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

Dimitris Fotakis* Vardis Kandiros* Thanasis Lianeas* Nikos Mouzakis*
Panagiotis Patsilinakos* Stratis Skoulakis*

Abstract

Computing an equilibrium of a game is of central interest in Algorithmic Game Theory. For (atomic) Congestion Games, computing a pure Nash equilibrium (PNE) is PLS-complete for the general case, while for single-commodity Network Congestion Games computing the PNE that minimizes the potential function reduces to a min-cost flow computation [8]. We study the complexity of computing a PNE in Weighted Congestion Games with affine and linear delays, where O(1)-approximate Nash equilibria can be computed in polynomial time [4] and the only known PLS-hardness results follow from those for unweighted Congestion Games.

We first show that it is PLS-complete to compute a PNE even in single-commodity Weighted Network Congestion Games on series-parallel networks with linear latencies, a result indicating that equilibrium computation for Weighted Congestion Games can be significantly more difficult than their unweighted counterparts. Note that for unweighted Congestion Games on single-commodity series parallel networks with general latency functions, a PNE can be computed by a simple greedy algorithm. For that reduction, we use exponential coefficients on the linear latency functions. To investigate the impact of weighted players only, we restrict the game so that the linear latencies can only have polynomial coefficients. We show that it is PLS-complete to compute a PNE in Weighted Network Congestion Games with all edges having the identity function. For the latter, none of the known PLS-complete problems seems as a natural initial problem for the reduction. Hence, we consider the problem of finding a maximal cut in NodeMaxCut, a natural special case of MaxCut where the weights of the edges are the product of the weights of their endpoints. Via an involved reduction, we show that NodeMaxCut is PLS-complete, so that we can use it as our initial problem. The result and the reduction themselves are of independent interest and they may have more implications in proving other complexity results related to PLS.

1 Introduction

Among the strongest notions in Game Theory is that of a Nash equilibrium. Although its existence is always guaranteed in normal form games if mixed strategies are allowed, its computation is more often inefficient than efficient. This is true even if the game admits a potential function and any improving step by a player reduces the value of the potential, in which case we additionally know that a Nash equilibrium on *pure strategies* always exists. In such *potential games*, a pure Nash equilibrium (or simply equilibrium) can be computed by letting the players one by one play their best response, until they eventually reach an equilibrium, which of course may take exponentially

*National Technical University of Athens: {fotakis,vkandiros,lianeas,nmouzakis,patsilinak,sskoul}@corelab.ntua.gr

many steps. In a sense, the described *best response* dynamics is equivalent to searching for an extremum by doing local improving steps.

Polynomial Local Search (PLS) is a complexity class defined by Johnson et al. [16] to capture the difficulty of certain search problems. It consists of problems that admit local search heuristics that are guaranteed to converge to a local optimum, like the problem of finding an equilibrium in potential games described above. There are many natural complete problems for PLS (see, e.g., Michiels et al. [17] for a survey) many of which are or can be regarded as the problem of computing an equilibrium in a potential game. That said, PLS fully captures the difficulty of such problems. Complete problems particularly relevant to this work are LocalMaxCut and CircuitFlip which we will describe later, and the problem of computing an equilibrium in Congestion Games.

Congestion Games (CGs) are one of the most widely studied classes of games in Algorithmic Game Theory. In CGs players compete over resources trying to minimize their costs. More precisely, there exists a set of resources E and the strategy of each selfish agent is a subset of resources $s_i \in 2^E$ that she selects. The cost of each agent i for a given strategy vector $s = (s_i, s_{-i})$ is the cumulative congestion on the each resources that she selects to use. More precisely, the cost of agent i is $C_i(x_i, x_{-i}) = \sum_{e \in s_i} \ell_e(x_e)$, where x_e is the number of agents participating in resource e and $\ell_e(x)$ is latency function of resource e. Usually agents are not allowed to select any subset of resources. The permitted subsets may differ from agent to agent and are denoted by $S_i \subseteq 2^E$. The case where all the S_i's are the same set is referred as *Symmetric CGs*.

Congestion Games are of interest not only due to their obvious connection to resource sharing in networks and distributed systems with selfish users, but also due to their unique game theoretic properties. Congestion games are potential games, i.e., there exists a potential function mapping each strategy profile to a real number, $\Phi(s) = \sum_{e \in E} \sum_{x=1}^{x_e} \ell_e(x)$, so that whenever an agent changes her strategy to reduce her cost, the potential function decreases. The latter not only provides a proof of existence of an equilibrium, but also provides a pseudo-polynomial algorithm for computing one (best response dynamics). Unfortunately a polynomial algorithm for computing an equilibrium is highly unlikely to exist since this problem was proven to be PLS-complete [8].

Although computing an equilibrium in general CGs is PLS-complete, there are important special cases in which an equilibrium can be computed efficiently. A notable example concerns computing an equilibrium in single-commodity Network CGs. In this type of games, there exists a network with latency functions on its edges together with an origin node o and a target node d. The strategy space of each agent is the set of $o - d$ paths . Fabrikant et al. [8] provide an efficient algorithm for computing an equilibrium, which is based on solving a suitable min-cost flow problem. Another interesting special case where approximate equilibrium can be efficiently computed by approximate best response dynamics concerns a general class of Symmetric CGs [5].

An important generalization of CGs is that of Weighted CGs where the players are weighted, putting a different load on the resources they choose. Since Weighted CGs are a generalization of CGs computing an equilibrium is a harder task. Many recent works study algorithms for computing an approximate equilibrium in general CGs (see the related work section). The reason for this is twofold: At first, computing an exact equilibrium is for sure PLS-hard and, second, Weighted CGs with general latency functions are not potential games, meaning that exact equilibria may not even exist.

Our work is motivated by the fact that the forementioned results do not provide a vivid picture of the differences between weighted and unweighted CGs, since in the general setting both are hard. In order get a better understanding of these differnces we examine the computational hardness

of computing equilibria in Weighted CGs for the cases in which an equilibrium can be computed efficiently for their unweighted counterparts.

This work deals with the problem of computing equilibria in Weighted Network CGs with linear latency functions on the links of the network. The choice of linear functions ensures the existence of a potential function [11], rendering the problem of computing equilibrium in PLS. We prove PLS-completeness both for the single-commodity, i.e., all agents want to travel from a common origin o to a common destination d, and to the multi-commodity case where each agent i has a different pair (o_i, d_i) of origin-destination nodes. Obviously, the second type of games is a generalization of the first. The reason that two different PLS-reductions are provided is that in the single-commodity case we study a more general set of latency functions by allowing them to have exponential coefficients, while in the multi-commodity case all the linear latency functions have coefficient 1. We highlight that in the case of equally weighted players both equilibria can be efficiently computed. In the first case via a min-cost flow computation, while in the second using *best response dynamics*, which converges in a polynomial number of steps.

Contribution. First we show that computing an equilibrium even in single-commodity Weighted Network CGs with linear latency functions is PLS-complete (Theorem 1). Our result reveals a huge gap between the weighted and the unweighted case for this class of CGs, since Papadimitriou et al. [8] designed a polynomial time algorithm for computing equilibrium in single-commodity (unweighted) Network CGs with general increasing latency functions. For the proof we reduce from LocalMaxCut, one of the most significant PLS-complete problems. We highlight that our reduction not only uses simple linear latency functions (i.e., of the form $\ell(x) = ax$), but also the network topology used for our reduction is that of series-parallel networks.

Trying to further understand the differences between weighted and unweighted CGs, we turn our attention to the case where all resources/links have the identity latency function (i.e., for all e: $\ell_e(x) = x$) and only the players' weights are allowed to be exponential. The reason for this choice is that an equilibrium in general (unweighted) CGs with this type of functions can be efficiently computed using best response dynamics. We manage to show that computing an equilibrium even in multi-commodity Weighted Network CGs where all links have the identity function is PLS-complete (Theorem 2). For the above reduction, none of the already known PLS-complete problems seemed suitable, since their intrinsic difficulty lies in the interactions between players, while the complexity of our problem stems from the weights of the players themselves. This should not be a surprise, since in general CGs with this type of latency functions and equally weighted agents an equilibrium can be efficiently computed. Thus, in order to prove Theorem 2 we introduce as an intermediate problem a simple variant of LocalMaxCut that we call LocalNodeMaxCut.

LocalNodeMaxCut is a natural special case of LocalMaxCut where the weights of the edges are related in a specific way. Namely, each node has a positive weight and the weight of each edge is the product of the weights of its endpoints. As in LocalMaxCut, the goal is to find a maximal cut of the edges, i.e., a partition of the nodes so that the total weight of the edges traversing the cut cannot be increased by moving any single node from one side of the partition to the other. Interestingly enough, LocalNodeMaxCut is suitable for the reduction we want due to the weights on the nodes, while at the same time it is expressive enough to be PLS-complete.

Our major technical contribution is proving that LocalNodeMaxCut is PLS-complete. To establish its PLS-hardness we did not use a problem that lies on any of the "reduction paths" departing from LocalNodeMaxCut, as this seemd unnatural since LocalNodeMaxCut is a very restrictive special case of LocalMaxCut. Instead we turned to the very first problem proved

to be PLS-complete, i.e., CircuitFlip. Note that a similar problem to LocalNodeMaxCut, that of computing a local optimum of an unweighted graph, was shown by Schäffer and Yannakakis [18] to be P-complete.

In LocalNodeMaxCut it is very crucial that nodes see other nodes in the same way. More precisely, any two nodes see the same weight on any third node, in contrast to LocalMaxCut where the weights on the edges may be distributed so that nodes see common neighboring nodes differently. This lack of freedom is very restrictive in many places of the PLS-completeness of LocalMaxCut but we manage to overcome it via more sophisticated ideas combined with gadgets already used by Elsässer and Tscheuschner [6] and Gairing and Savani [12]. Our technical work, though, is difficult to be explained at a high level and so, instead of getting into its details here, we highlight it at the end of Section 4. The ideas used there can be adapted to simplify proofs in existing work [6; 12] and we conjecture that they also can be used to prove hardness of approximation results for Weighted Network CGs with "natural" latency functions, thus complementing the existing results for the unweighted case [19].

Related Work. While Weighted CGs do not always possess exact equilibria, there is a large amount of literature concerning cases of Weighted CGs in which exact or approximate equilibria can be efficiently computed. Fotakis et al. [11] show that in the case of linear latencies the existence of an equilibrium is guaranteed by a potential argument. Even-Dar et al. [7] study different best-response schemes for weighted congestion games in series of parallel links. In case of identity latency functions it was proven that the max-weight priority best response policy reaches equilibrium in linear steps. Goldberg [15] analyzes the convergence time to a Nash equilibrium of a randomized local search distributed protocol in the case of series-parallel links with different capacities. Gairing et al. [13] present a polynomial algorithm for computing Nash equilibrium in the restricted case of parallel links, i.e., each agent can select a subset of the overall links. Caragiannis et al. [3] and Giannakopoulos et al. [14] design efficient algorithms for computing approximate equilibria in Weighted CGs with polynomial latencies. More algorithmic results for approximate equilibria in Weighted CGs are provided by Fanelli and Moscardelli [9] and Feldotto et al. [10].

To capture the difficulty of local search problems, Johnson et al. [16] define the class PLS that consists of problems admitting a polynomial time algorithm for computing a better neighbor solution. They prove that all such problems can be reduced to a general optimization problem named CircuitFlip. Later, Yannakakis and Schäeffer [18] used CircuitFlip to prove that LocalMaxCut is PLS-complete. LocalMaxCut searches for a cut in an edge-weighted undirected graph such that the cumulative weight of the edges traversing the cut cannot be increased by changing the set of any single node. It was used to establish the PLS-completeness of computing equilibria in general unweighted CGs [8]. In the same paper PLS-completeness of computing an equilibrium for multi-commodity Network CGs was proven through a very involved proof. Ackermann et al. [1] provide a much simpler PLS-completeness proof of the previous result via a reduction from LocalMaxCut.

From a technical point of view, most related to our work are the PLS-completeness results for variants of LocalMaxCut by Elsässer and Tscheuschner [6] and Gairing and Savani [12] who adapted the original construction of Schäffer and Yannakakis [18]. Gairing and Savani [12] establish PLS-completeness of computing a Nash equilibrium in hedonic games, while Elsässer and Tscheuschner [6] show that LocalMaxCut remains PLS-complete even if the underlying graph has nodes of degree at most five. Skopalik and Vöcking [19] proved that it PLS-complete to compute an α-approximate Nash equilibrium in general unweighted congestion games.

Organization. The paper is organized as follows. In Section 3 we present proof sketches for the hardness results regarding computing equilibria in single-commodity Weighted CGs and multi-commodity Weighted CGs. In Section 4, we give an overall presentation of the way our LocalNodeMaxCut PLS-reduction works, while omitting certain details. The fully detailed proofs are deferred to the appendix, in sections A and B, respectively.

2 Preliminaries

We first define three local search problems, i.e., problems where the goal is to find a solution which is locally optimal with respect to the values of all other solutions in its neighborhood.

Local-Max-Cut. An instance of MaxCut consists of an edge-weighted graph $H(N, A)$, where N is the set of nodes and A is the set of edges. The weight of edge $a \in A$ is denoted by w_a. A *cut* of H is a subset $S \subseteq N$. The *weight* of a cut S is the sum of the weights of the edges crossing it, i.e., $W(S) = \sum_{a \in C(S)} w_a$, where $C(S) = \{\{u, v\} \in A : u \in S \text{ and } v \in N \setminus S\}$. The *neighborhood* of a cut S is the set $ND(S)$ that contains exactly all the cuts that can be obtained by moving one node from S to $N \setminus S$ or one node from $N \setminus S$ to S. A maximal cut is a cut S such that $W(S) \geq W(S')$, for all $S' \in ND(S)$. Given an edge-weighted graph H, LocalMaxCut is the problem of finding a maximal cut of H.

