On the Complexity of Reconfiguration Problems

Takehiro Ito[a,*], Erik D. Demaine[b], Nicholas J. A. Harvey[c], Christos H. Papadimitriou[d], Martha Sideri[e], Ryuhei Uehara[f], Yushi Uno[g]

[a] *Graduate School of Information Sciences, Tohoku University, Aoba-yama 6-6-05, Sendai, 980-8579, Japan.*
[b] *MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar St., Cambridge, MA 02139, USA.*
[c] *Department of Combinatorics and Optimization, University of Waterloo, 200 University Ave. West, Waterloo, Ontario N2L 3G1, Canada.*
[d] *Computer Science Division, University of California at Berkeley, Soda Hall 689, EECS Department, Berkeley, CA 94720, USA.*
[e] *Department of Computer Science, Athens University of Economics and Business, Patision 76, Athens 10434, Greece.*
[f] *School of Information Science, JAIST, Asahidai 1-1, Nomi, Ishikawa 923-1292, Japan.*
[g] *Graduate School of Science, Osaka Prefecture University, 1-1 Gakuen-cho, Naka-ku, Sakai 599-8531, Japan.*

Abstract

Reconfiguration problems arise when we wish to find a step-by-step transformation between two feasible solutions of a problem such that all intermediate results are also feasible. We demonstrate that a host of reconfiguration problems derived from NP-complete problems are PSPACE-complete, while some are also NP-hard to approximate. In contrast, several reconfiguration versions of problems in P are solvable in polynomial time.

Keywords: approximation, graph algorithm, PSPACE-complete, reachability on solution space

1. Introduction

Consider the bipartite graph with weighted vertices in Figure 1(a) (both solid and dotted edges). It models a situation in which power stations with fixed capacity (the square vertices) provide power to customers with fixed demand (the round vertices). It can be seen as a feasible solution of a particular instance of a search problem which we may call the POWER SUPPLY problem [8, 10]: Given a bipartite graph $G = (U, V, E)$ with weights on the vertices, can G be partitioned into subtrees, each of which contains exactly one vertex from U, such that the sum of the demands of the V vertices (customers) in each subtree is no more than the capacity of the U vertex (power station) in it?

But suppose now that we are given *two* feasible solutions of this instance (the leftmost and rightmost ones in Figure 1), and we are asked: Can the solution on the left be transformed into the solution on the right *by moving only one customer at a time, and always remaining feasible?* This problem, which we call the POWER SUPPLY RECONFIGURATION problem, is an exemplar of the kind of problems we discuss in this paper. (In this particular instance, it turns out that the answer is "yes"; see Figure 1.) As one may have expected, the most basic reconfiguration problem is the SATISFIABILITY RECONFIGURATION problem: Given a CNF formula and two satisfying truth assignments $\mathbf{s}_0$ and $\mathbf{s}_t$, are these connected in the subgraph of the hypercube induced by the satisfying truth assignments? This problem has been shown PSPACE-complete [3].

In more generality, *reconfiguration problems* have the following structure: Fix a search problem $\mathcal{S}$ (a polynomial-time algorithm which, on instance I and candidate solution y of length polynomial in that of I, determines whether y is a feasible solution of I); and fix a polynomially-testable symmetric *adjacency relation* A on the set of feasible solutions, that is, a polynomial-time algorithm such that, given an instance I of $\mathcal{S}$ and two feasible solutions y' and y'' of I, it determines whether y' and y'' are adjacent. (In almost all problems discussed in this paper, the feasible solutions can be considered as sets of elements, and two solutions are adjacent if their symmetric difference has size

*Corresponding author.
Email addresses: takehiro@ecei.tohoku.ac.jp (Takehiro Ito), edemaine@mit.edu (Erik D. Demaine), harvey@math.uwaterloo.ca (Nicholas J. A. Harvey), christos@cs.berkeley.edu (Christos H. Papadimitriou), sideri@aueb.gr (Martha Sideri), uehara@jaist.ac.jp (Ryuhei Uehara), uno@mi.s.osakafu-u.ac.jp (Yushi Uno)

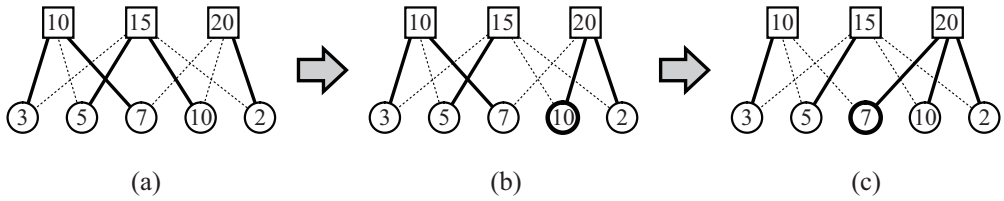


Figure 1: A sequence of feasible solutions for the POWER SUPPLY problem.

1.) The RECONFIGURATION PROBLEM FOR $\mathcal{S}$ AND A is the following computational problem: Given instance I of $\mathcal{S}$ and two feasible solutions y_0 and y_t of I, is there a sequence of feasible solutions $y_0, y_1, \ldots, y_t$ of I such that y_{i-1} and y_i are adjacent for $i = 1, 2, \ldots, t$?

Reconfiguration problems can also arise from optimization problems, if one turns the optimization problem into a search problem by giving a threshold. For example, the CLIQUE RECONFIGURATION problem is the following: Given a graph G, an integer k, and two cliques C_0 and C_t of G, both of size at least k, is there a way to transform C_0 into C_t via cliques, each of which results from the previous one by adding or subtracting a single node of G, without ever going through a clique of size less than $k - 1$?

Reconfiguration problems are useful and entertaining, have been coming up in recent literature [1, 3, 6, 9], and are interesting for a variety of reasons. First, they may reflect, as in the POWER SUPPLY RECONFIGURATION problem above, a situation where we actually seek to implement such a sequence of elementary changes in order to transform the current configuration to a more desirable one, in a context in which intermediate steps must also be fully feasible, and only restricted changes can occur — in our example, no two customers can change providers simultaneously, and we certainly do not wish customers to be without power. In a complex, dynamic environment in which changing circumstances affect the feasible solution of choice, determining whether such adaptation is possible may be crucial. Reconfiguration problems also model questions of *evolvability*: Can genotype y_0 evolve into genotype y_t via individual mutations which are each of adequate fitness? Here a genotype is considered feasible if its fitness is above a threshold, and two genotypes are considered adjacent if one is a simple mutation of the other. Finally, reconfiguration versions of constraint satisfaction problems (the first kind studied in the literature [3]) yield insights into the structure of the solution space, which may help in understanding heuristics, such as survey propagation, whose performance depends crucially on connectivity and other properties of the solution space.

In this paper, we embark on a systematic investigation of the complexity of reconfiguration problems. Our main focus is showing that a host of reconfiguration problems (including all those mentioned above and many more) are PSPACE-complete. The proof for the POWER SUPPLY RECONFIGURATION problem and those for certain other problems are explained in Section 2. We then point out in Section 3 that certain reconfiguration problems arising from problems in P (such as MINIMUM SPANNING TREE and MATCHING) can be solved in polynomial time. In Section 4 we show certain approximability and inapproximability results for reconfiguration problems. An extended abstract of the paper has been presented in [7].

2. PSPACE-completeness

In this section we show that a host of reconfiguration problems are PSPACE-complete. In Section 2.1 we first give a proof for the POWER SUPPLY RECONFIGURATION problem, and in Section 2.2 we then give proof sketches for certain other reconfiguration problems.

