Computing Stable Outcomes in Hedonic Games

Martin Gairing and Rahul Savani

Department of Computer Science
University of Liverpool
{m.gairing,rahul.savani}@liverpool.ac.uk

Abstract. We study the computational complexity of finding stable outcomes in symmetric additively-separable hedonic games. These coalition formation games are specified by an undirected edge-weighted graph: nodes are players, an outcome of the game is a partition of the nodes into coalitions, and the utility of a node is the sum of incident edge weights in the same coalition. We consider several natural stability requirements defined in the economics literature. For all of them the existence of a stable outcome is guaranteed by a potential function argument, so local improvements will converge to a stable outcome and all these problems are in PLS. The different stability requirements correspond to different local search neighbourhoods. For different neighbourhood structures, our findings comprise positive results in the form of polynomial-time algorithms for finding stable outcomes, and negative (PLS-completeness) results.

1 Introduction

Hedonic games were introduced in the economics literature as a model of coalition formation where each player cares only about those within the same coalition [9]. Such games can be used to model a variety of settings ranging from multi-agent coordination to group formation in social networks. This paper studies the computational complexity of finding stable outcomes in hedonic games. We consider the stability requirements introduced in [5], which includes a detailed discussion of real-life situations in which hedonic models are reasonable.

The literature has focused almost exclusively on the issue of the *existence* of stable outcomes. When computational complexity has been addressed, it has been in the context of deciding whether a stable outcome exists. This has been done under different utility functions and stability requirements [5, 3, 18]. An outcome is called *Nash-stable* if no player prefers to be in a different coalition. This is the most stringent stability requirement we consider: here a deviation depends only on the preferences of the deviating player. Less stringent stability requirements are achieved by restricting feasible deviations: a coalition may try to hold on to an attractive player or block the entry of an unattractive player.

We consider the case of hedonic games with *additively-separable* utilities, as they allow a succinct representation which is suitable for studying computational complexity. In this representation, a game is specified by an *edge-weighted graph*. In general this graph is directed, which allows non-symmetric preferences, but

S. Kontogiannis, E. Koutsoupias, P.G. Spirakis (Eds.): SAGT 2010, LNCS 6386, pp. 174–185, 2010.

then a stable outcome might not exist [5, 3]. In the sequel, a hedonic game is an *undirected* edge-weighted graph, so that preferences are *symmetric*. Every node is a player, and an outcome is a partition of the players into coalitions. For a given outcome, the utility of a player is the sum of the edges weights of the incident edges to nodes in the same coalition. For a symmetric additively-separable hedonic game, a Nash-stable outcome always exists by a simple potential function argument: the potential function is the total happiness of an outcome, i.e., the sum of players' utilities. Nash-deviations improve the potential. We define the problem NashStable as that of computing a Nash-stable outcome for an additively-separable hedonic game.

We also consider a less stringent stability requirement, called *individual stability*. Here the set of all feasible deviations for a given outcome is a subset of Nash deviations: a player can deviate to another coalition only if everyone in this coalition is happy to have her. We also consider an even less stringent stability requirement, called *contractual individual stability*. Here the set of all feasible deviations for a given outcome is a subset of individually-stable deviations: (in addition to the above requirement) a player can deviate only if everyone in the coalition she leaves is happy for her to leave. These stability requirements were introduced in [5]. The same potential function argument shows that individually-stable outcomes and contractually-individually-stable outcomes exist for symmetric additively-separable hedonic games, and indeed every Nash-stable outcome is also individually-stable and contractually-individually-stable. In each case, local improvements will find a stable outcome, and all the problems we consider are in the complexity class PLS (polynomial local search) [12]. Local search dynamics are desirable because they are *distributed*. Are they also efficient for hedonic games? If not, can we find efficient dynamics or centralized algorithms for finding stable outcomes?

Symmetric additively-separable hedonic games are closely related to party affiliation games, which are also specified by an undirected edge-weighted graph. In a party affiliation game each player must choose between one of two "parties"; each player's happiness is the sum of her edges to nodes in the same party; in a stable outcome no player would prefer to be in the other party. The problem PartyAffiliation is to find a stable outcome in such a game. If such an instance has only negative edges then it is equivalent to the problem LocalMaxCut, which is to find a stable outcome of a local max-cut game. In party affiliation games there are at most two coalitions, while in hedonic games any number of coalitions is allowed. Thus, whereas PartyAffiliation for instances with only negative edges is PLS-complete [16], NashStable is trivial in this case, as the outcome where all players are in singleton coalitions is Nash-stable. Both problems are trivial when all edges are non-negative, in which case the grand coalition of all players is Nash-stable. Thus, interesting hedonic games contain both positive and negative edges.