Local-Node-Max-Cut. A special case of MaxCut is that of NodeMaxCut where each node $i \in N$ has some weight w_i and the weight of edge $a = \{i, j\}$ is then $w_a = w_i w_j$. In such a case we denote LocalMaxCut by LocalNodeMaxCut. We work with this definition in Section 3.

Alternatively, LocalNodeMaxCut consists of an (unweighted) undirected graph $H(N, A)$, where N is the set of nodes and A is the set of edges. Each node $i \in N$ is a selfish agent and has a non-negative weight $w_i \geq 0$. Each node selects as value either 0 or 1 so as to minimize her cost, which is defined to be the total weight of her neighbors that play the same value. A 0-1 vector $v = (v_1, \ldots, v_{|N|})$ is a Nash Equilibrium if and only if for all nodes $i \in N$,

$$\sum_{j \in N_i \text{ and } v_i = v_j} w_j \leq \sum_{j \in N_i \text{ and } v_i \neq v_j} w_j,$$

where N_i denotes the set of nodes that share an edge with node i. In LocalNodeMaxCut the goal is to compute a 0-1 vector that is a Nash Equilibrium. We work with this definition in Section 4.

To see the equivalency of the two definitions observe that

$$\sum_{j \in N_i \text{ and } v_i = v_j} w_j \leq \sum_{j \in N_i \text{ and } v_i \neq v_j} w_j \Leftrightarrow \sum_{j \in N_i \text{ and } v_i = v_j} w_i w_j \leq \sum_{j \in N_i \text{ and } v_i \neq v_j} w_i w_j,$$

i.e., i is at her best response iff the weight of the cut only decreases if i switches sides.

Circuit-Value with Flip-Neighborhood. The very first problem proved to be PLS-complete is CircuitFlip, where an instance consists of a boolean circuit C with n input gates, m output gates and (wlog) only NOR gates. The *value* $C(I)$ of an n-bit input string I is the integer corresponding to the m-bit (binary) string in the output gates. Given a circuit C, CircuitFlip searches for a string I for which the value of the circuit cannot increase if we flip any of I's bits, i.e. for all I' with Hamming distance 1 from I, $C(I) \geq C(I')$. Strings with Hamming distance 1 are called *neighbors*.

Next, we move on to give some Complexity Theory related preliminaries.

Local Search Problems. An instance of a *local search problem* is a triple (S, f, N), where S is the set of *feasible solutions* (or simply *solutions*) of a combinatorial optimization problem, $f : S \to \mathbb{R}$ is a *cost* function for the solutions in S and $N : S \to 2^S$ is a *neighborhood* function on S returning for each $s \in S$ the set of its neighboring feasible solutions.

A *local search problem* Π_{LS} is specified by a set of local search problem instances and it is either a minimization or a maximization problem. The problem is to find for any given instance (S, f, N) a locally optimal solution $s^* \in S$, i.e., an $s^* \in S$ so that $f(s^*) \leq f(s)$ for a minimization problem or $f(s^*) \geq f(s)$ for a maximization problem. Examples of local search problems are LocalMaxCut and LocalNodeMaxCut defined below.

Polynomial-time Local Search (PLS). The *class NPO* consists of combinatorial optimization problems for which we can decide in polynomial time whether an input x defines a correct problem instance, for which the cost function is computable in polynomial time, and for which it is decidable in polynomial time whether a string s defines a feasible solution, where the size of each feasible solution is polynomially bounded in the input size.

Let Π_{LS} be a local search problem and let Π be the underlying combinatorial optimization problem. Local search problem Π_{LS} is in the *class PLS* (Polynomial-time Local Search) if $\Pi \in NPO$ and if two polynomial-time algorithms A and B exist that satisfy the following properties:

- For a problem instance (S, f, N) of Π_{LS}, algorithm A returns a solution $s \in S$.
- For a problem instance (S, f, N) of Π_{LS} and a solution $s \in S$, algorithm B decides whether s is a local optimum and if this is not the case, it returns a neighboring solution with better cost.

PLS Reductions and Completeness. Local search problem Π_{LS} is *PLS-reducible* to local search problem Π'_{LS}, if two polynomial-time algorithms ϕ_1 and ϕ_2 exist that satisfy the following properties:

- Algorithm ϕ_1 transforms a problem instance I of Π_{LS} into a problem instance $\phi_1(I)$ of Π'_{LS}.
- Algorithm ϕ_2 maps a problem instance $I = (S, f, N)$ of Π_{LS} and a solution $s' \in S'$ with $\phi_1(I) = (S', f', N')$ to a solution $s \in S$.
- For a problem instance I of Π_{LS}, we have that if $s' \in S'$ is a local optimum for $\phi_1(I) = (S', f', N')$, then $\phi_2(I, s')$ is a local optimum for I.

Note that if local search problem Π_{LS} is PLS-reducible to local search problem Π'_{LS}, then the existence of a polynomial-time algorithm for Π'_{LS} implies the existence of a polynomial-time algorithm for Π_{LS}. In other words, Π'_{LS} is at least as hard as Π_{LS}. Additionally, PLS-reductions are transitive, which means that if Π_{LS} is PLS-reducible to Π'_{LS} and Π'_{LS} is PLS-reducible to Π''_{LS}, then Π_{LS} is also PLS-reducible to Π''_{LS}.

To capture the hardest problems of the class we define *PLS-completeness*. A local search problem is *PLS-hard* if each problem $\Pi_{LS} \in PLS$ is PLS-reducible to it. By the above discussion, to prove that a local search problem Π_{LS} is PLS-hard it suffices to reduce a PLS-hard problem to it. A local search problem is *PLS-complete* if it is in PLS and is PLS-hard. Schäffer and Yannakakis [18] have proved that LocalMaxCut is PLS-complete (see Michiels et al. [17] for a survey), which directly implies that LocalNodeMaxCut is in PLS.

Next, we move on to give some Game Theory related preliminaries.

Weighted Congestion Games. A *Weighted Congestion Game* consists of n weighted players each having a positive weight w_i, a set of resources E together with a non-decreasing latency function $\ell_e : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ for each resource $e \in E$, and a non-empty strategy set $S_i \subseteq 2^E$ for every player $i \in [n]$, where $[n] = \{1, \ldots, n\}$. A subclass of Weighted Congestion Games is that of *Weighted Network Congestion Games* where there is an underlying (undirected) network with some of the vertices forming origin-destination pairs. The resources of the game are the links of the network. Each player i is assigned to some origin-destination pair o_i-d_i and her strategy set consists of all the o_i-d_i paths. Formally, a Weighted Network Congestion Game is a tuple $\mathcal{G} = (G(V, E), (c_e)_{e \in E}, (w_i)_{i \in [n]}, ((o_i, d_i))_{i \in [n]})$. In the special case where all players share a common origin-destination pair we have a single-commodity game, else we have a multi-commodity game.

Configurations, Costs and Equilibria. Player i chooses a strategy s_i in her strategy set S_i (an o_i-d_i path for network games). All these choices combined form a configuration. Formally, a *configuration* is a vector $(s_1, \ldots, s_n)$, with $s_i \in S_i$. In order to capture single player deviations, given an initial configuration $s = (s_1, \ldots, s_n)$ we denote by (s'_i, s_{-i}) the configuration where player i has changed her strategy from s_i to s'_i while all other players play according to s, i.e., player $j \neq i$ chooses s_i. Under a configuration $s = (s_1, \ldots, s_n)$ each resource gets some *congestion* x_e equal to the sum of the weights of the players choosing it, i.e., $x_e = \sum_{i : e \in s_i} w_i$, and has *cost* $\ell_e(x_e)$. Player i's *cost* under configuration s is $c_i(s) = \sum_{e \in s_i} \ell_e(x_e)$. A configuration s is a *Nash equilibrium* if no player has an incentive to deviate. Formally, s is a Nash equilibrium if $\forall i \in [n], \forall s'_i \in S_i : c_i(s) \leq c_i(s'_i, s_{-i})$. Instead of Nash equilibrium we may simply say *equilibrium*.

3 Computing Equilibria in Weighted Congestion Games

In this section we state and provide proof sketches for our PLS-completeness theorems. In the first part we do so for single-commodity Weighted Network Congestion Games on series parallel networks with linear latencies (Theorem 1) and in the second part we do so for multi-commodity Weighted Network Congestion Games with the identity function on all links (Theorem 2). For the complete proofs see sections A.1 and A.2 respectively.

Single-Commodity Weighted Network Congestion Games.
We prove that it is PLS-hard to compute an equilibrium of a Weighted Congestion Game, even if it is a Weighted Network Congestion Game on a series-parallel single-commodity network with linear latencies, i.e., with latency functions of the form $\ell_k(x) = a_k x$. We do so by reducing from LocalMaxCut.

Theorem 1. *Computing a Nash equilibrium in single-commodity Weighted Network Congestion Games with linear latency functions is PLS-complete.*

Proof sketch. We will reduce from the PLS-complete problem LocalMaxCut. Given an instance of LocalMaxCut we will construct a Weighted Network Congestion Game for which the Nash equilibria will correspond to maximal solutions of LocalMaxCut and vice versa.

To give the construction, let $H(N, A)$ be an edge-weighted graph of a LocalMaxCut instance and let $n = |N|$ and $m = |A|$. In the constructed Weighted Network Congestion Game 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

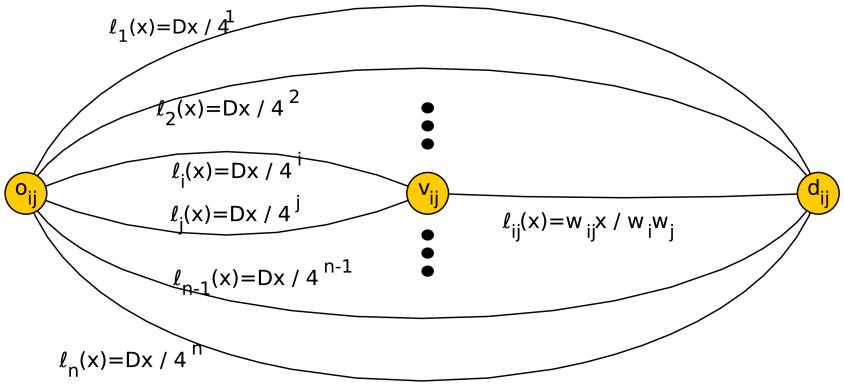

Figure 1: The series-parallel network F_{ij} that corresponds to edge $\{i,j\} \in A$

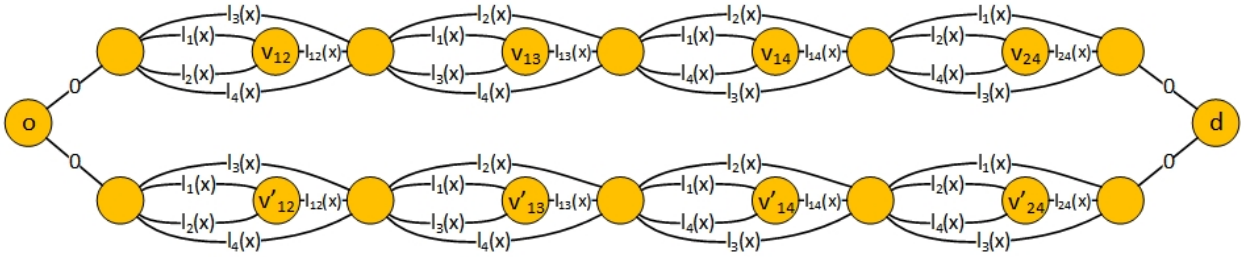

Figure 2: An example of the structure of the construction for a graph $H(N,A)$, with $N=\{1,2,3,4\}$ and $A=\{\{1,2\},\{1,3\},\{1,4\},\{2,4\}\}$. Each of the upper and lower parts consists of copies of F_{12}, F_{13}, F_{14} and F_{24} connected in series.

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.We 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 $\{i,j\} \in A$ let F_{ij} be the series-parallel graph that corresponds to edge $\{i,j\}$. F_{ij} is presented in Fig. 1, where D is assumed to be a (polynomially) big enough constant. An example graph 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 links 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 links, contains some links with latency function of the form $\frac{w_{ij}x}{w_i w_j}$. These links for path p_i^u or p_i^l are in one to one correspondence to the edges of node i in H and this is crucial for the proof.

By the choice of the players' weights and the latency functions' slopes, one can show that at a Nash equilibrium, a player of weight w_i chooses either p_i^u or p_i^l. The formal proof uses induction starting from larger weights. The heaviest players, i.e., players with weight w_n, have a dominant strategy to choose either p_n^u or p_n^l since $\ell_n(x)$ has a significantly smaller slope than all other $\ell_i(x)$'s, small enough so that even if all other players choose the same paths (reaching a load of at most $3\sum_{l=1}^{n} 16^l = \frac{16^{n+1}-1}{16-1}$), still players of weight w_n prefer p_n^u or p_n^l over all other paths. But then, p_n^u and p_n^l get a lot of weight load at equilibrium compared to the weight of lighter players. This makes the links on these paths look like they get some big additive constants, which makes them extremely

expensive for all lighter players and these players exclude them from their strategy space. That said, by the same reasoning, the players of weight w_{n-1} have a dominant strategy to choose either p^u_{n-1} or p^l_{n-1} and this inductively proves true for all $i \in [n]$.