2.1. POWER SUPPLY RECONFIGURATION

The POWER SUPPLY RECONFIGURATION problem was defined informally in the Introduction. An instance is given in terms of a bipartite graph $G = (U, V, E)$, where each vertex in U is called a *supply vertex* and each vertex in V is called a *demand vertex*. Each supply vertex $u \in U$ is assigned a positive integer $\sup(u)$, called the *supply of* u, while each demand vertex $v \in V$ is assigned a positive integer $\mathrm{dem}(v)$, called the *demand of* v. We wish to partition G into subtrees, by deleting edges from G, such that each subtree T has exactly one supply vertex whose supply is at least

the sum of demands of all demand vertices in T. We call an assignment $f : V \to U$ a *configuration of* G if there is an edge $\big(v, f(v)\big) \in E$ for each demand vertex $v \in V$. A configuration f of G is *feasible* if the following condition holds: for each supply vertex $u \in U$,

$$\sup(u) \geq \sum\Big\{\mathrm{dem}(v) \mid v \in V \text{ such that } f(v) = u\Big\}.$$

The *adjacency relation* on the set of feasible configurations is defined as follows: two feasible configurations f and f' are *adjacent* if $\big|\{v \in V : f(v) \neq f'(v)\}\big| = 1$, that is, f' can be obtained from f by changing the assignment of a single demand vertex. Given a bipartite graph $G = (U, V, E)$ and two feasible configurations f_0 and f_t of G, the POWER SUPPLY RECONFIGURATION problem is to determine whether there is a sequence of feasible configurations $f_0, f_1, \ldots, f_t$ of G such that f_{i-1} and f_i are adjacent for $i = 1, 2, \ldots, t$. Note that POWER SUPPLY RECONFIGURATION, as well as any reconfiguration problem defined in this paper, does not ask an actual reconfiguration sequence.

Figure 1 illustrates three feasible configurations of a bipartite graph G, where each supply vertex is drawn as a square, each demand vertex as a circle, and the supply or demand is written inside. Figure 1 also illustrates an example of a transformation from the feasible configuration in Figure 1(a) to one in Figure 1(c), where the demand vertex whose assignment was changed from the previous one is depicted by a thick circle.

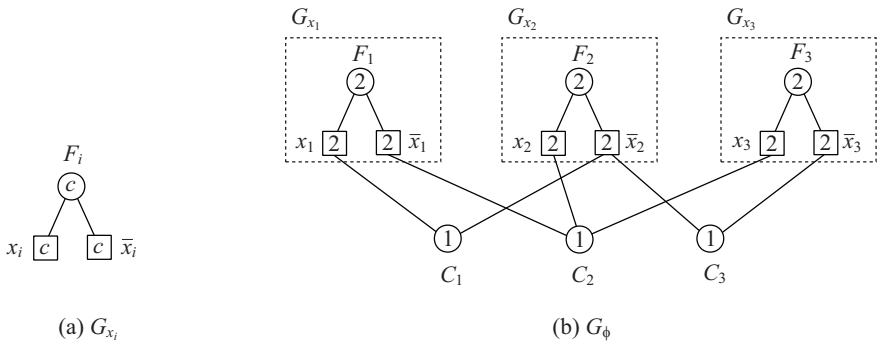
We have the following theorem.

Theorem 1. POWER SUPPLY RECONFIGURATION *is* PSPACE*-complete.*

PROOF. It is easy to see that this problem, as well as any reconfiguration version of a problem $\mathcal{S}$ in NP, can be solved in polynomial space, as follows. Since $\mathcal{S}$ is in NP, we can enumerate all feasible solutions of $\mathcal{S}$ in nondeterministic polynomial time. Since NP $\subseteq$ PSPACE [11, p. 148], this enumeration can be done in PSPACE. We then nondeterministically traverse the solutions that are adjacent with the current solution. (By the assumption, the adjacency can be checked in polynomial time for each enumerated solution.) Savitch's Theorem [12] says that this NPSPACE algorithm can be converted into a PSPACE algorithm.

We give a polynomial-time reduction from the SATISFIABILITY RECONFIGURATION problem to this problem. In that problem we are given a Boolean formula ϕ in conjunctive normal form, say with n variables $x_1, x_2, \ldots, x_n$ and m clauses $C_1, C_2, \ldots, C_m$, and two satisfying truth assignments $\mathbf{s}_0$ and $\mathbf{s}_t$ of ϕ. Then, we are asked whether there is a sequence of satisfying truth assignments, starting with $\mathbf{s}_0$ and ending in $\mathbf{s}_t$, and each differing from the previous one in only one variable. This problem is known to be PSPACE-complete [3]. One may assume without loss of generality that the formula ϕ has no clause which contains both positive and negative literals of the same variable. Let c be the maximum number of occurrences of a literal in the clauses, and hence each literal appears in at most c clauses in ϕ.

Given such an instance of SATISFIABILITY RECONFIGURATION, we construct an instance of POWER SUPPLY RECONFIGURATION as follows. We first make a *variable gadget* G_{x_i} for each variable x_i, $1 \leq i \leq n$; G_{x_i} is a binary tree with three vertices as illustrated in Figure 2(a); the root F_i is a demand vertex of demand c, and the two leaves x_i and $\overline{x_i}$ are supply vertices of supply c. Then the bipartite graph G_ϕ corresponding to the formula ϕ is constructed as follows. For



(a) G_{x_i} (b) G_ϕ

Figure 2: (a) Variable gadget G_{x_i}, and (b) bipartite graph G_ϕ corresponding to a Boolean formula ϕ with three clauses $C_1 = (x_1 \vee \overline{x}_2)$, $C_2 = (\overline{x}_1 \vee x_2 \vee x_3)$ and $C_3 = (\overline{x}_2 \vee \overline{x}_3)$, and hence $c = 2$.

each variable x_i, $1 \leq i \leq n$, add the variable gadget G_{x_i} to the graph; and, for each clause C_j, $1 \leq j \leq m$, add a demand vertex C_j of demand 1 to the graph. Finally, join a supply vertex x_i (or $\bar{x_i}$) in G_{x_i}, $1 \leq i \leq n$, with the clause demand vertex C_j, $1 \leq j \leq m$, if and only if the literal x_i (respectively, $\bar{x_i}$) is in the clause C_j. (See Figure 2(b) as an example.) Clearly, G_ϕ is a bipartite graph.

Consider a feasible configuration of G_ϕ. Then each demand vertex F_i, $1 \leq i \leq n$, must be assigned to one of x_i and $\bar{x_i}$; *a literal is considered false if F_i is assigned to its corresponding supply vertex.* Notice that, since supply vertices have supply c and the F_i's have demand c, a false-literal supply vertex cannot provide power to any of the other demand vertices. Hence, all clause demand vertices C_j, $1 \leq j \leq m$, must be assigned to true-literal supply vertices that occur in them. Since each literal x_i (or $\bar{x_i}$), $1 \leq i \leq n$, appears in at most c clauses in ϕ, the corresponding supply vertex x_i (respectively, $\bar{x_i}$) in G_{x_i} can provide power to all clause demand vertices C_j whose corresponding clauses have x_i (respectively, $\bar{x_i}$).

To complete the reduction, we now create two feasible configurations f_0 and f_t of G_ϕ corresponding to the satisfying truth assignments $\mathbf{s}_0$ and $\mathbf{s}_t$ of ϕ, respectively. Each demand vertex F_i, $1 \leq i \leq n$, is assigned to the supply vertex whose corresponding literal is false, while each clause demand vertex C_j, $1 \leq j \leq m$, is assigned to an arbitrary true-literal supply vertex adjacent to C_j. Since $\mathbf{s}_0$ and $\mathbf{s}_t$ are satisfying truth assignments of ϕ, both f_0 and f_t are feasible configurations of G_ϕ. This completes the construction of the corresponding instance of the POWER SUPPLY RECONFIGURATION problem.

We know that a feasible configuration of G_ϕ corresponds to a satisfying truth assignment of ϕ *plus* an assignment of each clause to a true literal. It is easy to see that this correspondence goes backwards: every satisfying truth assignment of ϕ can be mapped to at least one (in general, to exponentially many) feasible configurations of G_ϕ.

How about adjacent configurations — defined to be configurations differing in the assignment of just one demand vertex? One can easily observe that there are only two types of reassignments to go from a feasible configuration of G_ϕ to an adjacent one, as follows:

(1) One could change the assignment of a demand vertex F_i from x_i to $\bar{x_i}$, or vice versa, *if no clause demand vertex is currently assigned to supply vertices x_i or $\bar{x_i}$.*
(2) Alternatively, if a clause demand vertex C_j is adjacent to more than one true-literal supply vertex, then one could change the assignment of C_j from the current one to another.