Our Contribution. In this paper, we examine the complexity of computing stable outcomes in symmetric additively-separable hedonic games. We observe that NashStable, i.e., the problem of computing a Nash-stable outcome, is

PLS-complete. Here, we give a simple reduction from PartyAffiliation, which was shown to be PLS-complete in [16]. Our reduction relies on a method to ensure that all stable outcomes use exactly two coalitions (where in general there can be as many coalitions as players). In contrast, the problem CIS of finding a contractually-individually-stable outcome can be solved in polynomial time.

Moreover, we study IS, i.e., the problem of finding an individually-stable outcome. We show that if the outcome is restricted to contain at most two coalitions, an individually-stable outcome can be found in polynomial time. This suggests that a reduction showing PLS-hardness for IS cannot be as simple as for NashStable: one would need to construct hedonic games that allow three or more coalitions. In order to prove a hardness result, we increase the size of the neighbourhood, defining the search problem ISwithSwaps, which is similar to IS, but in addition to one-player deviations, two players can switch coalitions.

We define a restricted version of PartyAffiliation, called OneEnemyPartyAffiliation, in which each player dislikes at most one other player. Our main result is that OneEnemyPartyAffiliation is PLS-complete. This reduction is from CircuitFlip and is rather involved. We reduce OneEnemyPartyAffiliation to ISwithSwaps, which shows it is PLS-complete; we leave the complexity of IS open.

Related Work. Hedonic coalition formation games were first considered by [9]. [11] later surveyed coalition structures in game theory and economics. Based on [9], [5] formulated different stability concepts in the context of hedonic games, which are the basic definitions we use here. The general focus in the game theory community has been on characterizing the conditions for which stable outcomes exist. [6] showed that additively-separable and symmetric preferences guarantee the existence of a Nash-stable partition. They also showed that under certain different conditions on the preferences, the set of Nash-stable partitions can be empty but the set of individually-stable partitions is always non-empty.

[7] surveys algorithmic problems related to stable partitions. [3] showed that for hedonic games represented by an *individually rational list of coalitions*, the complexity of checking whether core-stable, Nash-stable or individual-stable partitions exist is NP-complete, and that every hedonic game has a contractually-individually-stable solution. Recently, [18] showed that for additively-separable hedonic games checking whether a core-stable, strict-core-stable, Nash-stable or individually-stable partition exists is NP-hard. [10] characterize the complexity of problems related to coalitional stability for hedonic games represented by hedonic nets, a succinct, rule-based representation based on marginal contribution nets.

The definition of party affiliation games we use appears in [2]. Recent work on local max cut and party affiliation games has focused on approximation [4, 8]; see also [15]. For surveys on the computational complexity of local search and the complexity class PLS, see [1, 14]. Our main PLS-completeness result (Theorem 2) uses ideas from [19] which in turn builds on [16].

2 Preliminaries

A symmetric additively-separable hedonic game is an undirected edge-weighted graph $G = (V, E, w)$. Every node $i \in V$ is a player. An outcome is a partition p of V into coalitions. Denote by $p(i)$ the coalition to which $i \in V$ belongs under p, and by $E(p(i))$ the set of edges $\{\{i, j\} \in E \mid j \in p(i)\}$. The utility of $i \in V$ under p is the sum of edges to others in the same coalition, $\sum_{e \in E(p(i))} w(e)$. We consider different levels of restrictions for player deviations; see [5].

Definition 1. *Consider an outcome p of a game $G = (V, E, w)$. The outcome is* Nash-stable *if and only if there exists no player i and coalition $c \neq p(i)$, possibly empty, such that*

$$\sum_{e \in E(p(i))} w(e) < \sum_{\{\{i,j\} \in E \ \mid \ j \in c\}} w(\{i, j\}) \ . \tag{1}$$

The outcome is individually-stable *if and only if there exists no player i and coalition $c \neq p(i)$, possibly empty, such that* (1) *holds and*

$$w(\{i, j\}) > 0 \ , \qquad \forall j \in c \ . \tag{2}$$

The outcome is contractually-individually-stable *if and only if there exists no player i and coalition $c \neq p(i)$, possibly empty, such that* (1) *and* (2) *hold and $w(\{i, j\}) < 0$ for all $j \in p(i)$.*

The search problems NashStable, IS, and CIS are to find a stable outcome of a hedonic game for the corresponding definition of stability. For Nash-stability, a player is allowed to deviate based only on her own utility, irrespective of others. Individual stability allows any player to block an unattractive individual from entering her coalition, i.e., a single negative edge to a coalition prevents a player switching to that coalition (although a stable outcome may contain negative edges). Contractual individual stability also allows any player to prevent an attractive player leaving her coalition, i.e., a single positive edge prevents a player leaving a coalition. Recall that all games we consider contain both positive and negative edges, else the problem of finding a stable outcome is easy.