Additionally, one can prove that p^u_i and p^l_i will have at least one player (of weight w_i). The underlying idea is that if wlog p^u_i has two players (of weight w_i) then the third player of weight w_i prefers to go to p^l_i, since it is going to be empty. 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 link in G (the edge with latency function $w_{ij}x/w_iw_j$) only if $\{i,j\} \in A$. The correctness of the reduction lies in the fact that players in G try to minimize their costs incurred by these type of links in the same way one wants to minimize the sum of the weights of the edges in each side of the cut when solving LocalMaxCut.

Given a maximal solution S of LocalMaxCut the proof shows that the configuration Q that for every $k \in S$ routes 2 players through p^u_k and 1 player through p^l_k and for every $k \in N \setminus S$ routes 1 player through p^u_k and 2 players through p^l_k is an equilibrium. Conversely, given an equilibrium Q the cut $S = \{k \in N : 2 \text{ players have chosen } p^u_k \text{ at } Q\}$ is a maximal solution of LocalMaxCut.

Assume that we are at equilibrium and consider a player of weight w_k that has chosen p^u_k and wlog p^u_k is chosen by two players (of weight w_k). By the equilibrium conditions the cost she computes for p^u_k is at most the cost she computes for p^l_k, which implies

$$\sum_{i=1}^{m} \frac{2D16^k}{4^k} + \sum_{\{k,j\}\in A} \frac{w_{kj}(2\cdot 16^k + x^u_j 16^j)}{16^k 16^j} \leq \sum_{i=1}^{m} \frac{2D16^k}{4^k} + \sum_{\{k,j\}\in A} \frac{w_{kj}(2\cdot 16^k + x^l_j 16^j)}{16^k 16^j}$$

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

$$\sum_{\{k,j\}\in A} w_{kj}x^u_j \leq \sum_{\{k,j\}\in A} w_{kj}x^l_j \Leftrightarrow \sum_{\{k,j\}\in A} w_{kj}(x^u_j - 1) \leq \sum_{\{k,j\}\in A} w_{kj}(x^l_j - 1) \tag{1}$$

Define $S = \{i \in N : x^u_i = 2\}$. By our assumption it is $k \in S$ and the left side of (1), i.e., $\sum_{\{k,j\}\in A} w_{kj}(x^u_j - 1)$, is the sum of the weights of the edges of H with one of its nodes being k and the other belonging in S. Similarly, the right side of of (1), i.e., $\sum_{\{k,j\}\in A} w_{kj}(x^l_j - 1)$ is the sum of the weights of the edges with one of its nodes being k and the other belonging in $N \setminus S$. But then (1) directly implies that for the (neighboring) cut S' where k goes from S to $N \setminus S$ 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 $S' \in ND(S)$ it is $W(S) \geq W(S')$ proving one direction of the claim. Observing that the argument works backwards the proof completes. □

Multi-Commodity Weighted Network Congestion Games.
We prove that it is PLS-hard to compute an equilibrium in Weighted Congestion Games, even if it is a Weighted Network Congestion Game with all latency functions equal to the identity function, i.e., for any link e, $\ell_e(x) = x$. This result is stronger in some aspect than that of Section A.1 since we allow only the weights of the players to be exponential. Note that if both the coefficients of the linear latency functions and the weights of the players are polynomial, then best response dynamics converges to an equilibrium in polynomial time. For the proof, we reduce from LocalNodeMaxCut which, as we prove in Theorem 6, is PLS-complete.

Theorem 2. *Computing a Nash equilibrium in multi-commodity Weighted Network Congestion Games with all links having the identity function as their latency function is PLS-complete.*

Proof sketch. We will reduce from the PLS-complete problem LocalNodeMaxCut. For an instance of LocalNodeMaxCut we will construct a multi-commodity Network Congestion Game where every equilibrium will correspond to a maximal solution of LocalNodeMaxCut and vice versa. Our construction draws ideas from Ackermann et al. [1].

Skipping many of the details (that can be found in Section A.2), in the constructed instance for every $i \in [n]$, Player i will essentially have two path choices at equilibrium, say p_i^u and p_i^l, that cost-wise dominate all others. For any $i, j \in [n]$, p_i^u and p_j^u (resp. p_i^l and p_j^l) may have in common only one link that itself belongs only to them, say link e_{ij}^u (resp. e_{ij}^l). These links (e_{ij}^u and e_{ij}^l) are present only if $\{i, j\}$ is an edge of H, i.e., $\{i, j\} \in A$, and have the identity function as their latency function. What holds is that for every $i \in [n]$, p_i^u's cost differs from p_i^l's cost only because of the e_{ij}^u's and e_{ij}^l's for the j's neighboring i in H.

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 $S = \{i \in [n] : \text{player } i \text{ has chosen } p_i^u\}$. The proof shows that S is a solution to LocalNodeMaxCut. 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. By defining S_i to be the neighbors of i in S, i.e. $S_i = \{j \in S : \{i, j\} \in A\}$, and N_i be the neighbors of i in N, i.e. $N_i = \{j \in N : \{i, j\} \in A\}$, $c_i^u \leq c_i^l$ translates to

$$K + \Big(\sum_{j \in N_i} w_i + \sum_{j \in S_i} w_j \Big) \leq K + \Big(\sum_{j \in N_i} w_i + \sum_{j \in N_i \setminus S_i} w_j \Big)$$

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

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

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 $N \setminus S$. Thus for S and S' it is $W(S') \leq W(S)$. A similar argument (or just symmetry) shows that if $i \in N \setminus S$ and we send i from $N \setminus S$ to S to form a cut S' it would again be $W(S') \leq W(S)$. Thus, for any $S' \in ND(S)$ it is $W(S) \geq W(S')$ showing that S is a solution to LocalNodeMaxCut. Observing that the argument works backwards we have that from an arbitrary solution of LocalNodeMaxCut we may get an equilibrium for the constructed Weighted Congestion Game instance. □

4 PLS-completeness of LocalNodeMaxCut

In this section we present the proof of Theorem 6 which states that LocalNodeMaxCut is *PLS*-complete. As discussed in the preliminaries, LocalNodeMaxCut is in PLS and thus it remains to prove its PLS-hardness. To do so, we reduce CircuitFlip to it. We remind that an instance of LocalNodeMaxCut is composed by an undirected graph $H(N, A)$, and the weights of each node $i \in N$, $w_i \geq 0$. Given a circuit C of the CircuitFlip, we construct an instance of LocalNodeMaxCut such as from any Nash Equilibrium we can compute in polynomial time a locally maximal solution for the CircuitFlip problem.

Given a circuit C of CircuitFlip, the node-weighted graph that we construct is a combination of different gadgets which themselves might be seen as smaller instances of LocalNodeMaxCut. These gadgets have different goals but what each of them intuitively does is receiving some information to some of its nodes, the "input" nodes, and transforming it while carrying it through its internal part towards the "output" nodes. These input and output nodes are the nodes through which the different gadgets are connected. The connections are not always simple and we will later go into more detail on how these connections are done.

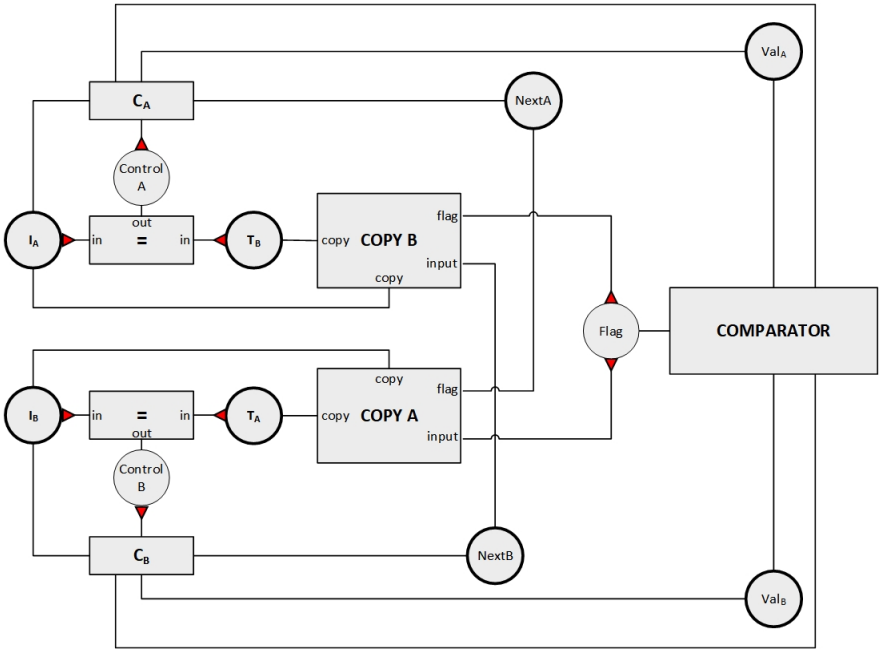

Figure 3: The high level construction of the NodeMaxCut *instance. Note that rectangles represent gadgets that will be defined below, circles represent nodes that take part in multiple gadgets, and bolded black circles represent a set of (n) such nodes. The computation gadgets are represented by C_A and C_B.*

Our construction follows a *flip-flop architecture* that has been previously used for reductions from CircuitFlip to LocalMaxCut and some of its variants [18; 12; 6], but requires more sophisticated implementations in many of its gadgets, since we deal with the special case of LocalNodeMaxCut. The most important differences and our major technical contributions are summarized at the end of the section. The constructed instance is presented at a high level in Figure 3. Next, we discuss the role of each separate gadget. The exact construction of each separate gadget is presented in the respective section of the appendix.

For a given circuit C of the CircuitFlip with n input gates and m output gates, we first construct the *Circuit Computing* gadgets $C\ell$ (for $\ell = A, B$). These gadgets are instances of LocalNodeMaxCut that simulate C in the following sense: I_ℓ is a set of n nodes whose values correspond to an "input string" of C. Val_ℓ is a set of m nodes whose values correspond to the output of circuit C with input string I_ℓ. $Next\ell$ is a set of n nodes that represent a neighbor string of I_ℓ with greater output value in C. More precisely, if the "input" string defined by the values of I_ℓ, has a neighbor solution (Hamming distance *1*) with *strictly* greater cost then the values of the n nodes in $Next\ell$ correspond to this neighbor string. In case such a neighbor string does not exist (which

means that I_ℓ is an optimal solution to CircuitFlip) the values of the nodes in $Next\ell$ equal I_ℓ.

These *Circuit Computing* gadgets have two separate functionalities: the *write mode* ($Control_\ell = 0$) and the *compute mode* ($Control_\ell = 1$). When C_ℓ is in the *write mode* the values of the input nodes I_ℓ are changed. When C_ℓ is in the *compute mode* the values of the nodes $Next\ell, Val_\ell$ are updated with the correct output values of the circuit C. To formalize the term *correct*, we introduce the following notation that we use through the section.

Definition 1. *For the given circuit* C *of* CircuitFlip. *We denote with:*

- Real-Val(I_ℓ) *the value of the circuit* C *with input string defined by the values of the nodes in* I_ℓ.
- Real-Next($I)_\ell$) *is a neighbor string (Hamming distance* 1*) of* I_ℓ *with strictly greater output value in circuit* C. *If such a string does not exist,* Real-Next(I_ℓ) $= I_\ell$.

The *Comparator* gadget compares Val_A and Val_B, which are intended to be *Real-Val*(I_A), *Real-Val*(I_B) and outputs *1* if $Val_A \leq Val_B$ or *0* otherwise. The result of this comparison is "*stored*" in the value of the *Flag* node. If $Flag = 1$ then I_B "*writes*" her better neighboring solution to I_A (symmetrically if $Flag = 0$). Intuitively, if this happens then in the next "*cycle*" I_A will have a better value and will write her improving solution to I_B. This goes on and on until no better neighbor solution exists and both I_A and I_B have the same output node values.

The *CopyB* gadget (respectively for *CopyX*") is responsible for writing the values of the nodes *NextB* to the nodes I_A when $Flag = 1$ (when $Val_A \leq Val_B$). When $Flag = 1$ the nodes T_B take the values of the nodes in *NextB*. Now if $I_A \neq NextB$ then the *Equality* gadget turns the value of the *ControlA* to 0 because $I_A \neq T_B$. Thus C_A enters *write mode* and the nodes in I_A adopt the values of the *NextB* nodes. Then *ControlA* becomes 1 since $I_A = T_B$ and C_A enters "*compute mode*". This means that values *Real-Next*(I_A), *Real-Val*(I_A) are written to the *output nodes NextA*, Val_A.

Before proceeding we present a proof-sketch of our reduction. The mathematically rigorous version of this proof is presented in the proof of Theorem 6 at the end of the section. We will prove that at any equilibrium of the instance of LocalNodeMaxCut of Figure 3 in which $Flag = 1$ (symmetrically if $Flag = 0$) three things hold:

1. $I_A = NextB$
2. $NextB =$ *Real-Next*(I_B)
3. *Real-Val*(I_B) $\geq$ *Real-Val*(I_A)

Once these claims are established we can be sure that the string defined by the values of nodes in I_B defines a locally optimal solution for CircuitFlip. This is because the above 3 claims directly imply that *Real-Val*(I_B) $\geq$ *Real-Val* (*Real-Next* (I_B)) which means that there is no neighboring solution of I_B with strictly greater cost. Obviously we establish symmetrically the above claims when $Flag = 0$.