Therefore, any sequence of adjacent feasible configurations of G_ϕ can be broken down to subsequences, intermittently with a reassignment of type (1) above; in each subsequence, every two adjacent configurations can go from one to another via a reassignment of type (2) above. Therefore, all feasible configurations in each subsequence correspond to the same satisfying truth assignment of ϕ, while any two consecutive subsequences correspond to adjacent satisfying truth assignments (namely, differing in only one variable).

Conversely, for given any sequence of adjacent satisfying truth assignments of ϕ, there is a corresponding sequence of adjacent feasible configurations of G_ϕ, obtained as follows: Consider a flip of a variable x_i from true to false. (A flip of x_i from false to true is similar.) Then we wish to change the assignment of the demand vertex F_i from the supply vertex $\bar{x_i}$ to x_i. (Remember that the literal to which F_i is assigned is considered false.) We first change the assignments of all clause demand vertices, which are currently assigned to x_i, to another true-literal supply vertex: since we are about to flip the variable x_i and we know that the truth assignment of ϕ after the flip will be also satisfiable, there must be a "second" true-literal supply vertex for every clause demand vertex currently assigned to x_i. After all such reassignments, we finally change the assignment of F_i from $\bar{x_i}$ to x_i.

It is now easy to see that there is a sequence of adjacent satisfying truth assignments of ϕ from $\mathbf{s}_0$ to $\mathbf{s}_t$ if and only if there is a sequence of adjacent feasible configurations of G_ϕ from f_0 to f_t. This completes the proof of Theorem 1. □

2.2. Other intractable reconfiguration problems

There is a wealth of reconfiguration versions of NP-complete problems which can be shown PSPACE-complete via extensions, often quite sophisticated, of the original NP-completeness proofs; in this subsection we only sample the realm of possibilities.

We have already defined the CLIQUE RECONFIGURATION problem in the Introduction as an example of a general scheme whereby any optimization problem can be transformed into a reconfiguration problem by giving a threshold (upper bound for minimization problems, lower bound for maximization problems) for the allowed values of the objective

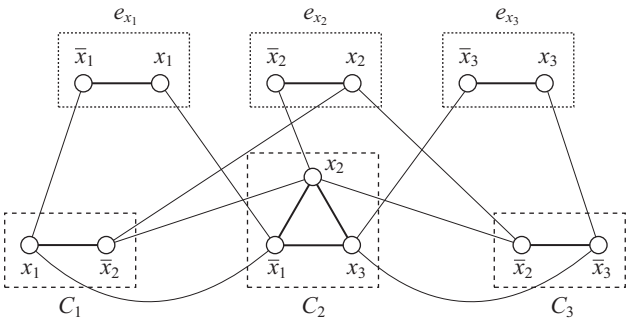


Figure 3: Graph $\rho(\phi)$ corresponding to a 3SAT formula ϕ with three clauses $C_1 = (x_1 \vee \bar{x}_2)$, $C_2 = (\bar{x}_1 \vee x_2 \vee x_3)$ and $C_3 = (\bar{x}_2 \vee \bar{x}_3)$.

function of the intermediate feasible solutions; the INDEPENDENT SET RECONFIGURATION and VERTEX COVER RECONFIGURATION problems are defined similarly. In the INTEGER PROGRAMMING RECONFIGURATION problem, we are given a 0-1 linear program seeking to maximize cx subject to $Ax \leq b$, and we consider two solutions adjacent if they only differ in one variable.

Theorem 2. *The following problems are* PSPACE*-complete*: INDEPENDENT SET RECONFIGURATION, CLIQUE RECONFIGURATION, VERTEX COVER RECONFIGURATION, SET COVER RECONFIGURATION, INTEGER PROGRAMMING RECONFIGURATION.

Proof sketch. We sketch a proof for the INDEPENDENT SET RECONFIGURATION problem. The reduction can be obtained by extending the well-known reduction from the 3SAT problem to the INDEPENDENT SET problem [11]. We construct a graph $\rho(\phi)$ from a given 3SAT formula ϕ with n variables and m clauses, as follows. (As in the proof of Theorem 1, we may assume without loss of generality that the formula ϕ has no clause which contains both positive and negative literals of the same variable.) For each variable x in ϕ, we add an edge e_x to the graph; its two endpoints are labeled x and $\bar{x}$. Then, for each clause C in ϕ, we add a clique of size $|C|$ to the graph; each node in the clique corresponds to a literal in the clause C. Finally, we add an edge between two nodes in different components if and only if the nodes correspond to opposite literals. (See Figure 3 as an example.) Then, it is easy to see that $\rho(\phi)$ has a maximum independent set of size $k = n + m$ if and only if ϕ is satisfiable; n nodes are chosen from the endpoints of n edges corresponding to the variables; *a literal is considered true if the corresponding endpoint is chosen.* Consider all independent sets of size $k = n + m$ in $\rho(\phi)$; they can be partitioned into subclasses of the form $\rho(\mathbf{s})$ corresponding to the satisfying truth assignments $\mathbf{s}$ of ϕ (the various independent sets in the subclass $\rho(\mathbf{s})$ correspond to the different possible ways to satisfy each clause by $\mathbf{s}$). It is easy to see that all independent sets in $\rho(\mathbf{s})$ are connected via intermediate independent sets of size at least $k-1$. Therefore, by similar arguments in the proof of Theorem 1, it is easy to observe that deciding whether two independent sets of size k in $\rho(\phi)$ can be transformed into one another via intermediate independent sets of size at least $k - 1$ is PSPACE-complete.

It is easy to see that a subset $I \subseteq V$ of vertices in a graph $G = (V, E)$ is an independent set of G if and only if I induces a clique in the complement of G. Also, I is an independent set of G if and only if $V \setminus I$ is a vertex cover of G [2, Lemma 3.1]. Thus, the result for INDEPENDENT SET RECONFIGURATION yields those for CLIQUE RECONFIGURATION and VERTEX COVER RECONFIGURATION. Then, the result for SET COVER RECONFIGURATION is immediate since it is a generalization of VERTEX COVER RECONFIGURATION. INTEGER PROGRAMMING RECONFIGURATION generalizes CLIQUE RECONFIGURATION via the well-known integer program for CLIQUE. □

One might compare our INDEPENDENT SET RECONFIGURATION problem with the SLIDING TOKEN problem, which is also known to be PSPACE-complete [6]. A *Token configuration T of a graph G* is an independent set of G such that a Token is placed on each vertex in T. In the SLIDING TOKEN problem, we are given a graph G and two Token configurations (independent sets) T_0 and T_t of G, both have the same number of Tokens, and we are asked whether there is a sequence of Token configurations of G, starting with T_0 and ending in T_t, and each resulting from the previous one by sliding only one Token from one vertex *to an adjacent vertex.* Therefore, the two problems have slightly different adjacency relations: in our INDEPENDENT SET RECONFIGURATION problem, a Token can "jump" from one vertex to any other vertex

Figure 4: (a) Initial independent set (Token configuration) and (b) target independent set (Token configuration).

if it results in an independent set of G; while, in the SLIDING TOKEN problem, we can just slide a Token *along an edge* of G. Consider the instance in Figure 4, where the vertices in independent sets (or Token configurations) are colored with black. Then, this is an Yes-instance for INDEPENDENT SET RECONFIGURATION with $k = 2$, but a No-instance for SLIDING TOKEN. However, the PSPACE-completeness proof for SLIDING TOKEN by [6] indeed works to prove our result for INDEPENDENT SET RECONFIGURATION. Then, we can prove that INDEPENDENT SET RECONFIGURATION and VERTEX COVER RECONFIGURATION remain PSPACE-complete even for planar graphs of maximum degree 3.

3. Reconfiguration Problems in P

Reconfiguration problems arise in relation to problems in P as well. For example, in the MINIMUM SPANNING TREE RECONFIGURATION problem, we are given an edge-weighted graph G, a threshold k, and two spanning trees of G, both of weight at most k, and wish to transform one tree into another via edge exchanges, without ever getting into a tree with weight $> k$. The MATCHING RECONFIGURATION problem is defined similarly (the formal definition will be given later). We show in this section that both problems can be solved in polynomial time.

