, as described earlier. Note that in every F_{ij} and for any $k \in [n]$ the latency function $\ell_k(x) = \frac{Dx}{k}$ appears in exactly one edge. With F_{ij} defined, an example of the structure of such a network G is given in Fig. 2.

Observe that in each of G_1 and G_2 there is a unique path that contains all the edges with latency functions $\ell_i(x)$, for $i \in [n]$, and call these paths p_i^u and p_i^l for the upper (G_1) and lower (G_2) copy respectively. Note that each of p_i^u and p_i^l in addition to those edges, contains some edges with latency function of the form $\frac{w_{ij}x}{w_i w_j}$. These edges for path p_i^u or p_i^l is in one to one correspondence to the edges of vertex i in H and this is crucial for the proof.

We go on to prove the correspondence of Nash equilibria in G to maximal cuts in H, i.e., solutions of MAX-CUT. We will first show that at a Nash equilibrium, a player of weight w_i chooses either p_i^u or p_i^l. Additionally, we prove that p_i^u and p_i^l will have at least one player (of weight w_i). This already provides a good structure of a Nash equilibrium and players of different weights, say w_i and w_j, may go through the same edge in G (the edge with latency function $w_{ij}x/w_i w_j$) only if $\{i, j\} \in E$. The correctness of the reduction lies in the fact that players in G try to minimize their costs incurred by these type of edges in the same way one wants to minimize the sum of the weights of the edges in each side of the cut when solving MAX-CUT.

To begin with, we will prove that at equilibrium any player of weight w_i chooses either p_i^u or p_i^l and at least one such player chooses each of p_i^u and p_i^l. For that, we will need the following proposition as a building block, which will also reveal a suitable value for D.

Proposition 5. *For some $i, j \in [n]$ consider F_{ij} (Fig. 1) and assume that for all $k \in [n]$, there are either one, two or three players of weight w_k that have to choose an $o_{ij} - d_{ij}$ path. At equilibrium,*

17

all players of weight w_k (for any $k \in [n]$) will go through the path that contains a edge with latency function $\ell_k(x)$.

Proof. The proof is by induction on the different weights starting from bigger weights. For any $k \in [n]$ call e_k the edge of F_{ij} with latency function $\ell_k(x)$ and call e_{ij} the edge with latency function $\frac{w_{ij}x}{w_iw_j}$. For some $k \in [n]$ assume that for all $l > k$ all players of weight w_l have chosen the path containing e_l and lets prove that this is the case for players of weight w_k as well. Since D is going to be big enough, for the moment ignore edge e_{ij} and assume that in F_{ij} there are only n parallel paths each consisting of a single edge.

Let the players be at equilibrium and consider any player, say player K, of weight w_k. The cost she computes on e_k is upper bounded by the cost of e_k if all players with weight up to w_k are on e_k, since by induction players with weight $> w_k$ are not on e_k at equilibrium. This cost is upper bounded by $c^k = \frac{D(3\sum_{l=1}^{k}16^l)}{4^k} = \frac{3D\frac{16^{k+1}-1}{16-1}}{4^k}$.

For any edge e_l for $l < k$, the cost that K computes is lower bounded by $c^< = \frac{D16^k}{4^{k-1}}$ since she must include herself in the load of e_l and the edge with the smallest slope in its latency function is e_{k-1}. But then $c^k < c^<$, since

$$c^k < c^< \Leftrightarrow \frac{3D\frac{16^{k+1}-1}{16-1}}{4^k} < \frac{D16^k}{4^{k-1}} \Leftrightarrow 48 \cdot 16^k - 3 < 60 \cdot 16^k$$

Thus, at equilibrium players of weight w_k cannot be on any of the e_l's for all $l < k$. On the other hand, the cost that K computes for e_l for $l > k$ is at least $c_l^> = \frac{D(16^l+16^k)}{4^l}$, since by induction e_l is already chosen by at least one player of weight w_l. But then $c^k < c_l^>$ since

$$\frac{3D\frac{16^{k+1}-1}{16-1}}{4^k} < \frac{D(16^l + 16^k)}{4^l} \Leftrightarrow$$
$$48 \cdot 16^k - 3 < 15\frac{16^l + 16^k}{4^{l-k}} \Leftrightarrow$$
$$48 \cdot 4^{l-k}16^k < 15 \cdot 16^l = 15 \cdot 4^{l-k}4^{l-k}16^k.$$

Thus, at equilibrium players of weight w_k cannot be on any of the e_l's for all $l > k$.

This completes the induction for the simplified case where we ignored the existence of e_{ij}, but lets go on to include it and define D so that the same analysis goes through. By the above, $c^< - c^k = \frac{D16^k}{4^{k-1}} - \frac{3D\frac{16^{k+1}-1}{16-1}}{4^k} > D$ and also for any $l > k$ it is $c_l^> - c^k = \frac{D(16^l+16^k)}{4^l} - \frac{3D\frac{16^{k+1}-1}{16-1}}{4^k} > D$ (this difference is minimized for $l = k + 1$). On the other hand the maximum cost that edge e_{ij} may have is bounded above by $c^{ij} = \frac{w_{ij}3\sum_{l=1}^{n}16^l}{w_iw_j}$, as e_{ij} can be chosen by at most all of the players and note that $c^{ij} \leq 16^{n+1}\max_{q,r\in[n]} w_{qr}$. Thus, one can choose a big value for D, namely $D = 16^{n+1}\max_{q,r\in[n]} w_{qr}$, so that even if a player with weight w_k has to add the cost of e_{ij} when computing her path cost, it still is $c^{ij} + c^k < c^<$ (since $c^< - c^k > D \geq c_{ij}$) and for all $l > k$: $c^{ij} + c^k < c_l^>$ (since $c_l^> - c^k > D \geq c^{ij}$), implying that at equilibrium all players of weight w_k may only choose the path through e_k. $\square$

Other than revealing a value for D, the proof of Porposition 5 reveals a crucial property: a player of weight w_k in F_{ij} strictly prefers the path containing e_k to the path containing e_l for any $l < k$, independent to whether players of weight $> w_k$ are present in the game or not. With this in

mind we go back to prove that at equilibrium any player of weight w_i chooses either p_i^u or p_i^l and at least one such player chooses each of p_i^u and p_i^l. The proof is by induction, starting from bigger weights.

Assume that by the inductive hypothesis for every $i > k$, players with weights w_i have chosen paths p_i^u or p_i^l and at least one such player chooses each of p_i^u and p_i^l. Consider a player of weight w_k, and, wlog, let her have chosen an $o - d$ path through G_1. Since at least one player for every bigger weight is by induction already in the paths of G_1 (each in her corresponding p_i^u), Proposition 5 and the remark after its proof give that in each of the F_{ij}'s the player of weight w_k has chosen the subpath of p_k^u, and this may happen only if her chosen path is p_k^u. It remains to show that there is another player of weight w_k that goes through G_2, which, with an argument similar to the previous one, is equivalent to this player choosing path p_k^l.

To reach a contradiction, let p_k^u be chosen by all three players of weight w_k, which leaves p_k^l empty. Since all players of bigger weights are by induction settled in paths completely disjoint to p_k^l, the load on this path if we include a player of weight w_k is upper bounded by the sum of all players of weight $< w_k$ plus w_k, i.e., $16^k + 3 \sum_{t=1}^{k-1} 16^t = 16^k + 3\frac{16^k-1}{16-1}$, which is less than the lower bound on the load of p_k^u, i.e., $3 \cdot 16^k$ (since p_k^u carries 3 players of weight 16^k). This already is a contradiction to the equilibrium property, since p_k^u and p_k^l share the exact same latency functions on their edges which, given the above inequality on the loads, makes p_k^u more costly than p_k^l for a player of weight w_k. To summarize, we have the following.

Proposition 6. *At equilibrium, for every $i \in [n]$ a player of weight w_i chooses either p_i^u or p_i^l. Additionally, each of p_i^u and p_i^l have been chosen by at least one player (of weight w_i).*

Finally, we prove that every equilibrium of the constructed instance corresponds to a maximal solution of MAX-CUT and vice versa. Given a maximal solution S of MAX-CUT we will show that the configuration Q that for every $k \in S$ routes 2 players through p_k^u and 1 player through p_k^l and for every $k \in V \setminus S$ routes 1 player through p_k^u and 2 players through p_k^l is an equilibrium. Conversely, given an equilibrium Q the cut $S = \{k \in V : 2 \text{ players have chosen } p_k^u \text{ at } Q\}$ is a maximal solution of MAX-CUT.

Assume that we are at equilibrium and consider a player of weight w_k that has chosen p_k^u and wlog p_k^u is chosen by two players (of weight w_k). By the equilibrium conditions the cost she computes for p_k^u is at most the cost she computes for p_k^l, which, given Proposition 6, implies

$$\sum_{i=1}^{m} \frac{2D16^k}{4^k} + \sum_{\{k,j\} \in E} \frac{w_{kj}(2 \cdot 16^k + x_j^u 16^j)}{16^k 16^j} \leq \sum_{i=1}^{m} \frac{2D16^k}{4^k} + \sum_{\{k,j\} \in E} \frac{w_{kj}(2 \cdot 16^k + x_j^l 16^j)}{16^k 16^j}$$

where x_j^u (resp. x_j^l) is either 1 or 2 (resp. 2 or 1) depending whether, for any $j : \{k, j\} \in E$, one or two players (of weight w_j) respectively have chosen path p_j^u. By canceling out terms, the above implies

$$\sum_{\{k,j\} \in E} w_{kj} x_j^u \leq \sum_{\{k,j\} \in E} w_{kj} x_j^l \Leftrightarrow \sum_{\{k,j\} \in E} w_{kj}(x_j^u - 1) \leq \sum_{\{k,j\} \in E} w_{kj}(x_j^l - 1) \tag{2}$$

Define $S = \{i \in V : x_i^u = 2\}$. By our assumption it is $k \in S$ and the left side of (2), i.e., $\sum_{\{k,j\} \in E} w_{kj}(x_j^u - 1)$, is the sum of the weights of the edges of H with one of its vertices being k and the other belonging in S. Similarly, the right side of of (2), i.e., $\sum_{\{k,j\} \in E} w_{kj}(x_j^l - 1)$ is the sum of the weights of the edges with one of its vertices being k and the other belonging in

$V \setminus S$. But then (2) directly implies that for the (neighboring) cut S' where k goes from S to $V \setminus S$ it holds $w(S) \geq w(S')$. Since k was arbitrary (given the symmetry of the problem), this holds for every $k \in [n]$ and thus for every $S' \in N(S)$ it is $w(S) \geq w(S')$ proving one direction of the claim. Observing that the argument works backwards we complete the proof. For the formal proof, to define function ϕ_2, given the constructed instance and one of its solutions, say s', ϕ_2 returns solution $s = \{k \in V : 2 \text{ players have chosen } p_k^u \text{ at } s'\}$. $\qquad\square$

A.2   The Proof of Theorem 2

We will reduce from the PLS-complete problem NODE-MAX-CUT. Our construction draws ideas from Ackermann et al. [1]. For an instance of NODE-MAX-CUT we will construct a multi-commodity network CG where every equilibrium will correspond to a maximal solution of NODE-MAX-CUT and vice versa. For the formal PLS-reduction, which needs functions ϕ_1 and ϕ_2, ϕ_1 returns the (polynomially) constructed instance described below and ϕ_2 will be revealed later in the proof.

We will use only the identity function as the latency function of every edge, but for ease of presentation we will first prove our claim assuming we can use constant latency functions on the edges. Then we will describe how we can drop this assumption and use only the identity function on all edges, and have the proof still going through.

Let $H(V, E)$ be the vertex-weighted graph of an instance of NODE-MAX-CUT with $n = |V|$ vertices and $m = |E|$ edges. The network weighted congestion game has n players, with player i having her own origin destination $o_i - d_i$ pair and weight w_i equal to the weight of vertex $i \in V$. In the constructed network there will be many $o_i - d_i$ paths for every player i but there will be exactly two paths that cost-wise dominate all others. At equilibrium, every player will choose one of these two paths that correspond to her. This choice for player i will be equivalent to picking the side of the cut that vertex i should lie in order to get a maximal solution of NODE-MAX-CUT.

The initial network construction is shown in Fig. 4. It has n origins and n destinations. The rest of the vertices lie either on the lower-left half (including the diagonal) of a $n \times n$ grid, which we call the upper part, or the upper-left half of another $n \times n$ grid, which we call the lower part. Other than the edges of the two half-grids that are all present, there are edges connecting the origins and the destinations to the two parts. For $i \in [n]$, origin o_i in each of the upper and lower parts connects to the first (from left to right) vertex of the row that has i vertices in total. For $i \in [n]$, destination d_i in each of the upper and lower parts connects to the i-th vertex of the row that has n vertices in total. To define the (constant) latency functions, we will need 2 big constants, say d and $D = n^3 d$, and note that $D \gg d$.

All edges that connect to an origin or a destination and all the vertical edges of the half-grids will have constant D as their latency function, and any horizontal edge that lies on a row with i vertices will have constant $i \cdot d$ as its latency function. To finalize the construction we will do some small changes but note that, as it is now, player i has two shortest paths that are far less costly (at least by d) than all other paths. These two paths are path p_i^u that starts at o_i, continues horizontally through the upper part for as much as it can and then continues vertically to reach d_i, and path p_i^l which does the exact same thing through the lower part (for an example see Fig. 4a). Each of p_i^u and p_i^l costs equal to $c_i = 2D + i(i-1)d + (n-i)D$. To verify this claim simply note that (i) if a path tries to go through another origin or moves vertically away from d_i in order to reach less costly horizontal edges, then it will have to pass through at least $(2 + i - 1) + 2$ vertical edges of cost D and its cost from such edges compared to p_i^u's and p_i^l's costs increases by at least

Figure 4: (a) The construction of the reduction of Theorem 2. As an example, in orange are the least costly $o_2 - d_2$ paths p_2^u (up) and p_2^l (down), each with cost equal to $2D + 2d + (n-2)D$. (b) The replacement of the red vertex at the i-th row and j-th column of the upper half-grid whenever edge $\{i, j\} \in E$. A symmetric replacement happens in the lower half-grid.

$2D = 2n^3d$, which is already more than paying all horizontal edges; and (ii) if it moves vertically towards d_i earlier than p_i^u or p_i^l then its cost increases by at least d, since it moves towards more costly horizontal edges.

To complete the construction if $\{i, j\} \in E$ (with wlog $i < j$) we replace the (red) vertex at position i, j of the upper and the lower half-grid $(1, 1$ is top left for the upper half-grid and lower left for the lower half-grid) with two vertices connected with an edge, say e_{ij}^u and e_{ij}^l respectively, with latency function $\ell_{ij}(x) = x$, where the first vertex connects with the vertices at positions $i, j - 1$ and $i - 1, j$ of the grid and the second vertex connects to the vertices at positions $i + 1, j$ and $i, j + 1$ (see also Fig. 4b). Note that if we take $d \gg \sum_{k \in [n]} w_k$, then, for any $i \in [n]$, paths p_i^u and p_i^l still have significantly lower costs than all other $o_i - d_i$ paths. Additionally, if $\{i, j\} \in E$ then p_i^u and p_j^u have a single common edge and p_i^l and p_j^l have a single common edge, namely e_{ij}^u and e_{ij}^l respectively, which add some extra cost to the paths (added to c_i defined above).

