JOURNAL OF COMPUTER AND SYSTEM SCIENCES 37, 79–100 (1988)

How Easy Is Local Search?

DAVID S. JOHNSON

AT & T Bell Laboratories, Murray Hill, New Jersey 07974

CHRISTOS H. PAPADIMITRIOU

Stanford University, Stanford, California and National Technical University of Athens, Athens, Greece

AND

MIHALIS YANNAKAKIS

AT & T Bell Maboratories, Murray Hill, New Jersey 07974

Received December 5, 1986; revised June 5, 1987

We investigate the complexity of finding locally optimal solutions to NP-hard combinatorial optimization problems. Local optimality arises in the context of local search algorithms, which try to find improved solutions by considering perturbations of the current solution ("neighbors" of that solution). If no neighboring solution is better than the current solution, it is locally optimal. Finding locally optimal solutions is presumably easier than finding optimal solutions. Nevertheless, many popular local search algorithms are based on neighborhood structures for which locally optimal solutions are not known to be computable in polynomial time, either by using the local search algorithms themselves or by taking some indirect route. We define a natural class PLS consisting essentially of those local search problems for which local optimality can be verified in polynomial time, and show that there are complete problems for this class. In particular, finding a partition of a graph that is locally optimal with respect to the well-known Kernighan–Lin algorithm for graph partitioning is PLS-complete, and hence can be accomplished in polynomial time only if local optima can be found in polynomial time for all local search problems in PLS.

1. INTRODUCTION

One of the few general approaches to difficult combinatorial optimization problems that has met with empirical success is *local* (or *neighborhood*) *search.* In a typical combinatorial optimization problem, each instance is associated with a finite set of feasible solutions, each feasible solution has a cost, and the goal is to find a solution of minimum (or maximum) cost. In order to derive a local search algorithm for such a problem, one superimposes on it a *neighborhood structure* that specifies a "neighborhood" for each solution, that is, a set of solutions that are, in some sense "close" to that solution. For example, in the traveling salesman problem (TSP), a classical neighborhood is the one that assigns to each tour the set of tours

0022-0000/88 $3.00

571/37/1-6

that differ from it in just two edges (this is called the *2-change* neighborhood). In the graph partitioning problem (given a graph with $2n$ vertices and weights on the edges, partiton the vertices into two sets of n vertices such that the sum of the weights of the edges going from one set to the other is minimized) a reasonable neighborhood would be the so-called "swap" neighborhood: Two partitions are neighbors if one can be obtained from the other by swapping two vertices.

Given a combinatorial optimization problem with a superimposed neighborhood structure, the local search heuristic operates as follows. Starting from an independently obtained initial solution, we repeatedly replace the current solution by a neighboring solution of better value, until no such neighboring solution exists, at which point we have identified a solution that is "locally optimal." Typically, we repeat this procedure for as many randomly chosen initial solutions as is computationally feasible and adopt the best local optimum found. Variants of this methodology have been applied to dozens of problems, often with impressive success. The much-publicized "simulated annealing" approach of [5] is just a new twist on this classical theme.

In local search, one is of course not limited to such simple neighborhood structures as those described above. Neighborhoods can be complex, asymmetric, and cost-dependent. For example, the most successful algorithm known for the TSP is the "λ-change" or "Lin–Kernighan" heuristic [7], in which the neighborhood of a tour consists of all tours that can be reached from it by a sequence of changes of edges, going to arbitrary depth, with the exponential explosion of the neighborhood controlled by a complex "greedy" criterion. Similarly, for the graph partitioning problem, the champion is the "Kernighan–Lin" local search algorithm [4], in which we go from a partition to a neighbor by a sequence of swaps. At each step of the sequence we choose a swap involving as-yet-unswapped vertices that yields the best cost differential (most positive, or least negative). Although the first few swaps may possibly worsen the solution, later swaps may more than make up for any initial loss. We can stop at any point with positive gain.

In all reported studies of the above-cited local search algorithms and their variants, local optima have been obtained from arbitrary initial solutions in very reasonable amounts of time (typical observed growths are low-order polynomials [7, 12]). There has been a debate on the *quality* of the optima [7, 9, 11], but the fact that local optima are easy to obtain has never been challenged. It has been shown [8] that in the 2-change neighborhood for the TSP we can have an exponentially long sequence of successive improvements, but the examples showing this are contrived and complex, and require that the procedure repeatedly choose the worst of its alternatives. There are no analogous examples known for λ-changes, or even for 3-changes (where neighboring tours differ by *three* edges, rather than just two).

Moreover, this exponential counterexample only rules out the obvious algorithm for finding a local optimum. There is no evidence that a clever criterion for choosing the next neighbor, or even a totally different constructive technique, might not produce a local optimum in polynomial time. The analogy with the simplex and

ellipsoid algorithms for linear programming is instructive here. The simplex algorithm is essentially a local search algorithm, where the "solutions" are vertices of a polytope and the neighbors of a solution are those solutions that can be reached from it by a single "pivot." The precise nature of the algorithm depends on the method used for choosing among neighbors when more than one neighbor offers an improvement. Most such "pivoting rules" have been shown to yield an exponential number of steps on certain pathological examples, and no pivoting rule has been proved to be immune from such examples. There are, however, polynomial-time algorithms for finding solutions that are locally optimal with respect to the simplex neighborhood structure, and the ellipsoid algorithm is one such. It differs from simplex algorithms in that it proceeds by an indirect approach and does not examine *any* vertices of the polytope except for the locally optimal one with which it terminates.

The example of linear programming is also interesting for another reason. Under the simplex neighborhood structure, local optimality implies global optimality. Thus determining a local optimum is a goal sufficient in itself. Another interesting problem in which local optimality suffices was suggested to us by Don Knuth [6] (and in fact was our initial motivation for this work). In this problem local optimality does not imply global optimality, but it is all we need for our desired application. We are given an $m \times n$ real matrix A, $m < n$, and we wish to find a nonsingular $m \times m$ submatrix B of A such that the elements of $B^{-1}A$ are all of absolute value at most 1. Since the elements of $B^{-1}A$ are ratios of determinants, finding the submatrix B with the largest determinant would do, but this is NP-complete [10]. However, it is easy to see that, according to Cramer's rule, all we need is a *locally optimal* submatrix, with column swaps as neighborhood. So, here is an NP-complete problem in which the important goal is to find *any* local optimum! No polynomial algorithm is known, but the local search heuristic (improve until locally optimum) has been observed always to converge after only a few iterations.

How easy is it to find a local optimum (in any and all of the above situations)? We can formalize this question by defining a new complexity class. This class, called PLS for polynomial-time local search, is made up of relations ("search problems" in the terminology of [1]) that associate instances of a combinatorial optimization problem (under a given neighborhood structure), with the local optima for those instances. To make this class meaningful, we must make certain assumptions on the problem and the neighborhood structure: First, given an instance (e.g., an $n \times n$ distance matrix in the traveling salesman problem), all solutions must have size bounded by a polynomial in the instance size, and we must be able to produce *some* solution (e.g., a cyclic permutation of n objects) in polynomial time. Second, given an instance and a solution, we must be able to compute the cost of the solution in polynomial time. Finally, given an instance and a solution, we must be able in polynomial time to determine whether that solution is locally optimal and, if not, to generate a neighboring solution of improved cost.