The result for the MINIMUM SPANNING TREE RECONFIGURATION problem can be obtained from the following more general proposition.

Proposition 1. *Let $\mathbf{M} = (S, \mathcal{B})$ be a matroid, and let $w : S \to \mathbb{R}$ be a weight function on S. Let B_0 and B_t be two bases in $\mathcal{B}$ such that $\max\{w(B_0), w(B_t)\} \leq k$. Then, there always exists a sequence of $|B_0 \setminus B_t|$ $(= |B_t \setminus B_0|)$ exchanges that transforms one into the other, without ever exceeding weight k, and maintaining a base at each step.*

PROOF. Since the adjacency relation is symmetric, we may assume without loss of generality that $w(B_0) \leq w(B_t)$. Since B_0 and B_t are bases, $|B_0| = |B_t|$ and hence let $m = |B_0 \setminus B_t| = |B_t \setminus B_0|$. The proposition trivially holds if $m = 1$. Therefore, by applying induction, it suffices to prove the following claim: there exists $y \in B_0 \setminus B_t$ and $z \in B_t \setminus B_0$ such that $B_0 - y + z$ is a base in $\mathcal{B}$ and $w(B_0 - y + z) \leq w(B_t)$, where we use the shorthand notation $B - y + z = (B \setminus \{y\}) \cup \{z\}$. Observe that $\left|(B_0 - y + z) \setminus B_t\right| = \left|B_t \setminus (B_0 - y + z)\right| = m - 1$ and $w(B_0 - y + z) \leq k$ if the claim holds.

By Brualdi's exchange property [14, Corollary 39.12a], we can always write $B_0 \setminus B_t = \{y_1, y_2, \ldots, y_m\}$ and $B_t \setminus B_0 = \{z_1, z_2, \ldots, z_m\}$ such that $B_0 - y_i + z_i$ is a base in $\mathcal{B}$ for every index i, $1 \leq i \leq m$. Suppose for a contradiction that

$$w(B_0 - y_i + z_i) = w(B_0) - w(y_i) + w(z_i) > w(B_t)$$

for all indices $i = 1, 2, \ldots, m$. Then, $w(z_i) - w(y_i) > w(B_t) - w(B_0)$, and hence

$$\begin{aligned} w(B_t) &= w(B_0) + \sum_{1 \leq i \leq m} \Big(w(z_i) - w(y_i)\Big) \\ &> w(B_0) + \sum_{1 \leq i \leq m} \Big(w(B_t) - w(B_0)\Big) \\ &= w(B_0) + m \cdot \Big(w(B_t) - w(B_0)\Big) \\ &\geq w(B_0) + \Big(w(B_t) - w(B_0)\Big) \\ &= w(B_t), \end{aligned}$$

a contradiction. Therefore, there must exist some index i such that $w(B_0 - y_i + z_i) \leq w(B_t)$, as required. □

In the MATCHING RECONFIGURATION problem, we are given an unweighted graph G, a threshold k, and two matchings M_0 and M_t of G, both of size at least k, and we are asked whether there is a sequence of matchings of G, starting with M_0 and ending in M_t, and each resulting from the previous one by either addition or deletion of a single edge in G, without ever going through a matching of size less than $k-1$.

Proposition 2. MATCHING RECONFIGURATION *can be solved in polynomial time.*

In the remainder of this section, as a proof of Proposition 2, we give a polynomial-time algorithm which solves MATCHING RECONFIGURATION.

We first introduce some terms. Let M be a matching of a graph G. A vertex v is called *M-covered* if v is incident with an edge in M; otherwise, v is called *M-exposed*. A path (or a cycle) of G is called *M-alternating* if the edges along the path (respectively, along the cycle) belong alternatively to M and not to M. An *M-augmenting path* is an M-alternating path whose endpoints are both M-exposed. For two matchings M and N of G, we denote by $M \triangle N$ the symmetric difference of M and N, that is, $M \triangle N = (M \setminus N) \cup (N \setminus M)$. A path (or a cycle) of G is called *(M,N)-alternating* if the edges along the path (respectively, along the cycle) belong alternatively to M and to N. The length of a path P in a graph is defined as the number of edges in P.

We may assume without loss of generality that $|M_0| \leq |M_t|$. Consider the subgraph H of G induced by all edges in $M_0 \triangle M_t$. Then, since M_0 and M_t are both matchings of G, each vertex in H has degree at most 2. Therefore, H consists of single edges, (M_0, M_t)-alternating paths and (M_0, M_t)-alternating cycles. The *greedy algorithm* for transforming M_0 into M_t is the following. Divide the components of H into the following four categories:

(1) single edges of $M_t \setminus M_0$;
(2) (M_0, M_t)-alternating paths which start and end with edges of $M_t \setminus M_0$;
(3) (M_0, M_t)-alternating cycles; and
(4) all the rest.

In this category order, transform M_0 into M_t by repeatedly adding edges of $M_t \setminus M_0$ and deleting edges of $M_0 \setminus M_t$ along each component of H. It is easy to see that intermediate matchings have size at least $|M_0| - 1$ ($\geq k-1$) for exchanging edges in Category (2). Therefore, we can always exchange the edges in Categories (1) and (2). Moreover, since each component in Categories (1) and (2) is an M_0-augmenting path, the matching M obtained by exchanging all edges in Categories (1) and (2) has size at least $|M_t|$ ($\geq |M_0|$). We then exchange the edges in an (M_0, M_t)-alternating cycle C in Category (3), as follows: we first delete an arbitrary edge in $C \cap M_0$, and then exchange the remaining edges along the obtained (M_0, M_t)-alternating path. Therefore, intermediate matchings have size at least $|M| - 2 \geq |M_t| - 2$ for exchanging the edges in Category (3). Similarly, the edges in Category (4) can be exchanged without ever going through a matching of size less than $|M_t| - 2$.

We show that the greedy algorithm correctly solves MATCHING RECONFIGURATION in polynomial time.

Case (a): $|M_t| \geq k+1$.

In this case, since the greedy algorithm transforms M_0 into M_t without ever going through a matching of size less than $|M_t| - 2$, all the intermediate matchings have size at least $|M_t| - 2 \geq k-1$, as required.

Case (b): $|M_t| = k$, and M_t is *not* a maximum matching of G.

In this case, we first transform M_t into a matching M'_t of size $k+1$ along an arbitrary M_t-augmenting path P; clearly, the intermediate matchings for exchanging the edges in P have size at least $|M_t| - 1 = k-1$. Then, the greedy algorithm can transform M_0 into M'_t so that all intermediate matchings are of size $\geq k-1$. Finally, we transform M'_t into M_t along the path P. In this way, a desired sequence always exists for this case.

Case (c): $|M_t| = k$, and M_t is a maximum matching of G.

Since $k \leq |M_0| \leq |M_t|$, M_0 is also a maximum matching of G. Then, H consists only of (M_0, M_t)-alternating paths with even-length and (M_0, M_t)-alternating cycles; otherwise, this contradicts that M_0 and M_t are both maximum matchings of G. Therefore, H contains components only of Categories (3) and (4).

Since every component in Category (4) is an even-length (M_0, M_t)-alternating path, each path starts with an edge of $M_t \setminus M_0$ and ends at an edge of $M_0 \setminus M_t$. It is easy to see that all intermediate matchings have size at least $|M_t| - 1 \geq k-1$ for exchanging edges in the path. Therefore, if H contains no component of Category (3), then the greedy algorithm can transform M_0 into M_t without ever going through a matching of size less than $k-1$.

Figure 5: (a) No-instance and (b) Yes-instance for MATCHING RECONFIGURATION, where $M_0 = \{e_1, e_3\}$, $M_t = \{e_2, e_4\}$ and $k = 2$ in both instances.

Suppose now that H contains components of Category (3). In this case, there does not always exist a desired sequence of matchings. (See Figure 5 as an example.) Nonetheless, existence can be determined in polynomial time by the following lemma.