Assume we are at equilibrium. By the above discussion player $i \in [n]$ may only have chosen p_i^u or p_i^l. Let $S = \{i \in [n] : \text{player } i \text{ has chosen } p_i^u\}$. We will prove that S is a solution to NODE-MAX-CUT. By the equilibrium conditions for every $i \in S$ the cost of p_i^u, say c_i^u, is less than or equal to the cost of p_i^l, say c_i^l. Given the choices of the rest of the players, and by defining S_i to be the neighbors of i in S, i.e. $S_i = \{j \in S : \{i, j\} \in E\}$, and V_i be the neighbors of i in V, i.e. $V_i = \{j \in V : \{i, j\} \in E\}$, $c_i^u \leq c_i^l$ translates to

$$\left(2D + i(i-1)d + (n-i)D\right) + \left(\sum_{j \in V_i} w_i + \sum_{j \in S_i} w_j\right)$$
$$\leq \left(2D + i(i-1)d + (n-i)D\right) + \left(\sum_{j \in V_i} w_i + \sum_{j \in V_i \setminus S_i} w_j\right),$$

21

with the costs in the second and fourth parenthesis coming from the e_{ij}'s for the different j's. This equivalently gives

$$\sum_{j \in S_i} w_j \leq \sum_{j \in V_i \setminus S_i} w_j \Leftrightarrow \sum_{j \in S_i} w_i w_j \leq \sum_{j \in V_i \setminus S_i} w_i w_j.$$

The right side of the last inequality equals to the weight of the edges with i as an endpoint that cross cut S. The left side equals to the weight of the edges with i as an endpoint that cross the cut S', where S' is obtained by moving i from S to $V \setminus S$. Thus for S and S' it is $w(S') \leq w(S)$. A similar argument (or just symmetry) shows that if $i \in V \setminus S$ and we send i from $V \setminus S$ to S to form a cut S' it would again be $w(S') \leq w(S)$. Thus, for any $S' \in N(S)$ it is $w(S) \geq w(S')$ showing that S is a solution to NODE-MAX-CUT. Observing that the argument works backwards we have that from an arbitrary solution of NODE-MAX-CUT we may get an equilibrium for the constructed weighted CG instance. For the formal part, to define function ϕ_2, given the constructed instance and one of its solutions, ϕ_2 returns solution $s = \{i \in [n] : \text{player } i \text{ has chosen } p_i^u\}$.

What remains to show is how we can almost simulate the constant latency functions so that we use only the identity function on all edges and, for every $i \in [n]$, player i still may only choose paths p_i^u or p_i^l at equilibrium. Observe that, since we have a multi-commodity instance we can simulate (exponentially large) constants by replacing an edge $\{j, k\}$ with a three edge path $j - o_{jk} - d_{jk} - k$, adding a complementary player with origin o_{jk} and destination d_{jk} and weight equal to the desired constant. Depending on the rest of the structure we may additionally have to make sure (by suitably defining latency functions) that this player prefers going through edge $\{o_{jk}, d_{jk}\}$ at equilibrium.

To begin with, consider any horizontal edge $\{j, k\}$ with latency function $i \cdot d$ (for some $i \in [n]$) and replace it with a three edge path $j - o_{jk} - d_{jk} - k$. Add a player with origin o_{jk} and destination d_{jk} with weight equal to $in^3 w$, where $w = \sum_{i \in [n]} w_i$, and let all edges have the identity function. At equilibrium no matter the sum of the weights of the players that choose this three edge path, the $o_{jk} - d_{jk}$ player prefers to use the direct $o_{jk} - d_{jk}$ edge or else she pays at least double the cost (middle edge vs first and third edges). Thus the above replacement is (at equilibrium) equivalent to having edge $\{j, k\}$ with latency function $3x + in^3 w = 3x + i \cdot d$, for $d = n^3 w$.

Similarly, consider any edge $\{j, k\}$ with latency function D and replace it with a three edge path $j - o_{jk} - d_{jk} - k$. Add a player with origin o_{jk} and destination d_{jk} with weight equal to $n^3 d$ and let all edges have the identity function. Similar to above, this replacement is (at equilibrium) equivalent to having edge $\{j, k\}$ with latency function $3x + n^3 d = 3x + D$, for $D = n^3 d$.

With these definitions, at equilibrium, all complementary players will go through the correct edges and, due to the complementary players, all edges that connect to an origin or a destination will have cost $\approx D$, all vertical edges of the half-grids will cost $\approx D$, and any horizontal edge that lies on a row with i vertices will cost $\approx i \cdot d$, where "$\approx$" means at most within $\pm 3w = \pm \frac{3d}{n^3}$ (note that w is the maximum weight that the $o_i - d_i$ players can add to each of the three edge paths). Additionally, for every $i \in [n]$, p_i^u and p_i^l are structurally identical, i.e., they have the same structure, identical complementary players on their edges and share the same latency functions. All the above make the analysis go through in the same way as in the simplified construction. $\square$

B   Missing Technical Details from the Analysis of BridgeGaps

In this section, we present an algorithm that computes approximate equilibria for NODE-MAX-CUT. Let $G(V, E)$ be vertex-weighted graph with n vertices and m edges, and consider any $\varepsilon > 0$.

The algorithm, called BRIDGEGAPS and formally presented in Algorithm 1, returns a $(1+\varepsilon)^3$-approximate equilibrium (Lemma 2) for G in time $O(\frac{m}{\varepsilon}\lceil\frac{n}{\varepsilon}\rceil^{2D_\varepsilon})$ (Lemma 1), where D_ε is the number of different rounded weights, i.e., the weights produced by rounding down each of the original weights to its closest power of $(1+\varepsilon)$. To get a $(1+\varepsilon)$-approximate equilibrium, for $\varepsilon < 1$, it suffices to run the algorithm with $\varepsilon' = \frac{\varepsilon}{7}$.

Description of the Algorithm. BRIDGEGAPS first creates an instance G' with weights rounded down to their closest power of $(1+\varepsilon)$, i.e., weight w_i is replaced by weight $w_i' = (1+\varepsilon)^{\lfloor\log_{1+\varepsilon} w_i\rfloor}$ in G', and then computes a $(1+\varepsilon)^2$-approximate equilibrium for G'. Observe that any $(1+\varepsilon)^2$-approximate equilibrium for G' is a $(1+\varepsilon)^3$-approximate equilibrium for G, since

$$\sum_{j \in V_i:s_i=s_j} w_j \le (1+\varepsilon) \sum_{j \in V_i:s_i=s_j} w_j' \le (1+\varepsilon)^3 \sum_{j \in V_i:s_i\ne s_j} w_j' \le (1+\varepsilon)^3 \sum_{j \in V_i:s_i\ne s_j} w_j,$$

where V_i denotes the set of vertices that share an edge with vertex i, with the first and third inequalities following from the rounding and the second one following from the equilibrium condition for G'.

To compute a $(1+\varepsilon)^2$-approximate equilibrium, BRIDGEGAPS first sorts the vertices in increasing weight order and note that, wlog, we may assume that $w_1' = 1$, as we may simply divide all weights by w_1'. Then, it groups the vertices so that the fraction of the weights of consecutive vertices in the same group is bounded above by $\lceil n/\varepsilon\rceil$, i.e., for any i, vertices i and $i+1$ belong in the same group if and only if $\frac{w_{i+1}'}{w_i'} \le \lceil\frac{n}{\varepsilon}\rceil$. This way, groups g_j are formed on which we assume an increasing order, i.e., for any j, the vertices in g_j have smaller weights than those in g_{j+1}.

The next step is to bring the groups closer together using the following process which will generate weights w_i''. For all j, all the weights of vertices on heavier groups, i.e., groups $g_{j+1}, g_{j+2}, \ldots$, are divided by $d_j = \frac{1}{\lceil n/\varepsilon\rceil}\frac{w_{j+1}^{min}}{w_j^{max}}$ so that $\frac{w_{j+1}^{min}/d_j}{w_j^{max}} = \lceil\frac{n}{\varepsilon}\rceil$, where w_{j+1}^{min} is the smallest weight in g_{j+1} and w_j^{max} is the biggest weight in g_j. For vertex i, let the resulting weight be w_i'', i.e., $w_i'' = \frac{w_i'}{\Pi_{j\in I_i} d_j}$, where I_i contains the indexes of groups below i's group, and keep the increasing order on the vertex weights. Observe that by the above process for any i: $\frac{w_{i+1}''}{w_i''} \le \lceil\frac{n}{\varepsilon}\rceil$, either because i and $i+1$ are in the same group or because the groups are brought closer together. Additionally if i and $i+1$ belong in different groups then $\frac{w_{i+1}''}{w_i''} = \lceil\frac{n}{\varepsilon}\rceil$, implying that for vertices i, i' in different groups with $w_{i'}'' > w_i''$ it is $\frac{w_{i'}''}{w_i''} \ge \lceil\frac{n}{\varepsilon}\rceil$. Thus, if we let D_ε be the number of different weights in G', i.e., $D_\varepsilon = |\{w_i' : i \text{ vertex of } G'\}|$, then the maximum weight w_n'' is $w_n'' = \frac{w_n''}{w_{n-1}''}\frac{w_{n-1}''}{w_{n-2}''}\cdots\frac{w_2''}{w_1''} \le \lceil\frac{n}{\varepsilon}\rceil^{D_\varepsilon}$

In a last step, using the w'' weights, BRIDGEGAPS starts from an arbitrary configuration (a 0-1 vector) and lets the vertices play ε-best response moves, i.e., as long as there is an index i of the vector violating the $(1+\varepsilon)$-approximate equilibrium condition, BRIDGEGAPS flips its bit. When there is no such index BRIDGEGAPS ends and returns the resulting configuration.

Lemma 1. *For any $\varepsilon > 0$, BRIDGEGAPS terminates in time $O(\frac{m}{\varepsilon}\lceil\frac{n}{\varepsilon}\rceil^{2D_\varepsilon})$*

Proof. We are going to show the claimed bound for the last step of BRIDGEGAPS since all previous steps can be (naively) implemented to end in $O(n^2)$ time.

The proof relies on a potential function argument. For any $\vec{s} \in \{0,1\}^n$, let

$$\Phi(\vec{s}) = \frac{1}{2}\sum_{i\in V}\sum_{j\in V_i:s_i=s_j} w_i'' w_j''.$$

23

Algorithm 1: BRIDGEGAPS, computing $(1+\varepsilon)^3$-approximate equilibria

---

Input: A NODE-MAX-CUT instance $G(V,E)$ with n vertices and weights $\{w_i\}_{i\in[n]}$ sorted
  increasingly with $w_1 = 1$, and an $\varepsilon > 0$.

Output: A vector $\vec{s} \in \{0,1\}^n$ partitioning the vertices in two sets.

1 for $i \in [n]$ **do** $w_i := (1+\varepsilon)^{\lfloor \log_{1+\varepsilon} w_i \rfloor}$

2 groups$:= 1$;

  insert w_1 **into** g_{groups};   ▷ Assign the weights into groups $\{g_j\}_{j\in[groups]}$

  for $i \in \{2,...,n\}$ **do**

   | **if** $\frac{w_i}{w_{i-1}} > \lceil \frac{n}{\epsilon} \rceil$ **then** groups++;
   | **insert** w_i **into** g_{groups};

3 for $j \in \{2,...,groups\}$ **do**

   | $w_j^{min} :=$ minimum weight of group g_j;   ▷ Bring the groups $\lceil \frac{n}{\varepsilon} \rceil$ close
   | $w_{j-1}^{max} :=$ maximum weight of group g_{j-1};
   | $d_j = \frac{1}{\lceil n/\varepsilon \rceil} \frac{w_{j+1}^{min}}{w_j^{max}}$
   | **for** $w_i \in g_j \cup \ldots \cup g_{group}$ **do** $w_i := w_i / d_j$;

4 $\vec{s}:=$ an arbitrary $\{0,1\}^n$ vector;

5 For all i, let $V_i = \{j : \{i,j\} \in E\}$ be the neighborhood of i in G;

  while $\exists i : \sum_{j\in V_i : s_i = s_j} w_j > (1+\varepsilon) \sum_{j\in V_i : s_i \neq s_j} w_j$ **do**

   | $s_i := 1 - s_i$;   ▷ Moves towards equilibrium

6 return $\vec{s}$.

Since for the maximum weight w_n'' it is $w_n'' \leq \lceil \frac{n}{\varepsilon} \rceil^{D_\varepsilon}$, it follows that $\Phi(\vec{s}) \leq m \lceil \frac{n}{\varepsilon} \rceil^{2D_\varepsilon}$. On the other hand whenever an ε-best response move is made by BRIDGEGAPS producing $\vec{s}'$ from some $\vec{s}$, Φ decreases by at least ε, i.e., $\Phi(\vec{s}) - \Phi(\vec{s}') \geq \varepsilon$. This is because, if i is the index flipping bit from $\vec{s}$ to $\vec{s}'$, then by the violation of the $(1+\varepsilon)$-equilibrium condition

$$w_i'' \sum_{j\in V_i : s_i = s_j} w_j'' \geq w_i''(1+\varepsilon) \sum_{j\in V_i : s_i \neq s_j} w_j'' \Rightarrow \sum_{j\in V_i : s_i = s_j} w_i'' w_j'' - \sum_{j\in V_i : s_i' = s_j'} w_i'' w_j'' \geq \varepsilon,$$

since $\sum_{j\in V_i : s_i' = s_j'} w_i'' w_j'' \geq 1$, and

$$\Phi(\vec{s}) - \Phi(\vec{s}') = \sum_{j\in V_i : s_i = s_j} w_i'' w_j'' - \sum_{j\in V_i : s_i' = s_j'} w_i'' w_j'' \geq \varepsilon.$$

Consequently, the last step of the algorithm will do at most $\frac{m \lceil \frac{n}{\varepsilon} \rceil^{2D_\varepsilon}}{\varepsilon}$ ε-best response moves.   □

Lemma 2. *For any vertex-weight graph G and any $\varepsilon > 0$, BRIDGEGAPS returns a $(1+\varepsilon)^3$-approximate equilibrium for NODE-MAX-CUT in G.*

Proof. Clearly, BRIDGEGAPS terminates with a vector $\vec{s}$ that is a $(1+\varepsilon)$-approximate equilibrium for the instance with the w'' weights. It suffices to show that $\vec{s}$ is a $(1+\varepsilon)^2$-approximate equilibrium

for G', i.e., the instance with the w' weights, since this will directly imply that $\vec{s}$ is a $(1+\varepsilon)^3$-approximate equilibrium for G, as already discussed at the beginning of the description of the algorithm.