Remark 1. *Since our construction in Figure 3 is an instance of* LocalNodeMaxCut *and the bitwise complement of an equilibrium is also equilibrium the term* Flag $= 1$ *seems meaningless. In the construction of the* Circuit Computing *gadget in Section B.2 (and in the constructions of all the presented gadgets) there also exist two* supernodes, *a* 1*-node and a* 0*-node, with huge weight that share an edge. As a result, at any equilibrium these nodes have opposite values. The term*

Flag $= 1$ *means that the* Flag *node has the same value with the* 1*-node. This notation is also used in subsequent lemmas and always admits the same interpretation. As one can see in the appendix, the construction of the gadgets assume nodes with values always* 0 *or* 1*. This can be easily established by connecting one such node with its complementary* supernode.

In the rest of the section we present the necessary lemmas to make the above presented proof sketch rigorous. To do so, we follow a three step approach. We first present the exact behavior of the *Circuit Computing* gadgets C_A, 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, C_B are the basic primitives of our reduction and are based on the gadgets introduced by Schäffer and Yannakakis [18] to establish PLS-completeness of LocalMaxCut. This type of gadgets can be constructed so as to simulate any boolean circuit C. The most important nodes are those corresponding to the input and the output of the simulated circuit C and are denoted as I, O. The other important node is the *Control* that switches between the *write* and the *compute mode* of the gadget. Figure 4 is an abstract depiction of this type of gadgets. The properties of the gadget are described in Theorem 3. Its proof is presented in Section B.2, where the exact construction of the gadget is presented. We first introduce some convenient notation that will help us throughout the proof of completeness.

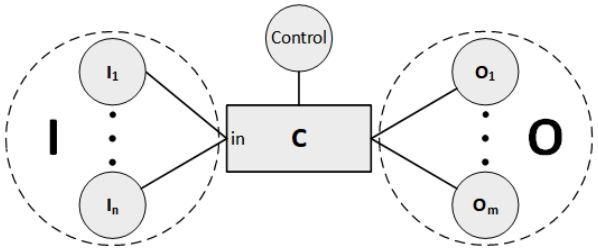

Figure 4: Circuit Computing *gadgets. The big, dashed "vertices" named I and O, represent all the input and output nodes respectively. This type of ("hyper"-)node is represented in the rest of the figures with a bold border.*

Definition 2. *Let an instance of* LocalNodeMaxCut *and a specific equilibrium of this instance. The* bias *that node i experiences with respect to the subset $N' \subseteq N$ is*

$$\left| \sum_{j \in N_i^1 \cap N'} w_j - \sum_{j \in N_i^0 \cap N'} w_j \right|$$

where N_i^0 is the set of neighbors of node i that choose 0 *(respectively for N_i^1)*

Bias is a key notion in the subsequent analysis. The gadgets presented in Figure 3 are subset of nodes of the overall instance. Each gadget is composed by the "input nodes", the internal nodes and the "output nodes". 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 equilibrium of the overall instance, the output nodes of the gadget experience some bias towards some values that depend on the values of the input nodes of the gadget. Since the

output nodes of a gadget may also participate as input nodes 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 equilibrium the bias that a node experiences from a gadget in which it is an output node, is greater than the sum of the biases of the gadgets in which it participates as input node.

Theorem 3 describes the equilibrium behavior of the input nodes I_ℓ and output nodes $Next\ell, Val_\ell$ of the *CircuitComputing* gadgets C_ℓ.

Theorem 3. *At any equilibrium of the* LocalNodeMaxCut *of Figure 3.*

1. *If* $Control\ell = 1$ *and the nodes of* $Next\ell$,Val_ℓ *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 node in* I_ℓ *experiences* 0 *bias from the internal nodes of* C_ℓ.
3. $Control\ell$ *experiences* $w_{Control\ell}$ *bias from the internal nodes of* C_ℓ.

Case 1 of Theorem 3 describes the *compute mode* of the *Circuit Computing* gadgets. At any equilibrium with $ControlA = 1$, and with the output nodes of C_A being indifferent with respect to other gadgets, then C_A computes its output correctly. Note that because the nodes in $NextA, ValA$ are also connected with internal nodes of other gadgets (*CopyX"* and *Comparator* gadgets) that may create bias towards the opposite value, the second condition is indispensable. Case 2 of Theorem 3 describes the *write mode*. If at an equilibrium $ControlA = 0$ then the nodes 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 3 describes the minimum bias that the equality gadget must pose to the Control nodes 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 $Control\ell$ nodes 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 nodes in C_ℓ gadgets.

Remark 2. *We remark that our construction of the* Circuit Computing *gadgets presented in Section B.2 ensures Theorem 3 for selecting* $w_{Control}$ *arbitrarily smaller than the weights for the internal nodes of the* Circuit Computing *gadgets. As we shall see up next this is crucial for our reduction and, as we discuss in the end of the section, this is a major difference with the respective* Circuit Computing *gadgets of other reductions in this vein [18; 12; 6].*

Solving the Feedback problem.
The purpose of this section is to establish the first case of the above presented claims i.e. at any equilibrium of LocalNodeMaxCut 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 4.

Theorem 4. *Let an equilibrium of the instance of* LocalNodeMaxCut *described in Figure 3.*

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

- *If* Flag $= 0$ *then* $I_B =$ NextA

We next present the necessary lemmas for proving Theorem 4.

Lemma 1. *Let an equilibrium of the overall* LocalNodeMaxCut *instance of Figure 3. Then*

- $ControlA = (I_A = T_B)$
- $ControlB = (I_B = T_A)$

In Section B.3 we present the construction of the equality gadget. This gadget is specifically designed so that at any equilibrium, its internal nodes create bias to $ControlA$ towards the value of the predicate $(I_A = T_B)$. Notice that if we multiply all the internal nodes 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 2). Lemma 1 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 3, the bias that $ControlA$ experiences from C_A is $w_{ControlA}$. As a result, the equilibrium 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 LocalNodeMaxCut 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 nodes in I_A takes the values of the nodes in $NextB$. This is formally stated in Lemma 2.

Lemma 2. *At any equilibrium point of the* LocalNodeMaxCut *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

In Section B.4, we present the construction of the *Copy* gadgets. At an equilibrium where $Flag = 1$, this gadget creates bias to the nodes in I_A, T_B nodes towards adopting the values of $NextB$. Since I_A, T_B also participate in the *Equality* gadget in order to establish Lemma 2 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 nodes of $CopyB$ with a sufficiently large constant. The "indifference" of the nodes in I_A, T_B with respect to the values of the internal nodes of the *Equality* gadget is denoted in Figure 3 by the *red marks* between the nodes in I_A, T_B and the *Equality* gadget.

In Case 2 of Lemma 2 the additional condition $ControlA = 0$ is necessary to ensure that $I_A = NextB$. The reason is that the bias of the *Copy* gadget to the nodes in I_A is sufficiently larger than the bias of the *Equality* gadget to the nodes in I_A, but not necessarily to the bias of the C_A gadget. The condition $ControlA = 0$ ensures 0 bias of the C_A gadget to the nodes I_A, by Theorem 3. As a result the values of the nodes in I_A are determined by the values of their neighbors in the $CopyB$ gadget.

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

Correctness of the Output Nodes.
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 5. *At any equilibrium of the instance of* LocalNodeMaxCut *of Figure 3:*

- *If* Flag $= 1$
  1. Real-Val$(I_A) \leq$ Real-Val(I_B)
  2. NextB $=$ Real-Next(I_B)
- *If* Flag $= 0$
  1. Real-Val$(I_B) \leq$ Real-Val(I_A)
  2. NextA $=$ Real-Next(I_A)

At first we briefly explain the difficulties in establishing Theorem 5. 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 4, the correctness of the nodes in $NextA$ or Val_A cannot be guaranteed. The reason is that in order to apply Theorem 3, $NextA$ and Val_A should experience 0 bias with respect to any other gadget they are connected to. But at an equilibrium, these nodes 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 nodes, i.e. $NextA = Real\text{-}Next(I_A)$ and $Val_A = Real\text{-}Val(I_A)$, is ensured by the design of the *CopyX"* and the *Comparator* gadgets. Apart from their primary role these gadgets are specifically designed to cause 0 bias to the output nodes of the *Circuit Computing* gadget to which the better neighbor solution is written. In other words at any equilibrium in which $Flag = 1$ and any node 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 5 and permits the application of Case 1 of Theorem 3. Lemma 3 and 4 formally state these "green marks".

Lemma 3. *At any equilibrium point of the* LocalNodeMaxCut *instance of Figure 3:*

- *If* Flag $= 1$ *then any node in* NextA *experience* 0 *bias with respect to the* CopyX" *gadget.*
- *If* Flag $= 0$ *then then any node in* NextB *experience* 0 *bias with respect to the* CopyB *gadget.*

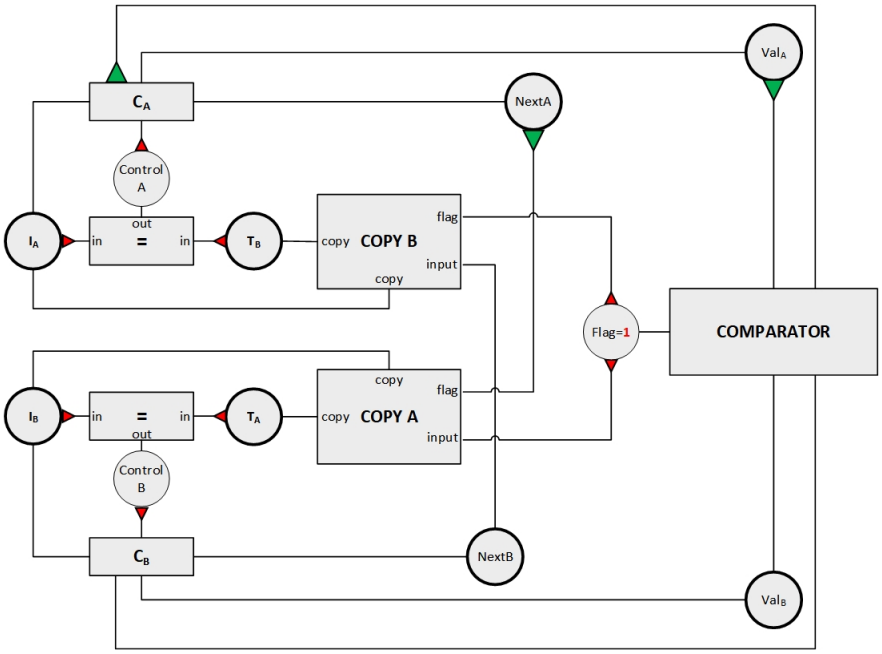

Figure 5: Since $Flag = 1$ any internal node of the C_2 gadget has 0 bias with respect to all the other gadgets. As a result, Theorem 3 applies.

Lemma 4. *Let an equilibrium of the instance of* LocalNodeMaxCut *of Figure 3:*

- *If* Flag $= 1$ *then all nodes of C_A experience* 0 *bias to the* Comparator *gadget.*
- *If* Flag $= 0$ *then all nodes of C_B experience* 0 *bias to the* Comparator *gadget.*

Remark 3. *The reason that in Lemma 4 we refer to all nodes of C_A (respectively C_B) and not just to the nodes in Val_A (respectively Val_B) is that in the constructed instance of* LocalNodeMaxCut *of Figure 3, we connect internal nodes of the C_A gadget with internal nodes of the* Comparator *gadget. This is the only point in our construction where internal nodes of different gadgets share an edge and is denoted in Figure 3 and 5 with the direct edge between the C_A gadget and the* Comparator *gadget.*

Now using Lemma 3 and Lemma 4 we can prove the correctness of the output nodes *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 nodes in *NextB*, Val_B when $Flag = 0$).

Lemma 5. *Let an equilibrium of the instance of* LocalNodeMaxCut *of Figure 3:*

- *If* Flag $= 1$ *then* NextA $=$ Real-Next(I_A), $\text{Val}_A =$ Real-Val(I_A).
- *If* Flag $= 0$ *then* NextB $=$ Real-Next(I_B), $\text{Val}_B =$ Real-Val(I_B).

Proof. We assume that $Flag = 1$ (for $Flag = 0$ the exact same arguments hold). By Theorem 4 we have $I_A = NextB$ and by Lemma 2 we have that $T_B = NextB$. As a result, $I_A = T_B$ and by Lemma 1 $ControlA = 1$. Lemma 3 and Lemma 4 guarantee that the nodes 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 3 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 nodes in Val_B and $NextB$ when $Flag = 1$. We remind again that, even if at an equilibrium $ControlB = 1$, we could not be sure about the correctness of the values of these output nodes due to the bias their neighbors in the *CopyB* and the *Comparator* gadget (Theorem 3 does not apply). The *Comparator* gadget plays a crucial role in solving this last problem. Namely, it also checks whether the output nodes 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 nodes of the C_A, C_B gadgets with the internal nodes of the *Comparator* gadget (Figure 3: edges between C_A, C_B and *Comparator*).

Lemma 6. *At any equilibrium of the* NodeMaxCut *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 nodes *NextB*, i.e. $NextB = Real\text{-}Next(I_B)$, is not guaranteed by application of Theorem 3 (as in the case of correctness of $NextA, Val_A$), but from the construction of the *Comparator* gadget. Lemma 6 is proven in Section B.5 where the exact construction of this gadget is presented. Notice that Lemma 6 says nothing about the correctness of the values of the output nodes 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 $(Real\text{-}Val(I_A) \leq Real\text{-}Val(I_B))$ 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 LocalNodeMaxCut. The crucial differences between our *Comparator* and the *Comparator* of the previous reductions [18; 12; 6] are discussed in the end of the section. Lemma 7 formally states the robustness of the outcome of the *Comparator* even with "wrong values" in the nodes of Val_B and is proven in Section B.5.