Lemma 1. *Suppose that both M_0 and M_t are maximum matchings of G, and let $k = |M_0| = |M_t|$. Then, there exists a sequence of matchings which transforms M_0 into M_t so that all intermediate matchings have size at least $k - 1$ if and only if, for every (M_0, M_t)-alternating cycle C, there exists an M_0-alternating path in G starting with an M_0-exposed vertex and ending at a vertex in C.*

For the example in Figure 5(b), the (M_0, M_t)-alternating cycle $\{e_1, e_2, e_3, e_4\}$ has such an M_0-alternating path $\{e_5\}$. By Lemma 1 one can easily determine whether there exists a desired sequence for Case (c) in polynomial time; we simply check if there exists such an M_0-alternating path P in G, assuming that each vertex in an (M_0, M_t)-alternating cycle is the endpoint of P.

From now on, we prove Lemma 1 to complete the proof of Proposition 2. We first show a useful fact, which is a part of the Edmonds-Gallai decomposition [14].

For a graph $G = (V, E)$, let

$$D(G) = \{v \in V \mid \text{there exists a maximum matching } N \text{ of } G \text{ in which } v \text{ is } N\text{-exposed}\}.$$

For a maximum matching M of G, let

$$\mathrm{EVEN}(M) = \{v \in V \mid \text{there exists an even-length } M\text{-alternating path from an } M\text{-exposed vertex to } v\}.$$

Note that we regard an M-alternating path of length 0 as even-length path, and hence $\mathrm{EVEN}(M)$ contains all M-exposed vertices. We have the following lemma.

Lemma 2. *For every maximum matching M of a graph G, $\mathrm{EVEN}(M) = D(G)$.*

PROOF. We first show that $\mathrm{EVEN}(M) \subseteq D(G)$. Let v be an arbitrary vertex in $\mathrm{EVEN}(M)$. Then, there exists an even-length M-alternating path P from an M-exposed vertex to v. Consider the matching $M' = M \triangle P$. (Note that $M' = M$ if P is an M-alternating path of length 0.) Since the length of P is even, M' is also a maximum matching of G and v is M'-exposed. We thus have $v \in D(G)$.

We then show that $\mathrm{EVEN}(M) \supseteq D(G)$. Let v be an arbitrary vertex in $D(G)$. If v is M-exposed, then $v \in \mathrm{EVEN}(M)$, of course. Suppose now that v is M-covered. Since $v \in D(G)$, there exists a maximum matching N of G in which v is N-exposed. Consider the subgraph $H_{M,N}$ of G induced by all edges in $M \triangle N$. Then, since M and N are both maximum matchings of G, $H_{M,N}$ consists only of (M, N)-alternating paths with even-length and (M, N)-alternating cycles. Since v is M-covered and N-exposed, v must be an endpoint of an even-length (M, N)-alternating path P. Clearly, the other endpoint of the path P is M-exposed (and N-covered), and hence $v \in \mathrm{EVEN}(M)$. □

Lemma 2 immediately implies the following corollary.

Corollary 1. *For every two maximum matchings M and N of G, $\mathrm{EVEN}(M) = \mathrm{EVEN}(N)$.*

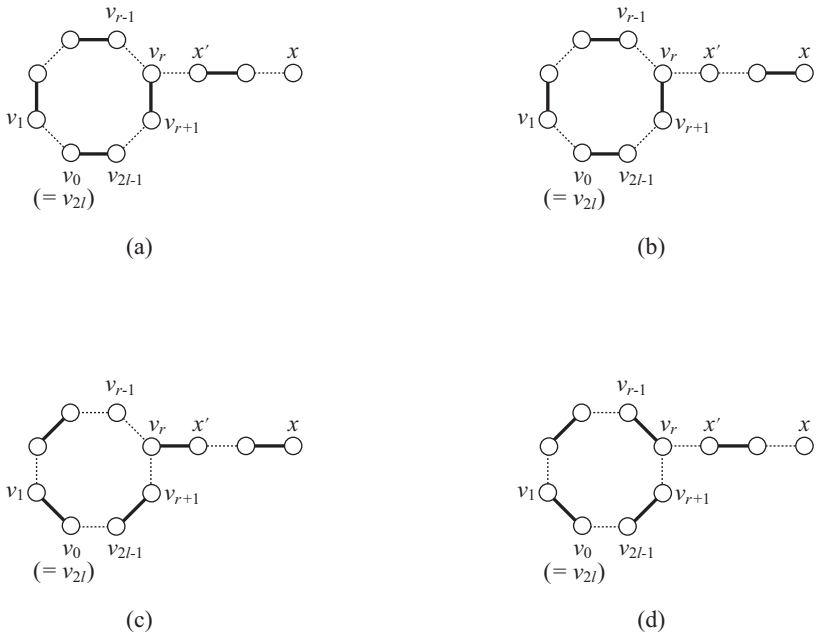


Figure 6: Exchanging the edges in an (M_0, M_t)-alternating cycle $C = \{v_0, v_1, \ldots, v_{2l}\}$ using an M_0-alternating path P starting with an M_0-exposed vertex x and ending at $v_r \in C$, where each edge in a matching is drawn as a thick line.

We are now ready to prove Lemma 1.

[Proof of Lemma 1]
Necessity: Suppose that, for every (M_0, M_t)-alternating cycle, there exists an M_0-alternating path in G starting with an M_0-exposed vertex and ending at a vertex in the cycle. It suffices to show that we can exchange the edges in Category (3) such that all intermediate matchings are of size $\geq k - 1$.

Let $C = \{v_0, v_1, \ldots, v_{2l}\}$ be an (M_0, M_t)-alternating cycle where $v_{2l} = v_0$, and suppose that there exists an M_0-alternating path P starting with an M_0-exposed vertex x and ending at v_r in C. (See Figure 6(a).) Let x' be the vertex in P adjacent with v_r, as illustrated in Figure 6(a). Note that, since v_r is in C, the edge (x', v_r) is not in M_0. Then, we exchange the edges in C as follows: first, exchange the edges of the path $\{x, \ldots, x'\}$ along P, and obtain a matching M in which x' is M-exposed (see Figs. 6(a) and (b)); then, exchange the edges of the path $\{x', v_r, v_{r+1}, \ldots, v_{r-1}\}$ in this order (see Figs. 6(b) and (c)); finally, exchange the edges of the path $\{v_{r-1}, v_r, x', \ldots, x\}$ in this order (see Figs. 6(c) and (d)). Clearly, all intermediate matchings have size $\geq k - 1$.

Let M' be the matching of G obtained by the edge exchanges above. Let $E(C)$ be the set of edges in C. Since $M_0 \cap M' = M_0 \setminus E(C)$, we can exchange the edges of each (M_0, M_t)-alternating cycle independently. In this way, we can exchange the edges of all components of Category (3) such that all intermediate matchings are of size $\geq k - 1$, and hence there exists a way to transform M_0 into M_t without ever going through a matching of size less than $k - 1$.

Sufficiency: Suppose that Category (3) contains an (M_0, M_t)-alternating cycle C such that there is no M_0-alternating path in G starting with an M_0-exposed vertex and ending at a vertex in C. Then, no vertex in C is contained in EVEN(M_0). Suppose for a contradiction that there is a sequence of matchings which transforms M_0 into M_t such that all intermediate matchings are of size $\geq k - 1$. Let $M_0, M_1, \ldots, M_t$ be such a sequence of matchings whose length (*i.e.* the number of intermediate matchings) is minimum. Let M_q be the first matching in the sequence for which we remove an edge (u, v) of M_0 that belongs to C. Then, since k is equal to the maximum size of a matching in G, we clearly have $|M_{q-1}| = k$, $|M_q| = k - 1$ and

$$\text{Exposed}(M_q) = \text{Exposed}(M_{q-1}) \cup \{u, v\},$$

where Exposed(M) is the set of all M-exposed vertices in G for a matching M of G. Since all intermediate matchings are of size $\geq k-1$, the matching M_{q+1} must be obtained from M_q by adding some edge (y, z). Note that y and z must be both in Exposed(M_q). If both y and z are also in Exposed(M_{q-1}), then this contradicts the fact that M_{q-1} is a maximum matching of G. We thus assume that $y = u$. If $z = v$, then $M_{q-1} = M_{q+1}$; this contradicts that $M_0, M_1, \ldots, M_t$ is a minimum-length sequence. Therefore, z is some vertex in Exposed(M_{q-1}). But then, the path $\{z, u, v\}$ is an even-length M_{q-1}-alternating path. Since z is M_{q-1}-exposed and M_{q-1} is a maximum matching of G, v is in EVEN(M_{q-1}). By Corollary 1, EVEN(M_0) = EVEN(M_{q-1}) and hence $v \in$ EVEN(M_0). This contradicts that no vertex in C is contained in EVEN(M_0). □

Besides MATROID RECONFIGURATION and MATCHING RECONFIGURATION, it turns out that all polynomial-time solvable special cases of SATISFIABILITY, as characterized by Schaefer [13], give rise to polynomially solvable reconfiguration problems:

Theorem 3 ([3]). SATISFIABILITY RECONFIGURATION *for linear, Horn, dual Horn and 2-literal clauses are all in* P.