3 Computational Complexity of Finding Stable Outcomes

We start with the least restrictive condition under which player deviations are allowed, i.e., Nash deviations. Here a player is allowed to change her coalition whenever this improves her utility. By a very simple reduction from PartyAffiliation we observe the following:

Observation 1. NashStable *is PLS-complete.*

Proof. Consider an instance of PartyAffiliation which is represented as an edge weighted graph $G = (V, E, w)$. We augment G by introducing two new

players, called *supermodels*. Every player $i \in V$ has an edge of weight $W > \sum_{e \in E} |w_e|$ to each of the supermodels. The two supermodels are connected by an edge of weight $-M$, where $M > |V| \cdot W$. By the choice of M the two supermodels will be in different coalitions in any Nash-stable outcome of the resulting hedonic game. Moreover, by the choice of W, each player will be in a coalition with one of the supermodels. The fact that edges to supermodels have all the same weight directly implies a one-to-one correspondence between the Nash-stable outcomes in the hedonic game and in the party affiliation game. □

Now that we have PLS-completeness under the least restrictive deviation condition, it is natural to ask about stable outcomes under more restrictive conditions. We proceed with the most restrictive version that we consider.

Proposition 1. CIS *can be solved in* $\mathcal{O}(|E|)$ *time. Moreover, local improvements converge in at most* $2|V|$ *steps.*

It is easy to construct stable CIS partitions. The reason for this comes from the very restrictive conditions under which deviations are allowed. We now study deviation conditions which are less restrictive than in CIS but more restrictive than Nash deviations. Recall that in an individually-stable outcome a player is always allowed to leave a coalition but only allowed to enter if no player in the new coalition is connected to her by a negative edge. It is an interesting open problem whether IS is PLS-hard. The following result implies that for a PLS-hardness reduction we need to use at least three coalitions (unless PLS $\subseteq$ P), unlike the reduction for NashStable (Observation 1). Let 2-IS be the problem of computing an individually-stable outcome when at most two coalitions are allowed.

Proposition 2. 2-IS *can be solved in polynomial time.*

Proof. We assume that there is at least one negative edge. Otherwise, the grand coalition is Nash-stable. The algorithm goes as follows:

> Start with any bipartition. Move nodes with incident negative edges so that they have a negative edge to the other coalition. In each of the two coalitions, contract all nodes with negative incident edges into a single node and call the contracted nodes s and t. For any other node the new edge weights to s and t are the sum of the original edge weights. Now (ignoring all edges between s and t) compute a min cut between s and t via a max flow algorithm and assign the nodes accordingly.

After the first stage, all nodes that we are about to contract have a negative edge to the other coalition. So they are not allowed to join the other coalition. This property is preserved by contraction. Afterwards, the flow algorithm operates only on positive edges and computes a global minimum cut between s and t. Thus, the cut also maximizes the total happiness of all non-contracted nodes, so none of these nodes has an incentive to switch coalitions. All performed steps of the algorithm can be done in polynomial time. □

What makes the problem easy in the case of two coalitions? The reason is simple: negative edges block deviations. This leads to an interesting question. What happens when we allow players to *swap* coalitions? Certainly, this increases the PLS-neighbourhood, and (in general) reduces the number of stable outcomes. We define an extended neighbourhood that includes *swaps*.

Definition 2. *In a* swap *two players swap coalitions. A swap is* improving *if at least one of the players becomes strictly better off and neither gets worse off.*

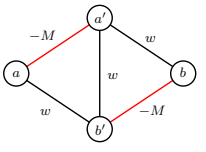
The new neighbourhood is comprised of single-player (IS type) deviations *and* swaps. Observe that in a solution where no player can improve by a single-player deviation, only swaps of two players connected by a negative edge can give rise to local improvements. With this larger neighbourhood we prove the following, which is the main result.

Theorem 2. ISwithSwaps *is PLS-complete.*

We prove this in two parts. We use the fact that in OneEnemyPartyAffiliation every node is incident to at most one negative edge to reduce this problem to ISwithSwaps by replacing negative edges with a simple local gadget (Lemma 1). Then our main result is that OneEnemyPartyAffiliation is PLS-complete (Theorem 3).

Lemma 1. OneEnemyPartyAffiliation *can be reduced in polynomial time to* ISwithSwaps.

Proof. We start with a party affiliation game where every player dislikes at most one other player. We add supermodels to enforce only two coalitions. We replace a negative edge (a, b) of weight $-w$ with the following gadget.



