Node-Max-Cut and the Complexity of Equilibrium in Linear Weighted Congestion Games

Dimitris Fotakis
National Technical University of Athens, Greece
fotakis@cs.ntua.gr

Vardis Kandiros
Massachusetts Institute of Technology, Cambridge, MA, USA
kandiros@mit.edu

Thanasis Lianeas
National Technical University of Athens, Greece
lianeas@corelab.ntua.gr

Nikos Mouzakis
National Technical University of Athens, Greece
nmouzakis@corelab.ntua.gr

Panagiotis Patsilinakos
National Technical University of Athens, Greece
patsilinak@corelab.ntua.gr

Stratis Skoulakis
Singapore University of Technology and Design, Singapore
efstratios@sutd.edu.sg

Abstract

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

2012 ACM Subject Classification Theory of computation → Exact and approximate computation of equilibria

Keywords and phrases PLS-completeness, Local-Max-Cut, Weighted Congestion Games, Equilibrium Computation

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.50

Category Track A: Algorithms, Complexity and Games

Related Version The full version of the paper is available at `https://arxiv.org/abs/1911.08704`.

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 50; pp. 50:1–50:19

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Funding This work was supported by the Hellenic Foundation for Research and Innovation (H.F.R.I.) under the "First Call for H.F.R.I. Research Projects to support Faculty members and Researchers and the procurement of high-cost research equipment grant", project: BALSAM (project id: 1424). *Stratis Skoulakis*: Supported by NRF 2018, Fellowship NRF-NRFF2018-07. Part of this work was carried out while the author was a PhD student at the National Technical University of Athens.

1 Introduction

Motivated by the remarkable success of local search in combinatorial optimization, Johnson et al. introduced [25] the complexity class Polynomial Local Search (PLS), consisting of local search problems with polynomially verifiable local optimality. PLS includes many natural complete problems (see e.g., [28, App. C]), with Circuit-Flip [25] and Max-Cut [33] among the best known ones, and lays the foundation for a principled study of the complexity of local optima computation. In the last 15 years, a significant volume of research on PLS-completeness was motivated by the problem of computing a pure Nash equilibrium of potential games (see e.g., [1, 34, 20] and the references therein), where any improving deviation by a single player decreases a potential function and its local optima correspond to pure Nash equilibria [29].

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

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

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

On the other hand, the complexity of equilibrium computation for weighted CGs is not well understood. All the hardness results above carry over to weighted CGs, since they generalize standard CGs (where the players have unit weight). But on the positive side, we only know how to efficiently compute a PNE for weighted CGs on parallel links with general latencies [16] and for weighted CGs on parallel links with identity latency functions and asymmetric strategies [19]. Despite the significant interest in (exact or approximate) equilibrium computation for CGs (see e.g., [4, 5, 6, 27] and the references therein), we do not understand how (and to which extent) the complexity of equilibrium computation is affected by player weights. This is especially true for weighted CGs with linear latencies, which admit a potential function and their equilibrium computation is in PLS.

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

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

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

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

A major part of our technical effort concerns reducing Circuit-Flip to Node-Max-Cut, thus showing that computing a local optimum of Node-Max-Cut is PLS-complete (Section 5). Since Node-Max-Cut is a very restricted special case of Max-Cut, we have to start from a PLS-complete problem lying before Local-Max-Cut on the "reduction paths" of PLS. The reduction is technically involved, due to the very restricted nature of the problem. In Node-Max-Cut, every vertex contributes to the cut value of its neighbors with the same weight, and differentiation comes only as a result of the different total weight in the neighborhood of each vertex. To deal with this restriction, we combine some new carefully constructed gadgets with the gadgets used by Schäffer and Yannakakis [33], Elsässer and Tscheuschner [9]. and Gairing and Savani [20]. In general, as a very restricted special case of Max-Cut, Node-Max-Cut is a natural and convenient starting point for future PLS reductions, especially when one wants to show hardness of equilibrium computation for restricted classes of games that admit weighted potential functions (e.g., as that in [17]). So, our results may serve as a first step towards a better understanding of the complexity of (exact or approximate) equilibrium computation for weighted potential games.