Consider any index i and let V_i^h be the neighbors of i that belong in the heaviest group among the neighbors of i. By the $(1+\varepsilon)$-approximate equilibrium condition it is

$$\sum_{j \in V_i \setminus V_i^h : s_i = s_j} w_j'' + \sum_{j \in V_i^h : s_i = s_j} w_j'' \leq (1+\varepsilon)\Big(\sum_{j \in V_i \setminus V_i^h : s_i \neq s_j} w_j'' + \sum_{j \in V_i^h : s_i \neq s_j} w_j'' \Big). \tag{3}$$

Recalling that for every j, $w_j'' = \frac{w_j'}{\Pi_{k \in I_j} d_k}$, where I_j contains the indexes of groups below j's group, and letting $D = \Pi_{k \in I_j} d_k$, for a $j \in V_i^h$, gives

$$\sum_{j \in V_i \setminus V_i^h : s_i = s_j} w_j' + \sum_{j \in V_i^h : s_i = s_j} w_j' \leq D\Big(\sum_{j \in V_i \setminus V_i^h : s_i = s_j} w_j'' + \sum_{j \in V_i^h : s_i = s_j} w_j'' \Big) \tag{4}$$

On the other hand for any j and j', if j' belongs in a group lighter than j then by construction $\frac{w_j''}{w_{j'}''} \geq \frac{n}{\varepsilon}$ (recall the way the groups were brought closer), which gives $nw_{j'}'' \leq \varepsilon w_j''$, yielding

$$\sum_{j \in V_i \setminus V_i^h : s_i \neq s_j} w_j'' + \sum_{j \in V_i^h : s_i \neq s_j} w_j'' \leq (1+\varepsilon) \sum_{j \in V_i^h : s_i \neq s_j} w_j'' \tag{5}$$

Using equations (4), (3) and (5), in this order, and that $D \sum_{j \in V_i^h : s_i \neq s_j} w_j'' = \sum_{j \in V_i^h : s_i \neq s_j} w_j' \leq \sum_{j \in V_i : s_i \neq s_j} w_j'$, we get

$$\sum_{j \in V_i : s_i = s_j} w_j' \leq (1+\varepsilon)^2 \sum_{j \in V_i : s_i \neq s_j} w_j'$$

as needed. $\qquad\square$

Remark. We observe the following trade off: we can get a $(1+\varepsilon)^2$-approximate equilibrium if we skip the rounding step at the beginning of the algorithm but then the number of different weights D_ε and thus the running time of the algorithm may increase. Also, if Δ is the maximum degree among the vertices of G, then replacing $\frac{n}{\varepsilon}$ with $\frac{\Delta}{\varepsilon}$ in the algorithm and following a similar analysis gives $O(\frac{m}{\varepsilon}\lceil\frac{\Delta}{\varepsilon}\rceil^{2D_\varepsilon})$ running time.

C   Missing Technical Details from the Proof of Theorem 4

In the following sections, we present all the technical details for the proof of Theorem 4. Recall that our NODE-MAX-CUT instance is composed of the following gadgets:

1. *Leverage* gadgets that are used to transmit nonzero bias to vertices of high weight.

2. Two *Circuit Computing* gadgets that calculate the values and next neighbors of solutions.

3. A *Comparator* gadget.

4. Two *Copy* gadgets that transfer the solution of one circuit to the other, and vice versa.

5. Two *Control* gadgets that determine the (write or compute) mode in which the circuit operates.

Note that whenever we wish to have a vertex of higher weight that dominates all other vertices of lower weight, we multiply its weight with 2^{kN} for some constant k. We then choose N sufficiently large so that, for all k, vertices of weight 2^{kN} dominate all vertices of weight $2^{(k-1)N}$. Henceforth, we will assume N has been chosen sufficiently large.

In what follows, when we refer to the value of the circuit C, we mean the value that the underlying CIRCUIT-FLIP instance C would output, given the same input. Moreover, when we refer to the value of a vertex, we mean the side of the cut the vertex lies on. There are two values, 0 and 1, one for each side of the cut.

As usual in previous work, we assume two *supervertices*, a 1-vertex and a 0-vertex that share an edge and have a huge weight, which dominates the weight of any other vertex, e.g., 2^{1000N}. As a result, at any local optimum, these vertices take complementary values. When we write, e.g., that $Flag = 1$ or $Control = 1$, we mean that $Flag$ or $Control$ takes the same value as the 1-vertex. This convention is used throughout this section, always with the same interpretation. Our construction assumes that certain vertices always take a specific value, either 0 or 1. We can achieve this by connecting such vertices to the *supervertex* of complementary value.

C.1   Outline of the Proof of Theorem 4

We start with a proof-sketch of our reduction, where we outline the key properties of our gadgets and the main technical claims used to establish the correctness of the reduction.

At a high level, the proof of Theorem 4 boils down to showing that at any local optimum of the NODE-MAX-CUT instance of Figure 3, in which $Flag = 1$, the following hold:

1. $I_A = NextB$

2. $NextB = Real\text{-}Next(I_B)$

3. $Real\text{-}Val(I_B) \geq Real\text{-}Val(I_A)$

Once these claims are established, we can be sure that the string defined by the values of vertices in I_B defines a locally optimal solution for CIRCUIT-FLIP. This is because the claims above directly imply that $Real\text{-}Val(I_B) \geq Real\text{-}Val\left(Real\text{-}Next\left(I_B\right)\right)$, which means that there is no neighboring solution of I_B of strictly better value. Obviously, we establish symmetrically the above claims when $Flag = 0$.

In the remainder of this section, we discuss the main technical claims required for the proof of Theorem 4. To do so, we follow a three step approach. We first discuss the behavior of the *Circuit Computing* gadgets C_A and C_B. We then reason why $I_A = NextB$, which we refer to as the *Feedback Problem*. Finally we establish the last two claims, which we refer to as *Correctness of the Outputs*.

Circuit-Computing Gadgets. The *Circuit Computing* gadgets C_A and C_B are the basic primitives of our reduction. They are based on the gadgets introduced by Schäffer and Yannakakis [33] to establish PLS-completeness of MAX-CUT. This type of gadgets can be constructed so as to simulate any Boolean circuit C.
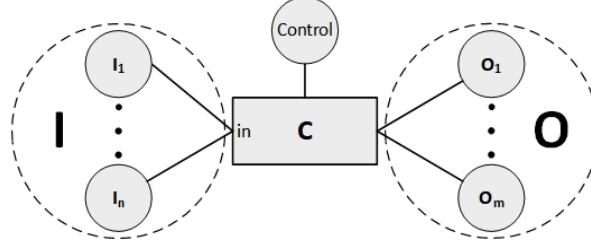
Figure 5: *Circuit Computing* gadgets. The dashed circles, called I and O, represent all input and output vertices, respectively. This type of ("hyper"-)vertex is represented in the rest of the figures with a bold border.

The most important vertices are those corresponding to the input and the output of the simulated circuit C and are denoted as I, O. Another important vertex is *Control*, which allows the gadget to switch between the write and the compute mode of operation. Figure 5 is an abstract depiction of the *Circuit Computing* gadgets.

The main properties of the gadget are described in Theorem 7. Its proof is presented in Section C.3, where the exact construction of the gadget is presented. We recall here the definition of bias, first discussed in Section 5.

Definition (Bias)**.** The *bias* that a vertex i experiences with respect to $V' \subseteq V$ is

$$\left| \sum_{j \in V_i^1 \cap V'} w_j - \sum_{j \in V_i^0 \cap V'} w_j \right| ,$$

where V_i^0 (resp. V_i^1) is the set of neighbors of vertex i on the 0 (resp. 1) side of the cut.

Bias is a key notion in the subsequent analysis. The gadgets presented in Figure 3 are a subset of the vertices of the overall instance. Each gadget is composed by the "input vertices", the internal vertices and the "output vertices". Moreover as we have seen each gadget stands for a "circuit" with some specific functionality (computing, comparing, copying e.t.c.). Each gadget is specifically constructed so as at any local optimum of the overall instance, the output vertices of the gadget experience some bias towards some values that depend on the values of the input vertices of the gadget. Since the output vertices of a gadget may also participate as input vertices at some other gadgets, it is important to quantify the bias of each gadget in order to prove consistency in our instance. Ideally, we would like to prove that at any local optimum the bias that a vertex experiences from a gadget in which it is an output vertex, is greater than the sum of the biases of the gadgets in which it participates as input vertex.

Theorem 7 describes the local optimum behavior of the input vertices I_ℓ and output vertices $Next\ell, Val_\ell$ of the $CircuitComputing$ gadgets C_ℓ.

Theorem 7. *At any local optimum of the* NODE-MAX-CUT *of Figure 3.*

1. *If $Control_\ell = 1$ and the vertices of $Next\ell, Val_\ell$ experience 0 bias from any other gadget beyond C_ℓ then:*

    - $Next\ell = \text{Real-Next}(I_\ell)$

- $\text{Val}_\ell = \text{Real-Val}(I_\ell)$

2. *If $Control_\ell = 0$ then each vertex in I_ℓ experiences 0 bias from the internal vertices of C_ℓ.*

3. *$Control_\ell$ experiences $w_{Control\ell}$ bias from the internal vertices of C_ℓ.*

Case 1 of Theorem 7 describes the *compute mode* of the *Circuit Computing* gadgets. At any local optimum with $ControlA = 1$, and with the output vertices of C_A being indifferent with respect to other gadgets, then C_A computes its output correctly. Note that because the vertices in $NextA, ValA$ are also connected with internal vertices of other gadgets (*CopyA* and *Comparator* gadgets) that may create bias towards the opposite value, the second condition is indispensable. Case 2 of Theorem 7 describes the *write mode*. If at a local optimum $ControlA = 0$ then the vertices in I_A have 0 bias from the C_A gadget and as a result their value is determined by the biases of the *CopyB* gadget and the *Equality* gadget. Case 3 of Theorem 7 describes the minimum bias that the equality gadget must pose to the Control vertices so as to make the computing gadget flip from one mode to the other. As we shall see, the weights $w_{ControlA} = w_{ControlB} = w_{Control}$ are selected much smaller than the bias the $Control\ell$ vertices experience due to the *Equality* gadgets, meaning that the *Equality* gadgets control the *write mode* and the *compute mode* of the *Circuit Computing* gadgets no matter the values of the vertices in C_ℓ gadgets.

Solving the Feedback problem. Next, we establish the first of the claims above, i.e., at any local optimum of NODE-MAX-CUT instance of Figure 3 in which $Flag = 1$, $NextB$ is written to I_A and vice versa when $Flag = 0$. This is formally stated in Theorem 8.

Theorem 8. *Let a local optimum of the NODE-MAX-CUT instance in Figure 3.*

- *If $\text{Flag} = 1$ then $I_A = \text{NextB}$*

- *If $\text{Flag} = 0$ then $I_B = \text{NextA}$*

We next present the necessary lemmas for proving Theorem 8.

Lemma 3. *Let a local optimum of the NODE-MAX-CUT instance in Figure 3. Then,*

- *$ControlA = (I_A = T_B)$*

- *$ControlB = (I_B = T_A)$*

In Section C.4, we present the construction of the *Equality* gadget. This gadget is specifically designed so that at any local optimum, its internal vertices generate bias to $ControlA$ towards the value of the predicate $(I_A = T_B)$. Notice that if we multiply all the internal vertices of the equality gadget with a positive constant, the bias $ControlA$ experiences towards value $(I_A = T_B)$ is multiplied by the same constant (see Definition). Lemma 3 is established by multiplying these weights with a sufficiently large constant so as to make this bias larger than $w_{ControlA}$. We remind that by Theorem 7, the bias that $ControlA$ experiences from C_A is $w_{ControlA}$. As a result, the local optimum value of $ControlA$ is $(I_A = NextB)$ no matter the values of $ControlA$'s neighbors in the C_1 gadget. The *red mark* between $ControlA$ and the C_A gadget in Figure 3 denotes the "indifference" of $ControlA$ towards the values of the C_A gadget (respectively for $ControlB$).

In the high level description of the NODE-MAX-CUT instance of Figure 3, when $Flag = 1$ the values of $NextB$ is copied to I_A as follows: At first T_B takes the value of $NextB$. If $I_A \neq T_B$ then $ControlA = 0$ and the C_A gadget switches to *write mode*. Then the vertices in I_A takes the values of the vertices in $NextB$. This is formally stated in Lemma 4.

28

Lemma 4. *At any local optimum point of the* NODE-MAX-CUT *instance of Figure 3:*

- *If* $\text{Flag} = 1$, *i.e.* NextB *writes on* I_A, *then*

    1. $T_B = \text{NextB}$
    2. *If* $Control A = 0$ *then* $I_A = T_B = \text{NextB}$

- *If* $\text{Flag} = 0$ *i.e.* NextA *writes on* I_B, *then*

    1. $T_A = \text{NextA}$
    2. *If* $Control B = 0$ *then* $I_B = T_A = \text{NextA}$

In Section C.5, we present the construction of the *Copy* gadgets. At a local optimum where $Flag = 1$, this gadget creates bias to the vertices in I_A, T_B vertices towards adopting the values of $NextB$. Since I_A, T_B also participate in the *Equality* gadget in order to establish Lemma 4 we want to make the bias of the *CopyB* gadget larger than the bias of the *Equality* gadget. This is done by again by multiplying the weights of the internal vertices of *CopyB* with a sufficiently large constant. The "indifference" of the vertices in I_A, T_B with respect to the values of the internal vertices of the *Equality* gadget is denoted in Figure 3 by the *red marks* between the vertices in I_A, T_B and the *Equality* gadget.

In Case 2 of Lemma 4 the additional condition $Control A = 0$ is necessary to ensure that $I_A = NextB$. The reason is that the bias of the *Copy* gadget to the vertices in I_A is sufficiently larger than the bias of the *Equality* gadget to the vertices in I_A, but not necessarily to the bias of the C_A gadget. The condition $Control A = 0$ ensures 0 bias of the C_A gadget to the vertices I_A, by Theorem 7. As a result the values of the vertices in I_A are determined by the values of their neighbors in the *CopyB* gadget.

Proof of Theorem 8. Let a local optimum in which $Flag = 1$. Let us assume that $I_A \neq NextB$. By Case 1 of Lemma 4, $T_B = NextB$. As a result, $I_A \neq T_B$, implying that $Control A = 0$ (Lemma 3). Now, by Case 2 of Lemma 4 we have that $I_A = NextB$, which is a contradiction. The exact same analysis holds when $Flag = 0$. □

Correctness of the Output Vertices. In the previous section we discussed how the *Feedback problem* ($I_A = NextB$ when $Flag = 1$) is solved in our reduction. We now exhibit how the two last cases of our initial claim are established.