Here M is sufficiently large so that a and a' (as well as b and b') have to be in different coalitions and thus can only swap coalitions. Thus, if $a = b$, then both a and b receive a payoff of $-w$ from the original edge and 0 from the gadget.

On the other hand, if $a \neq b$, then both a and b receive a payoff of 0 from the original edge and w from the gadget. So we shifted the payoffs of a and b by w. Observe that the payoff of a' and b' is always w. So they will never block a swap. Thus, we didn't change the PLS neighbourhood of a and b. □

In order to complete the proof of Theorem 2, we show that OneEnemyPartyAffiliation is PLS-complete. Our proof is by reduction from the well known PLS-complete CircuitFlip problem (cf. [16]).

Definition 3. *An instance of* CircuitFlip *is a boolean circuit with n inputs and n outputs. A feasible solution is an assignment to the inputs and the value of a solution is the output treated as a binary number. The neighbourhood of an assignment consists of all assignments obtained by flipping exactly one input bit. The objective is to maximise the value.*

Theorem 3. OneEnemyPartyAffiliation *is PLS-complete.*

Proof. We reduce from CircuitFlip. Let C be an instance of CircuitFlip with inputs $V_1, \ldots, V_n$, outputs $C_1, \ldots, C_n$, and gates $G_1, \ldots, G_N$. We make the following simplifying assumptions about C: (i) The gates are topologically ordered so that if the output of G_i is an input to G_j then $i > j$. (ii) All gates are NOR gates with fan-in 2. (iii) $G_1, \ldots, G_n$ is the output and $G_{n+1}, \ldots, G_{2n}$ is the (bitwise) negated output of C with G_1 and G_{n+1} being the most significant bits. (iv) $G_{2n+1}, \ldots, G_{3n}$ outputs a (canonical) better neighbouring solution if $V_1, \ldots, V_n$ is not locally optimal.

We use two complete copies of C. One of them represents the current solution while the other ones represents the next (better) solution. Each copy gives rise to a graph. We will start by describing our construction for one of the two copies and later show how they interact. Given C construct a graph G_C as follows:

We have nodes $v_1, \ldots, v_n$ representing the inputs of C, and nodes g_i representing the output of the gates of C. We will also use g_i to refer to the whole gate. For $i \in [n]$, denote by $w_i := g_{2n+i}$ the nodes representing the better neighbouring solution. Recall that $g_1, \ldots, g_n$ represent the output of C while $g_{n+1}, \ldots, g_{2n}$ correspond to the negated output.

In our party affiliation game we use 0 and 1 to denote the two coalitions. We slightly abuse notation by using $u = \kappa$ for $\kappa \in \{0, 1\}$ to denote that node u is in coalition κ. In the construction, we assume the existence of nodes with a fixed coalition. This can be achieved as in the proof of Observation 1 with the help of supermodels. We use 0 and 1 to refer to those constant nodes. In the graphical representation (cf. Figure 1), we represent those constants by square nodes.

We follow the exposition of [16] and [19] and use types to introduce our construction. Nodes may be part of multiple types. In general types are ordered w.r.t. decreasing edge weights. So earlier types are more important. Different types will serve different purposes.

Type 1: Check Gates. For each gate g_i we have a three-part component as depicted in Figure 1(a). The inputs of g_i, denoted $I_1(g_i)$ and $I_2(g_i)$, are either inputs of the circuit or outputs of some gate with larger index. The main purpose of this component is to check if g_i is correct, i.e., $g_i = \neg(I_1(g_i) \wedge I_2(g_i))$, and to set $z_i = 1$ if g_i is incorrect. The α, β, γ, δ and λ nodes are local nodes for the gate. A gate can be in two operational modes, called *gate push regimes.* Type 7 will determine in which of the following push regimes a gate is.

Definition 4 (Gate push regimes). *In the* RESET GATE regime $\alpha_{i,1}$, $\alpha_{i,2}$, $\gamma_{i,1}$ *and* $\gamma_{i,2}$ *get a bias towards* 1 *while* $\lambda_{i,1}, \lambda_{i,2}, \beta_{i,1}, \beta_{i,2}, \beta_{i,3}, \delta_{i,1}, \delta_{i,2}$ *and* $\gamma_{i,3}$ *get a bias towards* 0. *In the* FIX GATE regime *we have opposite biases.*

Type 2: Propagate Flags. In order to propagate incorrect values for the z variables we interconnect them as in Figure 1(b) by using the topological order on the gates. Observe that for any locally optimal solution $z_i = 1$ enforces $z_j = 1$ for all $j < i$. The component is also used to (help to) fix the gates in order and to RESET them in the opposite order. Node z_{N+1} is for technical convenience.