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

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

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

Johnson et al. [25] introduced the complexity class PLS and proved that Circuit-Flip is PLS-complete. Subsequently, Schäffer and Yannakakis [33] proved that Max-Cut is PLS-complete. From a technical viewpoint, our work is close to previous work by Elsässer and

Tscheuschner [9] and Gairing and Savani [20], where they show that Local-Max-Cut in graphs of maximum degree five [9] and computing a PNE for hedonic games [20] are PLS-complete, and by Ackermann et al. [1], where they reduce Local-Max-Cut to computing a PNE in network congestion games.

2 Basic Definitions and Notation

Polynomial-Time Local Search (PLS). A *polynomial-time local search* (PLS) problem L [25, Sec. 2] is specified by a (polynomially recognizable) set of instances I_L, a set $S_L(x)$ of feasible solutions for each instance $x \in I_L$, with $|s| = O(\text{poly}(|x|)$ for every solution $s \in S_L(x)$, an objective function $f_L(s,x)$ that maps each solution $s \in S_L(x)$ to its value in instance x, and a *neighborhood* $N_L(s,x) \subseteq S_L(x)$ of feasible solutions for each $s \in S_L(x)$. Moreover, there are three polynomial-time algorithms that for any given instance $x \in I_L$: (i) the first generates an initial solution $s_0 \in S_L(x)$; (ii) the second determines whether a given s is a feasible solution and (if $s \in S_L(x)$) computes its objective value $f_L(s,x)$; and (iii) the third returns either that s is *locally optimal* or a feasible solution $s' \in N_L(s,x)$ with better objective value than s. If L is a maximization (resp. minimization) problem, a solution s is locally optimal if for all $s' \in N_L(s,x)$, $f_L(s,x) \geq f_L(s',x)$ (resp. $f_L(s,x) \leq f_L(s',x)$). If s is not locally optimal, the third algorithm returns a solution $s' \in N_L(s,x)$ with $f(s,x) < f(s',x)$ (resp. $f(s,x) > f(s',x)$). The complexity class PLS consists of all polynomial-time local search problems. By abusing the terminology, we always refer to polynomial-time local search problem simply as local search problems.

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

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

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

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

In this work, we study Max-Cut and Node-Max-Cut as local search problems under the flip neighborhood. Then, they are referred to as Local-Max-Cut and Local-Node-Max-Cut. The neighborhood $N(S)$ of a cut $(S, V \setminus S)$ consists of all cuts $(S', V \setminus S')$ where S and

S' differ by a single vertex. Namely, the cut S' is obtained from S by moving a vertex from one side of the cut to the other. A cut S is locally optimal if for all $S' \in N(S)$, $w(S) \geq w(S')$. In Local-Max-Cut (resp. Local-Node-Max-Cut), given an edge-weighted (resp. vertex-weighted) graph, the goal is to compute a locally optimal cut. Clearly, both Max-Cut and Node-Max-Cut belong to PLS. In the following, we abuse the terminology and refer to Local-Max-Cut and Local-Node-Max-Cut as Max-Cut and Node-Max-Cut, for brevity, unless we need to distinguish between the optimization and the local search problem.