4. Approximation

We have seen that an optimization problem gives rise to a reconfiguration problem by bounding the objective of intermediate configurations. In turn, we can get a natural optimization problem if we try to *optimize the worst objective among all configurations* in the reconfiguration sequence. For example, in the problem that we call the MAXMIN CLIQUE RECONFIGURATION problem, we are given a graph and two cliques C_0 and C_t, and we are asked to maximize the minimum size of any clique in a sequence which transforms C_0 into C_t by additions and removals of single nodes. In this section, we give some inapproximability and approximability results for such optimization problems.

4.1. Inapproximability

In this subsection, we show inapproximability results for two max-min type reconfiguration problems.

We first give the following theorem for the MAXMIN CLIQUE RECONFIGURATION problem.

Theorem 4. MAXMIN CLIQUE RECONFIGURATION *cannot be approximated within any constant factor unless* P = NP.

PROOF. We give a polynomial-time reduction in an approximation-preserving manner from the (ordinary) CLIQUE problem to this problem. For a given graph G with n nodes, we construct a new graph G' with $3n$ nodes as the corresponding instance of MAXMIN CLIQUE RECONFIGURATION: a set of n nodes is connected as G, while two new sets of n nodes are connected each as a clique (these two cliques of G' are called C_0 and C_t); finally, there are edges in G' between each new node and each node in G.

Consider any sequence of cliques of G', each resulting from the previous one by insertion or deletion of a single node, starting from C_0 and ending in C_t. We claim that one of them will be a clique of G — this follows directly from the absence of any edges between C_0 and C_t. Conversely, for every clique C of G, there exists a sequence from C_0 to C_t via C: add the nodes of C to the clique C_0 and obtain the clique $C_0 \cup C$, then remove those of C_0 and obtain C, then add those of C_t and obtain $C \cup C_t$, and finally remove those of C and obtain C_t. Since $|C_0| = |C_t| = n$ and $|C| \leq n$, the minimum clique size in the sequence is the size of C, and hence solving (or approximating) this instance of MAXMIN CLIQUE RECONFIGURATION is the same as solving (respectively, approximating) the CLIQUE problem for G. Since it is known that CLIQUE cannot be approximated within any constant factor unless P = NP [4], the result follows. □

In the MAXMIN MAXSAT RECONFIGURATION problem, we are given a SAT formula and two truth assignments $\mathbf{s}_0$ and $\mathbf{s}_t$ (which are not necessarily satisfying), and we are asked to maximize the minimum number of clauses satisfied by any truth assignment in a path in the hypercube between $\mathbf{s}_0$ and $\mathbf{s}_t$. Then, a similar argument establishes the following theorem.

Theorem 5. MAXMIN MAXSAT RECONFIGURATION *cannot be approximated within a factor better than* $\frac{15}{16}$ *unless* P = NP.

Proof. We give a polynomial-time reduction in an approximation-preserving manner from the (ordinary) maxsat problem to this problem. Suppose that we are given an instance ϕ of maxsat with n variables $x_1, x_2, \ldots, x_n$ and m clauses $C_1, C_2, \ldots, C_m$. We construct a new formula ϕ' in which each clause C_j, $1 \leq j \leq m$, is replaced by $(C_j \vee y \vee z)$ where y and z are new variables, and the additional clause $(\overline{y} \vee \overline{z})$ *with weight* m. Notice that every truth assignment of ϕ' with $z \neq y$ satisfies all $2m$ clauses, and hence the truth assignments $\mathbf{s}_0 : z = 1, y = 0, x_1 = x_2 = \cdots = x_n = 1$ and $\mathbf{s}_t : z = 0, y = 1, x_1 = x_2 = \cdots = x_n = 0$ are both satisfying all $2m$ clauses.

For each truth assignment $\mathbf{s}$ of the original formula ϕ, let $\mathbf{s}'$ be a truth assignment of the corresponding formula ϕ' such that $z = y$ (namely, either $z = y = 0$ or $z = y = 1$) and each x_i, $1 \leq i \leq n$, is as in $\mathbf{s}$. Then, it is easy to see that there is a path in the $(n+2)$-dimensional hypercube from $\mathbf{s}_0$ to $\mathbf{s}_t$ via $\mathbf{s}'$ such that $y \neq z$ in all intermediate truth assignments except for $\mathbf{s}'$. Clearly, every truth assignment, except for $\mathbf{s}'$, in the path satisfies all $2m$ clauses, and hence the objective value for the path is the number of clauses satisfied by $\mathbf{s}'$.

Consider now an optimal path in the $(n+2)$-dimensional hypercube between $\mathbf{s}_0$ and $\mathbf{s}_t$. Since at $\mathbf{s}_0 : z = 1, y = 0$ and at $\mathbf{s}_t : z = 0, y = 1$, there must exist a truth assignment $\mathbf{s}^*$ on this path such that $z = y$. Since the clause $(\overline{y} \vee \overline{z})$ has weight m and the path is assumed optimal, it must be that $z = y = 0$. Thus, the remaining variables x_i, $1 \leq i \leq n$, must spell an optimal satisfying truth assignment of the original formula ϕ. Hence, from the optimal value OPT' for the corresponding instance of maxmin maxsat reconfiguration, we can compute the optimal value OPT for the original instance ϕ of maxsat: since at $\mathbf{s}^* : z = y = 0$, we have

$$\mathrm{OPT} = \mathrm{OPT}' - m. \tag{1}$$

Suppose now that we have an α-approximation for maxmin maxsat reconfiguration, and hence we can compute an approximate value A' for the corresponding instance such that

$$A' \geq \alpha \cdot \mathrm{OPT}'. \tag{2}$$

One may assume without loss of generality that $A' \geq m$; otherwise there must exist at least one truth assignment such that $z = y = 1$ in the path; but, by replacing all such truth assignments with $z = y = 0$, we can easily obtain a better objective $\geq m$. Thus, there exists a truth assignment for the original formula ϕ which satisfies a number $(A' - m)$ of clauses. Let $A = A' - m$. By Eqs. (1) and (2) we have

$$A \;=\; A' - m \geq \alpha \cdot \mathrm{OPT}' - m = \alpha \cdot \mathrm{OPT} + (\alpha - 1)m. \tag{3}$$

Since $m \geq \mathrm{OPT}$, by Eq. (3) we have $A \geq (2\alpha - 1) \cdot \mathrm{OPT}$. Therefore, we can obtain a $(2\alpha - 1)$-approximation for maxsat, from an α-approximation for maxmin maxsat reconfiguration. Since it is known that maxsat cannot be approximated within a factor better than $\frac{7}{8}$ unless P = NP [5], the result follows. □

4.2. Approximability

In this subsection, we show approximability results for two min-max type reconfiguration problems.

In the minmax set cover reconfiguration problem, we are given an universal set U, a family $\mathcal{S}$ of subsets of U, each of subsets has a nonnegative cost, and two covers C_0 and C_t of U, where a *cover* C of U is a subfamily of $\mathcal{S}$ whose union is U. Then, we are asked to minimize the maximum total cost of any cover in a sequence which transforms C_0 into C_t via covers of U, each of which results from the previous one by adding or deleting a single set in $\mathcal{S}$.