The resulting class PLS of relations lies somewhere between the search problem analoges P_S and NP_S of P and NP. (We shall define these classes more formally in

Section 2.) Where eaxtly is PLS? In particular, is it true that $PLS = P_S$ (and thus we can always find local optima in polynomial time). Is it true that $PLS = NP_S$ (and thus for some problems in PLS, local optima cannot be found in polynomial time unless $P = NP$). Or is it possible that PLS is distinct from both P_S and NP_S? Practically all the empirical evidence would lead us to conclude that finding locally optimal solutions is much easier than solving NP-hard problems. Moreover, as we shall see in Section 2, no problem in PLS can be NP-hard unless NP = co-NP. Thus it seems unlikely that $PLS = NP_S$. On the other hand, if $PLS = P_S$, then presumably there must be some general approach to finding local optima, and no such approach has yet been discovered. Note that such an approach would have to be at least as clever as the ellipsoid method, since linear programming is in PLS.

The question of $P_S = ?PLS = ?NP_S$ thus has no obvious answers and suggests no natural conjectures, a property it shares with the more general open question of how hard a "total function" in NP_S can be. (PLS-problems are total since locally optimal solutions must exist by definition, given that the set of solutions is finite.) In the interest of understanding these issues, we proceed as any complexity theorist would with a new class: We look for *complete problems.*

Of course, in this situation the conventional concepts of reduction are inadequate. Intuitively, to reduce a local search problem A to another one B, we must not only be able to map instances of A to instances of B, we must also be able to recover a local optimum of A from a local optimum of B. In Section 2 we formalize this notion of reduction together with that of PLS, P_S, and NP_S, and make some elementary observations about the relations between these classes. In Section 3 we sketch a proof that the following "generic" problem is complete for PLS: Given a circuit with many inputs and outputs, find an input whose output (when viewed as a binary integer) cannot be reduced by flipping any single bit of the input. The proof is unusually complex for a generic problem. This is because, in our definition of a problem in PLS, we use *three* algorithms, not one. Much of the complexity of the reduction lies in "absorbing" two of them into the third (the computation of the cost).

Proving the existence of a generic complete problem is only the first step in establishing the significance of a new complexity class. We must also tie the class to problems that are of interest in their own right, either for theoretical or practical reasons. In Section 4 we exhibit our first such discovery, a natural and well-studied local search problem that is complete for PLS: the one based on the Kernighan–Lin neighborhood structure for the graph partitioning problem [4]. The proof is rather subtle, in that it must deal with delicate issues not present in ordinary reductions. In Section 5 we conclude by mentioning some additional results and open problems.

2. The Class PLS

A *polynomial-time local search (PLS) problem* L can be either a *maximization* or a *minimization* problem and is specified as follows: As with all computational

problems, L has a set D_L of instances, which can be taken to be a polynomial-time recognizable subset of $\{0, 1\}^*$. For each instance x (in the TSP, the encoding of a distance matrix), we have a finite set $F_L(x)$ of *solutions* (tours in the TSP), which are considered also as strings in $\{0, 1\}^*$, without loss of generality all with the same polynomially bounded length $p(|x|)$. For each solution $s \in F_L(x)$ we have a non-negative integer *cost* $c_L(s, x)$ and also a subset $N(s, x) \subseteq F_L(x)$ called the *neighborhood* of s. The remaining constraints on L are provided by the fact that the following three polynomial-time algorithms A_L, B_L, and C_L, must exist. Algorithm A_L, given $x \in D_L$, produces a particular standard solution $A_L(x) \in F_L(x)$. Algorithm B_L, given an instance x and a string s, determines whether $s \in F_L(x)$, and if so computes $c_L(s, x)$, the cost of the solution. Algorithm C_L, given an instance x and a solution $s \in F_L(x)$, has two possible types of output, depending on s. If there is any solution $s' \in N(s, x)$ with better cost than that for s (i.e., such that $c_L(s', x) < c_L(s, x)$ if L is a minimization problem or such that $c_L(s', x) > c_L(s, x)$ if it is a maximization problem), C produces such a solution. Otherwise it reports that no such solutions exist and hence that s is *locally optimal.*

Observe that there is a "standard" local search algorithm implicit in the definition of L:

1. Given x, use A_L to produce a starting solution $s = A(x)$.
2. Repeat until locally optimal:

   Apply algorithm C_L to x and s.

   If C_L yields a better cost neighbor s' of s, set $s = s'$.

Since the set of solutions is finite, this algorithm must halt, i.e., there always exists at least one local optimum for a PLS problem L. But how long does it take to find one? If the set of solution values is polynomially bounded, as it would be in the unweighted versions of many optimization problems (e.g., find the independent set with the most vertices), then the algorithm will clearly terminate in polynomial time. Many interesting PLS-problems have exponential ranges of solution values, however. As remarked in the Introduction, for some of these problems the standard algorithm has been shown to take exponential time. (The next proof contains an example.) However, although the algorithm itself might take exponential time, one might hope to obtain its output more quickly by other means. This gives rise to the following "standard algorithm problem": "Given x, find the local optimum s that would be output by the standard local search algorithm for L on input x." Unfortunately, a general shortcut yielding polynomial-time algorithms for standard algorithm problems seems unlikely, given the following observation:

LEMMA 1. *There is a PLS problem L whose standard algorithm problem is NP-hard.*

Proof. Consider the following local search problem L with the same instances as SATISFIABILITY. The "solutions" for an instance x with n variables are truth

assignments to those variables, viewed as elements of $\{0,1\}^n$, and hence as integers from 0 to 2^n-1. The neighborhood of a solution $s>0$ is simply $N(s,x)=\{s-1\}$; the neigborhood of the 0 solution is empty. The cost $c_L(s,x)$ is 0 if s corresponds to a satisfying truth assignment, and otherwise is the integer s itself. The initial solution $A(x)$ is simply 2^n-1, the truth assignment with all variables true. The goal is to minimize cost. Given these definitions, it is clear that L is a PLS problem. Now observe that, given an instance x of SATISFIABILITY (for which we may assume without loss of generality that 0 is not a satisfying truth assignment), the standard local search algorithm for L will output 0 if and only if x is not satisfiable. Thus the standard algorithm problem for L is NP-hard, and L is the desired member of PLS. ∎

Note that even if finding thge precise local optimum output by the standard algorithm is hard, this does not imply that it is hard to find *some* local optimum. In the above problem, 0 is always a local optimum. Thus the most interesting computational problem associated with a PLS problem L is the following: *Given x, find some locally optimal solution $s\in F_L(x)$* (any one). The proper formal context for studying the complexity of PLS problems is thus not in terms of decision problems or of function computations, but in terms of relations and (in the terminology of [1]) "search problems."

A *search problem* is simply a relation R over $\{0,1\}^*\times\{0,1\}^*$. An algorithm "solves" a search problem R if, when given an $x\in\{0,1\}^*$, it either returns a y such that $(x,y)\in R$ or reports (correctly) that no such y exists. In the case of a PLS problem L, the relation is $R_L=\{(x,y)\colon x\in D_L$ and $y\in F_L(x)$ and is locally optimal$\}$. With a slight abuse of notation, let *PLS* denote the class of all relations R_L arising from PLS-problems.