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

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

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

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

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

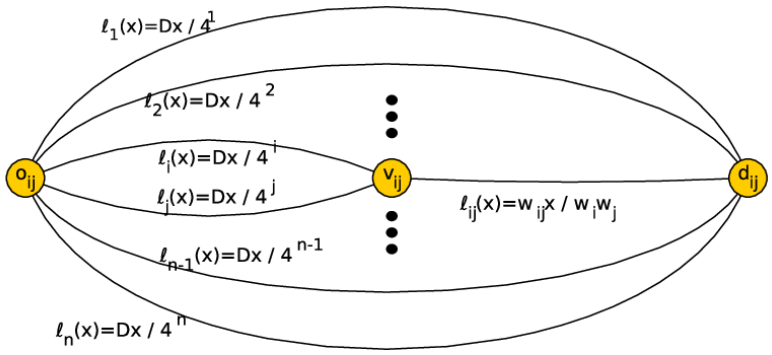

Figure 1 The series-parallel network F_{ij} that corresponds to edge $\{i,j\} \in A$.

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

3 Hardness of Computing Equilibria in Weighted Congestion Games

We next show that computing a PNE in weighted congestion games with linear latencies is PLS-complete either for single-commodity series-parallel networks or for multi-commodity networks with identity latency functions.

3.1 Weighted Congestion Games on Series-Parallel Networks

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

Proof sketch. Membership in PLS follows from the potential function argument of [17]. To show hardness, we present a reduction from Local-Max-Cut.

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

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

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

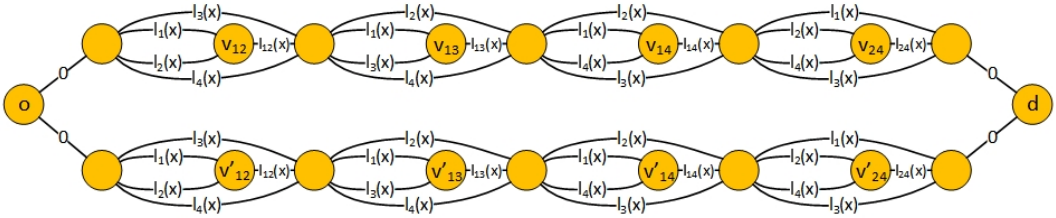

Figure 2 An example of the network G constructed in the proof of Theorem 1 for graph $H(V, A)$, with $V = \{1, 2, 3, 4\}$ and $A = \{\{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 4\}\}$. G is a parallel composition of two parts, each consisting of the smaller networks F_{12}, F_{13}, F_{14} and F_{24} (see also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

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

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

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

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

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

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

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

3.2 Weighted Congestion Games with Identity Latency Functions

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

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

Proof sketch. We use a reduction from Local-Node-Max-Cut, which as we show in Theorem 4, is PLS-complete. Our construction draws ideas from [1].

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

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

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

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

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

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

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

4 Computing Approximate Equilibria for Node-Max-Cut

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

Given a vertex-weighted graph $G(V, E)$ with n vertices and m edges, our algorithm, called BridgeGaps, computes a $(1+\varepsilon)^3$-approximate equilibrium for a Node-Max-Cut, for any $\varepsilon > 0$, in $(m/\varepsilon)(n/\varepsilon)^{O(D_\varepsilon)}$ time, where D_ε is the number of different vertex weights in G, when the weights are rounded down to powers of $1+\varepsilon$. We next sketch the algorithm and the proof of Theorem 3.

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

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

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

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

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

5 PLS-Completeness of Node-Max-Cut

In this section we sketch the proof of Theorem 4 and highlight the main differences of our reduction from known PLS reductions to Max-Cut [9, 20, 33].