Type 1 and 2 components are the same for both copies. In the following we describe how the copies interact. We denote the two copies of C by C^0 and C^1 and also use superscripts to distinguish between them for nodes of type 1 and 2.

Type 3: Set/Reset Circuits. The component of type 3 interconnects the z-flags from the two circuits C^0, C^1. This component is depicted in Figure 1(c) and has multiple purposes. First, it ensures that in a local optimum d^0 and d^1 are not both 1. Second, at the appropriate time, it triggers to reset the circuit with smaller output. And third, it locks d^0 or d^1 to 1 and resets them back to 0 at the appropriate times.

The z and y nodes can also be in two different operational modes called COMPUTE regime and RESET regime which is determined by Type 6.

Definition 5 (Circuit push regimes). *Let $\kappa \in \{0,1\}$. In the* COMPUTE regime for z^κ *all z_i^κ get a bias to 0 for all $0 \leq i \leq N+1$ and y^κ gets a bias to 1. In the* RESET regime for z^κ *we give opposite biases.*

Type 4: Check Outputs. This component compares the current output of the two circuits and gives incentive to set one of the nodes d^0 or d^1 to 1 for which the output of the corresponding circuit is smaller. For all $i \in [n]$, we have edges $(d^0, g^0_{n+i}), (d^0, g^1_i), (d^1, g^1_{n+i}), (d^1, g^0_i)$ and $(0, g^0_{n+i}), (1, g^1_i), (0, g^1_{n+i}), (1, g^0_i)$ of weight 2^{2n+1-i}. To break symmetry we have edges $(0, d^0), (1, d^1)$ of weight 2^n.

Type 5: Feedback Better Solution. This component is depicted in Figure 1(d). It is used to feedback the improving solution of one circuit to the input of the other circuit. Its operation is explained in Lemma 3.

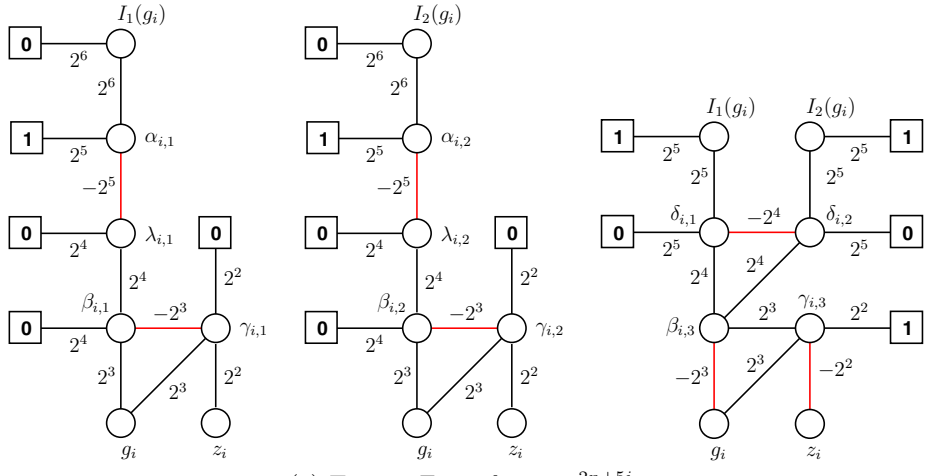
For the remaining types we use the following lemma and definition which are analogous to those in [19, 13].

Lemma 2. *For any polynomial-time computable function $f : \{0,1\}^k \mapsto \{0,1\}^m$ one can construct a graph $G_f(V_f, E_f, w)$ having the following properties: (i) there exist $s_1, \ldots, s_k$, $t_1, \ldots, t_m \in V_f$ with no negative incident edge, (ii) each node in V_f is only incident to at most one negative edge, (iii) $f(s) = t$ in any Nash-stable solution of the party affiliation game defined by G_f.*

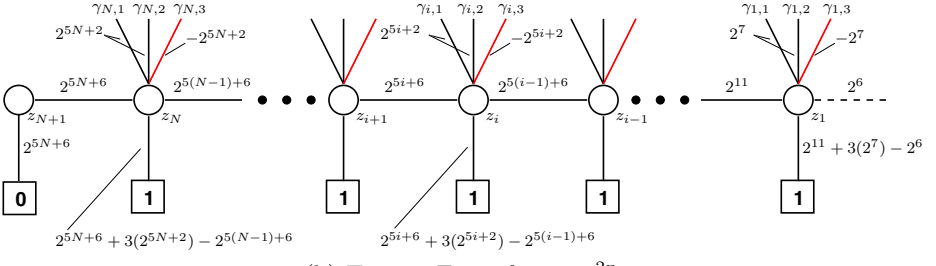
Definition 6. *For a polynomial-time computable function $f : \{0,1\}^k \mapsto \{0,1\}^m$ we say that G_f as constructed in Lemma 2 is a graph that* looks *at $s_1, \ldots, s_k \in V_f$ and* biases *$t_1, \ldots, t_m \in V_f$ according to the function f.*