PLS is related to the search problem analogs of P and NP as follows. P_S and NP_S are both classes of relations $R\subseteq\{0,1\}^*\times\{0,1\}^*$ such that if $(x,y)\in R$, then $|y|$ is polynomially bounded in $|x|$. Such a relation R is in P_S if there is a polynomial-time algorithm that solves it. It is in NP_S if there is a polynomial-time *nondeterministic* algorithm that recognizes $D_R=\{x$: there is a y such that $(x,y)\in R\}$ and is such that every accepting computation outputs a y satisfying $(x,y)\in R$. It is not difficult to show the following:

LEMMA 2. *$P_S=NP_S$ if and only if $P=NP$.*

Note that any $R\in\mathrm{P}_S$ can be formalized as a PLS problem: Let A be a polynomial time algorithm that solves R. If $x\in D_R$, then $F_R(x)$ is defined to be the singleton set consisting of the y returned by A on input x. If A reports that there is no y such that $(x,y)\in R$, then $F_R(x)$ is the singleton set $\{\gamma\}$, where γ is some special symbol. We define $N(y,x)$ to be the empty set in both cases, thus making y a trivial local optimum. Furthermore, any relation in PLS is in NP_S by the existence of the third algorithm in the definition of "PLS problem." Thus we have the following:

LEMMA 3. $P_S \subseteq PLS \subseteq NP_S$.

A second observation is the following, which in light of Lemma 1 strongly suggests that finding *some* local optimum for a PLS problem L is an "easier" task than finding the local optimum that is output by the standard algorithm for L.

LEMMA 4. *If a PLS problem L is NP-hard, then NP = co-NP.*

Proof. If L is NP-hard, then by definition there is an algorithm A for an NP-complete problem X that calls an algorithm for L as a subroutine and takes polynomial time (if the time spent executing the subroutine is ignored). But the existence of such an algorithm implies that we can verify that x is a no-instance for X in nondeterministic polynomial time: simply guess a computation of A on input x, including the inputs and outputs of the calls to the subroutine for L. The validity of the computation of A outside of the subroutines can be checked in polynomial time because A is deterministic; the validity of the subroutine outputs can be verified using the polynomial-time algorithm C_L (whose existence is implied by the fact that L is in PLS) to check whether the output is really a locally optimal solution for the input. Thus the set of "no"-instances of X is in NP, i.e., $X \in$ co-NP. Since X was NP-complete, this implies that NP = co-NP. ∎

(We remark in passing that the above proof can be modified to work for any "total" function in NP_S, not just members of PLS. Technically, we say a relation R in NP_S is *total* if its domain $D_R = \{x$: there is a y such that $(x, y) \in R\}$ is in P. We have already observed that problems in PLS are total in this sense. Note, however, that there are relations in NP_S that do not appear to be total. For example, consider the relation $\{(x, y)$: x is a graph and y is a Hamiltonian cycle in $x\}$. Although this relation is in NP_S, its domain will be in P only if P = NP.)

We say that a problem L in PLS is *PLS-reducible* to another, K, if there are polynomial-time computable functions f and g such that (a) f maps instances x of L to instances $f(x)$ of K, (b) g maps (solution of $f(x)$, x) pairs to solutions of x, and (c) for all instances x of L, if s is a local optimum for instance $f(x)$ of K, then $g(s, x)$ is a local optimum for x. Note that this notion of reduction has the standard desirable properties.

LEMMA 5. *If L, K, and J are problems in PLS such that L PLS-reduces to K and K PLS-reduces to J, then L PLS-reduces to K.*

LEMMA 6. *If L and K are problems in PLS such that L PLS-reduces to K, and if there is a polynomial-time algorithm for finding local optima for K, then there is also a polynomial-time algorithm for finding local optima for L.*

We say that a problem L in PLS is *PLS-complete* if every problem in PLS is PLS-reducible to L. In the next section we prove our first PLS-completeness result.

3. A First PLS-Complete Problem

The circuit computation problem introduced informally in the previous section will be called "FLIP." It can be described in terms of the formal definition of PLS as follows: Instances are interpreted as feedback-free Boolean circuits made up of *and*, *or*, and *not* gates. Given such a circuit x with m inputs and n outputs, a solution in $F_{\text{FLIP}}(x)$ is any bit vector s with m components. It has m neighbors: the m strings of length m with Hamming distance one from s, i.e., the strings that can be obtained from x by changing exactly one bit. Having fixed x, the cost of a solution s is defined as $\sum_{j=1}^{n} 2^j y_j$, where y_j is the jth output of the circuit with input s. Intuitively, this PLS problem asks for an input such that the output cannot be improved lexicographically by flipping a single input bit. To complete the specification of FLIP: algorithm A_{FLIP} returns the all-1 vector, the cost-computation algorithm B_{FLIP} is straightforward from the above, and algorithm C_{FLIP} returns the best of the m neighbors of s (ties broken lexicographically) if s is not locally optimal.

Formally, we view FLIP as a minimization problem. Note, however, that the minimization version of the problem and the corresponding maximization version (MAXFLIP) are equivalent as local search problems. To convert an instance of one to an instance of the other that has the same local optima, simply add an extra level of logic to the circuit x that flips each output variable (changing 1's to 0's and vice versa). This fact will be used in the proof of the following theorem, and also implies that the theorem holds for MAXFLIP as well as FLIP.

THEOREM 1. *FLIP is PLS-complete.*

Proof. Consider a PLS problem L. Without loss of generality, we assume that L is a minimization problem, and show how to PLS-reduce it to FLIP. (If L is a maximization problem, an analogous argument will PLS-reduce L to MAXFLIP, and this reduction composed with the reduction from MAXFLIP to FLIP will yield the desired PLS-reduction from L to FLIP.) Also without loss of generality, we assume that, for each instance x, $F_L(x)$ consists entirely of strings of length $p(x)$, no two of which are within Hamming distance 1 of each other. We first PLS-reduce L to an intermediate PLS problem M that differs from L only in the neighborhood structure: in M no solution has more than one neighbor. If s is locally optimal for x then $N_M(s, x)$ is empty; otherwise the single neighbor of s is the output of C_L given inputs s and x. Note that we can take $A_M = A_L$, $B_M = B_L$, and $C_M = C_L$.

We next PLS-reduce M to a second intermediate problem Q that has the same instances as L and M, but has the same neighborhood structure as FLIP, i.e., all strings of a given polynomially bounded length are solutions and any two strings at Hamming distance 1 are mutual neighbors. Suppose that the stipulated length of solutions for instance x of L (and hence of M) is $p = p(|x|)$. Then solutions for x in Q are of length $2p+2$. Although all such strings will be called "solutions," only

certain specified ones will be inexpensive enough to be candidates for local optima. For a solution u of L, the possible candidates are as follows:

(a) $uu11$, in which case $c_Q(uu11, x) = (2p+4)\, c_L(u, x) + 2$.

(b) $uu10$, in which case $c_Q(uu10, x) = (2p+4)\, c_L(u, x) + 1$.

(c) $uu00$, in which case $c_Q(uu00, x) = (2p+4)\, c_L(u, x)$.

(d) $uv00$, where u is not a local optimum and v is a string on the shortest Hamming path from u to its (single) neighbor w in M. The cost $c_Q(uv00, x)$ is $(2p+4)\, c_L(w, x) + (p+2) + h + 2$, where h is the Hamming distance between v and w ($h = 0$ is allowed).

(e) $uw10$, where w is as above and $c_Q(uw10, x) = (2p+4)\, c_L(w, x) + (p+2) + 1$.

(f) $uw11$, where w is as above and $c_Q(uw11, x) = (2p+4)\, c_L(w, x) + (h+2)$.

(g) $vu11$, where v is any string of length p other than u. The cost $c_Q(vu11, x)$ is $(2p+4)\, c_L(u, x) + h + 2$, where h is the Hamming distance between v and u.