Theorem 9. *At any local optimum of the instance of* NODE-MAX-CUT *of Figure 3:*

- *If* $\text{Flag} = 1$

    1. Real-Val$(I_A) \leq$ Real-Val(I_B)
    2. $\text{NextB} = \text{Real-Next}(I_B)$

- *If* $\text{Flag} = 0$

    1. Real-Val$(I_B) \leq$ Real-Val(I_A)
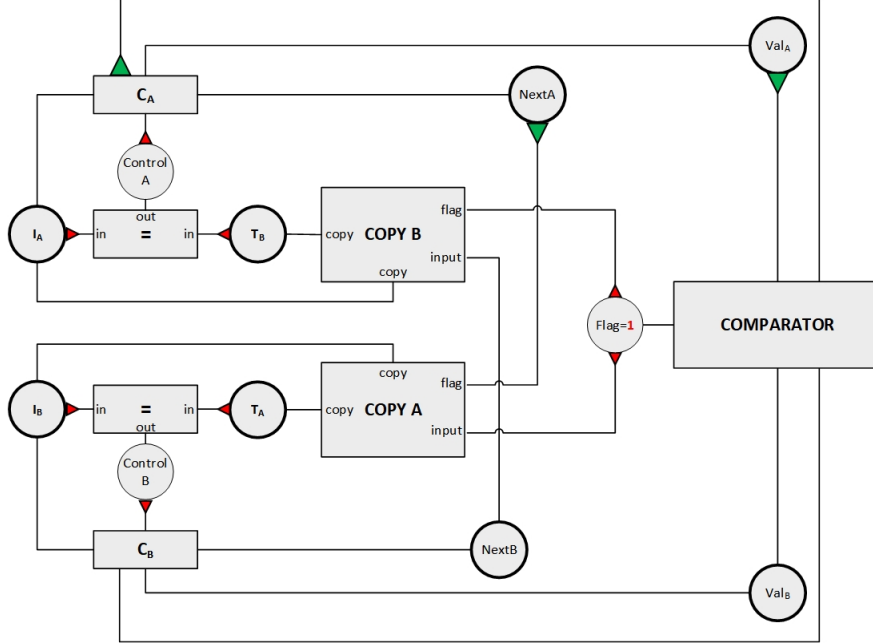   2. $\text{NextA} = \text{Real-Next}(I_A)$

29

Figure 6: Since $Flag = 1$, any internal vertex of the C_B gadget has 0 bias with respect to all the other gadgets. As a result, Theorem 7 applies.

At first we briefly explain the difficulties in establishing Theorem 9. In the following discussion we assume that $Flag = 1$, since everything we mention holds symmetrically for $Flag = 0$. Observe that if $Flag = 1$ we know nothing about the value of $ControlB$ and as a result we cannot guarantee that $NextB = Real\text{-}Next(I_B)$ or $Val_B = Real\text{-}Val(I_B)$. But even in the case of C_A where $ControlA = 1$ due to Theorem 8, the correctness of the vertices in $NextA$ or Val_A cannot be guaranteed. The reason is that in order to apply Theorem 7, $NextA$ and Val_A should experience 0 bias with respect to any other gadget they are connected to. But at a local optimum, these vertices may select their values according to the values of their *heavily weighted neighbors* in the $CopyA$ and the $Comparator$ gadget.

The correctness of the values of the output vertices, i.e. $NextA = Real\text{-}Next(I_A)$ and $Val_A = Real\text{-}Val(I_A)$, is ensured by the design of the $CopyA$ and the $Comparator$ gadgets. Apart from their primary role these gadgets are specifically designed to cause 0 bias to the output vertices of the *Circuit Computing* gadget to which the better neighbor solution is written. In other words at any local optimum in which $Flag = 1$ and any vertex in C_A: the total weight of its neighbors (belonging in the $CopyA$ or the $Comparator$ gadget) with value 1 equals the total weight of its neighbors (belonging in the $CopyA$ or the $Comparator$ gadget) with value 0.

The latter fact is denoted by the *green marks* in Figure 6 and permits the application of Case 1 of Theorem 7. Lemma 5 and 6 formally state these "green marks".

Lemma 5. *At any local optimum of the* NODE-MAX-CUT *instance of Figure 3:*

- *If* Flag $= 1$, *then any vertex in* NextA *experience* 0 *bias with respect to the* CopyA *gadget.*

- *If* Flag $= 0$ *then then any vertex in* NextB *experience* 0 *bias wrt. the* CopyB *gadget.*

Lemma 6. *Let a local optimum of the instance of* NODE-MAX-CUT *of Figure 3:*

- *If* Flag *= 1 then all vertices of* C_A *experience* 0 *bias wrt. the* Comparator *gadget.*

- *If* Flag *= 0 then all vertices of* C_B *experience* 0 *bias wrt. the* Comparator *gadget.*

Remark. The reason that in Lemma 6 we refer to all vertices of C_A (respectively C_B) and not just to the vertices in Val_A (respectively Val_B) is that in the constructed instance of NODE-MAX-CUT of Figure 3, we connect internal vertices of the C_A gadget with internal vertices of the *Comparator* gadget. This is the only point in our construction where internal vertices of different gadgets share an edge and is denoted in Figure 3 and 6 with the direct edge between the C_A gadget and the *Comparator* gadget.

Now using Lemma 5 and Lemma 6 we can prove the correctness of the output vertices $NextA$, Val_A when $Flag = 1$ i.e. $NextA = Real\text{-}Next(I_A)$ and $Val_A = Real\text{-}Val(I_B)$ (symmetrically for the vertices in $NextB$, Val_B when $Flag = 0$).

Lemma 7. *Let a local optimum of the instance of* NODE-MAX-CUT *of Figure 3:*

- *If* Flag $= 1$ *then* NextA $= $ Real-Next(I_A), Val$_A = $ Real-Val(I_A).

- *If* Flag $= 0$ *then* NextB $= $ Real-Next(I_B), Val$_B = $ Real-Val(I_B).

Proof. We assume that $Flag = 1$ (for $Flag = 0$ the exact same arguments hold). By Theorem 8 we have $I_A = NextB$ and by Lemma 4 we have that $T_B = NextB$. As a result, $I_A = T_B$ and by Lemma 3 $ControlA = 1$. Lemma 5 and Lemma 6 guarantee that the vertices in $NextA$, Val_A of C_A experience 0 bias towards all the other gadgets of the construction and since $ControlA = 1$, we can apply Case 1 of Theorem 7 i.e. $Val_A = Real\text{-}Val(I_A)$ and $NextA = Real\text{-}Next(I_A)$. □

Up next we deal with the correctness of the values of the output vertices in Val_B and $NextB$ when $Flag = 1$. We remind again that, even if at a local optimum $ControlB = 1$, we could not be sure about the correctness of the values of these output vertices due to the bias their neighbors in the $CopyB$ and the *Comparator* gadget (Theorem 7 does not apply). The *Comparator* gadget plays a crucial role in solving this last problem. Namely, it also checks whether the output vertices in $NextB$ have correct values with respect to the input I_B and if it detects incorrectness it outputs 0. This is done by the connection of some specific internal vertices of the C_A, C_B gadgets with the internal vertices of the *Comparator* gadget (Figure 3: edges between C_A, C_B and *Comparator*).

Lemma 8. *At any local optimum of the* NODE-MAX-CUT *instance of Figure 3:*

- *If* Flag $= 1$ *then* NextB $= $ Real-Next(I_B)

- *If* Flag $= 0$ *then* NextA $= $ Real-Next(I_A)

We highlight that the correctness of values of the output vertices $NextB$, i.e. $NextB = Real\text{-}Next(I_B)$, is not guaranteed by application of Theorem 7 (as in the case of correctness of $NextA, Val_A$), but from the construction of the *Comparator* gadget. Lemma 8 is proven in Section C.6 where the exact construction of this gadget is presented. Notice that Lemma 8 says nothing about the correctness of the values of the output vertices in Val_B. As we latter explain this cannot be guaranteed in our construction. Surprisingly enough, the *Comparator* outputs the right outcome of the predicate

31

($Real\text{-}Val(I_A) \leq Real\text{-}Val(I_B)$) even if $Val_B \neq Real\text{-}Val(I_B)$. The latter is one of our main technical contributions in the reduction that reveals the difficulty of Node-Max-Cut. The crucial differences between our *Comparator* and the *Comparator* of the previous reductions [33, 20, 9] are discussed in the end of the section. Lemma 9 formally states the robustness of the outcome of the *Comparator* even with "wrong values" in the vertices of Val_B and is proven in Section C.6.

Lemma 9. *At any local optimum of the* Node-Max-Cut *instance of Figure 3:*

- *If* $Flag = 1$, $NextA = Real\text{-}Next(I_A)$, $Val_A = Real\text{-}Val(I_A)$ *and* $NextB = Real\text{-}Next(I_B)$ *then* $Real\text{-}Val(I_A) \leq Real\text{-}Val(I_B)$.

- *If* $Flag = 0$, $NextB = Real\text{-}Next(I_B)$, $Val_B = Real\text{-}Val(I_B)$ *and* $NextA = Real\text{-}Next(I_A)$ *then* $Real\text{-}Val(I_B) \leq Real\text{-}Val(I_A)$.

We are now ready to prove Theorem 9.

Proof of Theorem 9. Let a local optimum of the instance of Figure 3 with $Flag = 1$ (respectively for $Flag = 0$). By Lemma 8, $NextB = Real\text{-}Next(I_B)$ and thus Case 1 is established. Moreover by Lemma 7, $NextA = Real\text{-}Next(I_A)$ and $Val_A = Real\text{-}Val(I_A)$. As a result, Lemma 9 applies and $Real\text{-}Val(I_A) \leq Real\text{-}Val(I_B)$ (Case 2 of Theorem 9) □

Putting Everything Together. Having established Theorem 8 and 9, the *PLS*-completeness of Node-Max-Cut follows easily. For the sake of completeness, we show how we can put everything together and conclude the proof Theorem 4.

Proof of Theorem 4. For a given circuit C of the Circuit-Flip, we can construct in polynomial time the instance of Node-Max-Cut of Figure 3. Let a local optimum of this instance. Without loss of generality, we assume that $Flag = 1$. Then, by Theorem 8 and Theorem 9, $I_A = NextB$, $NextB = Real - Next(I_B)$ and $Real - Val(I_A) \leq Real - Val(I_B)$. Hence, we have that

$$
\begin{aligned}
Real - Val(I_B) &\geq Real - Val(I_A) \\
&= Real - Val(NextB) \\
&= Real - Val\left(Real - Next(I_B)\right)
\end{aligned}
$$

But if $I_B \neq Real - Next(I_B)$, then $Real - Val(I_B) > Real - Val\left(Real - Next(I_B)\right)$ which is a contradiction. Thus $I_B = Real - Next(I_B) = I_B$, meaning that the string defined by the values of I_B is a locally optimal solution for the Circuit-Flip problem. □

In the following sections, we present the gadget constructions in detail and all the formal proofs missing from this section.

C.2 The Leverage Gadget

The *Leverage* gadget is a basic construction in the PLS completeness proof. This gadget solves a basic problem in the reduction. Suppose that we have a vertex with relatively small weight A and we want to bias a vertex with large weight B. For example, the large vertex might be indifferent towards its other neighbors, which would allow even a small bias from the small vertex to change
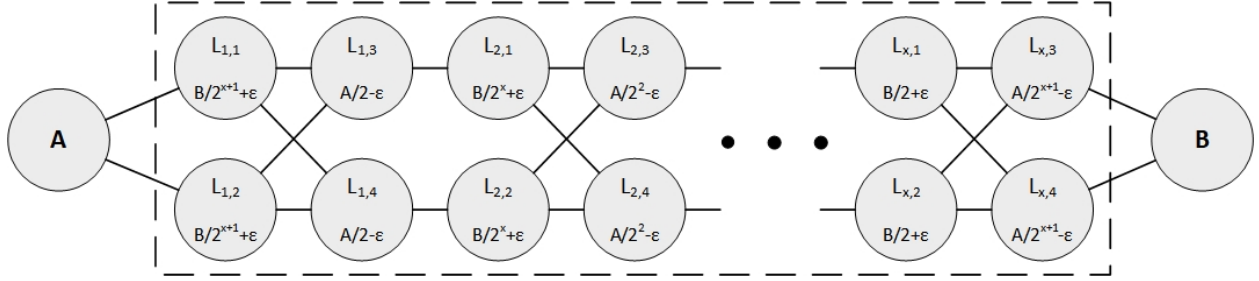
Figure 7: The *Leverage* Gadget.

its state. We would also like to ensure that the large vertex does not bias the smaller one with very large weight, in order for the smaller to retain its value.

This problem arises in various parts of the PLS proof. For example, we would like the outputs of a circuit to be fed back to the inputs of the other one. The outputs have very small weight compared to the inputs, since the weights drop exponentially in the *Circuit Computing* gadget. We would like the inputs of circuit B to change according to the outputs of circuit A and not the other way around. Another example involves the *Equality* Gadget, which influences the $Control_\ell$ of the *Circuit Computing* gadget. The vertices of the Gadget have weights of the order of 2^{10N}, while the control vertices of the *Circuit Computing* gadget are of the order of 2^{100N}. We would like the output of the gadget to bias the $Control_\ell$ vertices, while also remaining independent from them.

Let's get back to the original problem. A naive solution would be to connect vertex A directly with vertex B. However, this would result in vertex B biasing vertex A due to the larger weight it possesses. For example, if we connected $Control_A$ with the control variables of circuit B, then they would always bias $Control_A$ with a very large weight, rendering the entire *Equality* gadget useless. We would like to ensure that vertex A biases B with a relatively small weight, while also experiencing a small bias from it.

The solution we propose is a *Leveraging* gadget that is connected between vertices A and B. It's construction will depend on the weights A and B, as well as the bias that we would like B to experience from A. Before describing the construction, we discuss it's functionality on a high level.

As shown in Figure 7, we place the gadget between the vertices A and B. We use two parameters x, ε in the construction. We first want to ensure that vertex A experiences a small bias from the gadget. This is why we put vertices $L_{1,1}, L_{1,2}$ at the start with weight $B/2^{x+1} + \varepsilon$, which puts a relatively small bias. We want these vertices to be dominated by A. This is why vertices $L_{1,3}, L_{1,4}$ have combined weight less than A. However, these vertices cannot directly influence B, since it's weight dominates the weights of $L_{1,1}, L_{1,2}$. For this reason, we repeat this construction $x+1$ times, until vertices $L_{x,1}, L_{x,2}$, whose combined weight is slightly larger than B. This means that vertices $L_{x,3}, L_{x,4}$ are not dominated by B and can therefore be connected directly with it. The details of the proof are given below.

Lemma 10. *If the input vertex A of a leverage gadget with output vertex B, parameters x, ε, has value 1, then the output vertex experiences bias $w_A/2^x + 2\varepsilon$ towards 0, while the input vertex A experiences bias $w_B/2^x - 2\varepsilon$ towards 1. If A has value 0, then B experiences the same bias towards 1, while A is biased towards 0.*

Proof. We first consider the vertices $L_{1,1}, L_{1,2}$. They both experience bias w_A towards the opposite value of A, which is greater than the remaining weight of their neighbors $2w_A - 2\varepsilon$, and hence they
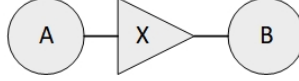
33

Figure 8: *Leveraging Gadget* notation

are both dominated to take the opposite value of A. Similarly, the vertices $L_{1,3}, L_{1,4}$ are now biased to take the opposite values of $L_{2,1}, L_{2,2}$ with bias at least $w_B/2^x + 2\varepsilon$, which is greater than the remaining neighbors of $w_B/2^x + \varepsilon$. Hence, both $L_{1,3}, L_{1,4}$ have the same value as A in any local optimum. In a similar way, we can prove that, in any local optimum $L_{i,3} = L_{i,4} = A$, and therefore B experiences bias $w_A/2^x + 2\varepsilon$ towards the opposite value of A, while A experiences bias at most $w_B/2^x - 2\varepsilon$ from this gadget. $\qquad\square$

Note that the above lemma works for any value of ε. This means that we can make the bias that B experiences arbitrarily close to $w_B/2^x$. For all cases where such a *Leverage* gadget is used, it is implied that $\varepsilon = 2^{-1000N}$ which is smaller than all other weights in the construction. Hence, we only explicitly specify the x parameter and, for simplicity, such a *Leverage* gadget is denoted as below schematically.