Lemma 7. *At any equilibrium of the* LocalNodeMaxCut *instance of Figure 3:*

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

$$Real - Val(I_A) \leq Real - Val(I_B)$$

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

$$Real - Val(I_B) \leq Real - Val(I_A)$$

We are now ready to prove Theorem 5.

Proof of Theorem 5. Let an equilibrium of the instance of Figure 3 with $Flag = 1$ (respectively for $Flag = 0$). By Lemma 6, $NextB = Real\text{-}Next(I_B)$ and thus Case 1 is established. Moreover by Lemma 5, $NextA = Real\text{-}Next(I_A)$ and $Val_A = Real\text{-}Val(I_A)$. As a result, Lemma 7 applies and $Real\text{-}Val(I_A) \leq Real\text{-}Val(I_B)$ (Case 2 of Theorem 5) □

Having established Theorem 4 and 5 the *PLS*-completeness of LocalNodeMaxCut follows easily. For the sake of completeness we present the proof of Theorem 6 that we describe in the beginning of the section.

Theorem 6. LocalNodeMaxCut *is PLS-complete.*

Proof. For a given circuit C of the CircuitFlip, we can construct in polynomial time the instance of LocalNodeMaxCut of Figure 3. Let an equilibrium of this instance. Without loss of generality, we assume that $Flag = 1$. Then, by Theorem 4 and Theorem 5, $I_A = NextB$, $NextB = Real - Next(I_B)$ and $Real - Val(I_A) \leq Real - Val(I_B)$. As a result, 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 $I_B \neq Real - Next(I_B)$, by Definition 1 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 CircuitFlip problem. □

Our technical contributions

As already mentioned the construction of Figure 3 follows the *flip-flop architectures* of previous PLS reductions [18; 12; 6]. In all these reduction the selection of the weights in the connecting edges played a crucial role. We briefly discuss the points of our reduction, which are at the heart of the PLS-completeness proof of LocalNodeMaxCut.

Our *Circuit Computing* gadgets are based on the *Circuit Computing* gadgets introduced by Schäffer and Yannakakis [18] (and used by Elsässer et al for similar problems [12; 6]). There is one major modification of these gadgets, which concerns Case 3 of Theorem 3. Our *Circuit Computing* gadgets can be constructed with the weight $w_{Control}$ being arbitrarily smaller than the weights of the other nodes of the *Circuit Computing* gadget. The importance of this property for our reduction is twofold: At first, Case 3 guarantees the *Control* node will have very small *bias* with respect to the *Circuit Computing* gadget. This is very important since we were able to select the weights in the *Equality* gadget large enough so as to control the *write* and the *compute* mode but at the same time small enough so as not to affect the values of "*input nodes*" I_A, I_B. The second crucial reason that necessitates $w_{Control}$ being small is that *Control* should follow the "*output*" of the *Equality* gadget and not vice versa. It is here that the main difference between the LocalNodeMaxCut problem and the original LocalMaxCut arises. The major difference between the *Circuit Computing* gadgets introduced by Schäffer and Yannakakis [18] and ours, is that the *Control* node has small bias with respect to the *Circuit Computing* gadget but weight comparable to the weights of the internal nodes of the *Circuit Computing* gadget. All these reductions ensure that *Control* does not influence its neighbors (outside the circuit computing gadget) by selecting sufficiently small weights in the edges, something that cannot be done in the LocalNodeMaxCut problem. Our first main technical contribution is establishing Cases 1 and 2 with the *Control* node having arbitrarily smaller weight than the other nodes in the gadget. To achieve so we design a *Leverage* gadget that is internally used in the *Circuit Computing* gadget. This gadget reduces the influence of a node with large weight to a node with small weight and is presented in Section B.1.

The second important difference of our reduction concerns the *red marks* between the *Flag* and the *Copy* gadgets in Figure 3. These *red marks* indicate that *Flag* node takes the output of the *Comparator* gadget. These marks are more tricky to establish than the other *red marks* in Figure 3. The reason is that both the *Comparator* and the *Copy* gadgets have input nodes in $NextA, NextB, Val_A, Val_B$ which have weights of the same order of magnitude as output nodes of the *Circuit Computing* gadgets. As a result, this time the weights of the internal nodes in the *Comparison* gadget cannot be selected sufficiently larger than the respective weights of the *Copy*

gadgets. In the work of Schäffer and Yannakakis [18; 12; 6] the bias of the *Flag* from the *Copy* gadgets was made negligible to the respective bias from the *Comparator* by connecting the *Flag* with the *Copy* gadgets with edges of sufficiently small weight. This is the second point where the absence of weights on the edges plays a major role. We overcome this difficulty by again using the *Leverage* gadget internally in the *Copy* gadget. Now the *Leverage* gadget decreases the influences of the internal nodes of the *Copy* gadgets to the *Flag* making its bias much smaller than the bias of the *Comparator*. This is formally established in Lemma 7.

Our final major technical contribution concerns the design of the *Comparator* gadget. As stated in Lemma 7 our *Comparator* outputs ($\textit{Real-Val}(I_A) \geq \textit{Real-Val}(I_A)$) even if some "*input nodes*" of the *Comparator* have incorrect values. In the previous reductions the *Comparator* guaranteed correctness both on the values of the nodes in $NextB$, Val_B by the use of the suitable weights on the connecting edges. Having the correct "input values", the comparison step is quite straightfoward to implement. This decoupled architecture of the *Comparator* could not be implemented with a LocalNodeMaxCut instance, due to the absence of edge-weights, and we had to deploy an *all at once Comparator* that ensures correctness to some of its input nodes so as to perform correctly the comparison step.

5 Conclusions and Future Work

In this paper we proved that weighted CGs are significantly harder than unweighted ones, even for cases where unweighted CGs are easy. Alongside these results we proved that LocalNodeMaxCut is PLS-complete. We note that the reductions we presented in Section 3 are both tight in the sense defined by Schäffer and Yannakakis [18], which means they preserve certain characteristics of the transition graph. In particular, for the first reduction we have that there are instances of weighted CGs such that any best response sequence has exponential length, as well as the problem of computing the equilibrium reached from a given initial state being PSPACE-hard.

However, we note that our reduction of CircuitFlip to LocalNodeMaxCut is not tight. To see why this is true, one has to consider that the *Copy* and *Equality* gadgets do not guarantee that the *Circuit Computing* gadget will not enter *compute* mode before the full input is changed. Hence, we might "jump ahead" and reach an equilibrium faster than CircuitFlip would allow, preventing the reduction from being tight.

Future directions include addressing the case where both constraints we considered coincide, i.e., the case of single-commodity weighted CGs on series-parallel networks with the identity function on all links. Is this setting constrained enough so that a polynomial algorithm for computing equilibria is possible or is it still PLS-hard? Another promising direction is using the ideas and reductions introduced here to obtain hardness of approximating equilibria for weighted CGs, similar to the work of Skopalik and Vöcking [19]. Lastly, it would be interesting to examine whether, in the context of smoothed analysis, perturbing the weights of nodes would lead to smoothed (quasi-)polynomial running time of best response sequences (similar to Angel et al. [2]).

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 *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 429–437. ACM, 2017.

[3] Ioannis Caragiannis, Angelo Fanelli, Nick Gravin, and Alexander Skopalik. Efficient computation of approximate pure nash equilibria in congestion games. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 532–541, 2011. doi: 10.1109/FOCS.2011.50. URL https://doi.org/10.1109/FOCS.2011.50.

[4] Ioannis Caragiannis, Angelo Fanelli, Nick Gravin, and Alexander Skopalik. Approximate pure nash equilibria in weighted congestion games: Existence, efficient computation, and structure. *ACM Trans. Economics and Comput.*, 3(1):2:1–2:32, 2015.

[5] Steve Chien and Alistair Sinclair. Convergence to approximate nash equilibria in congestion games. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007*, pages 169–178.

[6] Robert Elsässer and Tobias Tscheuschner. Settling the complexity of local max-cut (almost) completely. In *Automata, Languages and Programming - 38th International Colloquium, ICALP*, pages 171–182, 2001.

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

[8] Alex Fabrikant, Christos H. Papadimitriou, and Kunal Talwar. The complexity of pure nash equilibria. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 604–612, 2004. doi: 10.1145/1007352.1007445. URL https://doi.org/10.1145/1007352.1007445.

[9] Angelo Fanelli and Luca Moscardelli. On best response dynamics in weighted congestion games with polynomial delays. In *Internet and Network Economics, 5th International Workshop, WINE 2009*, pages 55–66.

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

[11] Dimitris Fotakis, Spyros C. Kontogiannis, and Paul G. Spirakis. Selfish unsplittable flows. *Theor. Comput. Sci.*, 348(2-3):226–239, 2005.

[12] Martin Gairing and Rahul Savani. Computing stable outcomes in hedonic games. In *Algorithmic Game Theory - Third International Symposium, SAGT*, pages 174–185, 2010.

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

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

[15] Paul W. Goldberg. Bounds for the convergence rate of randomized local search in a multiplayer load-balancing game. In *Proceedings of the Twenty-Third Annual ACM Symposium on Principles of Distributed Computing, PODC 2004*, pages 131–140.

[16] 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.

[17] Wil Michiels, Emile Aarts, and Jan Korst. *Theoretical Aspects of Local Search*. Springer Publishing Company, Incorporated, 1st edition, 2010.

[18] Alejandro A. Schäffer and Mihalis Yannakakis. Simple local search problems that are hard to solve. *SIAM J. Comput.*, 20(1):56–87, 1991.

[19] Alexander Skopalik and Berthold Vöcking. Inapproximability of pure nash equilibria. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 355–364, 2008. doi: 10.1145/1374376.1374428. URL https://doi.org/10.1145/1374376.1374428.

A The Proofs of the Theorems of Section 3

A.1 The proof of Theorem 1

We will reduce from the PLS-complete problem LocalMaxCut and given an instance of MaxCut we will construct a Weighted Network Congestion Game for which the Nash equilibria will correspond to maximal solutions of LocalMaxCut 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 $H(N, A)$ be an edge-weighted graph of a LocalMaxCut instance and let $n = |N|$ and $m = |A|$. In the constructed Weighted Network Congestion Game 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 $\{i, j\} \in A$ 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 $n+1$ links. For any $k \in [n]$ other than i, j there is an $o_{ij} - d_{ij}$ link 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 $o_{ij} - v_{ij}$ links, 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 $v_{ij} - d_{ij}$ link with latency function $\ell_{ij}(x) = \frac{w_{ij}x}{w_i w_j}$, where w_{ij} is the weight of edge $\{i, j\} \in A$ 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 link. 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 links 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 links, contains some links with latency function of the form $\frac{w_{ij}x}{w_i w_j}$. These links for path p_i^u or p_i^l is in one to one correspondence to the edges of node 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 LocalMaxCut. 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 link in G (the edge with latency function $w_{ij}x/w_i w_j$) only if $\{i, j\} \in A$. The correctness of the reduction lies in the fact that players in G try to minimize their costs incurred by these type of links in the same way one wants to minimize the sum of the weights of the edges in each side of the cut when solving LocalMaxCut.

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 7. *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, all players of weight w_k (for any $k \in [n]$) will go through the path that contains a link 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 link of F_{ij} with latency function $\ell_k(x)$ and call e_{ij} the link with latency function $\frac{w_{ij}x}{w_i w_j}$. 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 link e_{ij} and assume that in F_{ij} there are only n parallel paths each consisting of a single link.

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 link 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 link 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

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

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 link e_{ij} may have is bounded above by $c^{ij} = \frac{w_{ij}3\sum_{l=1}^{n} 16^l}{w_i w_j}$, 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 that goes through e_k. □

Other than revealing a value for D, the proof of Porposition 7 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 7 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 links 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 8. *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 LocalMaxCut and vice versa. Given a maximal solution S of LocalMaxCut 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 $k \in N \setminus S$ routes 1 player through p_k^u and 2 players through p_k^l is an equilibrium. Conversely, given an equilibrium Q the cut $S = \{k \in N : 2 \text{ players have chosen } p_k^u \text{ at } Q\}$ is a maximal solution of LocalMaxCut.