Each "noncandidate" s will have its cost determined as follows. Recall that since L is in PLS, there is a polynomial-time algorithm B_L for computing $c_L(s, x)$. Let $q(|x|)$ be a polynomial bound on the running time of B_L in terms of $|x|$ (since $|s|$ is polynomially bounded in $|x|$, such a q must exist). Then all solutions s for x satisfy $c_L(s, x) \leqslant 2^{q(|x|)}$. Let $Z = (4p+4)(2^{q(|x|)})$, and let a be the standard solution returned by algorithm A_L. If s is a noncandidate solution and h is the Hamming distance between s and $aa11$, then $c_Q(s, x) = Z + h$. Thus there is a downhill path from any noncandidate solution to $aa11$, and so only candidate solutions can be locally optimal. Furthermore, (a) through (g) have been designed so that once a candidate solution has been reached, a local optimization algorithm based on the flip neighborhood must simulate the standard algorithm for M, with the set of local optima being precisely those solutions $uu00$ for which u was locally optimal for L. (Readers may verify this for themselves, using the fact that all solutions of L are at least Hamming distance 2 apart by assumption.) (This one-to-one correspondence between local optima is actually a stronger property than we need for a PLS-reduction from M to Q, but will certainly suffice, and has additional consequences; see Corollary 1.1.) To complete the definition of Q, let algorithm A_Q return the "standard" solution 1^{2p+2}, and observe that a polynomial-time algorithm B_Q can be derived in a straightforward manner from the polynomial-time algorithms B_M and C_M, and that a polynomial-time C_Q is a straightforward adaption of B_Q.

The last step of our proof is a PLS-reduction from Q to FLIP. This is now relatively straightforward, as all the complexity of Q resides in computing its cost function. Our reduction works by constructing, for a given instance x of Q, a polynomial-size circuit with $2p(|x|) + 2$ inputs that computes $c_Q(s, x)$ for all solutions s of x, with its outputs describing the answer in binary notation. This can be done since the algorithm B_Q for computing costs runs in polynomial-time (by construction). ∎

COROLLARY 1.1. (a) *The standard algorithm problem for FLIP is NP-hard, and* (b) *There are instances of FLIP for which the standard algorithm requires exponential time.*

Proof. These follow from Lemma 1, the one-to-one correspondence between local optima implicit in our proof of Theorem 1, and the fact that our construction in the proof of Lemma 1 forces the standard algorithm for FLIP to simulate the standard algorithm for the problem being reduced to FLIP, provided that the starting solution for the FLIP standard algorithm, 1^{2p+2}, is precisely *aa*11, where *a* is the starting solution for the problem being reduced to FLIP. (Note that this is the case for the problem L of Lemma 1.) ∎

4. A WELL-KNOWN LOCAL SEARCH PROBLEM THAT IS PLS-COMPLETE

The *local optimum for Kernighan–Lin* (*LOKL*) problem is based on a famous local search heuristic for the well-studied *graph partitioning* problem. In the graph partitioning problem, we are given a graph $G=(V, E)$ with weights $w(e)$ on the edges. A solution is any partition of V into two equal subsets A and B, and the cost $c(A, B)$ is the sum of the weights of all edges going from A to B. Our goal is to find a partition of minimum cost.

The LOKL neighborhood structure for this problem is highly data-dependent, and its definition is built up as follows. A *swap* of partition (A, B) is a partition (C, D) such that A and C have symmetric difference 2, i.e., such (C, D) is obtained from (A, B) by swapping one element of A with one element of B. (C, D) is a *greedy* swap if $c(C, D)-c(A, B)$ is minimized over all swaps of (A, B). If in fact (C, D) is the lexicographically smallest over all greedy swaps, we say that (C, D) is the lexicographic greedy swap of (A, B). Let (A_i, B_i) be a sequence of partitions, each of which is a swap of the one preceding it. We call it *monotonic* if the differences A_i-A_0 and B_i-B_0 are monotonically increasing (that is, no vertex is switched back to its original set). Finally, we say that a partition (C, D) is a neighbor of (A, B) in the LOKL neighborhood structure if it occurs in the (unique) maximal monotonic sequence of lexicographic greedy swaps starting with (A, B). Note that such a sequence will consist of $|V|/2+1$ partitions, with the last one equalling (B, A). Thus each partition has $|V|/2$ neighbors. The Kernighan–Lin algorithm performs local search over this neighborhood structure, with the added proviso that if more than one neighbor offers an improvement over the current partition, we must choose a neighbor that offers the *best* improvement.

(The above description is not enough to allow us to specify the output of algorithms A_{LOKL} and C_{LOKL}. The original paper [4] does not specify a unique starting partition, nor does it say what is to be done when there is more than one neighbor offering the best improvement. In consequence, various implementations differ on these points. The paper also fails to specify the underlying neighborhood structure as precisely as we do here; it does not say what to do when ties are

encountered in the process of building a maximal monotone sequence of swaps, whereas we have specified a lexicographic tie-breaking rule. Our PLS-completeness result will in fact be independent of the choice of tie-breaking rules and the precise definitions of A_{LOKL} and C_{LOKL}.)

As an intermediary in our proof, we consider a variant on LOKL with a considerably larger neighborhood structure, called the *weak* LOKL problem, or WLOKL. WLOKL is like LOKL, except that a partition (C, D) is a neighbor of (A, B) if it can be obtained from it by a monotonic sequence of greedy swaps, all except the first of which are lexicographic. In other words, we exhaust all ties at the first step, although we resolve ties lexicographically from then on. Since there can be at most $O(|V|^2)$ swaps tied at the first step, each partition has at most $O(|V|^3)$ neighbors in the WLOKL neighborhood structure. (Note that all the LOKL neighbors are contained in the WLOKL neighborhood, and so if a partition is locally optimal with respect to WLOKL, it must also be locally optimal with respect to LOKL.)

THEOREM 2. *WLOKL is PLS-complete.*

Proof. As with FLIP, the maximization and minimization versions of WLOKL are equivalent (the same also holds for LOKL). To transform an instance $G=(V, E)$, w of one version to an instance $G=(V, E')$, w' of the other with the same local optima, simply replace E by $E'=\{(u, v): u, v \in V \text{ and } u \neq v\}$, extend w to the larger set by defining $w(e)=0$ for all $e \in E'-E$, and then define $w'(e)=W-w(e)$ for all $e \in E'$, where $W=\sum_{e \in E} w(e)$. We prove that WLOKL is PLS-complete by PLS-reducing MAXFLIP to the maximization version of WLOKL (using maximization versions simplifies our notation).

Let the circuit C be an instance of MAXFLIP. We will show how to construct a corresponding instance of the maximization version of WLOKL. Let N be the number of *and* and *or* gates contained in C (as we shall see, we can ignore the *not* gates). We assume without loss of generality that $N>6$.

The first step of our construction is designed to get around the fact that, while the sets A and B are interchangeable in the definition of graph partitioning (i.e., $c(A, B)=c(B, A)$), the truth values in MAXFLIP are not. (If y is the input vector to circuit C obtained from x by changing each "true" variable to "false" and vice versa, then it need not be the case that $c_{\text{MAXFLIP}}(y)$ equals $c_{\text{MAXFLIP}}(x)$.) To "symmetrize" MAXFLIP, we represent the circuit C by an instance of NOT-ALL-EQUAL-SATISFIABILITY (NAES), which does display a symmetry between "true" and "false." As with the standard SATISFIABILITY problem, instances of NAES are defined over a set U of *variables*, each instance being a collection X of clauses, each clause being a subset of the set $\{u, \bar{u}: u \in U\}$ of literals over U. The definition of "satisfying truth assignment" is different from that for ordinary SATISFIABILITY, however. A truth assignment to the variables *satisfies* an instance X if and only if every clause has at least one true and one false literal. (Thus interchanging the values of true and false in an NAES-satisfying truth

assignment yields another NAES-satisfying truth assignment.) Given our circuit C, we shall construct an NAES instance X_C whose satisfying truth assignments correspond to the valid computations of C.