C.3   The Circuit Computing Gadget

Each of the two computing circuits is meant to both calculate the value of the underlying CIRCUIT-FLIP instance, as well as the best neighboring solution. For technical reasons one of the two circuits will need to output the complement of the value instead of the value itself, so that comparison can be achieved later with a single vertex.

In this section we present the gadgets that implement the above circuits in a NODE-MAX-CUT instance. The construction below is similar to the constructions of Schäffer and Yannakakis used to prove MAX-CUT PLS-complete [33]. Since NOR is functionally complete we can implement any circuit with a combination of NOR gates. In particular, each NOR gate is composed of the gadgets below. Each such gadget is parameterized by a variable n, and a NOR gadget with parameter n is denoted NOR(n). Since we wish for earlier gates to dominate later gates we order the gates in reverse topological order, so as to never have a higher numbered gate depend on a lower numbered gate. The ith gate in this ordering corresponds to a gadget $NOR(2^{N+i})$. Note that the first gates of the circuit have high indices, while the final gates have the least indices.

We take care to number the gates so that the gates that each output the final bit of the value of the circuit are numbered with the n lowest indexes, i.e. the gate of the kth bit of the value corresponds to a gate $NOR(2^{N+k})$. This is necessary so that their output vertices can be used for comparing the binary values of the outputs.

The input vertices of these gadgets are either an input vertex to the whole circuit or they are the output vertex of another NOR gate, in which case they have the weight prescribed by the previous NOR gate. The input vertices of the entire circuit (which are not the output vertices of any NOR gate) are given weight 2^{5N}.

Moreover, we have y_i^1, z_i^1 vertices which are meant to bias the internal vertices of each gadget and determine its functionality. Specifically, $a_i^1, a_i^2, c_i^1, c_i^2, v_i, b_i^3$ are biased to have the same value as y_i^1, while $b_i^1, b_i^2, d_i^1, d_i^2, c_i^3$ are biased to have the same value as z_i^1. This is achieved by auxiliary vertices of weight 2^{-200N}, shown in Figure 10.
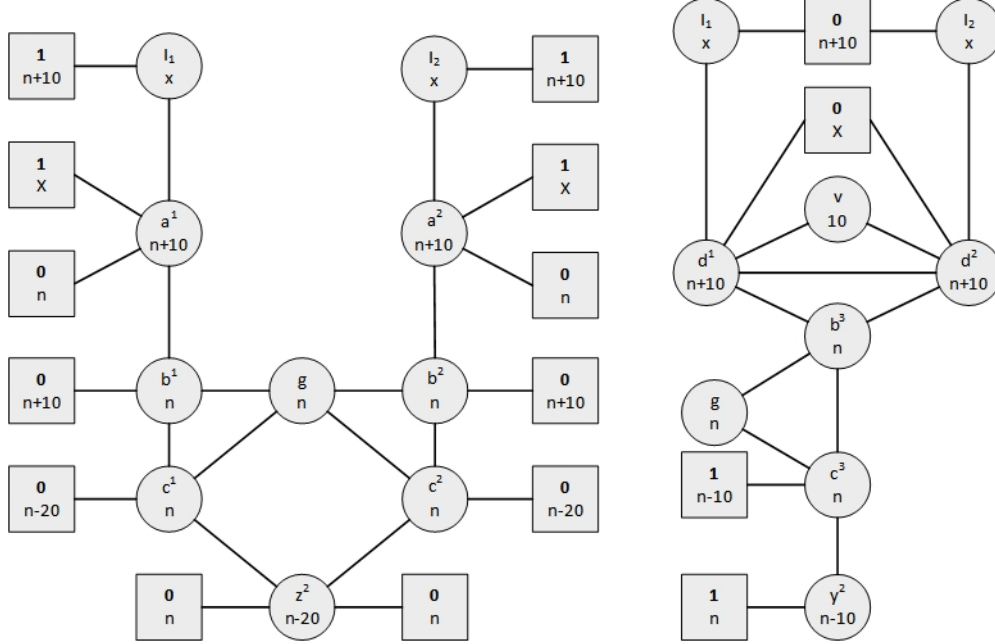
Figure 9: The NODE-MAX-CUT instance implementing a NOR(n) gadget.

We also have auxiliary vertices ρ of weight 2^{-500N} that bias the output vertex g_i to the correct NOR output value. Note that these vertices have the lowest weight in the entire construction.

These control vertices, y^1, z^1, y^2, z^2 are meant to decide the functionality of the gadget. We say that the y, z vertices have their natural value when $y = 1$ and $z = 0$. We say they have their unnatural value when $y = 0$ or $z = 0$. In general, when these vertices all have their natural values the NOR gadget is calculating correctly and when they have their unnatural values the circuit's inputs are indifferent to the gadget.

Unlike Schäffer and Yannakakis [33], we add two extra control variable vertices y^3, z^3 to each such NOR gadget, both of weight $n - 50$. The reason is to ascertain that in case of incorrect calculation at least one y variable will have its unnatural value. Otherwise, it would be possible, for example, to have an incorrect calculation with only z^2 being in an unnatural state.

These NOR gadgets are not used in isolation, but instead compose a larger computing circuit. As Schäffer and Yannakakis do ([33]), we connect each of the control variables z^i, y^i of the above construction so as to propagate their natural or unnatural values depending on the situation. The connection of these gadgets is done according to the ordering we established earlier. Recall that the last m gates correspond to gadgets calculating the value bits, the n gates before them correspond to the output gates of the next neighbor, and the rest are internal gates of the circuit.

These gadgets' function is twofold. Firstly, they detect a potential error in a NOR calculation and propagate it to further gates, if the control variables have their unnatural values. Second, if the control variables have their unnatural values, they insulate the inputs so that they are indifferent to the gadget and can be changed by any external slight bias.

Furthermore, all the vertices of these gadgets are all multiplied by a 2^{100N} weight, except the vertices of the NOR gadget corresponding to the final bits of the value which are multiplied by 2^{90N}. This is so that a possible error in the calculation of the next best neighbor supersedes any
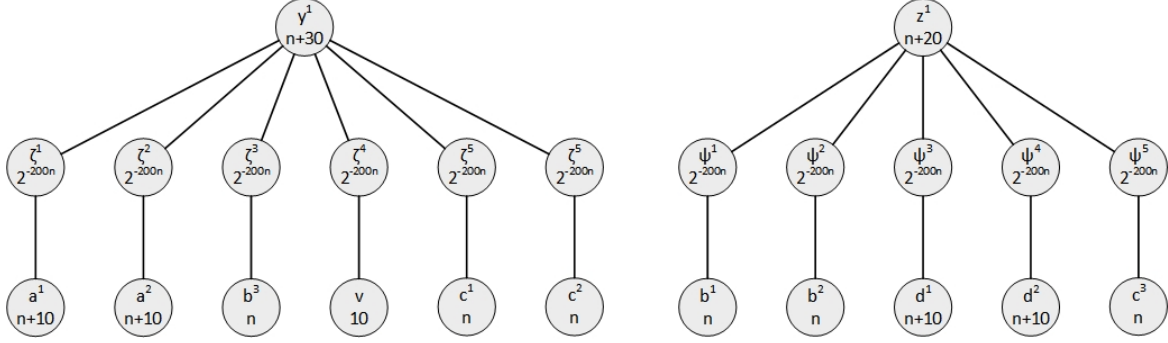
Figure 10: Local bias to internal vertices from y_i^1, z_i^1
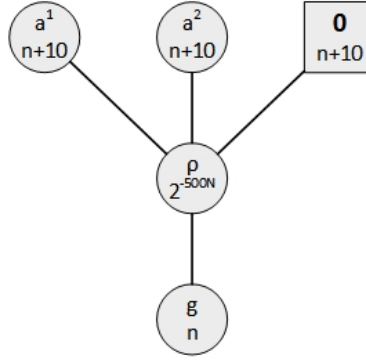

Figure 11: Extremely small bias to NOR output value.

possible result of the comparison. The auxiliary vertices introduced above, which are meant to induce small biases to internal vertices, are not multiplied by anything.

Lastly, for technical simplicity, we have a single vertex for each computing circuit meant to induce bias to all control variable vertices y, z at the same time. The topology of the connection is presented below.

We now prove the properties of these gadgets.

Lemma 11. *At any local optimum, if $z_i^1 = 1$ and $y_i^1 = 0$, then $I_1(g_i), I_2(g_i)$ are indifferent with respect to the gadget G_i.*

Proof. Since $z_i^1 = 1$ and $y_i^1 = 0$, by the previous lemma, $z_i^2 = 1$ and $y_i^2 = 0$ Since $y_i^1 = 0$, $a_i^1, a_i^2, c_i^1, c_i^2, v, b_i^3$ have an $\epsilon = 2^{-200N}$ bias towards 0. Since $z_i^1 = 1$, d, c_i^3, b_i^1, b_i^2 have an $\epsilon = 2^{-200N}$ bias towards 1. Assume $g_i = 0$. Then b_i^1 has bias at least $2^{100N} \cdot (2 \cdot 2^i + 10) + 2^{-200N}$ towards 1 which dominates his best response. Hence, $b_i^1 = 1$ in this case. Now a_i^1 has bias at least $w(I1) + 2^{100N} \cdot 2^i + 2^{-200N}$ towards 0 which also dominates. Therefore, $a_i^1 = 0$. Similarly, $a_i^2 = 0$. Moreover, c_i^3 has y_i^2 and g as neighbors which are both 0 so it can take its preferred value of $c_i^3 = 1$. Assume both d_i^1, d_i^2 are 0. Then $v = 1$ and $b_i^3 = 1$ and hence at least one of d_i^1, d_i^2 would have incentive to change to 1. If $d_i^1 = 0, d_i^2 = 1$ then $v = 0$ due to its 2^{-200N} bias. Also, $b_i^3 = 0$ because $d_i^1 = 0$, $d_i^2 = 1$, $g = 0$, $c_i^3 = 1$ balance each other out $b_i^3 = 0$ due to its 2^{-200N} bias. Since d_i^1 experiences at least $2^{100N} \cdot 2 \cdot 2^i + 10 + w(I_i^1) + 2^{-200N}$ bias towards 1 it can only be $d_i^1 = 1$ in local optimum. Hence, if $g = 0 \implies d_i^1 = d_i^2 = 1, a_i^1 = a_i^2 = 0$. Assume $g_i = 1$. Then c_i^1 experiences bias
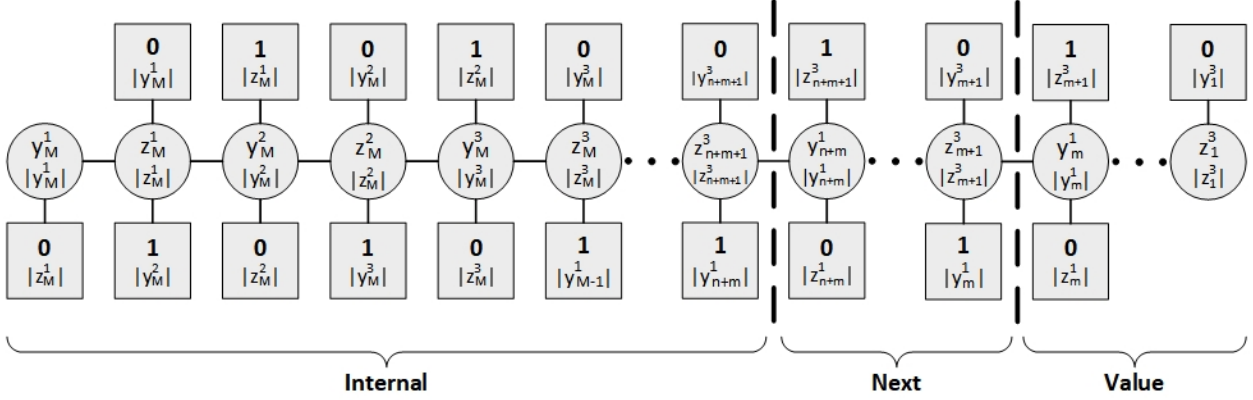
36

Figure 12: Connecting the control vertices of the NOR gadgets. Recall that M is the number of total gates in the circuit, n is the number of solution bits and m is the number of value bits. Note that the gates are ordered in reverse, i.e the first gates have highest index.
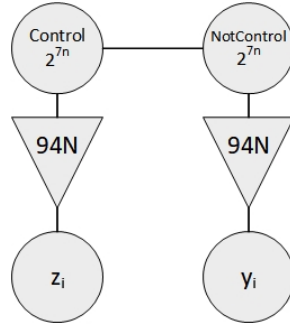

Figure 13: We use a single vertex Control to bias all control vertices y, z. Note that this vertex is connected with the y, z vertices through leverage gadgets

towards 0 from g_i and z_i^2 which together with the 2^{-200N} bias from y_i^1 means that his dominant strategy is to take the value 0. Now b_i^1 experiences bias from $c_i^1 = 0$ towards 1 as well as bias $2^{100N} \cdot 2^i + 10$ towards 1. Along with the 2^{-200N} bias from z_i^1 we have that $b_i^1 = 1$ in any local optimum. Similarly, $b_i^2 = 1$ by symmetry. Hence, in this case as well a_i^1 is 0 in any local optimum. Similarly, we get that $a_i^2 = 0$ in any local optimum.

Assume both $d_i^1 = d_i^2 = 0$ then, as above, we have that $b_i^3 = 1$ and hence at least one of the d would gain the edge of weight 2^{-200N} by taking the value 1. Hence, at least one d is equal to 1 and $b_i^3 = 0$ since it is indifferent with respect to g, c_i^3. Since v is now indifferent with respect to $d_i^1 = 0, d_i^2 = 1$ it takes its preferred value $v = 0$. Since $b_i^3 = v = 0$ we have that both d_i^1, d_i^2 must take their preferred values $d_i^1 = d_i^2 = 1$ in any local optimum.

In both cases both $a_i^1 = a_i^2 = 0, d_i^1 = d_i^2 = 1$ and hence $I_1(g_i), I_2(g_i)$ are indifferent with respect to the gadget. $\qquad \square$

Lemma 12. *If gate G_i is incorrect, then $z_i^2 = 1$. If $y_i^2 = 0$ then $z_i^2 = 1$. If $z_i^2 = 1$, then for all $j < i$ $z_j^1 = z_j^2 = z_j^3 = 1$ and $y_j^1 = y_j^2 = y_j^3 = 0$.*

Proof. There are two possibilities if G_i is incorrect. Either one of the inputs $I_1(g_i), I_2(g_i)$ is 1 and

$g_i = 1$ or both $I_1(g_i) = I_2(g_i) = 0$ and $g_i = 0$.