▶ **Theorem 4.** *Local-Node-Max-Cut is PLS-complete.*

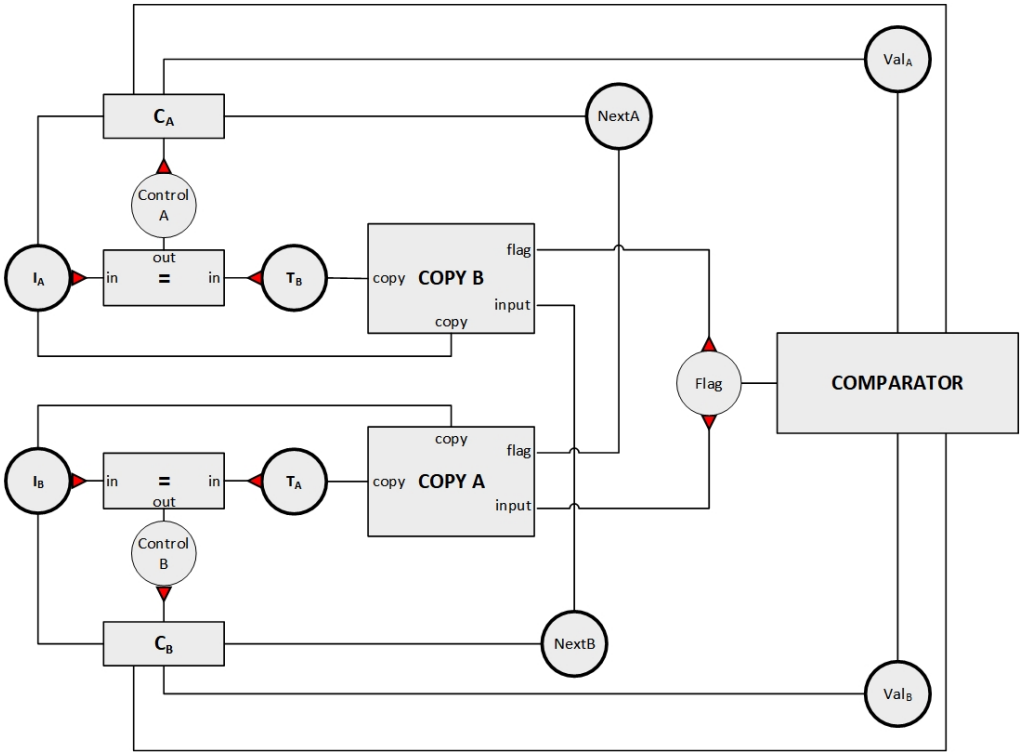

Figure 3 The general structure of the Node-Max-Cut instance constructed in the proof of Theorem 4. Rectangles denote the main gadgets, circles denote vertices that participate in multiple gadgets, and circles with bold border denote groups of n such vertices. The small red triangles are used to indicate the "information flow" through the instance.

As discussed in Section 2, the local search version of Node-Max-Cut is in PLS. To establish the PLS-hardness of Node-Max-Cut, we present a reduction from Circuit-Flip. We start with an outline of our construction and a brief discussion of its technical novelty. Then, in Section 5.1, we discuss our gadget constructions in more detail and present the key technical steps towards the proof of Theorem 4.

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

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

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

Given a circuit C, we construct two *Circuit Computing* gadgets C_ℓ (ℓ always stands for either A or B), which are instances of Node-Max-Cut that simulate circuit C in the following sense: Each C_ℓ has a set I_ℓ of n "input" vertices, whose (cut) values correspond to the input string of circuit C, and a set Val_ℓ of m "output" vertices, whose values correspond to the output string of C on input I_ℓ. There is also a set $Next\ell$ of n vertices whose values correspond to a n-bit string in the neighborhood of I_ℓ of circuit value larger than that of I_ℓ (if the values of Nextℓ coincide with the values of I_ℓ, I_ℓ is locally optimal). Lastly, there is a (set of) vertices $Control_\ell$ that control the behavior of the gadget.

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

Our Circuit Computing gadgets C_A and C_B are based on the gadgets of Schäffer and Yannakakis [33] (see also Figure 4 for an abstract description of them). Their detailed construction is described in the next section and their properties are summarized in Theorem 5.