In the final three types we look at and bias nodes from the lower types already defined. For the final types we do not give explicit edge weights. In order that the "looking" has no side-effects on the operation of the lower types, we scale edge weights in these types such that any edge weight of lower type is larger than the sum of the edge weights of all higher types. More precisely, for $j \in \{5,6,7\}$, the weight of the smallest edge of type j is larger than the sum of weights of all edges of types $(j+1), \ldots, 8$.

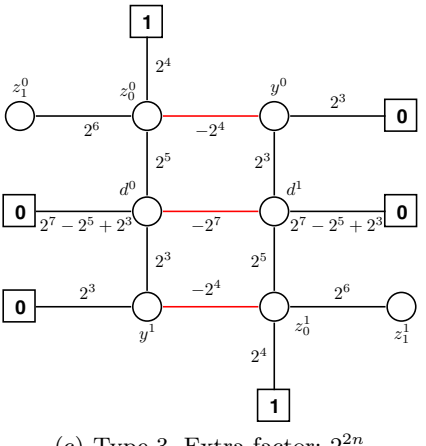
In the following, denote by $C(v)$ the value of circuit C of the CircuitFlip instance on input $v = (v_i)_{i \in [n]}$ and $w(v)$ the better neighbouring solution. Both are functions as in Definition 6.



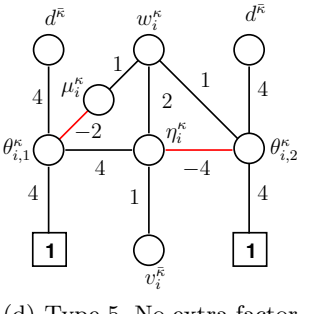
(a) Type 1. Extra factor: 2^{2n+5i}



(b) Type 2. Extra factor: 2^{2n}



(c) Type 3. Extra factor: 2^{2n}



(d) Type 5. No extra factor

Fig. 1. Components of type 1,2,3, and 5. Edge weights have to be multiplied by the factors given above.

Type 6: Change Push Regimes for z. The component of type 6 looks at v^0, v^1, d^0, d^1, η^0 and η^1 (type 5) and biases z_i^0, z_i^1, y^0 and y^1 as follows. z^0 is put in the COMPUTE regime if at least one of the following 3 conditions is fulfilled: (i) $C(v^0) \geq C(v^1)$, (ii) $w(v^1) = v^0$, or (iii) $w(v^1) \neq \eta^1 \wedge d^0 = 1$. Else z^0 is put into the RESET regime. Likewise z^1 is put in the COMPUTE regime if at least one of the following three conditions is fulfilled: (i) $C(v^0) < C(v^1)$, (ii) $w(v^0) = v^1$, or (iii) $w(v^0) \neq \eta^0 \wedge d^1 = 1$. Else z^1 is put into the RESET regime. Note that conditions (i) and (ii) are important for normal computation, while (iii) is needed to overcome bad starting configurations.

Type 7: Change Push Regimes for Gates. For each $i \in [N]$ and $\kappa \in \{0,1\}$, if $z_{i+1}^\kappa = 0$ we put the local variable of g_i^κ in the FIX GATE regime and in the RESET GATE regime otherwise.

Type 8: Fix Incorrect Gate. For each $i \in [N]$ and $\kappa \in \{0,1\}$, the components of type 8 give a tiny offset to g_i^κ for computing correctly. For each gate g_i^κ we look at $\alpha_{i,1}^\kappa, \alpha_{i,2}^\kappa$ and bias g_i^κ to $\neg(\alpha_{i,1}^\kappa \wedge \alpha_{i,2}^\kappa)$.

This completes our construction. We proceed by showing properties of Nash-stable outcomes. Each of the following six lemmas should be read with the implicit clause: "*In every Nash-stable outcome.*"

Lemma 3. *Let $\kappa \in \{0,1\}$, then the following holds for all $i \in [n]$:*
(a) If $d^{\overline{\kappa}} = 0$ then w_i^κ is indifferent w.r.t. edges of type 5.
(b) If $d^{\overline{\kappa}} = 1$ then $\eta_i^\kappa = w_i^\kappa$.