In the first case, without loss of generality we have that $I_1(g_i) = 1$. This means that vertex a_i^1 is biased towards value 0 with weight at least $2 \cdot 2^{i+1} \cdot 2^{100N}$ by $I_1(g_i)$ and constant vertex 1. This bias is greater than the weight of all the other neighbors of a_i^1 combined. Hence, in local optimum, $a_i^1 = 0$. Hence, vertex b_i^1 is biased towards value 1 with weight at least $2 \cdot |a_i^1|$, which is greater than the total weight of all the other neighbors of b_i^1 combined. Hence, $b_i^1 = 1$. Similarly, we can argue that $a_i^2 = 0$ and $b_i^2 = 1$ if $I_2(g_i) = 1$.

Since $b_i^1 = 1$ and $g_i = 1$, vertex c_i is biased towards 0 with weight at least $2 \cdot 2^i \cdot 2^{100N}$, which is greater than the total weight of all the other neighbors of c_i^1 combined. Hence, $c_i^1 = 0$. We now focus on vertex z_i^2. Its neighbors are two vertices of weight $2^i \cdot 2^{100N}$ with constant value 0, vertices c_i^1, c_i^2, y_i^3 and a constant vertex 1 with weights $2^{100N} \cdot (2^i - 50)$ and some auxiliary vertices of negligible weight. If $c_i^1 = 0$, then z_i^2 is biased towards 1 with weight at least $3 \cdot 2^i \cdot 2^{100N}$, which is greater than the weight of the remaining neighbors combined. Hence, in local optimum, $z_i^2 = 1$. Hence, the claim has been proved in this case. If $I_2(g_i) = 1$, the proof is analogous.

Now suppose $I_1(g_i) = I_2(g_i) = 0$ and $g_i = 0$. Since $I_1(g_i) = 0$, vertex d_i^1 is biased towards 1 with weight at least $2 \cdot 2^{i+1} \cdot 2^{100N}$, which is greater that the weight of all its other neighbors combined. Hence, $d_i^1 = 1$. Similarly, we can prove that $d_i^2 = 1$. This means that vertex b_i^3 is biased towards 0 with weight at least $2 \cdot (2^i + 10) \cdot 2^{100N}$, which is greater than the weight of its other vertices combined. This implies that $b_i^3 = 0$. By the same reasoning, $v_i = 0$. Since $b_i^3 = g_i = 0$, vertex c_i^3 is biased towards 1 with weight at least $2 \cdot 2^i \cdot 2^{100N}$, which is greater than the weight of its other vertices combined. Hence, $c_i^3 = 1$. Now we focus on vertex y_i^2. Its neighbors are a vertex of weight $2^i \cdot 2^{100N}$ with constant value 1, vertex c_i^3 with weight $2^i \cdot 2^{100N}$, z_i^1, a constant vertex 1 both with weight $2^{100N} \cdot (2_2^i 0)$, z_i^2 and a constant 0 with weight $2^{100N} \cdot 2^i - 10$ and some auxiliary vertices of negligible weight. Hence, y_i^2 is biased towards 0 with weight at least $2 \cdot 2^i \cdot 2^{100N}$, which is greater than the weight of the remaining neighbors combined. Hence, $y_i^2 = 0$.

We are now going to prove that if $y_i^2 = 0$, then $z_i^2 = 1$, which concludes the proof for this case and is also the second claim of the lemma. We first notice that z_i^2 is never biased towards 0 by the vertices of the NOR gadget. Hence, if the bias of the remaining vertices is towards 1, then $z_i^2 = 1$ in local optimum. We notice that vertices y_i^2 and constant vertex 0 bias our vertex with weight $2 \cdot (2^i - 10) \cdot 2^{100N}$, which is greater that any potential bias by vertices y_i^3 and constant 1 in the chain of total weight $2 \cdot 2^{100N} \cdot (2^i - 50)$. Hence, $z_i^2 = 1$.

It remains to prove the last claim of the Lemma. It suffices to show that when a z_i in the chain is 1, the next y_{i+1} will be 0 and the claim will follow inductively. By a similar argument to the one used for the second claim, vertex y_{i+1} is not biased towards 1 by any vertex in the NOR gadget. However, it experiences bias towards 0 from vertex z_i and constant vertex 1, which is greater than any other potential bias from its other neighbors. Hence, $y_{i+1} = 0$ and the claim follows. □

Lemma 13. *Suppose $z_i^1 = 0$ and $y_i^1 = 1$. If g_i is correct then z^2 and y^2 are indifferent with respect to the other vertices of the gate G_i. If g_i is incorrect then g_i is indifferent with respect to the other vertices of the gate G_i, but gains the vertex ρ of weight 2^{-500N}.*

Proof. Assume g_i is corrrect.

Assume at least one of $I_1(g_i), I_2(g_i)$ is equal to 1, say $I_1(g_i) = 1$, hence at least one of d_i^1, d_i^2, assume d_i^1, is equal to 0. This is because otherwise it would experience bias from its neighbors $I_1(g_i), d_i^2$ towards 0 which, along with the bias from the auxiliary vertex between y_i^1 and d_i^1, would dominate it towards 0. Therefore, b_i^3 experiences bias towards 1 from both g_i, which is correct, and

d_i^1, which means, along with the bias from y_i^1, its equal to 1. Hence, c_i^3 must be equal to 0, since it is dominated by the bias from b_i^3, the constant vertex of 1 and its auxiliary bias from z_i^1. Hence, c_i^3 is 0 and y_i^2 is indifferent.

Assume $I_1(g_i) = 0, I_2(g_i) = 0$. Hence, $d_i^1 = 1, d_i^2 = 1$, which means that $b_i^3 = 0$. Since g is correct, it must be $g = 1$, and therefore $c_i^3 = 0$, since it can take its preferred value of 0, towards which it is biased by z_i^1. Therefore, y_i^2 is indifferent to this gadget.

Moreover, since $g_i = 0$ it must be that $c_i^1 = c_i^2 = 1$ since they both have g_i and a constant 0 vertex as their neighbors, which along with the bias from y_i, dominates their bias. Since z_i^2 neighbors with two 1 vertices and two 0 vertices it is indifferent with respect to this gadget.

In all cases, when g_i is correct, y_i^2, z_i^2 are indifferent.

Assume g_i is incorrect.

Assume that at least one of $I_1(g_i), I_2(g_i)$ is equal to 1. Similarly to above, $d_i^1 = 0$. Since, g_i is incorrect c_i^3 will be 0 due to the bias from g_i, the constant vertex 1 and z_i^1. Hence, $b_i^3 = 1$. The vertex g_i is therefore indifferent with respect to this gadget.

Furthermore, since $I_1(g_i) = 1$, $a_i^1 = 0$ and $b_i^1 = 1$. Since $g_i = 1$ we have that $c_i^3 = 0$. Also, since $I_2(g_i) = 0$, a_i^2 must take its preferred value of 1, and hence b_i^2 takes its preferred value of 0. Similarly, c_i^2 can also take its preferred value of 1. Overall, g_i is connected to $b_i^1 = 1, c_i^1 = 0, b_i^2 = 0, c_i^2 = 1$ and hence is indifferent

Assume both $I_1(g_i) = I_2(g_i) = 0$. Then $d_i^1 = d_i^2 = 1$, which means that $b_i^3 = 0$, and since $g_i = 0$, we have that $c_i^3 = 1$. Hence, g_i is indifferent with respect to this gadget.

Because $I_1(g_i) = I_2(g_i) = 0$, we have that $a_i^1 = a_i^2 = 1$ since they can take their preferred values. Moreover, $b_i^1 = b_i^2 = 0$ since they are biased to 0 by z_i^2. Given that $g_i = 0$ it must be that both $c_i^1 = c_i^2 = 1$. Therefore, g_i indifferent in this case as well.

Since in all cases that g_i is incorrect, it is indifferent with respect to this gadget, it will adhere to the bias that the auxiliary gadget connecting a_i^1, a_i^2, g_i gives to g_i. If both I_i is 1 then $a_i^1 = 0$ and if $I_i = 0$ then $a_i^1 = 1$. In all cases, $a_i = \neg I_i$. Hence, the auxiliary gadget gives bias to g_i towards 0, except when both $I_1(g_i) = I_2(g_i) = 0$ in which case it biases g_i towards 1. This means that g_i has a 2^{-500N} bias towards its NOR value. □

Lemma 14. *If $Control = 1$ then all y, z vertices have a 2^{-87N} bias towards their natural values. If $Control = 0$ then all y, z vertices have a 2^{-87N} bias towards their unnatural values.*

Proof. The $NotControl$ vertices are dominated by $Control$'s bias of 2^{7N} and hence have the opposite value. By Lemma 10 we have that $Control$ and $NotControl$ experience at most 2^{6N} bias, while the y, z vertices experience 2^{-87N} bias towards the values opposite $Control$ and $NotControl$, which proves the claim. □

Lemma 15. *Assuming all vertices of the computing circuit gadget are in local optimum and have no external biases. If $Control = 1$ then $\forall i, z_i^1 = 0, y_i^1 = 1, z_i^2 = 0, y_i^2 = 1, z_i^3 = 0, y_i^3 = 1$. If $Control = 0$ then $\forall i z_i^1 = 1, y_i^1 = 0, z_i^2 = 1, y_i^2 = 0, z_i^3 = 1, y_i^3 = 0$.*

Proof. If $Control = 1$, consider the highest k such that y_k or z_k have their unnatural values, i.e. $y_k = 0$ or $z_k = 1$. Since $Control = 1$ all y, z vertices experience a bias towards their unnatural biases by Lemma 14. Since the bias that biases them towards their unnatural values is greater than the weight of the internal vertices connected to y^1, z^1, y^3, z^3 it must be that one of the vertices y^2, z^2 have unnatural values. However, by Lemma 12 all control vertices for $j < k$ are also unnatural. Assume that the output vertices of G_i are only internal to the circuit, i.e. no vertex except those

belonging to the computing gadget is connected to them. Since by Lemma 11, unnatural values for y_i^1, z_i^1 imply that the input vertices of gates G_i are indifferent to G_i, the vertex g_i would be dominated by the bias from the auxiliary vertex ρ. If g_k is an output vertex by the assumption has no external bias. Hence, in this case, g_k can take the correct value, which is a contradiction since g_k having the correct value would mean y_k^2, z_k^2 can take their natural bias by Lemma 13.

If $Control = 0$, consider the least k such that the y, k control vertices have their natural values. Since the $y_i^1, z_i^1, y_i^3, z_i^3$ vertices are dominated by the 2^{-87N} bias ensured by Lemma 14 we have that the only vertices with natural values can be y_i^2, z_i^2. However, even these vertices can only be biased towards their unnatural values since, by lemmas 12 and 13, even if the gate is correct y_i^2, z_i^2 are indifferent with respect to the NOR gadget. □

Having proved the above auxiliary lemmas, we can finally prove the theorem specifying the behaviour of our computing circuits.

Theorem 7. *At any local optimum of the* NODE-MAX-CUT *of Figure 3.*

1. *If $Control_\ell = 1$ and the vertices of $Next\ell, Val_\ell$ experience 0 bias from any other gadget beyond C_ℓ then:*

   - $\text{Next}\ell = \text{Real-Next}(I_\ell)$
   - $\text{Val}_\ell = \text{Real-Val}(I_\ell)$

2. *If $Control_\ell = 0$ then each vertex in I_ℓ experiences 0 bias from the internal vertices of C_ℓ.*

3. *$Control_\ell$ experiences $w_{Control\ell}$ bias from the internal vertices of C_ℓ.*

Proof.

1. Since $Control\ell = 1$ and since we assumed no vertex experiences any external bias, by lemma 15 we have that all y, z have their natural values and hence all gates compute correctly, by lemma 12. Therefore, $Next\ell = Real - Next(I_\ell)$ and $Val_\ell = Real - Val(I_\ell)$.

2. Since $Control\ell = 0$, by lemma 15 all y, z have their unnatural values. Since all NOR gadgets have unnatural control vertices we have that their inputs are indifferent with respect to the gadgets. Hence, the claim that they are unbiased follows.

3. The $Control\ell$ vertex is connected to a vertex $NotControl\ell$, of weight $W_{NotControl\ell} = 2^{7N}$, as well as to several leverage gadgets, which contribute bias at most $2^{100N-94N} = 2^{6N}$. Hence, the 2^{7N} bias dominates.

□

C.4   The Equality Gadget

The *Equality* Gadgets are used to check whether the next best neighbor of a circuit has been successfully transferred to the input of the other circuit. The output of the *Equality* gadget is connected to the control variables of the circuit that should receive the new input. If the new input has not been transferred, the output of this gadget biases the *Control* vertex towards 0, which biases the internal control vertices towards unnatural values. This enables the inputs of the circuit to change successfully to the next solution. When the new solution is transferred, the output of
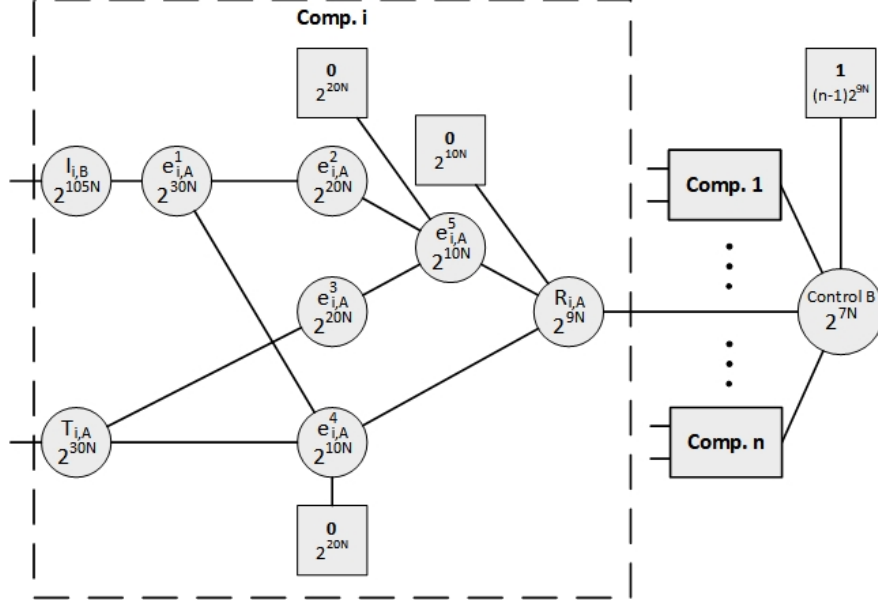
Figure 14: This gadget performs equality check for the bits $I_{i,B}$ and $T_{i,A}$. If they are equal, $R_{i,A} = 0$ at local optimum. We have n such gadgets for each of the two circuits. The n gadgets are connected to produce the final output, which is *ControlB*.

the gadget changes, in order to bias the control vertices towards their natural values, so that the computation can take place.