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 8, implies

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

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

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

Define $S = \{i \in N : x_i^u = 2\}$. By our assumption it is $k \in S$ and the left side of (2), i.e., $\sum_{\{k,j\}\in A} w_{kj}(x_j^u - 1)$, is the sum of the weights of the edges of H with one of its nodes being k and the other belonging in S. Similarly, the right side of of (2), i.e., $\sum_{\{k,j\}\in A} w_{kj}(x_j^l - 1)$ is the sum of the weights of the edges with one of its nodes being k and the other belonging in $N \setminus S$. But then (2) directly implies that for the (neighboring) cut S' where k goes from S to $N \setminus S$ 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 $S' \in ND(S)$ 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 proo, to define function ϕ_2, given the constructed instance and one of its solutions, say s', ϕ_2 returns solution $s = \{k \in N : 2 \text{ players have chosen } p_k^u \text{ at } s'\}$. □

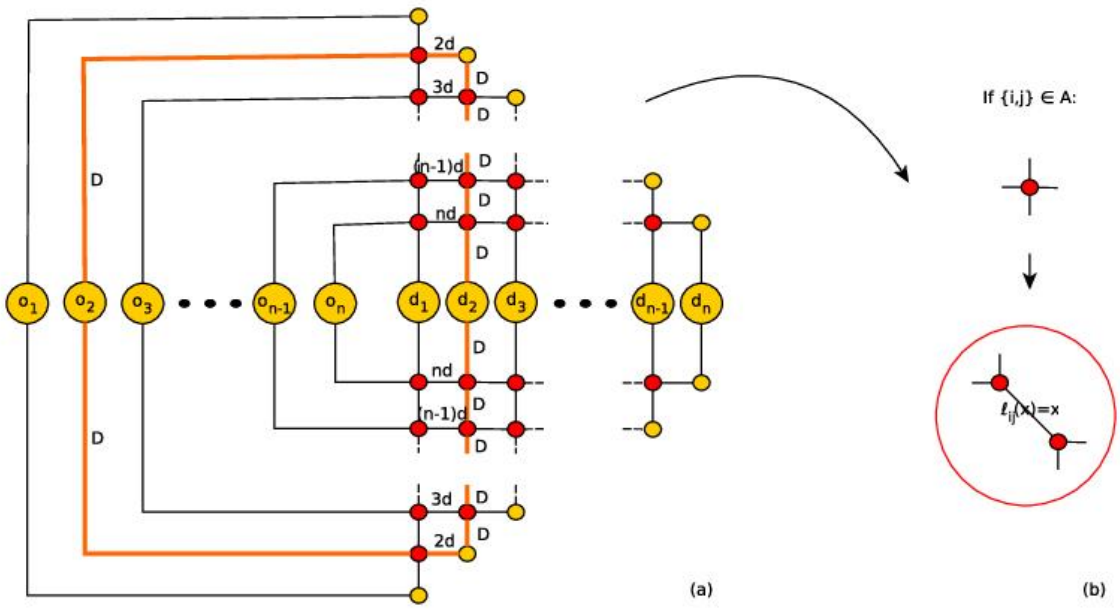

Figure 6: (a) The construction of the reduction of Theorem 2. As an example, in orange are the least costly $o_2 - d_2$ paths p_2^u (up) and p_2^l (down), each with cost equal to $2D + 2d + (n-2)D$. (b) The replacement of the red node at the i-th row and j-th column of the upper half-grid whenever edge $\{i, j\} \in A$. A symmetric replacement happens in the lower half-grid.

A.2 The proof of Theorem 2

We will reduce from the PLS-complete problem LocalNodeMaxCut. Our construction draws ideas from Ackermann et al. [1]. For an instance of LocalNodeMaxCut we will construct a multi-commodity Network Congestion Game where every equilibrium will correspond to a maximal solution of LocalNodeMaxCut 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 link, but for ease of presentation we will first prove our claim assuming we can use constant latency functions on the links. Then we will describe how we can drop this assumption and use only the identity function on all links, and have the proof still going through.

Let $H(N, A)$ be the node-weighted graph of an instance of LocalNodeMaxCut and let $n = |N|$ and $m = |A|$. The Weighted Network Congestion Game will have n players, with player i having her own origin destination $o_i - d_i$ pair and weight w_i equal to the weight of node $i \in N$. 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 node i should lie in order to get a maximal solution of LocalNodeMaxCut.

The initial network construction is shown in Fig. 6. 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 links of the two half-grids that are all present, there are links 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 links that connect to an origin or a destination and all the vertical links of the half-grids will have constant D as their latency function, and any horizontal link 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. 6a). 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 links, then it will have to pass through at least $(2 + i - 1) + 2$ vertical links 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 $2D = 2n^3 d$, which is already more than paying all horizontal links; 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 links.

To complete the construction if $\{i, j\} \in A$ (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 a link, 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. 6b). 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 $\{i, j\} \in A$ then p_i^u and p_j^u have a single common link and p_i^l and p_j^l have a single common link, 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 LocalNodeMaxCut. 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. $S_i = \{j \in S : \{i, j\} \in A\}$, and N_i be the neighbors of i in N, i.e. $N_i = \{j \in N : \{i, j\} \in A\}$, $c_i^u \le c_i^l$ translates to

$$\Big(2D + i(i-1)d + (n-i)D\Big) + \Big(\sum_{j \in N_i} w_i + \sum_{j \in S_i} w_j\Big) \le \Big(2D + i(i-1)d + (n-i)D\Big) + \Big(\sum_{j \in N_i} w_i + \sum_{j \in N_i \setminus S_i} w_j\Big)$$

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

$$\sum_{j \in S_i} w_j \le \sum_{j \in N_i \setminus S_i} w_j \Leftrightarrow \sum_{j \in S_i} w_i w_j \le \sum_{j \in N_i \setminus S_i} w_i w_j.$$

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 $N \setminus S$. Thus for S and S' it is $W(S') \le W(S)$. A similar argument (or just symmetry) shows that if $i \in N \setminus S$ and we send i from $N \setminus S$ to S to form a cut S' it would again be $W(S') \le W(S)$. Thus, for any $S' \in ND(S)$

it is $W(S) \geq W(S')$ showing that S is a solution to LocalNodeMaxCut. Observing that the argument works backwards we have that from an arbitrary solution of LocalNodeMaxCut we may get an equilibrium for the constructed Weighted Congestion Game 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 links 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 a link $\{j, k\}$ with a three link path $j - o_{jk} - d_{jk} - k$, adding a 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 link $\{o_{jk}, d_{jk}\}$ at equilibrium.

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

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

With these definitions, at equilibrium, all complementary players will go through the correct links and, due to the complementary players, all links that connect to an origin or a destination will have cost $\approx D$, all vertical links of the half-grids will cost $\approx D$, and any horizontal link 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 link 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 links and share the same latency functions. All the above make the analysis go through in the same way as in the simplified construction. □

B Details for the PLS-completeness Proof of LocalNodeMaxCut

In the following sections we fully present all the details of the construction for the proof of Theorem 6. Recall that our NodeMaxCut instance is composed of the following gadgets:

1. *Leverage* gadgets that are used to transmit nonzero bias to nodes of high weight.
2. Two *Circuit Computing* gadgets A,B 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 ther other, and vice versa.
5. Two controller gadgets that decide which circuit should enter *write* or *compute* mode.

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

Moreover, when we have constant nodes of a certain value (i.e. pinned to 1) we connect them with one of two *supernodes*. Supernodes are nodes of huge weight that share an edge and as a result at any equilibrium these nodes have opposite value. In particular, these supernodes have weight 2^{1000N} which dominates the weight of any other node, given we chose N as described above. The term $Control = 1$ means that the $Control$ node has the same value with the 1-node.

When we reference the value of a circuit, we will mean the value that the underlying CircuitFlip instance would output given the same input.

When we reference the value of a node we will mean the side of the cut it lies on. There are two values, $0, 1$ for each side of the cut.

B.1 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 node with relatively small weight A and we want to bias a node with large weight B. For example, the large node might be indifferent towards its other neighbors, which would allow even a small bias from the small node to change its state. We would also like to ensure that the large node 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 2 to change according to the outputs of circuit 1 and not the other way around. Another example involves the *Equality* Gadget, which influences the $Control_\ell$ of the *Circuit Computing* gadget. The nodes of the Gadget have weights of the order of 2^{10N}, while the control nodes of the *Circuit Computing* gadget are of the order of 2^{100N}. We would like the output of the gadget to bias the $Control_\ell$ nodes, while also remaining independent from them.

Let's get back to the original problem. A naive solution would be to connect node A directly with node B. However, this would result in node B biasing node A due to the larger weight it possesses. For example, if we connected *Control1* with the control variables of Circuit 2, then they would always bias *Control1* with a very large weight, rendering the entire *Equality* gadget useless. We would like to ensure that node 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 nodes 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 nodes A and B. We use two parameters x, ϵ in the construction. We first want to ensure that node A experiences a small bias from the gadget. This is why we put nodes $L_{1,1}, L_{1,2}$ at the start with weight $B/2^{x+1} + \epsilon$, which puts a relatively small bias. We want these nodes to be dominated by A. This is why nodes $L_{1,3}, L_{1,4}$ have combined weight less than A. However, these nodes 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 nodes $L_{x,1}, L_{x,2}$, whose combined weight is slightly larger than B. This means that nodes $L_{x,3}, L_{x,4}$

are not dominated by B and can therefore be connected directly with it. The details of the proof are given below.

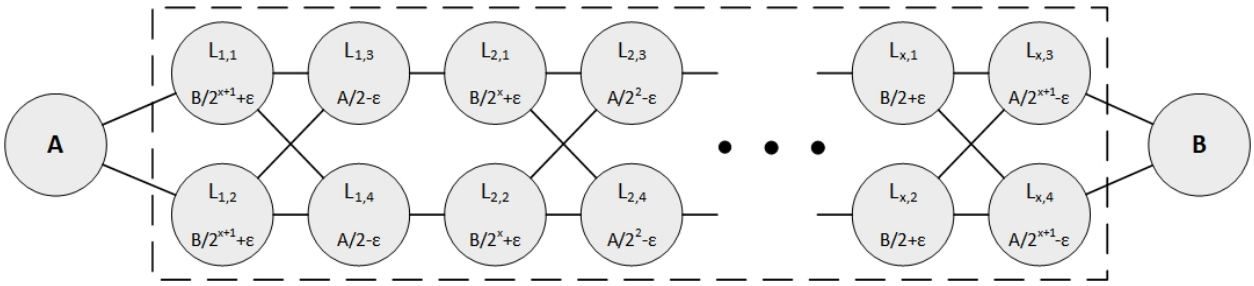

Figure 7: The Leveraging Gadget

Lemma 8. *If the input node A of a leverage gadget with output node B, parameters x, ϵ, has value 1, then the output node experiences bias $w_A/2^x + 2 * \epsilon$ towards 0, while the input node A experiences bias $w_B/2^x - 2 * \epsilon$ 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 nodes $L_{1,1}, L_{1,2}$. They both experience bias w_A towards the opposite value of A, which is greater than the remaining weight of their neighbors $2 * w_A - 2 * \epsilon$, and hence they are both dominated to take the opposite value of A. Similarly, the nodes $L_{1,3}, L_{1,4}$ are now biased to take the opposite values of $L_{2,1}, L_{2,2}$ with bias at least $w_B/2^x + 2 * \epsilon$, which is greater than the remaining neighbors of $w_B/2^x + \epsilon$. Hence, both $L_{1,3}, L_{1,4}$ have the same value as A in any equilibrium. In a similar way, we can prove that, in any equilibrium $L_{i,3} = L_{i,4} = A$, and therefore B experiences bias $w_A/2^x + 2 * \epsilon$ towards the opposite value of A, while A experiences bias at most $w_B/2^x - 2 * \epsilon$ from this gadget. □

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 $\epsilon = 2^{-1000N}$ 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.

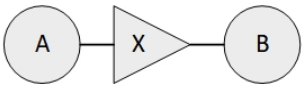

Figure 8: Leveraging Gadget *notation*

B.2 Circuit Computing Gadget

Each of the two computing circuits is meant to both calculate the value of the underlying CircuitFlip 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 node.

In this section we present the gadgets that implement the above circuits in a LocalNodeMaxCut instance. The construction below is similar to the constructions of Schäffer and Yannakakis

used to prove LocalMaxCut PLS-complete ([18]). 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 nodes can be used for comparing the binary values of the outputs.

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

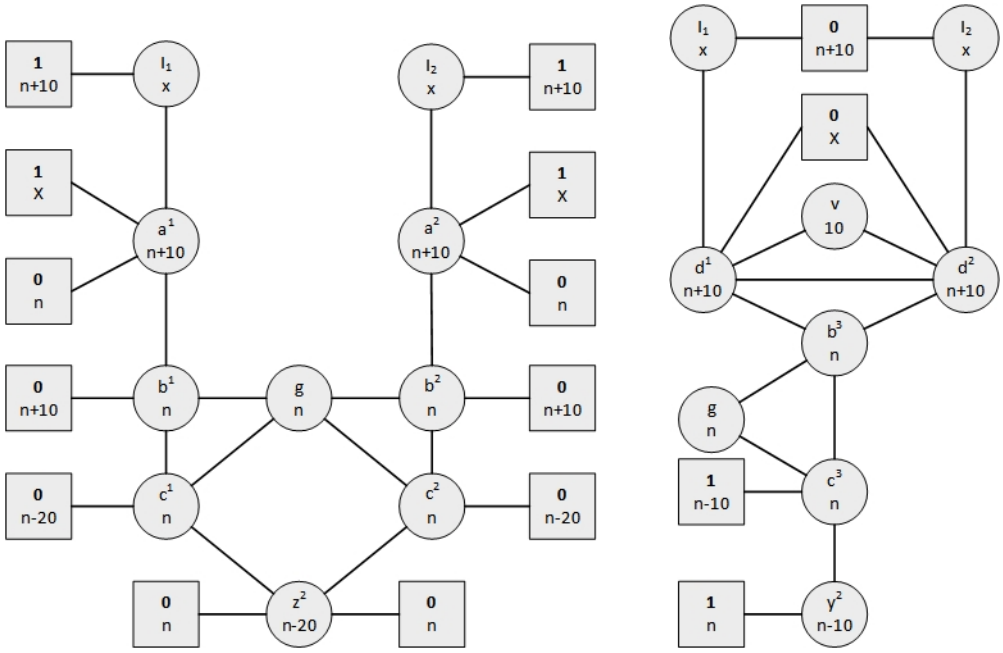

Figure 9: The NodeMaxCut *instance implementing a NOR(n) gadget.*

Moreover, we have y_i^1,z_i^1 nodes which are meant to bias the internal nodes 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 nodes of weight 2^{-200N}, shown in Figure 10.