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

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

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

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

Our Circuit Computing gadgets are based on similar gadgets used e.g., in [33]. A key difference is that our Circuit Computing gadgets are designed to operate with w_{Control} (i.e., weight of the Control_ℓ vertex) arbitrarily smaller than the weight of any other vertex in the Circuit Computing gadget. Hence, the Control vertex can achieve a very small bias wrt. the Circuit Computing gadget (see Theorem 5, Case 3), which in turn, allows us to carefully balance the weights in the Equality gadget. The latter weights should be large enough, so as to control the write and compute modes of C_ℓ, and at the same time, small enough, so as to avoid interference with the values of the input vertices I_ℓ. The second important reason for setting w_{Control} sufficiently small is that we need the Control vertex to follow the "output" of the Equality gadget, and not the other way around.

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

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

Another important difference concerns the implementation of the *red marks* (denoting the information flow) between Flag and the Copy gadgets, in Figure 3. They indicate that the value of Flag should agree with the output of the Comparator gadget. This part of the information flow is difficult to implement, because the Comparator gadget and the Copy gadgets receive input from NextA, NextB, Val_A and Val_B, where the vertex weights are comparable to the weights of the output vertices in the Circuit Computing gadgets. As a result, the weights of the vertices inside the Comparator gadget cannot become sufficiently larger than the weights of the vertices inside the Copy gadgets. [33, 20, 9] connect Flag to the Copy gadgets with edges of sufficiently small weight, which makes the bias of Flag from the Copy gadgets negligible compared against its bias from Comparator. Again, the *Leverage* gadget comes to our rescue. We use it internally in the Copy gadgets, in order to decrease the influence of the vertices inside the Copy gadgets to Flag. As a result, Flag's bias from the Copy gadgets becomes much smaller than its bias from Comparator (see Lemma 10).

Another key technical difference concerns the design of the Comparator gadget. As stated in Lemma 10 and explained below, the Comparator gadget manages to compute the result of the comparison $\text{Real-Val}(I_A) \leq \text{Real-Val}(I_B)$, even if some input vertices may have incosistent values. In previous work [33, 20, 9], the Comparator guarantees correctness of the values in both NextB and Val_B using appropriately chosen edge weights. With the correctness of the input values guaranteed, the comparison is not hard to implement. It is not clear if this decoupled architecture of the Comparator gadget can be implemented in Node-Max-Cut, due to the special structure of edge weights. Instead, we implement a new *all at once* Comparator, which ensures correctness to a subset of its input values enough to perform the comparison correctly (see also the discussion after Lemma 11 on this point).

5.1 A Technical Overview of the Proof of Theorem 4

In this section, we outline the gadget constructions and the key technical claims used in the proof of Theorem 4. As shown in Figure 3, our Node-Max-Cut instance consists of the following gadgets:

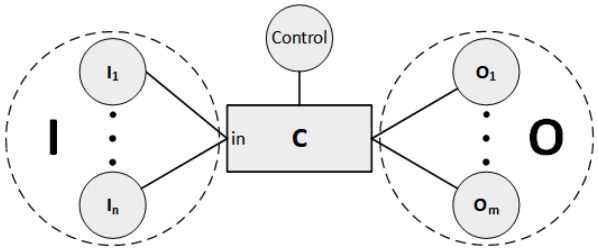

Figure 4 The Circuit Computing gadgets. The dashed circles, labeled I and O, represent all input and output vertices, respectively.

1. Two Circuit Computing gadgets that calculate the values and next neighbors of solutions.
2. Two Copy gadgets that transfer the solution of one circuit to the other, and vice versa.
3. Two Equality gadgets that determine the (write or compute) mode in which the Circuit Computing gadgets operate.
4. A Comparator gadget.

Unlike previous similar reductions in the literature (see e.g.,[33, 20, 9]), we introduce each gadget separately rather than through types of diminishing weight. This is because in Node-Max-Cut, weights are associated with vertices and each vertex may be part of multiple gadgets.

The Circuit Computing Gadgets. The Circuit Computing gadgets C_A and C_B are the basic primitives of our reduction. They are based on the gadgets introduced by Schäffer and Yannakakis [33] to establish PLS-completeness of Max-Cut. This type of Circuit Computing gadgets can be constructed so as to simulate any Boolean circuit C.