Since we have two possible directions, both from Circuit A to Circuit B and vice versa we need two copies of the gadgets described in this section.

We will now describe the function of the *Equality* Gadget when Circuit A gives feedback to Circuit B. The *Equality* Gadget takes as inputs the T_A vertices from the *CopyA* Gadgets and I_B and simply checks whether they are equal. Due to Lemma 4, at local optimum, the T_A vertices have the same value as *NextA* which we want to transfer. One might try to connect *NextA* as input to the *Equality* gadget. The reason we avoid this construction is that we do not want the output vertices of the *Circuit Computing* gadget C_A to experience any bias from this gadget, because the computation changes their value with very small bias. For this reason, we connect T_A vertices to the input that are dominated by η_A vertices. The input vertices I_B are dominated by either the vertices in the NOR gadgets or η_A, hence we can connect them directly as inputs to the gadget.

For each bit of the next best neighbor, we construct a gadget as in Figure 14, which performs the equality check for the i-th bit of the next best neighbor. The idea for this construction is very simple: the weights decrease as we come closer to the output, so that the input values dominate the final result. If the inputs are equal, the final value will be 0. Notice that we have put and intermediate vertex between I_B and the gadget to ensure that the two input vertices will have equal weight. A detailed analysis is provided in the proof of Lemma 3.

Now that we have gadgets to perform bit wise equality checks, we need to connect them all to produce the output of the *Equality* gadget. This is done by the construction of Figure 14 Essentially, the idea is that if all the bits are equal, all the comparison results will be 0 and will dominate the *ControlB* to take 1. If at least one result is 1, then together with the constant vertex 1 will bias

ControlB to take value 0.

We now prove the main lemma concerning the *Equality* Gadget, which states that at local optimum, the output of the *Equality* will be 1 if and only if the two inputs to the gadget are equal.

Lemma 3. *Let a local optimum of the* Node-Max-Cut *instance in Figure 3. Then,*

- $ControlA = (I_A = T_B)$

- $ControlB = (I_B = T_A)$

Proof. For simplicity we only prove the second claim, since the first follows by similar arguments. We first focus on on the behavior of a single *Equality* gadget. We would like to prove that $R_{i,A} = 0$ if and only if $I_{i,B} = T_{i,A}$.

We first observe that vertex $e^1_{i,A}$ is biased with weight 2^{105N} by $I_{i,B}$, which is greater that the bias from its other neighbor $e^2_{i,A}$. Hence, at local optimum it is always the case that $e^1_{i,A} = \neg I_{i,B}$. Moreover, vertices $e^2_{i,A}$ and $e^3_{i,A}$ essentially function as the complements of $e^1_{i,A}$ and $T_{i,A}$. This is because they are biased with weight 2^{30N} by them, which is greater than the bias by vertex $e^5_{i,A}$. Hence, $e^2_{i,A} = \neg e^1_{i,A}$ and $e^3_{i,A} = \neg T_{i,A}$.

We first examine the case where $I_B = T_A$. Then $e^1_{i,A} = \neg T_{i,A}$. Since these vertices have equal weights, vertex $e^4_{i,A}$ experiences 0 total bias from them and is biased by constant vertex 0 with weight 2^{20N} and by R_2 with weight 2^{9N}. Therefore, $e^4_{i,A} = 1$. By the previous observations we have that $e^2_{i,A}$ and $e^3_{i,A}$ have opposite values, which means that $e^5_{i,A}$ has bias 0 from these two vertices. Is also has bias 2^{20N} by constant 0 and 2^{9N} from $R_{i,A}$. Hence, $e^5_{i,A} = 1$. Vertices $e^4_{i,A}$ and $e_{5.i}$ bias vertex $R_{i,A}$ towards 0 with weight $2 \cdot 2^{20N}$, which is greater than the bias from constant 0 and *ControlB*. As a result, we have that $R_{i,A} = 0$ and the argument is complete in this case.

Now we examine the case where $I_{i,B} \neq T_{i,A}$. Assume that $I_{i,B} = 1$, the other case follows similarly. Then, $e^1_{i,A} = 0$, $T_{i,A} = 0$, $e^2_{i,A} = 1$, $e^3_{i,A} = 1$. This means that $e^5_{i,A}$ is biased with weight at least $2 \cdot 2^{30N}$ towards 1, which is greater than the combined weight of $R_{i,A}$ and constant 0. Therefore, $e^5_{i,A} = 0$. Now we observe that $R_{i,A}$ is biased with weight at least $2 \cdot 2^{20N}$ towards 1 by vertices $e^5_{i,A}$ and constant 0, which is greater that the combined weight of $e^4_{i,A}$ and *ControlB*. Hence, $R_{i,A} = 1$ in this case. If $I_{i,B} = 0$, then we could prove similarly that $e^4_{i,A} = 0$, which implies that $R_{i,A} = 1$ by the same argument.

We will now prove that *ControlB* takes the appropriate value. First of all, we observe that *ControlB* is connected with *NotControlB*, (part of the *Circuit Computing* gadget) which has weight 2^{7N} and with $R_{i,A}$ vertices which have weight 2^{9N}. It is also biased with weight slightly more than 2^{6N} by each of the control variables y_i due to the leverage gadget. This means that for N large enough *ControlB* is dominated by the behavior of the $R_{i,A}$ vertices. Suppose that $I_{i,B} = T_{i,A}$ for all i, $1 \leq i \leq n$. By the preceding calculations, we have that $R_{i,A} = 0$ for all i. Hence, *ControlB* experiences total bias $n \cdot 2^{9N}$ towards 1, which is greater than the weight of constant vertex 1. Thus, *ControlB* = 1 in this case. Now suppose that there exists a j, $1 \leq j \leq n$, such that $I_{2,j} = \neg T_{j,A}$. By the preceding calculations, $R_{j,A} = 1$. Hence, vertex *ControlB* is biased by vertices $R_{j,A}$ and constant 1 towards 0 with weight at least $(n-1) \cdot 2^{9N} + 2^{9N} = n \cdot 2^{9N}$, which is greater than the combined weight of all the other $R_{i,A}$'s. Therefore, *ControlB* = 0 in this case and the proof is complete. □

C.5 The Copy Gadget

The *Copy* Gadgets transfer the values of the next best Neighbor of a circuit to the input of the other circuit. This is fundamental for the correct computation of the local optimum. There are some technical conditions that these gadgets should satisfy, which we discuss in the following.

The purpose of the *Copy* Gadgets is twofold. Firstly, when the *Flag* vertex has value 1, they are meant to give the inputs of Circuit B a slight bias to take the values of the best flip neighbor that Circuit A offers, that is *NextA*. Secondly, in this case they are meant to give zero bias to the output vertices of Circuit A that calculate the best flip neighbors. This is because when vertex *Flag* is 1, the input of circuit A is going to change, which means that the NOR gates of this circuit will compute the new values. A consequence of the functionality of the NOR gadgets is that the outputs of a gadget are only biased towards the correct value with a very small weight. This is because the gadget is constructed in a way that allows these vertices to be indifferent to all of their neighbors when the time comes to change their value. As a result, if we connect the output vertices with other gadgets, we have to ensure that they will experience zero bias from them in order for the computation to take place properly. Since the outputs of Circuit A that produce the next best neighbor are connected to the *Copy* Gadgets, we should ensure that they will experience zero bias when vertex *Flag* is 1, so that they can change properly. A similar functionality should be implemented when vertex *Flag* is 0.

Next, we present the gadgets that implement the above functionality. There are two *Copy* gadgets with similar topology, *CopyA* and *CopyB*. For simplicity, we only describe the details of *CopyA*. The gadget takes as input the value of vertex *Flag*, which determines whether a value should be copied or whether the outputs of Circuit A should experience zero bias. It also takes as input *NextA*, which is the next best neighbor calculated by Circuit A. The output of the gadget is a bias to vertices I_B and T_A towards adopting the value of *NextA*.

At this point, one might wonder why we didn't just connect the output of the *CopyA* gadget to the input I_B. This is because the value of I_B also depends on the control variables. If the control variables of the input gates have natural values, then the inputs experience great bias from the gate, making it impossible for their values to change by the *Copy* Gadget. Hence, the *Copy* Gadget gives a slight bias to vertex T_A, which is an input to an auxiliary circuit that compares it with I_B (i.e the *Equality* gadget) . If they are not equal, this means that the output has not been transferred yet. In this case, the output of the gadget is given a suitable value to bias the control vertices towards unnatural values. When this happens, the inputs I_B can change to the appropriate values.

Note that we have one of the above gadgets for each of the bits of the next best neighbor solution that the *Circuit Computing* gadgets output.

We have a gadget of Figure 15 for each of the m bits of the next best neighbor. Vertex $F_{i,A}$ has a very large weight in order to dominate the behavior of $\eta_{i,A}$. However, we do not want this vertex to influence the behavior of *Flag*. For this reason, we connect *Flag* with $F_{i,A}$ using a *Leveraging* gadget. Notice that the behavior of $F_{i,A}$ is dominated by *Flag* by weight at least 2^{50N}. Another important point is that we connect the output of the *CopyA* gadget with the input of Circuit B using another *Leveraging* gadget. This is due to the fact that the weight of the input vertices is of the order of 2^{105N}, which is far more than the weight of $\eta_{i,A}$. Hence, we do not want the input vertices to influence the value of $\eta_{i,A}$, while also ensuring that the *Copy* gadget gives a slight bias to the inputs I_B towards the value of *NextA*.

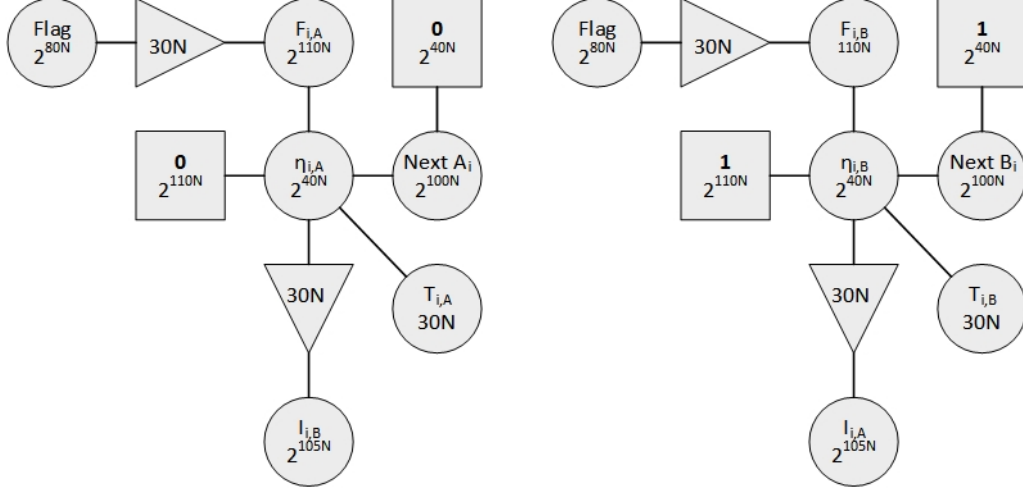We now prove Lemma 4, which makes precise the already stated claims about the function of

Figure 15: The gadgets that copy the values from one circuit to the other

the *Copy* Gadgets.

Lemma 4. *At any local optimum point of the* NODE-MAX-CUT *instance of Figure 3:*

- *If* Flag $= 1$, *i.e.* NextB *writes on* I_A, *then*

  1. $T_B = $ NextB
  2. *If* $ControlA = 0$ *then* $I_A = T_B = $ NextB

- *If* Flag $= 0$ *i.e.* NextA *writes on* I_B, *then*

  1. $T_A = $ NextA
  2. *If* $ControlB = 0$ *then* $I_B = T_A = $ NextA

Proof. We prove the claim for $Flag = 1$. The case $Flag = 0$ is identical.

We begin with the first claim. Due to the leveraging gadget, vertex $F_{i,B}$ experiences bias from $Flag$ which is slightly less than 2^{50N}. Hence, it is biased towards 0 with weight at least 2^{49N}. This is greater than the weight of $\eta_{i,B}$, which is the other neighbor of $F_{i,B}$. Hence, $F_{i,B} = 0$ at local optimum. Now vertex $\eta_{i,B}$ experiences zero total bias from vertices $F_{i,B}$ and constant 1 and biases 2^{100N} by $NextB_i$, 2^{30N} by $T_{i,B}$ and slightly more that 2^{75N} by the input $I_{i,A}$ due to leveraging, which means that its value at local optimum will be determined by $NextB_i$. Specifically, $\eta_{i,B} = \neg NextB_i$ at local optimum. Now, vertex $T_{i,B}$ experiences bias 2^{40N} from $\eta_{i,B}$ and biases of the order of 2^{7N} from the gates of the controller gadget. Hence, $T_{i,B}$ has bias towards $NextB_i$ equal to $w_{\eta_{i,B}}$ and will take this value at local optimum.

To prove the second claim, we use the already proven fact that $\eta_{i,B} = \neg NextB_i$ when $Flag = 1$. Due to the *Leverage* gadget, vertex I_i experiences bias slightly less than 2^{10N} from vertex $\eta_{i,B}$. Since $ControlA = 0$, by Lemma 7, we have that $I_{i,A}$ is indifferent with respect to the gadget C_A, and will therefore take the value of $\neg \eta_{i,B} = NextB_i = T_{i,B}$ □

Lemma 5. *At any local optimum of the* NODE-MAX-CUT *instance of Figure 3:*

44

- *If* Flag $= 1$, *then any vertex in* NextA *experience* 0 *bias with respect to the* CopyA *gadget.*

- *If* Flag $= 0$ *then then any vertex in* NextB *experience* 0 *bias wrt. the* CopyB *gadget.*

Proof. We notice that due to leveraging, vertex $F_{i,A}$ of gadget *CopyA* experiences bias slightly less than 2^{50N} from vertex $Flag = 1$. This dominates its behavior, since the other neighbor $\eta_{i,A}$ has weight that is orders of magnitude smaller. Hence, $F_{i,A} = 0$. Now, vertex $\eta_{i,A}$ experiences total bias $2 * 2^{110N}$ from vertices $F_{i,A}$ and constant 0, 2^{100N} from $NextA_i$, 2^{30N} from $T_{i,A}$ and slightly more than 2^{75N} from $I_{i,B}$ due to the *Leverage* gadget used. This means that $\eta_{i,A} = 1$. Now we are ready to prove our claim. Vertex $NextA_i$ is connected to vertices $\eta_{i,A}$ and constant 0 of gadget $CopyA_i$. They have the same weight and opposite values at local optimum. This means that $NextA_i$ has 0 bias with respect to $CopyA_i$, i.e it is indifferent.

The case for $Flag = 0$ follows symmetrically. $\square$

C.6 The Comparator Gadget

The purpose of the *Comparator* gadget is to implement the binary comparison between the bits of the values of the two circuits. At the same time we need to ensure that the vertices of the losing circuit (i.e the circuit with the lower value) are indifferent with respect to the *Comparator* gadget, so that Lemma 7 can be applied.