Theorem 6. *There is a linear-time* 2*-approximation algorithm for* minmax set cover reconfiguration.

Proof. For a cover C of U, we denote by $\omega(C)$ the sum of costs of all subsets in C. Consider an optimal sequence $C_0, C_1, \ldots, C_t$ for an instance of minmax set cover reconfiguration. Let OPT be the objective value for the sequence, and hence $\mathrm{OPT} = \max\{\omega(C_i) \mid 0 \leq i \leq t\}$. Therefore, we clearly have

$$\max\{\omega(C_0), \omega(C_t)\} \leq \mathrm{OPT}. \tag{4}$$

As our approximation solution, we consider the following sequence of covers: (i) add the subsets in $C_t \setminus C_0$ one by one to C_0, and obtain the cover $C_0 \cup C_t$ of U; (ii) delete the subsets in $C_0 \setminus C_t$ one by one from $C_0 \cup C_t$, and obtain C_t. Clearly, our approximate value A is $A = \omega(C_0 \cup C_t)$, and hence by Eq. (4) we have

$$A = \omega(C_0 \cup C_t) \leq \omega(C_0) + \omega(C_t) \leq 2 \cdot \max\{\omega(C_0), \omega(C_t)\} \leq 2 \cdot \mathrm{OPT}.$$

This completes the proof of Theorem 6. □

Returning to the POWER SUPPLY problem, there is a natural optimization version of the problem, in which the constraint that the total demand of all demand vertices in each subtree T be within the supply of the supply vertex in T is replaced by a "soft" criterion: we allow that the total demand in T exceeds the supply in T, but wish to minimize the sum of the "deficient power" of all supply vertices in the graph.

We now define the MINMAX POWER SUPPLY RECONFIGURATION problem. For a configuration f of a bipartite graph $G = (U, V, E)$ and a supply vertex $u \in U$, the *deficient power* $d(f, u)$ *of* u *on* f is defined as follows:

$$d(f,u) = \sum \left\{ \mathrm{dem}(v) \mid v \in V \text{ such that } f(v) = u \right\} - \sup(u).$$

If f is infeasible, then there is at least one supply vertex u such that $d(f, u) > 0$. On the other hand, if f is feasible, then $d(f, u) \leq 0$ for all supply vertices $u \in U$; in fact, a nonpositive deficient power $d(f, u)$ represents the *marginal power* of u on f. The *cost* $c(f)$ *of a configuration* f is defined as follows:

$$c(f) = \sum_{u \in U} |d(f,u)|.$$

Note that $c(f)$ contains the marginal power of supply vertices, because it is difficult to change the supplies quickly and hence we waste the marginal power. Clearly, $c(f) = \sum_{u \in U} \sup(u) - \sum_{v \in V} \mathrm{dem}(v)$ for every feasible configuration f of G. In the problem that we call the MINMAX POWER SUPPLY RECONFIGURATION problem, we are given a bipartite graph $G = (U, V, E)$ and two feasible configurations f_0 and f_t of G, and we are asked to minimize the maximum cost of any configuration in a sequence which transforms f_0 into f_t by reassignments of single demand vertices. Then, we have the following observation.

Observation 1. *The objective value for a sequence which transforms* f_0 *into* f_t *is* $\sum_{u \in U} \sup(u) - \sum_{v \in V} \mathrm{dem}(v)$ *if and only if all configurations in the sequence are feasible. Moreover, such a sequence is optimal if it exists.*

In the remainder of this subsection, we give a linear-time 2-approximation algorithm for the MINMAX POWER SUPPLY RECONFIGURATION problem if a given bipartite graph G has exactly two supply vertices. We first show that the problem is strongly NP-hard even for more restricted instances.

Lemma 3. MINMAX POWER SUPPLY RECONFIGURATION *is strongly* NP*-hard, even for the restricted problem consisting of instances on a complete bipartite graph with exactly two supply vertices.*

PROOF. We give a polynomial-time reduction from the 3-PARTITION problem [2] to this problem for a complete bipartite graph with exactly two supply vertices. In 3-PARTITION, we are given a positive integer bound b, and a set A of $3m$ elements $a_1, a_2, \ldots, a_{3m}$; each element $a_i \in A$ has a positive integer size $s(a_i)$ such that $b/4 < s(a_i) < b/2$ and such that $\sum_{a \in A} s(a) = mb$. Then, the 3-PARTITION problem is to determine whether A can be partitioned into m disjoint subsets $A_1, A_2, \ldots, A_m$ such that $\sum_{a \in A_j} s(a) = b$ for each j, $1 \leq j \leq m$. 3-PARTITION is known to be strongly NP-complete [2].

For a given instance of 3-PARTITION, we first construct a complete bipartite graph $G = (U, V, E)$ with $|U| = 2$, as follows: U consists of two supply vertices u_1 and u_2 such that $\sup(u_1) = mb$ and $\sup(u_2) = (m+1)b$; and V consists of $4m$ demand vertices $v_1, v_2, \ldots, v_{3m}$ and $b_1, b_2, \ldots, b_m$ such that $\mathrm{dem}(v_i) = s(a_i)$ for each i, $1 \leq i \leq 3m$, and $\mathrm{dem}(b_j) = b$ for each j, $1 \leq j \leq m$. We then give two feasible configurations f_0 and f_t of G, as follows:

$$f_0(x) = \begin{cases} u_1 & \text{if } x = v_i,\ 1 \leq i \leq 3m; \\ u_2 & \text{if } x = b_j,\ 1 \leq j \leq m, \end{cases}$$

and

$$f_t(x) = \begin{cases} u_2 & \text{if } x = v_i,\ 1 \leq i \leq 3m; \\ u_1 & \text{if } x = b_j,\ 1 \leq j \leq m. \end{cases}$$

Clearly, $d(f_0, u_1) = d(f_t, u_1) = 0$ and $d(f_0, u_2) = d(f_t, u_2) = -b$ (that is, only the supply vertex u_2 has an amount b of marginal power), and hence $c(f_0) = c(f_t) = b$.

It is easy to see that there exists a desired partition $\{A_1, A_2, \ldots, A_m\}$ for a given instance of 3-PARTITION if and only if there exists a sequence which consists of only feasible configurations of G for the corresponding instance of MINMAX POWER SUPPLY RECONFIGURATION. Therefore, by Observation 1 we can answer whether the set A has a desired partition by determining whether the optimal value is b or not for the corresponding instance of MINMAX POWER SUPPLY RECONFIGURATION. □

By Lemma 3 it is very unlikely that the MINMAX POWER SUPPLY RECONFIGURATION problem can be solved even in pseudo-polynomial time. However, the problem can be solved in linear time for the following special case.

Suppose in the remainder of this subsection that we are given a bipartite graph $G = (U, V, E)$ having exactly two supply vertices u_1 and u_2. (Note that G is not necessarily complete.) For two given feasible configurations f_0 and f_t of G, let $W = \{v \in V \mid f_0(v) \neq f_t(v)\}$, that is, W is the set of demand vertices which must be reassigned to the other supply vertex. Notice that all (demand) vertices in W are adjacent to both the two supply vertices. Let v^* be a demand vertex in W having the maximum demand, that is, $\mathrm{dem}(v^*) = \max\{\mathrm{dem}(v) \mid v \in W\}$. Then, we have the following lemma.

Lemma 4. *If $c(f_0) \geq 2 \cdot \mathrm{dem}(v^*)$, then an optimal sequence for the instance consists of only feasible configurations of G, and it can be found in linear time.*

PROOF. Suppose without loss of generality that $W \neq \emptyset$. If all demand vertices in W are assigned to the same supply vertex u on f_0, then we just reassign the demand vertices in W from u to the other one by one. Notice that all intermediate configurations are feasible since both f_0 and f_t are feasible. Therefore, we may assume in the following that each of the two supply vertices has at least one demand vertex in W.