The most important vertices are those corresponding to the input and the output of the simulated circuit C. Another important vertex is Control, which allows the gadget to switch between the write and the compute mode of operation. Figure 4 is an abstract depiction of the Circuit Computing gadgets. Theorem 5 describes the local optimum behavior of the input vertices I_ℓ and output vertices Nextℓ, Val$_\ell$ of the Circuit Computing gadgets C_ℓ.

▶ **Theorem 5.** *At any local optimum of the Node-Max-Cut instance in Figure 3, the following hold:*

1. *If Control$_\ell = 1$ and the vertices in Nextℓ, Val$_\ell$ experience 0 bias from any other gadget beyond C_ℓ, then Nextℓ = Real-Next(I_ℓ) and Val$_\ell$ = Real-Val(I_ℓ).*
2. *If Control$_\ell = 0$, then each vertex in I_ℓ experiences 0 bias from the internal vertices of C_ℓ.*
3. *Control$_\ell$ experiences w_{Control_ℓ} bias from the internal vertices of C_ℓ.*

Case 1 of Theorem 5 describes the *compute mode* of the Circuit Computing gadgets. At any local optimum with Control$_A = 1$ and with the output vertices of C_A being indifferent wrt. other gadgets, C_A computes its output correctly. Case 2 of Theorem 5 describes the *write mode*. If at a local optimum Control$_A = 0$, the vertices in I_A have 0 bias from the C_A gadget. As a result, their value is determined by the biases of the CopyB gadget and the Equality gadget. Case 3 of Theorem 5 bounds the bias that the Equality gadget poses to the Control vertices, so as to make the computing gadget flip from one mode to the other.

The Copy and Equality Gadgets. The Copy and Equality gadgets are responsible for transferring the output of one Circuit Computing gadget to the other. More specifically, the Equality gadget takes as input the nodes I_A, I_B, T_A, T_B and assures that the Control vertices

always have the correct values. Recall that as shown in Theorem 5, Case 3, the bias that the Equality gadget exerts is enough to dominate the values of the Control nodes. Using that, we can prove the following lemma:

▶ **Lemma 6.** *For any local optimum of the* Node-Max-Cut *instance in Figure 3, we have that* $Control_A = (I_A = T_B)$ *and* $Control_B = (I_B = T_A)$.

The Copy gadget transfers the values of the NextB and NextA to I_A and I_B, respectively, when each gadget is in its *write mode*. This behavior is summarized by the following:

▶ **Lemma 7.** *At any local optimum point of the* Node-Max-Cut *instance in Figure 3, the following hold:*

1. *If* $Flag = 1$, *i.e., NextB* writes *on* I_A, *then (i)* $T_B = NextB$, *and (ii) if* $Control_A = 0$, *then* $I_A = T_B = NextB$.
2. *If* $Flag = 0$ *i.e., NextA* writes *on* I_B, *then (i)* $T_A = NextA$, *and (ii) if* $Control_B = 0$, *then* $I_B = T_A = NextA$.

Additionally, the Copy gadget has the property of leaving the NextB and NextA vertices unbiased, when Flag has certain values. This is necessary so that whenever a Circuit Computing gadget is about to compute, the second condition of Theorem 5, Case 1, can apply and allow the computation to take place.

▶ **Lemma 8.** *At any local optimum of the* Node-Max-Cut *instance in Figure 3:*

- *If* $Flag = 1$, *then any vertex in NextA experiences* 0 *bias from the CopyA gadget.*
- *If* $Flag = 0$, *then any vertex in NextB experiences* 0 *bias from the CopyB gadget.*

Having established the above properties of the Copy and Equality gadgets, we can show the following theorem which asserts that at an local optimum, one of the two circuits has taken the other's output as input.