In particular, the output vertices that correspond to the bits of the value, presented in section C.3, with weights $2^{90N} \cdot 2^{N+i}$ are each connected as below.

Note that the output bits of the second circuit B are the complement of their true values, in order to achieve comparison with a single bit. The weight of the *Flag* vertex is 2^{80N}

To see why the value of vertex *Flag* implements binary comparison one needs to consider four cases: In the first two, where the ith bits are both equal, the total bias *Flag* experiences is zero, since it experiences bias towards a certain bit as well as the complement of said bit. In the other two, where one bit is 1 and the other is 0, the *Flag* vertex will experience 2^i bias towards either value, which will supersede all lower bits.

However, the *Comparator* gadget is meant not only to implement comparison between values, but also to detect whether a circuit is computing wrongly and, hence, to fix it. To this end we connect the following control vertices to the vertex *Flag*: the control vertices $y^3_{m+1,A}$ for circuit A and $z^3_{m+1,B}$ for circuit B, where $m + 1$ is the last NOR gadget before the bits of the values (recall that we have m value bits and that $w_{y^3_{i,A}} = w_{z^3_{i,B}} = 2^{100N} \cdot (2^{N+m+1} - 50))$ (see Figure 12), as well as the control vertices $y^3_{i,A}, z^3_{i,B}, \forall i \leq m$ for each NOR gadget that corresponds to an output bit of the value (which have weight $w_{y^3_{i,A}} = w_{z^3_{i,B}} = 2^{90N} \cdot (2^{N+i} - 50))$. The vertices $y^3_{m+1,A}$ and $z^3_{m+1,B}$ are used to check whether the next best neighbor has been correctly computed. If it isn't, these vertices dominate *Flag*, due to their large weight of 2^{100N} compared to the weight of the value bits, which is of the order of 2^{90N}. The control vertices of the output bits of the value are used in a more intricate way to ensure that even if one to the results is not correct, the output of the comparison is the desired one. Details are provided in Lemma 16. All these vertices are connected in such a way that a control vertex with unnatural value, biases *Flag* towards fixing that circuit.

We prove the following properties:

Lemma 6. *Let a local optimum of the instance of* Node-Max-Cut *of Figure 3:*

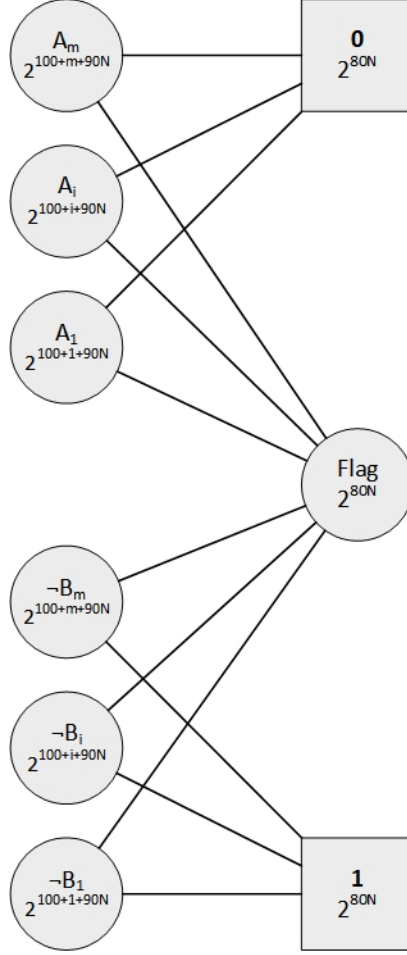- *If* Flag $= 1$ *then all vertices of* C_A *experience* 0 *bias wrt. the* Comparator *gadget.*

Figure 16: vertices of the *Comparator* gadget. Note that Circuit B is meant to output the comple-
ment of its true output.

- *If* Flag $= 0$ *then all vertices of C_B experience 0 bias wrt. the* Comparator *gadget.*

Proof. Suppose *Flag* $= 1$. Then the only vertices of C_A connected to the *Comparator* gadget are
the value output bits and certain control vertices, in such a way that they are connected to either
Flag or a constant vertex 0 of weight equal to *Flag*. In all cases, both biases cancel each other
out and the vertices of Circuit A are indifferent. Suppose *Flag* $= 0$. Then the only vertices of
C_B connected to *Flag* are also connected with a constant 1 vertex. Similarly to the first case, all
vertices of circuit B are indifferent with respect to the *Comparator* gadget when *Flag* $= 0$. ☐

We now prove the most important lemma of the *Comparator* gadget. Our goal is to compare
the output values of the two circuits, so that we change the input of the circuit with the smaller real
value. The main difficulty lies in that one or both of the circuits might produce incorrect bits in
their output. A simple idea would be to try to detect any incorrect output bits and influence *Flag*
accordingly, as we do with control variables $y^3_{m+1,A}$ and $z^3_{m+1,B}$. However, if the least significant bit
of a circuit is incorrect, the weight of the corresponding control vertex is exponentially smaller that
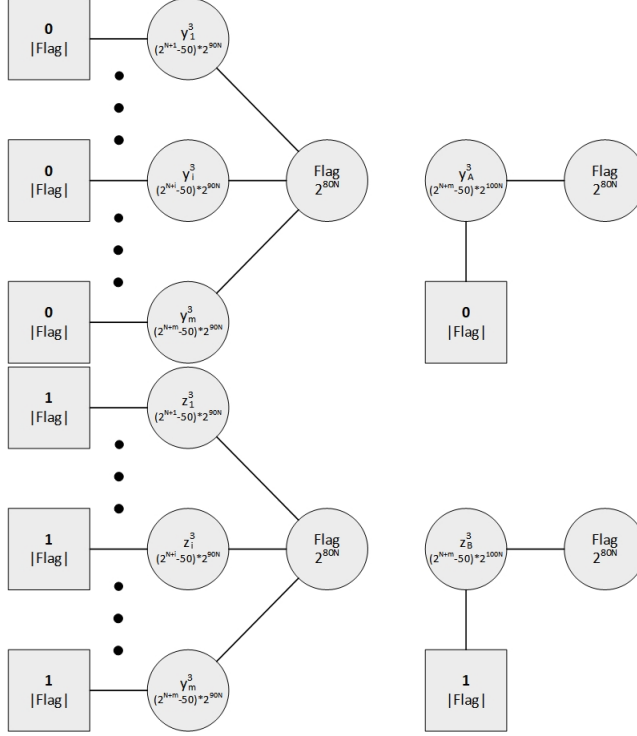the rest of the bits. Hence, it cannot dominate the outcome of the comparison. This means that

Figure 17: Connection between the *control vertices* and the *Flag* vertex.

sometimes we might be at a local optimum where some output vertices are incorrect. To alleviate this problem we propose this construction.

The idea behind this lemma is very simple: if it is guaranteed that the output of one of the circuits is correct and we know which bits of the other circuit *might* be wrong, we can still compare their true values. This is accomplished by an extension of the traditional comparison method, by also taking into account the control variables of the output bits and examining all the possible cases. This lemma is very useful in our proof, since by Lemma 7 we know that at least one of the circuits computes correctly in local optimum.

Lemma 16. *At any local optimum:*

Suppose that Flag $= 1$. *If* $\forall i, z^1_{i,A} = 0, y^1_{i,A} = 1, z^2_{i,A} = 0, y^2_{i,A} = 1, z^3_{i,A} = 0, y^3_{i,A} = 1$ *and* $\forall i > m, z^1_{i,B} = 0, y^1_{i,B} = 1, z^2_{i,B} = 0, y^2_{i,B} = 1, z^3_{i,B} = 0, y^3_{i,B} = 1$ *then*

$$Real\text{-}Val(I_A) \leq Real\text{-}Val(I_B)$$

Suppose that Flag $= 0$. *If* $\forall i, z^1_{i,B} = 0, y^1_{i,B} = 1, z^2_{i,B} = 0, y^2_{i,B} = 1, z^3_{i,B} = 0, y^3_{i,B} = 1$ *and* $\forall i > m, z^1_{i,A} = 0, y^1_{i,A} = 1, z^2_{i,A} = 0, y^2_{i,A} = 1, z^3_{i,A} = 0, y^3_{i,A} = 1$ *then*

$$Real\text{-}Val(I_B) \leq Real\text{-}Val(I_A)$$

Proof. Since for all gates that do not correspond to value bits (see Figure 12), we have that they possess natural values, and hence *Flag* is indifferent with respect to them, we only need to examine the final m gates that correspond to the value bits.

We denote the kth bit of Val_A, Val_B as A_k, B_k. B_k corresponds to the actual value of the kth bit of the circuit B instead of its complement for simplicity. The actual value of the vertex corresponding to B_k is the opposite. We also denote $z^2_{k,B}$ the control vertex corresponding to the bit B_k. We make the distinction between A_k, B_k and $Real(A_k), Real(B_k)$. These may be equal or different depending on whether the circuit calculated the kth bit correctly. By the assumption we know that $A_k = Real(A_k)$ since A calculates correctly. We do not know whether $B_k = Real(B_k)$, but we do know that $B_k \neq Real(B_k) \implies z^2_{k,B} = 1$.

We consider three cases.

In the case that $(A_k, B_k, z^2_{k,B}) \in (0,0,0), (1,1,0), (0,1,1)$, $Flag$ experiences bias at most $2 \cdot 2^{90N} \cdot (50)$ from this bit towards $Flag = 1$ in any of these cases. In this case, we have that either $Real(A_k) = Real(B_k)$ or $Real(A_k) < Real(B_k)$, depending on whether B_k calculated correctly. Either way, $Real(A_k) \leq Real(B_k)$.

In the case that $(A_k, B_k, z^2_{k,B}) \in (0,1,0)$, $Flag$ experiences bias $2 \cdot 2^{90N} \cdot (2^{N+k})$ from this bit towards $Flag = 1$. In this case, we have that $Real(A_k) < Real(B_k)$, since both calculate correctly.

In the case that $(A_k, B_k, z^2_{k,B}) \in (0,0,1), (1,0,0), (1,1,1), (1,0,1)$ then $Flag$ experiences bias at least $2 \cdot 2^{90N} \cdot (2^{N+k} - 50)$ towards $Flag = 0$ from this bit in any of these cases. In these cases, $Real(A_k)$ might be higher, but we will show that these cases can never matter.

Suppose k the highest i for which $(A_i, B_i, z^2_{k,B}) \notin (0,0,0), (1,1,0), (0,1,1)$.

If no such k exists then all bits must lie in the first case and hence $\forall k Real(A_k) \leq Real(B_k)$. Hence, $Real\text{-}Val(I_A) \leq Real\text{-}Val(I_B)$.

If for that k, $(A_k, B_k, z^2_{k,B}) \in (0,1,0)$, we know that $Real(A_k) < Real(B_k)$ while for all higher bits $k Real(A_k) \leq Real(B_k)$. This means that $Real\text{-}Val(I_A) < Real\text{-}Val(I_B)$, since the lower bits don't matter as long as we have a strict inequality in a high bit.

Lastly, if we have that $(A_k, B_k, z^2_{k,B}) \in (0,0,1), (1,0,0), (1,1,1), (1,0,1)$, we have that $Flag$ experiences bias at least $2 \cdot 2^{90N} \cdot (2^{N+k} - 50)$ towards $Flag = 0$ from this bit. Furthermore, it experiences bias at most $2 \cdot 2^{90N} \cdot (50)$ towards $Flag = 1$ from each bit higher that k. Each bit i lower than k causes bias at most $2 \cdot 2^{90N} \cdot (2^{N+i})$ each towards $Flag = 1$. In total, if we have m bits, we have at most $(m-k) \cdot 2 \cdot 2^{90N} \cdot (50) + \sum_{i<k} 2 \cdot 2^{90N} \cdot (2^{N+i}) \leq (m) \cdot 2 \cdot 2^{90N} \cdot (50) + 2 \cdot 2^{90N} \cdot (2^{N+k} - 2^N)$ towards $Flag = 1$ and at least $2 \cdot 2^{90N} \cdot (2^{N+k} - 50)$ towards $Flag = 0$. For N sufficiently high, the bias towards 0 would win, making $Flag$ no longer have 1 as its best response, which is a contradiction. Hence, the third case can not happen in a local optimum with $Flag = 1$.

The case for $Flag = 0$ is identical, with the only difference being we consider $y^2_{k,A}$ instead. $\square$

Lemma 8. *At any local optimum of the* NODE-MAX-CUT *instance of Figure 3:*

- *If* Flag $= 1$ *then* NextB $=$ Real-Next(I_B)

- *If* Flag $= 0$ *then* NextA $=$ Real-Next(I_A)

Proof. Assume a local optimum with $Flag = 1$ and $NextB \neq Real\text{-}Next(I_B)$. By Lemma 12 we have that Circuit B is computing incorrectly and hence the control vertex $z^3_{m+1,B}$ (i.e. the last gate before the value bits) has its unnatural value, which is $z^3_{m+1,B} = 1$.

Assume that, $Control A = 0$. Then by Lemma 4 we have that $I_A = T_B$, which by Lemma 3 we have $Control A = 1$, a contradiction. Hence, $Control A = 1$.

Therefore, since have that $Control A = 1$ and that $NextA = Real\text{-}Next(I_A)$, which by Lemma 15, implies that the corresponding vertex $y^3_{m+1,A}$ has its natural value $y^3_{m+1,A} = 1$.

This means that *Flag* experiences bias towards 0 at least $2 \cdot 2^{100N} \cdot (2^{N+m+1} - 50)$ from the vertices $z^3_{m+1,B}, y^3_{m+1,A}$, which dominates *Flag* to take value 0. This is a contradiction since we assumed that $Flag = 1$ at local optimum. Hence, if $Flag = 1$ then it must be that $NextB = Real\text{-}Next(I_B)$. Similarly, we can prove that if $Flag = 0$ then $NextA = Real\text{-}Next(I_A)$. $\square$

Lemma 9. *At any local optimum of the* NODE-MAX-CUT *instance of Figure 3:*

- *If* $Flag = 1$, NextA = Real-Next(I_A), Val$_A$ = Real-Val(I_A) *and* NextB = Real-Next(I_B) *then* $Real\text{-}Val(I_A) \leq Real\text{-}Val(I_B)$.

- *If* $Flag = 0$, NextB = Real-Next(I_B), Val$_B$ = Real-Val(I_B) *and* NextA = Real-Next(I_A) *then* $Real\text{-}Val(I_B) \leq Real\text{-}Val(I_A)$.

Proof. Assume that, $ControlA = 0$. Then by Lemma 4 we have that $I_A = T_B$, which by Lemma 3 we have $ControlA = 1$, a contradiction. Hence, $ControlA = 1$.

Since $Flag = 1$ and we have that $ControlA = 1$, by Lemma 15 all control vertices of C_A have their natural values. Furthermore, since by the proof of Lemma 8 we know that all control vertices of weight 2^{100N} have their natural values, we can apply Lemma 16. Therefore, $Real\text{-}Val(I_A) \leq Real\text{-}Val(I_B)$

The proof for $Flag = 0$ is identical. $\square$