In addition to the variables x_i, $1 \leqslant i \leqslant m$, for the inputs to the circuit C, we will have variables z_i, $1 \leqslant i \leqslant m$, for the implicit bottom level input "gates" of the circuit that take the input variables and propagate them upwards. Rounding out our set of variables, we will also have a variable a_j for the output of each *and* and *or* gate in the circuit. (We do not need variables for the *not* gates, since if a is the variable associated with one of the inputs to a *not* gate of C, we can simply use $\bar{a}$ for the output of the *not* gate, noting that $\bar{\bar{a}} = a$.) The NAES clauses are specified as follows:

(i) For each bottom level gate, we will have two NAES clauses: $(x_i, \bar{z}_i)$ and $(\bar{x}_i, z_i)$, $1 \leqslant i \leqslant m$.

(ii) For each *or* gate "$a = b$ *or* c" (with a the variable for the gate, and b and c the literals for its inputs), we have three NAES clauses: $(\bar{a}, b, c)$, $(a, \bar{b}, 0)$, and $(a, \bar{c}, 0)$.

(iii) For each *and* gate "$a = b$ and c," we have three NAES clauses: $(a, \bar{b}, \bar{c})$, $(\bar{a}, b, 0)$, and $(\bar{a}, c, 0)$.

It is easy to verify that a truth assignment of the variables gives consistent values to the input(s) and output of a gate if and only if it satisfies the corresponding NAES clauses, and hence the desired correspondence holds between satisfying truth assignments and valid computations of C.

We now construct a weighted graph $G = (V, E)$ corresponding to X_C and hence to C. In specifying G, we will make reference to the *levels* of various circuit elements (and their corresponding NAES clauses). These are defined as follows. The bottom level gates (the z_i) will be assigned level 1. Thereafter, the level of an *and* or an *or* gate is one more than the level of its highest level input, and the level of a *not* gate (for which there are no corresponding clauses) is the *same* as the level of its input. The maximal level attained by any gate will be called the *depth* of the circuit and will be denoted by d. Let $M = N^{2N}$.

The vertices in V are specified as follows: First, there are vertices T_0, T_1, F_0, and F_1 (T for "true," F for "false"). Then, for each input variable x_i we have two vertices, x_i and $\bar{x}_i$. Finally, for each "gate" variable g (where g is either an a_i or a z_i), we have four vertices, g, $\bar{g}$, g', and $\bar{g}'$. (The extra "primed" vertices are introduced so that each pair of gate vertices $\{g, \bar{g}\}$ will initially be more expensive to swap than the $\{x, \bar{x}\}$ input pairs.)

The edges in E are divided into two classes. First, there are the edges in the non-problem-specific superstructure: Connecting each $\{T_i, F_j\}$ pair is an edge of weight M^2. Connecting each remaining pair of complementary vertices (x_i and $\bar{x}_i$, g and $\bar{g}$, g' and $\bar{g}'$) is an edge of weight M. Connecting each pair $\{g, \bar{g}'\}$ and $\{\bar{g}, g'\}$ is an edge of weight $N^{2(d-1)} + (1/M^2)$.

Next, we have edges that reflect the precise contents of the NAES clauses: For

each clause of level i, $1 \leqslant i \leqslant d$, we connect the unprimed vertices corresponding to each pair of literals in that clause by an edge of weight $N^{2(d-i)}$. (If the constant "0" appears in the clause, the "literal" representing it in the above specification is the vertex F_1. If a given pair of literals is contained in more than one clause, the weight of the edge joining them is the sum of the weights arising from all clauses that contain them both.) Finally, we connect the variable representing the jth output gate of C to F_1 with an edge of weight $2^j/M$.

This concludes the construction of G. Note that it can be accomplished in polynomial time. Thus, to complete the proof that WLOKL is PLS-complete, we must show that there is a polynomial time algorithm that can construct a local optimum for C as an instance of MAXFLIP, given a local optimum for G as an instance of WLOKL. The existence of such an algorithm will be immediate once we show that a locally optimal partition (A, B) for G (with respect to the WLOKL neighborhood) has the following properties:

(1) The vertices T_0 and T_1 are in the same set, say A, and the vertices F_0 and F_1 are in the other set, B. Also, the vertices x_i and $\bar{x}_i$ are in opposite sets, and, for each gate variable g, the vertices g, g' are in the same set and the vertices $\bar{g}$, $\bar{g}'$ are in the opposite set. Thus the partition induces a truth assignment for all the variables: g (or x_i) is true if and only if g (x_i) is in A.

(2) The truth assignment induced by the partition satisfies the NAES instance X_C and hence is consistent with the circuit C.

(3) The values of the input variables form a local optimum for the MAXFLIP instance.

We shall prove these properties in turn, starting with property (1). The proof of the theorem will be complete once we have proved that property (3) holds. We shall in fact show that (1) and (2) hold for the LOKL neighborhood, which implies that they hold for WLOKL (and which will prove useful in our eventual proof that LOKL is PLS-complete). Only property (3) will require the full generality of WLOKL.

First we make some observations concerning the weights of the edges. Consider a vertex u corresponding to a gate variable g or its negation $\bar{g}$, and an adjacent vertex v other than the complementary vertex $\bar{u}$ or its mate $\bar{u}'$. That is, v either is F_1 or corresponds to another variable. If v corresponds to an input variable x_i, then u corresponds to the level 1 gate variable z_i, and the weight of the edge is N^{2d-2}. If v corresponds to another gate variable or is F_1, then the literals u and v can coexist in at most N clauses, all of level 2 or greater, so the weight of the edge $\{u, v\}$ is at most $N \cdot N^{2(d-2)} < N^{2d-2}$.

We now show that (1) holds if the partition (A, B) is locally optimal with respect to the LOKL neighborhood. First, suppose that T_i and F_j are in the same set, say A, for some i and j. If T_{1-i} and F_{1-j} are in the other set B, then swapping F_j and T_{1-i} will add the edges $\{T_i, F_j\}$ and $\{T_{1-i}, F_{1-j}\}$ to the partition, for a cumulative gain of $2M^2$. Since each of the edges we might lose from the partition (at most $4N$

of them), has weight less than M, this swap (and hence a greedy swap) will improve the partition, contradicting the assumption of local optimality. If $T_{1,i}$ is in set A along with T_i and F_j, then swapping F_j with any member of B other than $F_{1,j}$ will gain the edges $\{T_0, F_j\}$ and $\{T_1, F_j\}$, again for a local gain of $2M^2$ and a global net profit. An analogous argument holds if A contains $F_{1,j}$ as well as T_i and F_j. We thus conclude that T_i and F_j cannot be in the same set for any i, j, and hence T_0 and T_1 are in one set, say A, and F_0 and F_1 are in the other set B.