▶ **Theorem 9.** *At any a local optimum of the* Node-Max-Cut *instance in Figure 3, the following hold:*

- *If* $Flag = 1$, *then* $I_A = NextB$.
- *If* $Flag = 0$, *then* $I_B = NextA$.

Proof of Theorem 9. Let a local optimum in which Flag = 1. Let us assume that $I_A \neq$ NextB. Then, by Case 1 of Lemma 7, $T_B =$ NextB. As a result, $I_A \neq T_B$, which implies that $Control_A = 0$, by Lemma 6. Now, by Lemma 7, Case 1.ii, we have that $I_A =$ NextB, which is a contradiction. The same analysis can be applied in case where Flag = 0. ◀

The Comparator Gadget. The last important gadget in our reduction is the Comparator, whose construction is technically involved. We first recall that Nextℓ and Val$_\ell$ denote the actual values that the corresponding vertices have in our construction, while Real-Next(I_ℓ) and Real-Val(I_ℓ) denote the values that these vertices are supposed to have, assuming that the Circuit Computing gadgets operate correctly. We also recall that when Flag = 1 (resp. Flag = 0), the Circuit Computing gadget C_A (resp. C_B) recomputes its output values, based on the (possibly incorrect) outputs of the other Circuit Computing gadget C_B (resp. C_A). The construction of the Comparator gadget and the following lemmas ensure that at any local optimum of the Node-Max-Cut instance in Figure 3, the actual values of the vertices in Nextℓ and Val$_\ell$ are identical to Real-Next(I_ℓ) and Real-Val(I_ℓ).

First, connecting certain internal vertices of the Circuit Computing gadgets with the vertices in NextA and NextB and in the Comparator, we obtain the following pair of lemmas.

▶ **Lemma 10.** *At any local optimum of the* Node-Max-Cut *instance in Figure 3, the following hold:*

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

▶ **Lemma 11.** *At any local optimum of the* Node-Max-Cut *instance in Figure 3:*

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

We should highlight that in the proofs of Lemma 10 and Lemma 11, in the first case where Flag = 1 (the case where Flag = 0 is symmetric), the correctness of values of the output vertices NextB (i.e., that NextB = Real-Next(I_B), in the first case of Lemma 11) is not guaranteed by Theorem 5 (as it happens with the correctness of the values of the output vertices NextA and Val_A in the hypothesis of the first case in Lemma 10), but from the construction of the Comparator gadget. We should also highlight that Lemma 11 does not imply anything about the correctness of the Val_B values, as this cannot be guaranteed in our construction. Hence, (the first case of) Lemma 10 does not assume anything about Val_B (in particular, it does not assume that $\text{Val}_B = \text{Real-Val}(I_B)$, as one might have expected). However, the Comparator gadget manages to output the right outcome of the comparison $\text{Real-Val}(I_A) \leq \text{Real-Val}(I_B)$, even if $\text{Val}_B \neq \text{Real-Val}(I_B)$. The connections from internal vertices of the Circuit Computing gadgets to the vertices in NextA and NextB and in the Comparator, in Figure 3, are crucial towards establishing this property. So, Lemma 10 ensures the robustness of the outcome of the Comparator, even with possibly inconsistent values in the output vertices Val_B. This property of the Comparator gadget is among the key technical steps (and novelties) in our reduction and is indicative of the difficulty of showing that Local-Node-Max-Cut is PLS-hard.

Furthermore, similarly to the Copy gadget, the Comparator gadget leaves the vertices of the Circuit Computing gadgets unbiased wrt. certain values of the Flag vertex.

▶ **Lemma 12.** *At any local optimum of the instance of* Node-Max-Cut *in Figure 3:*

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

Now using Lemma 8 and Lemma 12, we can prove the correctness of the output vertices NextA and Val_A, when Flag = 1, and of the output vertices NextB and Val_B, when Flag = 0.