Lemma 4. *If g_i^κ is incorrect then $z_i^\kappa = 1$. If $z_i^\kappa = 1$ then $z_j^\kappa = 1$ for all $0 \leq j \leq i$ and $y^\kappa = 0$.*

Lemma 5. *If $z_{i+1}^\kappa = 1$ then the inputs $I_1(g_i^\kappa)$ and $I_2(g_i^\kappa)$ are indifferent with respect to the type 1 edges of gate g_i^κ.*

Lemma 6. *Suppose $z_{i+1}^\kappa = 0$ and $z_i^\kappa = 1$ for some index $1 \leq i \leq N$.*
(a) If g_i^κ is correct then $\gamma_{i,1}^\kappa = \gamma_{i,2}^\kappa = 0$ and $\gamma_{i,3}^\kappa = 1$.
(b) If g_i^κ is not correct then g_i^κ is indifferent w.r.t. edges of type 1 but w.r.t. the edges only in type 8 deviating would improve her happiness.

Lemma 7. *If $d^\kappa = 1$ and $d^{\overline{\kappa}} = 0$ then for all $1 \leq i \leq 2n$, node g_i^κ is indifferent w.r.t. edges in type 4.*

Lemma 8. *Suppose $d^\kappa = 1$ and $d^{\overline{\kappa}} = 0$.*

(a) If z^κ is in the COMPUTE regime then $z_i^\kappa = 0$ for all $0 \leq i \leq N+1$ and $y^\kappa = 1$.
(b) If z^κ is in the RESET regime then $z_i^\kappa = 1$ for all $0 \leq i \leq N+1$ and $y^\kappa = 0$.

We now continue with the proof of Theorem 3. Suppose we are in a Nash-stable outcome of the party affiliation game. For our proof we assume $C(v^0) \geq$

$C(v^1)$. We will point out the small differences of the other case afterwards. Since $C(v^0) \geq C(v^1)$, z^0 is in the COMPUTE regime, i.e., all z_i^0 are biased to 0 and y^0 is biased to 1 (by type 6). Thus, $z_{N+1}^0 = 0$.

The remainder of the proof splits depending on the coalition of z_1^0 and z_1^1. By Lemma 4 we know that $z_1^\kappa = 0$ implies that all gates in C^κ are correct.

$z_1^0 = 1$: By Lemma 4 we have $z_0^0 = 1$ and $y^0 = 0$. If $d^0 = d^1 = 0$ then d^0 is better off changing to 1 (by inspection of type 3 edges). If $d^0 = 1$ then Lemma 8(a) implies $z_1^0 = 0$, a contradiction. If $d^1 = 1$ and z^1 is in the RESET regime then by Lemma 8(b) and Lemma 5, v^1 is indifferent w.r.t. type 1 edges. Thus $v^1 = \eta^0$. But then either condition (ii) or (iii) for putting z^1 in the COMPUTE regime (cf. type 6) are fulfilled. So z^1 has to be in the COMPUTE regime. Lemma 8(a) then implies $z_1^1 = 0$. But then the neighbourhood of d^1 in type 3 is dominated by 0, a contradiction to $d^1 = 1$.

$z_1^0 = 0$ and $z_1^1 = 1$: By Lemma 4 we have $z_0^1 = 1$ and $y^1 = 0$. Since $C(v^0) \geq C(v^1)$ we know that z^0 is in the COMPUTE regime. So $z_1^0 = 0$ enforces $z_0^0 = 0$ and $y^0 = 0$. By inspection of type 3 edges we have $d^0 = 0$ and thus $d^1 = 1$. First assume that z^1 is in the RESET regime, then $z_i^1 = 1$ for all $0 \leq i \leq N+1$ and Lemma 5 says that the inputs of all gates g_i^1 are indifferent w.r.t. type 1 edges. In particular this holds for $v^1 = (v_i^1)_{i \in [n]}$, so $v^1 = \eta^0$. By Lemma 3(b), $\eta^0 = w^0$. Since $z_1^0 = 0$, C^0 is computing correctly and thus $w^0 = w(v^0)$. Combining this we get $v_1 = w(v^0)$ which contradicts our assumption that z^1 is in the RESET regime. Thus z^1 is in the COMPUTE regime. Since $d^1 = 1$ we can apply Lemma 8(a) to conclude $z_1^1 = 0$, a contradiction.

$z_1^0 = 0$ and $z_1^1 = 0$: By Lemma 4 we have $z_0^0 = z_0^1 = 0$ and $y^0 = y^1 = 1$. Moreover we know that both circuits are computing correctly. If $d^0 = 1$ then $d^1 = 0$ and d^0 is indifferent w.r.t. type 3 edges. Since both circuits are computing correctly and $C(v^0) \geq C(v^1)$, the type 4 edges enforce $d^0 = 0$. But then d^1 is indifferent w.r.t. type 3 edges and the type 4 edges enforce $d^1 = 1$. So, $d^0 = 0$ and $d^1 = 1$. If z^1 is in the RESET regime then Lemma 8(b) gives $z_1^1 = 1$, a contradiction. Thus, z^1 is in the COMPUTE regime. Since $d^1 = 1$ we can apply Lemma 3(b). This and the fact that C^0 is computing correctly implies $\eta^0 = w(v^0)$. So z^1 can only be in the COMPUTE regime if $v^1 = w(v^0)$. Since $C(v^0) \geq C(v^1)$ this implies that $v^0 = v^1$ is a local optimum for the circuit C.