Suppose now that two complementary vertices u and $\bar{u}$ are in the same set, say A; u is g or g' for some gate variable g, or x_i for some i, $1 \leqslant i \leqslant m$. Since A and B have equal size, there are two complementary vertices v and $\bar{v}$ in B. If we swap u and v we will gain the edges $\{u, \bar{u}\}$ and $\{v, \bar{v}\}$, each of weight M, and lose only at most $4N$ edges of weight less than N^{2d}. Since $(4N)(N^{2d}) = 4N^{2d+1} < M$, this means that (A, B) would not be locally optimal, a contradiction. Thus, all complementary vertices are separated. Suppose finally that g and g' are in different sets for some gate variable g, say $g \in A$ and $g' \in B$. Then $\bar{g}$ is in B and $\bar{g}'$ is in A. If we swap g' and $\bar{g}'$, we gain the edges $\{g, \bar{g}'\}$ and $\{\bar{g}, g'\}$, each of weight exceeding $N^{2(d-1)}$, and do not lose any edges, again an improvement and a contradiction. It follows that for every gate variable g, vertices g and g' are in the same set and vertices $\bar{g}$ and $\bar{g}'$ are in the other set. This establishes property (1) for LOKL and hence for WLOKL.

We now show that property (2) holds for any locally optimal partition (A, B), i.e., that the truth assignment induced by the partition satisfies the NAES instance X_C. Suppose that the induced truth assignment is not satisfying. We first observe that all the level 1 clauses must be satisfied. If not, then there would be some pair $x_i, \bar{z}_i$ on the same side of the partition, with $\bar{x}_i$ and z_i on the other. But then we could swap x_i and $\bar{x}_i$ (flip the truth value of x_i), gaining edges $\{x_i, \bar{z}_i\}$ and $\{\bar{x}_i, z_i\}$ of weight $N^{2(d-1)}$ each while losing no edges, implying that (A, B) is not locally optimal. Thus, any unsatisfied clause must be at level 2 or above. We will argue that a greedy swap from (A, B) must swap vertices g and $\bar{g}$ for some violated gate g at the lowest possible level, and that once this swap is made, swapping g' and $\bar{g}'$ will lead to a partition better than (A, B), once again violating local optimality of the definition of the LOKL neigborhood.

Consider possibilities for a greedy first swap. If we swap any two complementary vertices u and $\bar{u}$, we will incur a loss of at most $2(N^{2(d-1)} + (1/M^2)) + 4N(N^{2(d-1)}) < M$; the first term is from the loss of edges $\{u, \bar{u}'\}$ and $\{\bar{u}, u'\}$ and the second is from all other lost edges. On the other hand, if we swap two noncomplementary vertices, then, by the arguments in support of (1), the net loss will be at least M. Thus any greedy swap from (A, B) must involve complementary vertices.

Consider the clauses at levels $i \geqslant 2$. Note that each of these corresponds to an *and* or *or* gate and hence contains 3 literals. If such a clause is satisfied (i.e., has at least one true and one false literal), then the triangle of the clause contributes two edges of weight $N^{2(d-i)}$ to the weight of the partition. If the clause is not satisfied, then all three vertices are in the same set and the clause contributes nothing. Let us say that an *and* or *or* gate is "satisfied" if all three of the corresponding clauses are satisfied, and is "violated" otherwise. By the above arguments, our assumption that the truth

assignment induced by (A, B) is not satisfying means that there is at least one violated gate, and all violated gates have level $i \geq 2$. Note that a gate of level $i \geq 2$ that is satisfied contributes weight $6N^{2(d-i)}$, whereas a violated gate contributes at most $4N^{2(d-i)}$.

Suppose g is the variable for a violated gate at level $i \geq 2$. If we flip the truth value of g (swap g and $\bar{g}$), we gain at least 2 (and at most 6) edges of weight $N^{2(d-i)}$, as each clause for that gate contains either g or $\bar{g}$. We lose the edges $\{g, \bar{g}'\}$ and $\{\bar{g}, g'\}$, each with weight $N^{2(d-1)} + (1/M^2)$. In addition, flipping the value of g may result in the violation or satisfaction of other gates at higher levels (those with g or $\bar{g}$ as input). The loss or gain due to these new violations is at most $6N \cdot N^{2(d-(i+1))} < N^{2(d-i)}$, given our assumption that $N > 6$. (If g is an output gate, there is a possible extra loss or gain of $2^N/M < 1$ for the edge $\{g, F_1\}$, but this is negligible.) Thus flipping the value of a violated gate at level $i \geq 2$ results in a loss L satisfying

$$L_i^{LB} = 2N^{2(d-1)} + \frac{2}{M^2} - 7N^{2(d-i)} < L < 2N^{2(d-1)} + \frac{2}{M^2} - N^{2(d-i)} = L_i^{UB}.$$

Note that if $h > i$, then $L_h^{LB} - L_i^{UB} \geq N^{2(d-i)} - 7N^{2(d-h)}$, which is greater than 0 since $N^2 > 7$ when $N > 6$. Thus the smallest loss that can be incurred by flipping a violated gate's variable occurs for a violated gate of minimum level.

This loss is also less than the loss incurred by flipping a variable corresponding to a satisfied gate at any level i, for the latter loss will be at least $2N^{2(d-1)} + (2/M^2) + 2N^{2(d-i)} - N^{2(d-i)}$, which exceeds L_j^{UB} for all $j \geq 2$. Similarly, all the L_i^{UB} are less than the cost of flipping an input variable x_i ($2N^{2(d-1)}$ due to the loss of the two edges $\{x_i, \bar{z}_i\}$ and $\{\bar{x}_i, z_i\}$) and are less than the cost of flipping vertices g' and $\bar{g}'$ for some g, (because such a flip incurs a loss of at least $2N^{2(d-1)} + (2/M^2)$). Therefore, the first greedy swap (in the sequence of swaps generating LOKL neighbors of (A, B)) must swap two vertices g and $\bar{g}$, where g is a violated gate at the lowest possible level k, for a loss of at most L_k^{UB}.

If, after swapping such an $g, \bar{g}$ pair, we next swap the corresponding vertices g' and $\bar{g}'$, we will regain the edges $\{g, \bar{g}'\}$ and $\{\bar{g}, g'\}$ for an increase of $2N^{2(d-1)} + (2/M^2)$ and a net total gain of at least $N^{2(d-k)}$. This means that (A, B) has a neighbor of higher weight, a contradiction of its local optimality. Thus property (2) is established for LOKL (and hence for WLOKL).

Call a partition that satisfies properties (1) and (2) a *consistent* partition. To prove property (3), we shall show first that the weight of a consistent partition is equal to a constant term, depending only on the circuit C, plus $1/M$ times the cost of the solution to the MAXFLIP instance induced by that partition. The following edges contribute to the weight of a consistent partition: edges between F_i and T_j vertices with a total weight of $4M^2$; edges between the $2N + 3m$ pairs of complementary variables (two for each *and* and *or* gate plus three for each input variable) with a total weight of $(2N + 3m)M$; edges $\{g, \bar{g}'\}$ and $\{\bar{g}, g'\}$ with a total

weight of $2(N+m)(N^{2(d-1)}+(1/M^2))$; edges connecting vertices in the same NAES clause with a total weight of $2mN^{2(d-1)}+\sum_{i=2}^{d} 6c_i(N^{2(d-i)})$, where c_i is the number of *and* and *or* gates in C at level i, $2 \leqslant i \leqslant d$. All these weights sum up to a constant independent of the partition. In addition we have the edges connecting the true output variables to F_1. The total weight of these edges is the cost of the associated solution to the MAXFLIP instance, divided by M.