▶ **Lemma 13.** *At any local optimum of the instance of* Node-Max-Cut *of Figure 3:*

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

Proof. We only consider the case where Flag = 1 (the same analysis applies to the case where Flag = 0). By Theorem 9, I_A = NextB, and by Lemma 7, T_B = NextB. As a result, $I_A = T_B$, and by Lemma 6, $\text{Control}_A = 1$. Then, Lemmas 8 and 12 ensure that the vertices in NextA and Val_A of the Computing Gadget C_A experience 0 bias from all the other gadgets. Therefore, since $\text{Control}_A = 1$, we can apply Theorem 5, Case 1, and conclude that $\text{Val}_A = \text{Real-Val}(I_A)$ and that NextA = Real-Next(I_A). ◀

Concluding the Proof of Theorem 4. The technical lemmas above are summarized by the following key technical claim:

▶ **Theorem 14.** *At any local optimum of the instance of* Node-Max-Cut *of Figure 3:*
- *If* Flag $= 1$*, then (i)* Real-Val$(I_A) \leq$ Real-Val(I_B)*, and (ii)* NextB $=$ Real-Next(I_B)*.*
- *If* Flag $= 0$*, then (i)* Real-Val$(I_B) \leq$ Real-Val(I_A)*, and (ii)* NextA $=$ Real-Next(I_A)*.*

Proof of Theorem 14. We consider a local optimum of the instance in Figure 3 with Flag $= 1$ (the same argument applies when Flag $= 0$). By Lemma 11, NextB $=$ Real-Next(I_B), and thus, (ii) is established. Moreover by Lemma 13, NextA $=$ Real-Next(I_A) and Val$_A =$ Real-Val(I_A). Consequently, by Lemma 10, we conclude Real-Val$(I_A) \leq$ Real-Val(I_B). ◀

With theorems 9 and 14 at hand, the PLS-completeness of Local-Node-Max-Cut follows easily. For the sake of completeness, we show how we put everything together.

Proof of Theorem 4. For a given circuit C of Circuit-Flip, we construct in polynomial time the instance of Node-Max-Cut in Figure 3. We next consider any local optimum of this instance. Without loss of generality, we assume that Flag $= 1$. Then, by Theorem 9 and Theorem 14, $I_A =$ NextB, NextB $=$ Real-Next(I_B) and Real-Val$(I_A) \leq$ Real-Val(I_B). Hence, we obtain that

$$\text{Real-Val}(I_B) \geq \text{Real-Val}(I_A) = \text{Real-Val}(\text{NextB}) = \text{Real-Val}(\text{Real-Next}(I_B))\,.$$

If $I_B \neq$ Real-Next(I_B), then Real-Val$(I_B) >$ Real-Val(Real-Next(I_B)), which is a contradiction. Therefore, $I_B =$ Real-Next(I_B), meaning that the binary string defined by the values of I_B is a locally optimal solution for Circuit-Flip. ◀

6 Conclusions and Future Work

In this work, we showed that equilibrium computation in linear weighted congestion games is PLS-complete either on single-commodity series-parallel networks or on multi-commodity networks with identity latency functions, where computing an equibrium for (unweighted) congestion games is known to be easy. The key step for the latter reduction is to show that local optimum computation for Node-Max-Cut, a natural and significant restriction of Max-Cut, is PLS-complete. The reductions in Section 3 are both *tight* [33], thus preserving the structure of the local search graph. In particular, for the first reduction, we have that (i) there are instances of linear weighted congestion games on single-commodity series-parallel networks such that any best response sequence has exponential length; and (ii) that the problem of computing the equilibrium reached from a given initial state is PSPACE-hard.

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

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

References

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

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

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

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

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

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

7 Xi Chen, Chenghao Guo, Emmanouil-Vasileios Vlatakis-Gkaragkounis, Mihalis Yannakakis, and Xinzhi Zhang. Smoothed complexity of local Max-Cut and binary Max-CSP. In *Proc. of the 52nd Annual ACM Symposium on Theory of Computing (STOC 2020)*, 2020.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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