This finishes the proof in case $C(v^0) \geq C(v^1)$. The case $C(v^0) < C(v^1)$ is completely symmetric except here the conclusion $v^0 = v^1$ in the very last sentence contradicts $C(v^0) < C(v^0)$. So this case can't happen in a local optimum.

Note that throughout the construction we made sure that no node is incident to more than one negative edge. This completes the proof of Theorem 3. □

The instance produced by this reduction has the property that no node is indifferent between the two coalitions. This might be useful for other reductions.

Corollary 1. OneEnemyPartyAffiliation *is PLS-complete even if restricted to instances where no player is ever indifferent between the two coalitions.*

Theorem 3 and Lemma 1 together establish that ISwithSwaps is PLS-complete (Theorem 2). Throughout the proof we used only two coalitions. Since 2-IS can be solved in polynomial time (Proposition 2), a PLS-hardness result for IS would require more then two coalitions (unless PLS $\subseteq$ P). We leave the complexity of IS as an interesting open problem.

References

[1] Aarts, E.H.L., Lenstra, J.K.: Local Search in Combinatorial Optimization. Wiley Interscience, Hoboken (1997)

[2] Balcan, M.-F., Blum, A., Mansour, Y.: Improved equilibria via public service advertising. In: SODA, pp. 728–737 (2009)

[3] Ballester, C.: NP-completeness in hedonic games. Games and Economic Behavior 49(1), 1–30 (2004)

[4] Bhalgat, A., Chakraborty, T., Khanna, S.: Approximating pure Nash equilibrium in cut, party affiliation and satisfiability games. In: ACM Conference in Electronic Commerce (EC), pp. 132–146 (2010)

[5] Bogomolnaia, A., Jackson, M.O.: The stability of hedonic coalition structures. Games and Economic Behavior 38(2), 201–230 (2002)

[6] Burani, N., Zwicker, W.S.: Coalition formation games with separable preferences. Mathematical Social Sciences 45(1), 27–52 (2003)

[7] Cechlárová, K.: Stable partition problem. In: Encyclopedia of Algorithms. Springer, Heidelberg (2008)

[8] Christodoulou, G., Mirrokni, V.S., Sidiropoulos, A.: Convergence and approximation in potential games. In: STACS, pp. 349–360 (2006)

[9] Dreze, J.H., Greenberg, J.: Hedonic coalitions: Optimality and stability. Econometrica 48(4), 987–1003 (1980)

[10] Elkind, E., Wooldridge, M.: Hedonic coalition nets. In: Int. Conference on Autonomous Agents and Multiagent Systems, pp. 417–424 (2009)

[11] Greenberg, J.: Coalition structures. In: Aumann, R.J., Hart, S. (eds.) Handbook of Game Theory with Economic Applications II. Elsevier, Amsterdam (1994)

[12] Johnson, D.S., Papadimitriou, C.H., Yannakakis, M.: How easy is local search? Journal of Computer and System Sciences 37, 79–100 (1988)

[13] Monien, B., Tscheuschner, T.: On the power of nodes of degree four in the local max-cut problem. In: 7th International Conference on Algorithms and Complexity, CIAC (2010)

[14] Monien, B., Dumrauf, D., Tscheuschner, T.: Local search: Simple, successful, but sometimes sluggish. In: 37th International Colloquium on Automata, Languages and Programming, ICALP (2010)

[15] Orlin, J.B., Punnen, A.P., Schulz, A.S.: Approximate local search in combinatorial optimization. SIAM Journal of Computing 33(5), 1201–1214 (2004)

[16] Schäffer, A.A., Yannakakis, M.: Simple Local Search Problems that are Hard to Solve. SIAM Journal of Computing 20(1), 56–87 (1991)

[17] Sipser, M.: Introduction to the Theory of Computation. Thomson (2006)

[18] Sung, S.C., Dimitrov, D.: Computational complexity in additive hedonic games. European Journal of Operational Research 203(3), 635–639 (2010)

[19] Tscheuschner, T.: The local max-cut problem is PLS-complete even on graphs with maximum degree five (2010), `http://arxiv.org/abs/1004.5329`