Suppose now that the solution to MAXFLIP instance induced by the locally optimal partition (A, B) is not a local optimum, i.e., flipping the value of some input variable x_k to the circuit C yields an output of higher value. We shall again derive a contradiction. By (1) and (2), we know that (A, B) is a consistent partition. What are its neighbors under the WLOKL neighborhood? We first argue that any greedy swap from (A, B) must be between an input variable and its complement, and in fact all such swaps tie for best. The loss for such a swap is $2N^{2(d-1)}$, independently of which input variable is involved. By the argument for property (2), the only competitors are swaps between complementary vertices. However, by the same argument (and the fact that all gates are satisfied by a consistent partition), any such swap must cost at least $2N^{2(d-1)}+(2/M^2)$.

Thus the pair $x_k, \bar{x}_k$ is eligible to be the first swap in a sequence of greedy swaps leading to WLOKL neighbors of (A, B). We now argue that subsequent swaps in that sequence will lead to a new partition consistent with the new input assignment obtained by flipping the truth value of x_k. If flipping x_k improves the solution cost in the MAXFLIP instance, the new partition will have better cost itself.

After flipping the value of x_k, the gate $z_k = x_k$ becomes violated. If we flip z_k (i.e., swap z_k and $\bar{z}_k$), we incur an additional loss of at most $2N^{2(d-1)}+(2/M^2)-N^{2(d-1)}$ (by analogy with the arguments for levels $i>1$ in the proof of property (2)). Since all other gates are satisfied, swapping g and $\bar{g}$ (or g' and $\bar{g}'$) for any other gate will result in a larger loss. Also, swapping x_i and $\bar{x}_i$ for any $i \neq k$ will result in a loss of $2N^{2(d-1)}$, again larger than the loss for flipping z_k. The only better swap is to flip back x_i and $\bar{x}_i$, but this is not allowed by the WLOKL definition of neighborhood. Thus, in the second step of our swap sequence we will swap z_k and $\bar{z}_k$. In the third step the best choice is to swap z'_k and $\bar{z}'_k$ for a gain of $2N^{2(d-1)}+(2/M^2)$; any other swap incurs a loss.

At this stage, all gates at level 1 have consistent values. From now on, as long as the partition is inconsistent but all level 1 gates are satisfied, the next greedy swap will involve a variable a for a violated gate at the lowest possible level: first we will have to swap a and $\bar{a}$ and then we will have to swap a' and $\bar{a}'$. By the proof of property (2), such a swap will be best as far as cost is concerned. We only have to show in addition that the vertices are eligible for swapping, i.e., have not already been swapped once earlier in the sequence of swaps. This follows from the fact that violations only propagate upwards, and hence the lowest level of a violated gate is monotonically increasing. Thus the sequence of greedy swaps will continue until a partition is arrived at that is consistent with the new assignment to the input variables, and hence has better cost than (A, B). This contradicts the local optimality of (A, B), and hence proves that property (3) holds. ▮

We shall now extend our result for WLOKL to the LOKL neighborhood structure.

THEOREM 3. *LOKL is PLS-complete.*

Proof. In the previous construction, if we had a consistent partition then there was an m-way tie for the best first swap: flipping each of the m input variables. If the partition did not correspond to a local optimum for the corresponding MAXFLIP instance, then one of these m swaps led to an improved partition. We shall modify our construction so that there is no tie, i.e., so that there is a unique best first swap, one that flips an input variable x_k that improves the cost in the MAXFLIP instance. We do this by simulating a more elaborate circuit than C.

Our circuit K will have m inputs ($x_1, ..., x_m$, as in C), and $n+m$ outputs (the cost outputs $y_1, ..., y_n$ of C plus m additional outputs $w_1, ..., w_m$). It is constructed as follows: K contains $m+1$ copies of C; the ith copy computes the cost of the input vector obtained from $(x_1, ..., x_m)$ by flipping x_i, and the $m+1$st copy computes the cost of $(x_1, ..., x_m)$ itself. This last value goes to the outputs $y_1, ..., y_n$. However, on top of the $m+1$ copies of C is another circuit which compares the costs found by each and writes the input vector that gave the best value to the outputs $w_1, ..., w_m$. Ties are resolved lexicographically with the input vector $(x_1, ..., x_m)$ preceding all its neighbors. Hence the outputs $w_1, ..., w_m$ contain $(x_1, ..., x_m)$ if and only if it was locally optimal, and otherwise they contain the input variable values for a neighbor with better cost.

As described, K can clearly be constructed in polynomial time, given C. Given the circuit K, we construct a graph G as in the previous theorem. The only difference is that the new output variables $w_1, ..., w_m$ are not connected to F_1. Rather, for $1 \leqslant i \leqslant m$, we have "tie-breaking" edges $\{w_i, \bar{x}_i\}$ and $\{\bar{w}_i, x_i\}$ with weight $1/M^3$ each. This weight is too small to play any role in almost all the arguments. Thus, as before, a locally optimal partition (with respect to the LOKL neighborhood) induces a consistent truth assignment. (Recall that properties (1) and (2) in the previous proof were shown to hold for LOKL as well as WLOKL.)

Regarding property (3), the cost of a consistent partition is some constant, plus $1/M$ times the cost of the solution to the associated solution for MAXFLIP, plus at most $2m/M^3$ for tie-breaking edges. The last term clearly will not effect the relative rankings of two consistent partitions unless both have the same value for the first two terms. The only place where the tie-breaking edges come into play is in breaking a tie for the first swap in the definition of the LOKL neighborhood. Suppose (A, B) is a consistent partition, and let x be the associated truth assignment to the inputs of MAXFLIP. We shall show that if (A, B) is locally optimal for LOKL, then x is locally optimal for MAXFLIP.

Suppose that x is not locally optimal for the MAXFLIP instance. Then there is a single k such that $w_k \neq x_k$. By the same arguments used in the proof of the previous theorem, the first swap in the monotonic sequence of lexicographic swaps that determines the LOKL neighbors of (A, B) must involve x_i and $\bar{x}_i$ for some i. If

$i \neq k$, we incur a loss of $2N^{2(d-1)} + (2/M^3)$; the first term comes from the edges connecting x_i, $\bar{x}_i$ to z_i, $\bar{z}_i$, and the second comes from the edges to w_i, $\bar{w}_i$. On the other hand, since $w_k \neq x_k$, if we swap x_k and $\bar{x}_k$ we will lose only $2N^{2(d-1)} - (2/M^3)$. Thus, if x is not locally optimal for MAXFLIP, the first swap will flip an input variable that yields a better solution x' for MAXFLIP. Then, as in the previous proof, the sequence of swaps will progressively rectify the values of the violated gates until we reach the consistent partition associated with x', and this partition will have better cost than that for x, implying that (A, B) was not locally optimal, a contradiction.

Thus if (A, B) is locally optimal for our graph partitioning instance with respect to the LOKL neighborhood, its associated x is locally optimal for the original MAXFLIP instance. Hence our derivation of G from the MAXFLIP instance provides the required PLS-reduction, and LOKL is PLS-complete. ▮

Note that the proof of PLS-completeness does not depend on the tie-breaking rule for greedy swaps (the lexicographic tie-breaking rule we specified for LOKL is not used in our proof since we manage to avoid all ties). Also, completeness does not depend on our method for choosing the neighbor to move to when there is more than one better neighbor (another detail left to particular implementations of the Kernighan–Lin algorithm). It also extends to any generalization of LOKL to larger neighborhoods, so long as the neighborhood of a partition contains all LOKL neighbors. (A natural example would be the generalization in which a partition is a neighbor of (A, B) if it can be reached from (A, B) by a sequence of swaps, all but the *first* of which are greedy.)