Since f_0 is feasible, $c(f_0) = \mathrm{sup}(u_1) + \mathrm{sup}(u_2) - \sum_{v \in V} \mathrm{dem}(v)$ and the cost $c(f_0)$ denotes the sum of marginal power of the two supply vertices. Moreover, since the sum is at least $2 \cdot \mathrm{dem}(v^*)$, one of the two supply vertices has marginal power of at least $\mathrm{dem}(v^*)$. Therefore, we can change the assignment of at least one demand vertex $v \in W$ from the initial supply vertex to the target one, since $\mathrm{dem}(v) \leq \mathrm{dem}(v^*)$. Clearly, the resulting configuration f_1 is also feasible, and hence it satisfies $c(f_1) = c(f_0) \geq 2 \cdot \mathrm{dem}(v^*)$. In this way, by reassigning the demand vertices in W one by one, we can obtain a desired sequence $f_0, f_1, \ldots, f_t$ which consists of only feasible configurations. By Observation 1 the sequence is an optimal solution. The length of the sequence is $|W|$ ($\leq |V|$) since each demand vertex in W moves exactly once and any of the other demand vertices does not move in the sequence. We can thus find an optimal solution in linear time. □

Using Lemma 4, we have the following theorem.

Theorem 7. *There is a linear-time 2-approximation algorithm for* MINMAX POWER SUPPLY RECONFIGURATION *if a given bipartite graph has exactly two supply vertices.*

PROOF. Let OPT be the optimal value for a given instance of MINMAX POWER SUPPLY RECONFIGURATION. Since the demand vertex v^* having the maximum demand in W must be reassigned at least once in any sequence from f_0 to f_t, it is easy to observe that

$$\mathrm{OPT} \geq \mathrm{dem}(v^*). \tag{5}$$

By Lemma 4 it suffices to consider the case $c(f_0) < 2 \cdot \mathrm{dem}(v^*)$. Note that, since f_0 is feasible, $\mathrm{sup}(u_1) + \mathrm{sup}(u_2) < 2 \cdot \mathrm{dem}(v^*) + \sum_{v \in V} \mathrm{dem}(v)$ in this case. Consider a slightly modified instance in which the supplies of the two supply vertices are increased by the same amount ε so that the total supply is equal to $2 \cdot \mathrm{dem}(v^*) + \sum_{v \in V} \mathrm{dem}(v)$, that is, the supply $\overline{\mathrm{sup}}(u)$ of a supply vertex u in the modified instance is $\overline{\mathrm{sup}}(u) = \mathrm{sup}(u) + \varepsilon$ where

$$\varepsilon = \frac{1}{2}\left(2 \cdot \mathrm{dem}(v^*) + \sum_{v \in V} \mathrm{dem}(v) - \mathrm{sup}(u_1) - \mathrm{sup}(u_2)\right).$$

In the modified instance, both the configurations f_0 and f_t remain feasible and $\overline{c}(f_0) = \overline{c}(f_t) = 2 \cdot \mathrm{dem}(v^*)$, where $\overline{c}(f)$ denotes the cost of a configuration f in the modified instance. Therefore, by Lemma 4 we can find in linear time a sequence which consists of only feasible configurations for the modified instance; by Observation 1, the objective value is $2 \cdot \mathrm{dem}(v^*)$. Note that some configurations in the sequence may be infeasible for the original instance. Consider an arbitrary configuration f in the sequence which is infeasible for the original instance; let $V_1 \subseteq V$ be the set of demand vertices such that $f(v) = u_1$, and let $V_2 = V \setminus V_1$. Since f is feasible for the modified instance, we have

$$\overline{c}(f) = \left(\overline{\mathrm{sup}}(u_1) - \sum_{v \in V_1} \mathrm{dem}(v)\right) + \left(\overline{\mathrm{sup}}(u_2) - \sum_{v \in V_2} \mathrm{dem}(v)\right) = 2 \cdot \mathrm{dem}(v^*). \tag{6}$$

On the other hand, since f is infeasible for the original instance, exactly one of $d(f, u_1)$ and $d(f, u_2)$ is positive, say u_1; otherwise, either f is feasible or f_0 would be infeasible in the original instance. Then, we have

$$\begin{aligned} c(f) &= \left(\sum_{v \in V_1} \mathrm{dem}(v) - \mathrm{sup}(u_1)\right) + \left(\mathrm{sup}(u_2) - \sum_{v \in V_2} \mathrm{dem}(v)\right) \\ &= \left(\sum_{v \in V_1} \mathrm{dem}(v) - \overline{\mathrm{sup}}(u_1) + \varepsilon\right) + \left(\overline{\mathrm{sup}}(u_2) - \varepsilon - \sum_{v \in V_2} \mathrm{dem}(v)\right) \\ &\leq \overline{\mathrm{sup}}(u_2) - \sum_{v \in V_2} \mathrm{dem}(v) \end{aligned}$$

since $\sum_{v \in V_1} \mathrm{dem}(v) - \overline{\mathrm{sup}}(u_1) \leq 0$. Then, by Eq. (6) we have $c(f) \leq \bar{c}(f) = 2 \cdot \mathrm{dem}(v^*)$. By Eq. (5) we thus have $c(f) \leq 2 \cdot \mathrm{OPT}$. Since the cost of a feasible configuration is smaller than the cost of an infeasible configuration, the objective value of this sequence in the original instance is at most $2 \cdot \mathrm{OPT}$, as required. □

5. Open Problems

There are many open problems raised by this work, and we mention some of these below:

- Can the MATCHING RECONFIGURATION problem for edge-weighted graphs be solved also in polynomial time? We conjecture that the answer is positive.
- Is the TRAVELING SALESMAN RECONFIGURATION problem (where two tours are adjacent if they differ in two edges) PSPACE-complete?
- Are there better approximation algorithms for the MINMAX POWER SUPPLY RECONFIGURATION problem? Lower bounds?
- Are the problems in Section 4 PSPACE-hard to approximate (not just NP-hard)?

Acknowledgements

We thank the referees for their fruitful comments, one of which leads us to an improvement of the approximation ratio for MINMAX POWER SUPPLY RECONFIGURATION.

References

[1] P. Bonsma and L. Cereceda, Finding paths between graph colourings: PSPACE-completeness and superpolynomial distances, Theoretical Computer Science 410 (2009) 5215–5226.
[2] M. R. Garey and D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman, San Francisco, CA, 1979.
[3] P. Gopalan, P. G. Kolaitis, E. N. Maneva and C. H. Papadimitriou, The connectivity of Boolean satisfiability: computational and structural dichotomies, SIAM J. Computing 38 (2009) 2330–2355.
[4] J. Håstad, Clique is hard to approximate within $n^{1-\varepsilon}$, Acta Mathematica 182 (1999) 105–142.
[5] J. Håstad, Some optimal inapproximability results, J. ACM 48 (2001) 798–859.
[6] R. A. Hearn and E. D. Demaine, PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation, Theoretical Computer Science 343 (2005) 72–96.
[7] T. Ito, E. D. Demaine, N. J. A. Harvey, C. H. Papadimitriou, M. Sideri, R. Uehara and Y. Uno, On the complexity of reconfiguration problems, Proc. of ISAAC 2008, LNCS 5369 (2008) 28–39.
[8] T. Ito, E. D. Demaine, X. Zhou and T. Nishizeki, Approximability of partitioning graphs with supply and demand, J. of Discrete Algorithms 6 (2008) 627–650.
[9] T. Ito, M. Kamiński, E. D. Demaine, Reconfiguration of list edge-colorings in a graph, Proc. of WADS 2009, LNCS 5664 (2009) 375–386.
[10] T. Ito, X. Zhou and T. Nishizeki, Partitioning trees of supply and demand, International J. Foundations of Computer Science 16 (2005) 803–827.
[11] C. H. Papadimitriou, Computational Complexity, Addison-Wesley, 1994.
[12] W. J. Savitch, Relationships between nondeterministic and deterministic tape complexities, J. of Computer and System Sciences 4 (1970) 177–192.
[13] T. J. Schaefer, The complexity of satisfiability problems, Proc. of 10th ACM Symposium on Theory of Computing, pp. 216–226, 1978.
[14] A. Schrijver, Combinatorial Optimization: Polyhedra and Efficiency, Springer-Verlag, 2003.