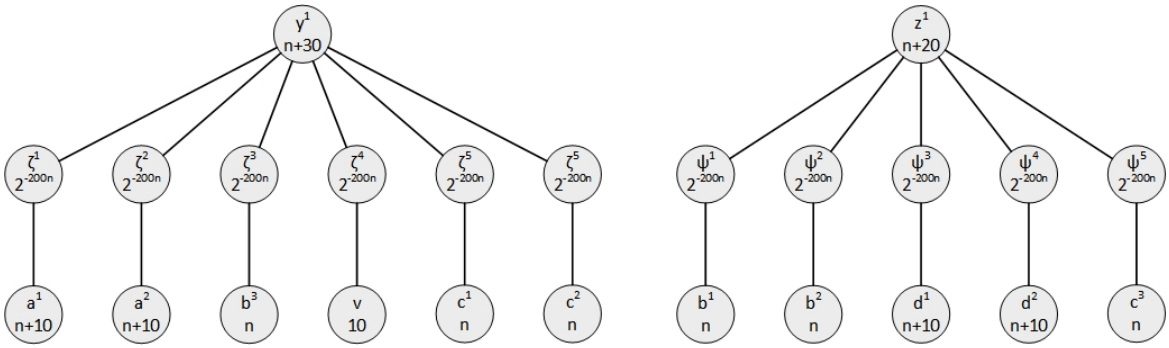

Figure 10: Local bias to internal nodes from y_i^1, z_i^1

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

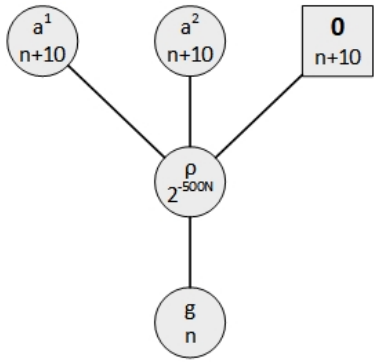

Figure 11: Extremely small bias to NOR output value.

These control nodes, y^1, z^1, y^2, z^2 are meant to decide the functionality of the gadget. We say that the y, z nodes 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 nodes 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 ([18]) we add two extra control variable nodes 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 ([18]), 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.

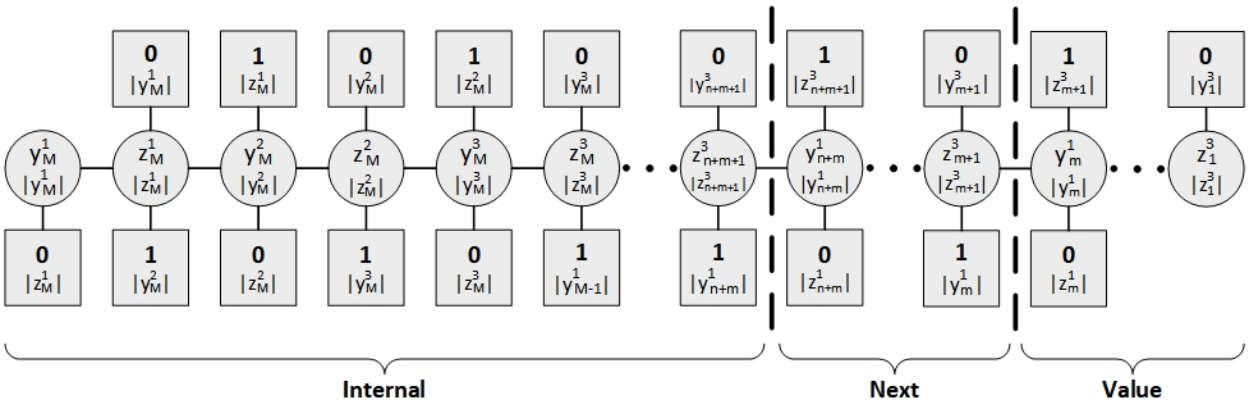

Figure 12: Connecting the control nodes of the NOR gadgets. Recall that M is the number of total gates in the circuit, n is the number of solution bits and m is the number of value bits. Note that the gates are ordered in reverse, i.e the first gates have highest index.

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 nodes of these gadgets are all multiplied by a 2^{100N} weight, except the nodes 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 possible result of the comparison. The auxiliary nodes introduced above, which are meant to induce small biases to internal nodes, are not multiplied by anything.

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

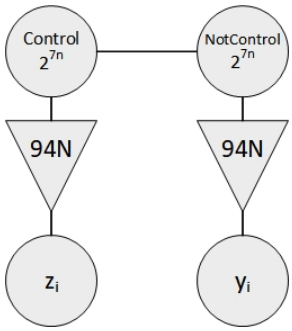

Figure 13: We use a single node Control to bias all control nodes y, z. Note that this node is connected with the y, z nodes through leverage gadgets

We now prove the properties of these gadgets.

Lemma 9. *In an equilibrium, 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 $2^{100N} * (2 * 2^i + 10) + 2^{-200N}$ towards 1 which dominates his best response. Hence, $b_i^1 = 1$ in this case. Now a_i^1 has bias at least $w(I1) + 2^{100N} * 2^i + 2^{-200N}$ 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 $2^{100N} * 2 * 2^i + 10 + w(I_i^1) + 2^{-200N}$ bias towards 1 it can only be $d_i^1 = 1$ in equilibrium. 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 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 $2^{100N} * 2^i + 10$ towards 1. Along with the 2^{-200N} bias from z_i^1 we have that $b_i^1 = 1$ in any equilibrium. Similarly, $b_i^2 = 1$ by symmetry. Hence, in this case as well a_i^1 is 0 in any equilibrium. Similarly, we get that $a_i^2 = 0$ in any equilibrium.

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 equilibrium.

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.

□

Lemma 10. *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 node a_i^1 is biased towards value 0 with weight at least $2 * 2^{i+1} * 2^{100N}$ by $I_1(g_i)$ and constant node 1. This bias is greater than the weight of all the other neighbors of a_i^1 combined. Hence, in equilibrium, $a_i^1 = 0$. Hence, node b_i^1 is biased towards value 1 with weight at least $2 * |a_i^1|$, 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 $g_i = 1$, node c_i is biased towards 0 with weight at least $2 * 2^i * 2^{100N}$, 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 node z_i^2. Its neighbors are two nodes of weight $2^i * 2^{100N}$ with constant value 0, nodes c_i^1, c_i^2 , y_i^3 and a constant node 1 with weights $2^{100N} * (2^i - 50)$ and some auxiliary nodes of negligible weight. If $c_i^1 = 0$, then z_i^2 is biased towards 1 with weight at least $3 * 2^i * 2^{100N}$, which is greater than the weight of the remaining neighbors combined. Hence, in equilibrium, $z_i^2 = 1$. 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 $I_1(g_i) = 0$, node d_i^1 is biased towards 1 with weight at least $2 * 2^{i+1} * 2^{100N}$, 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 node b_i^3 is biased towards

0 with weight at least $2 * (2^i + 10) * 2^{100N}$, which is greater than the weight of its other nodes combined. This implies that $b_i^3 = 0$. By the same reasoning, $v_i = 0$. Since $b_i^3 = g_i = 0$, node c_i^3 is biased towards 1 with weight at least $2 * 2^i * 2^{100N}$, which is greater than the weight of its other nodes combined. Hence, $c_i^3 = 1$. Now we focus on node y_i^2. Its neighbors are a node of weight $2^i * 2^{100N}$ with constant value 1, node c_i^3 with weight $2^i * 2^{100N}$, z_i^1,a constant node 1 both with weight $2^{100N} * (2_2^i 0)$,z_i^2 and a constant 0 with weight $2^{100N} * 2^i - 10$ and some auxiliary nodes of negligible weight. Hence, y_i^2 is biased towards 0 with weight at least $2 * 2^i * 2^{100N}$, 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 nodes of the NOR gadget. Hence, if the bias of the remaining nodes is towards 1, then $z_i^2 = 1$ in equilibrium. We notice that nodes y_i^2 and constant node 0 bias our node with weight $2 * (2^i - 10) * 2^{100N}$, which is greater that any potential bias by nodes y_i^3 and constant 1 in the chain of total weight $2 * 2^{100N} * (2^i - 50)$. 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, node y_{i+1} is not biased towards 1 by any node in the NOR gadget. However, it experiences bias towards 0 from node z_i and constant node 1, which is greater than any other potential bias from its other neighbors. Hence, $y_{i+1} = 0$ and the claim follows. □

Lemma 11. *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 nodes of the gate G_i. If g_i is incorrect then g_i is indifferent with respect to the other nodes of the gate G_i, but gains the node 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 node 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 node 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 node as their neighbors, which along with the bias from y_i, dominates their bias. Since z_i^2 neighbors with two 1 nodes and two 0 nodes 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 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 node 1 and z_i^1. Hence, $b_i^3 = 1$. The node 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 12. *If Control = 1 then all y, z nodes have a 2^{-87N} bias towards their natural values. If Control = 0 then all y, z nodes have a 2^{-87N} bias towards their unnatural values.*

Proof. The *NotControl* nodes are dominated by *Control*'s bias of 2^{7N} and hence have the opposite value. By lemma 8 we have that *Control* and *NotControl* experience at most 2^{6N} bias, while the y, z nodes experience 2^{-87N} bias towards the values opposite *Control* and *NotControl*, which proves the claim. □

Lemma 13. *Assuming all nodes of the computing circuit gadget are in equilibrium 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 nodes experience a bias towards their unnatural biases by lemma 12. Since the bias that biases them towards their unnatural values is greater than the weight of the internal nodes connected to y^1, z^1, y^3, z^3 it must be that one of the nodes y^2, z^2 have unnatural values. However, by lemma 10 all control nodes for $j < k$ are also unnatural. Assume that the output nodes of G_i are only internal to the circuit, i.e. no node except those belonging to the computing gadget is connected to them. Since by lemma 9, unnatural values for y_i^1, z_i^1 imply that the input nodes of gates G_i are indifferent to G_i, the node g_i would be dominated by the bias from the auxiliary node ρ. If g_k is an output node 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 11.

If *Control* = 0, consider the least k such that the y, k control nodes have their natural values. Since the $y_i^1, z_i^1, y_i^3, z_i^3$ nodes are dominated by the 2^{-87N} bias ensured by lemma 12 we have that the only nodes with natural values can be y_i^2, z_i^2. However, even these nodes can only be biased towards their unnatural values since, by lemmas 10 and 11, 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 3. *At any equilibrium of the* LocalNodeMaxCut *of Figure 3.*

1. *If* $Control\ell = 1$ *and the nodes of* $Next\ell$,Val_ℓ *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 node in* I_ℓ *experiences* 0 *bias from the internal nodes of* C_ℓ.

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

Proof.

1. Since $Control\ell = 1$ and since we assumed no node experiences any external bias, by lemma 13 we have that all y, z have their natural values and hence all gates compute correctly, by lemma 10. Therefore, $Next\ell = Real - Next(I_\ell)$ and $Val_\ell = Real - Val(I_\ell)$.

2. Since $Control\ell = 0$, by lemma 13 all y, z have their unnatural values. Since all NOR gadgets have unnatural control nodes we have that their inputs are indifferent with respect to the gadgets. Hence, the claim that they are unbiased follows.

3. The $Control\ell$ node is connected to a node $NotControl\ell$, 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.

□

B.3 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* node towards 0, which biases the internal control nodes 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 the gadget changes, in order to bias the control nodes 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 nodes from the *CopyA* Gadgets and I_B and simply checks whether they are equal. Due to Lemma 2, in equilibrium the T_A nodes 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 nodes 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 nodes to the input that are dominated by η_A nodes. The input nodes I_B are dominated by either the nodes 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 node between I_B and the gadget to ensure that the two input nodes will have equal weight. A detailed analysis is provided in the proof of Lemma 1.

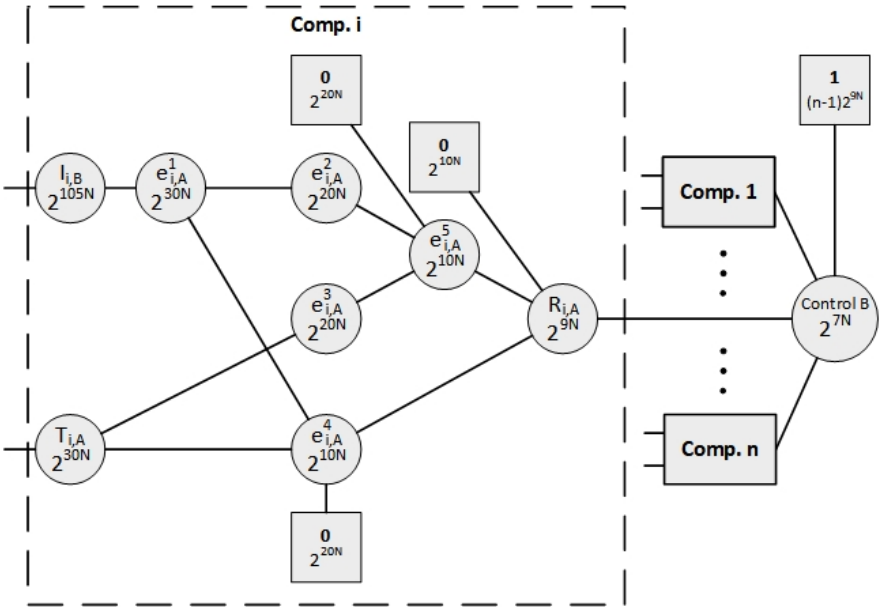

Figure 14: This gadget performs equality check for the bits $I_{i,B}$ and $T_{i,A}$. If they are equal, $R_{i,A} = 0$ in equilibrium. We have n such gadgets for each of the two circuits. The n gadgets are connected to produce the final output, which is ControlB.

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 node 1 will bias *ControlB* to take value 0.

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

Lemma 1. *Let an equilibrium of the overall* LocalNodeMaxCut *instance of 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 node $e^1_{i,A}$ 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, in equilibrium it is always the case that $e^1_{i,A} = \neg I_{i,B}$. Moreover,

nodes $e^2_{i,A}$ 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 node $e^5_{i,A}$. 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 nodes have equal weights, node $e^4_{i,A}$ experiences 0 total bias from them and is biased by constant node 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 nodes. Is also has bias 2^{20N} by constant 0 and 2^{9N} from $R_{i,A}$. Hence, $e^5_{i,A} = 1$. Nodes $e^4_{i,A}$ and $e_{5.i}$ bias node $R_{i,A}$ towards 0 with weight $2 * 2^{20N}$, 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 $2 * 2^{30N}$ 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 $2 * 2^{20N}$ towards 1 by nodes $e^5_{i,A}$ 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 $R_{i,A}$ nodes 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 $R_{i,A}$ nodes. 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 $n * 2^{9N}$ towards 1, which is greater than the weight of constant node 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, node *ControlB* is biased by nodes $R_{j,A}$ and constant 1 towards 0 with weight at least $(n-1) * 2^{9N} + 2^{9N} = n * 2^{9N}$, 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.

□

B.4 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* node 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 nodes of Circuit A that calculate the best flip neighbors. This is because when node *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 nodes to be indifferent to all of their neighbors when the time comes to change their value. As a result, if we connect the output nodes 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 node *Flag* is 1, so that they can change properly. A similar functionality should be implemented when node *Flag* is 0.

In this Section 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 node *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 nodes 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 node 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 nodes towards unnatural values. When this happens, the inputs I_B can change to the appropriate values.

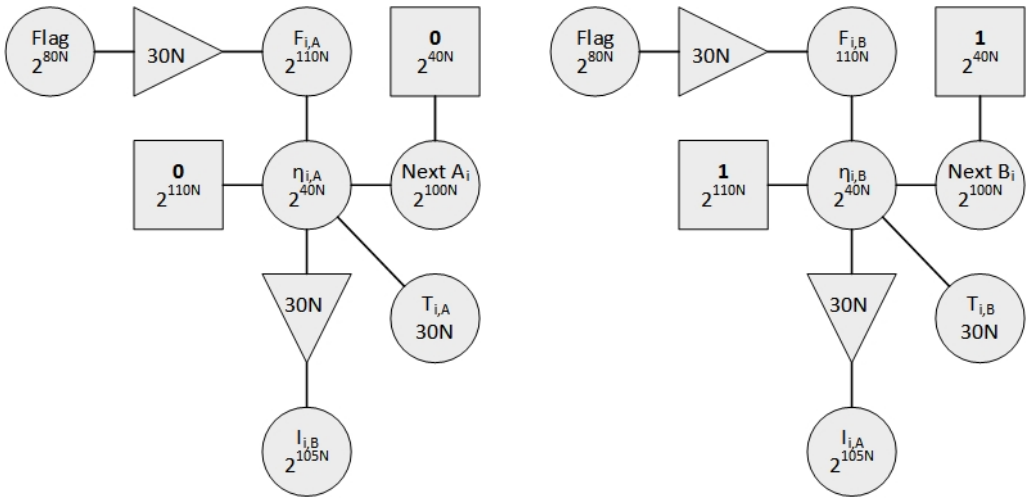

Figure 15: The gadgets that copy the values from one circuit to the other

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. Nodes $F_{i,A}$ has a very large weight in order to dominate the behavior of $\eta_{i,A}$. However, we do not want this node 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 nodes 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 nodes 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 2, which makes precise the already stated claims about the function of the *Copy* Gadgets.

Lemma 2. *At any equilibrium point of the* LocalNodeMaxCut *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, node $F_{i,B}$ 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 equilibrium. Now node $\eta_{i,B}$ experiences zero total bias from nodes $F_{i,B}$ 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 equilibrium will be determined by $NextB_i$. Specifically, $\eta_{i,B} = \neg NextB_i$ at equilibrium. Now, node $T_{i,B}$ 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 equilibrium.

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, node I_i experiences bias slightly less than 2^{10N} from node $\eta_{i,B}$. Since $ControlA = 0$, by Lemma 3, 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 3. *At any equilibrium point of the* LocalNodeMaxCut *instance of Figure 3:*

- *If* Flag $= 1$ *then any node in* NextA *experience* 0 *bias with respect to the* CopyX“ *gadget.*
- *If* Flag $= 0$ *then then any node in* NextB *experience* 0 *bias with respect to the* CopyB *gadget.*

Proof. We notice that due to leveraging, node $F_{i,A}$ of gadget *CopyA* experiences bias slightly less than 2^{50N} from node $Flag = 1$. 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, node $\eta_{i,X“}$ experiences total bias $2 * 2^{110N}$ from nodes $F_{i,A}$ 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. Node $NextA_i$ is connected to nodes $\eta_{i,A}$ and constant 0 of gadget $CopyA_i$. They have the same weight and opposite values at equilibrium. 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. □

B.5 *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 nodes of the losing circuit (i.e the circuit with the lower value) are indifferent with respect to the *Comparator* gadget, so that Lemma 3 can be applied.

In particular, the output nodes that correspond to the bits of the value, presented in section B.2, with weights $2^{90N} * 2^{N+i}$ are each connected as below.

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

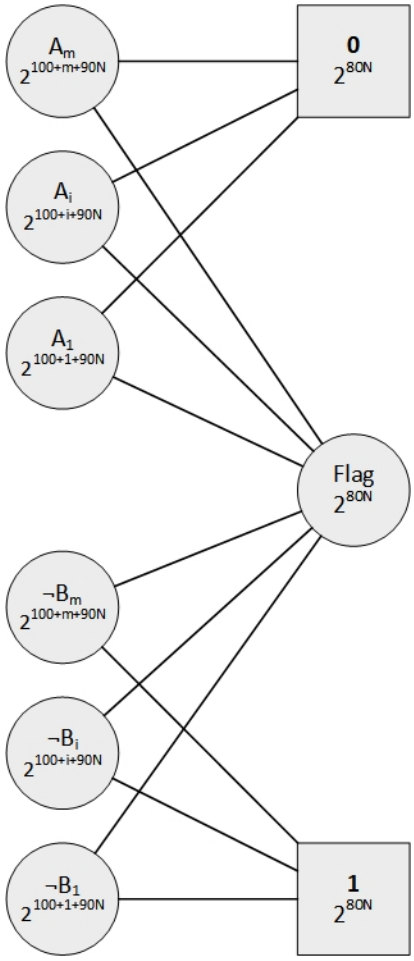

Figure 16: Nodes of the Comparator *gadget. Note that Circuit X" is meant to output the complement of its true output.*

To see why the value of node *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* node 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 nodes to the node *Flag*: the control nodes $y^3_{m+1,A}$ 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 $w_{y^3_{i,A}} = w_{z^3_{i,B}} = 2^{100N} * (2^{N+m+1} - 50)$) (see Figure 12), as well as the control nodes $y^3_{i,A}, z^3_{i,B}, \forall i \leq m$ for each NOR gadget that corresponds to an output bit of the value (which have weight $w_{y^3_{i,A}} = w_{z^3_{i,B}} = 2^{90N} * (2^{N+i} - 50)$). The nodes $y^3_{m+1,A}$ and $z^3_{m+1,B}$ are used to check whether the next best neighbor has been correctly computed. If it isn't, these nodes 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 nodes 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 14. All these nodes are connected in such a way that a control node with unnatural value, biases *Flag* towards fixing that circuit.

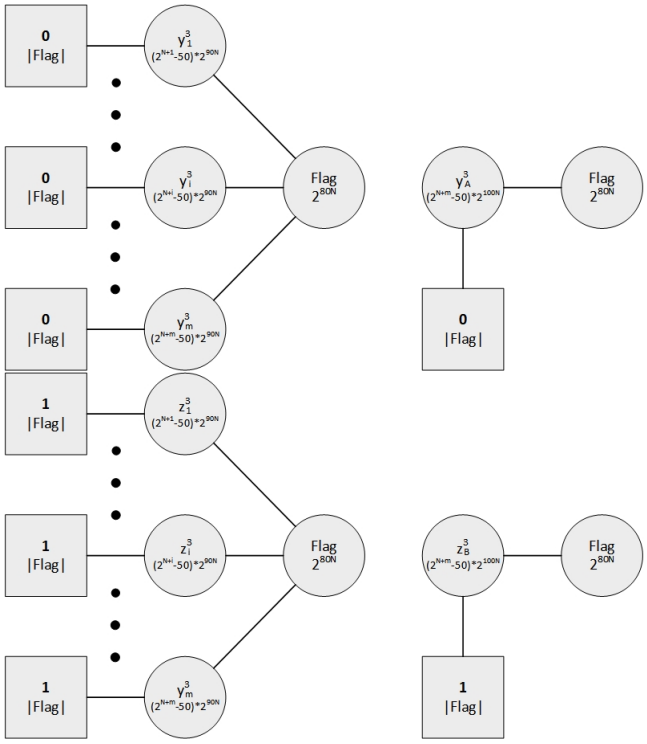

Figure 17: Connection between the control nodes *and the* Flag *node.*

We prove the following properties:

Lemma 4. *Let an equilibrium of the instance of* LocalNodeMaxCut *of Figure 3:*

- *If* Flag $= 1$ *then all nodes of* C_A *experience* 0 *bias to the* Comparator *gadget.*
- *If* Flag $= 0$ *then all nodes of* C_B *experience* 0 *bias to the* Comparator *gadget.*

Proof. Suppose *Flag* $= 1$. Then the only nodes of C_A connected to the *Comparator* gadget are the value output bits and certain control nodes, in such a way that they are connected to either *Flag* or

a constant node 0 of weight equal to *Flag*. In all cases, both biases cancel each other out and the nodes of Circuit A are indifferent. Suppose $Flag = 0$. Then the only nodes of C_B connected to *Flag* are also connected with a constant 1 node. Similarly to the first case, all nodes 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 node is exponentially smaller that the rest of the bits. Hence, it cannot dominate the outcome of the comparison. This means that sometimes we might be in equilibrium where some output nodes 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 5 we know that at least one of the circuits computes correctly in equilibrium.

Lemma 14. *At any equilibrium:*

Suppose 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* $Real - value(I_A) \leq Real - value(I_B)$

Suppose 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* $Real - value(I_B) \leq Real - value(I_A)$

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 node corresponding to B_k is the opposite. We also denote $z^2_{k,B}$ the control node 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 $2 * 2^{90N} * (50)$ 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 $2 * 2^{90N} * (2^{N+k})$ 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 $2 * 2^{90N} * (2^{N+k} - 50)$ 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, $Real - value(I_A) \leq Real - value(I_B)$.

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 $kReal(A_k) \leq Real(B_k)$. This means that $Real - value(I_A) < Real - value(I_B)$, 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 $2 * 2^{90N} * (2^{N+k} - 50)$ towards $Flag = 0$ from this bit. Furthermore, it experiences bias at most $2*2^{90N}*(50)$ towards $Flag = 1$ from each bit higher that k. Each bit i lower than k causes bias at most $2*2^{90N}*(2^{N+i})$ each towards $Flag = 1$. In total, if we have m bits, we have at most $(m-k)*2*2^{90N}*(50)+\sum_{i<k} 2*2^{90N}*(2^{N+i}) \leq (m)*2*2^{90N}*(50)+2*2^{90N}*(2^{N+k}-2^N)$ towards $Flag = 1$ and at least $2 * 2^{90N} * (2^{N+k} - 50)$ 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 an equilibrium with $Flag = 1$.

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

Lemma 6. *At any equilibrium of the* NodeMaxCut *instance of Figure 3:*

- *If* Flag $= 1$ *then* NextB $=$ Real-Next(I_B)
- *If* Flag $= 0$ *then* NextA $=$ Real-Next(I_A)

Proof. Assume an equilibrium with $Flag = 1$ and $NextB \neq Real - Next(I_B)$. By Lemma 10 we have that Circuit B is computing incorrectly and hence the control node $z^3_{m+1,B}$ (i.e. the last gate before the value bits) has its unnatural value, which is $z^3_{m+1,B} = 1$.

Assume that, $ControlA = 0$. Then by Lemma 2 we have that $I_A = T_B$, which by Lemma 1 we have $ControlA = 1$, a contradiction. Hence, $ControlA = 1$.

Therefore, since have that $ControlA = 1$ and that $NextA = Real - Next(I_A)$, which by Lemma 13, implies that the corresponding node $y^3_{m+1,A}$ has its natural value $y^3_{m+1,A} = 1$.

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

Similarly, we can prove that if $Flag = 0$ then $NextA = Real - Next(I_A)$ □

Lemma 7. *At any equilibrium of the* LocalNodeMaxCut *instance of Figure 3:*

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

$$Real - Val(I_A) \leq Real - Val(I_B)$$

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

$$Real - Val(I_B) \leq Real - Val(I_A)$$

Proof. Assume that, $ControlA = 0$. Then by Lemma 2 we have that $I_A = T_B$, which by Lemma 1 we have $ControlA = 1$, a contradiction. Hence, $ControlA = 1$.

Since $Flag = 1$ and we have that $ControlA = 1$, by Lemma 13 all control nodes of C_A have their natural values. Furthermore, since by the proof of Lemma 6 we know that all control nodes of weight 2^100N have their natural values, we can apply Lemma 14. Therefore, $Real - Val(I_A) \leq Real - Val(I_B)$

The proof for $Flag = 0$ is identical.

□