We remark that although a locally optimal partition induces a locally optimal solution to the FLIP instance being simulated in our construction, the converse is not true. That is, it is possible that an input vector x is locally optimal for the FLIP instance but the corresponding partition is not locally optimal. The reason is the following: Starting from the consistent partition (A, B) corresponding to x, the Kernighan–Lin algorithm will flip some input variable x_i (swap x_i and $\bar{x}_i$), depending on the tie-breaking rule in use. (Since x is locally optimal, we have $w_i = x_i$ for all i, and so all such flips generate the same loss.) The algorithm will then propagate the new values through the circuit until the consistent partition corresponding to the new input x' is reached. Since x is locally optimal, this new partition will be no better than the one for x. However, the algorithm does not stop here; it must continue until every vertex has been swapped exactly once. Thus, the algorithm will next flip some other input variable x_j, and then try to propagate the new values up the circuit. The propagation will not succeed if it encounters some violated gate variable that was already flipped in the first propagation. If this happens, then the partition will contain a violated gate forever after (or rather, until the T_i and F_i are swapped on the last two steps, returning us to the reversal (B, A) of our original partition). The presence of a violated gate will cause the cost to exceed that for any consistent partition, and so (A, B) will be locally optimal so long as the second propagation fails. That second propagation might succeed,

however, in which case the Kernighan–Lin algorithm will construct a consistent partition corresponding to an input vector that differs from x in two positions, and that hence is not a neighbor of x in MAXFLIP. Such a solution could have better cost than x even though x is locally optimal.

The above observation does not effect the validity of our proof of Theorem 2, since for a PLS-reduction between PLS problems we only need to have a mapping from local optima of the target problem to local optima of the source. Howver, our construction can be modified so that the correspondence is one-to-one, thus allowing us to prove that an analog of Corollary 1.1 holds for LOKL. The modification consists of augmenting the circuit K with an additional circuit that computes the parity of the input vector x, i.e., determines whether x has an even or odd number of 1's. When Kernighan–Lin flips an input variable, the output of this circuit will also flip. When Kernighan–Lin flips a second input variable, the output of the circuit will want to flip back, but this would involve swapping some vertex a second time, which is not allowed. Thus the second input flip can no longer be successfully propagated, and it follows that with this modification in our construction there *is* a one–one correspondence between local optima. Moreover, this modified construction forces the Kernighan–Lin algorithm to simulate the standard algorithm for MAXFLIP, provided we are given an appropriate starting partition (with all the x_i on the "True" side of the partition). Consequently we can conclude the following.

COROLLARY 3.1. (a) *It is NP-hard to find the solution constructed by the Kernighan–Lin algorithm from a given starting point, and* (b) *there are instances of (weighted) graph partitioning with starting partitions such that the Kernighan–Lin algorithm will take exponential time if started from those partitions.*

5. FURTHER RESULTS AND OPEN PROBLEMS

We have shown that one well-known local search algorithm, the Kernighan–Lin algorithm for graph partitioning, gives rise to a PLS-complete local search problem. One can naturally ask if such results can be proved for other algorithms and other optimization problems.

Such a result can, for instance, be proved for a "Kernighan–Lin-like" algorithm for the weighted independent set problem. Instances of this problem are graphs $G=(V, E)$ with weights assigned to the vertices (rather than to the edges as in graph partitioning). A solution is an independent set, i.e., a set $V' \subseteq V$ such that no two vertices in V' are joined by an edge in E. The cost of a solution V' is the sum of the weights of the vertices in V'. We wish to find an independent set of maximum weight.

Our local search algorithm, which we shall call "K-L," is based on a slightly different concept of "swap" than that used for graph partitioning by Kernighan and Lin. Here a swap involves taking a vertex v not in V', placing v in V', and then

removing from V' all vertices that are adjacent to v in G. By analogy with the Kernighan–Lin algorithm, our algorithm considers neighborhoods based on sequences of swaps. Call a swap *greedy* if it gives rise to the biggest possible increase in cost (or least possible loss, if no increase is possible). A *monotonic* sequence of greedy swaps will be one in which no vertex that is swapped into V' during the sequence is ever swapped out, with ties broken, say, lexicographically. The neighborhood used in the K-L algorithm consists of all those independent sets reachable by monotonic sequences of greedy swaps. Using techniques analogous to those used for proving Theorems 2 and 3, we can show that the local search problem for this neighborhood is PLS-complete.

Unfortunately, it is not clear that this result has much relevance in practice, because it is not clear that the K-L algorithm for a weighted independent set finds very good solutions (no one has yet investigated its performance in practice). What we would most like to prove is that the Lin–Kernighan λ-change algorithm for the traveling salesman problem is PLS-complete. There is ample evidence that this algorithm produces *very* good solutions [3, 7], so there is significant interest in understanding the complexity of its neighborhood structure. Moreover, like the Kernighan–Lin algorithm for graph partitioning (which, in fact, inspired it), the Lin–Kernighan algorithm has a neighborhood structure based on sequences of "greedy swaps," although here the definition of *swap* is of course different. Unfortunately, we at present see no way of extending our techniques to this problem.

Another open problem is the complexity of local search problems with simpler neighborhood structures involving a bounded number of "swaps," such as the 2-change and 3-change neighborhood structures for the TSP, the simple single-swap neighborhood for graph partitioning and independent set, and the problem of subdeterminant maximization mentioned in the Introduction. For only one of these five examples is it known that local optima can be found in polynomial time. (An $O(|V|^2)$ greedy algorithm is sufficient to construct an independent set that is locally optimal with respect to single swaps.) On the other hand, it is unlikely that any of the four remaining problems can be PLS-complete.

This is because for these problems the algorithms A_L, B_L, and C_L all run in logarithmic space, and thus do not appear to make full use of polynomial time local search. We conjecture that a local search problem cannot be PLS-complete unless the subproblem of verifying local optimality is itself logspace complete for P. Although this does not seem to follow directly from our definitions, we can at least show that both FLIP and LOKL have P-complete local optimality verification problems ("verification problems," for short).

The result for FLIP follows from the fact that it is P-complete to determine whether the output of a single-output monotone Boolean circuit is 1 or 0 [2]. (Recall that in a monotone circuit there are no *not* gates.) We actually give a transformation to MAXFLIP, but that will imply the result for FLIP as well. Suppose we are given a monotone circuit C with input values $x_1, \ldots, x_n$ as an instance of the monotone circuit value problem. Augment that circuit with an additional "latch" input that is *and*'d with each input before that input is fed to the later gates of C,

and set the latch input to 0. Note that this modified circuit C' can be constructed using logspace given C, and that the output value for C' must be 0. Further note that the output value will continue to be 0 as long as the latch input is not flipped. Thus the input $x_1, ..., x_n, 0$ will be locally optimal if and only if the output is 0 when the latch input is 1. But this will be true if and only if the output of C on inputs $x_1, ..., x_n$ is 0. Thus we have reduced the monotone circuit value problem to the local optimality verification problem for MAXFLIP, and the latter is complete for P.

To see that a similar result holds for LOKL, we look at the proof of Theorem 2 for WLOKL. The construction of the graph corresponding to the augmented circuit C' can be accomplished in logspace, given C'. Moreover, the consistent partition corresponding to the input $x_1, ..., x_n, 0$ can also be constructed in logspace. (Note that in general the construction of the consistent partition corresponding to a particular input is itself P-complete, since it solves the circuit value problem as a byproduct. With the latch input at 0, however, all gates except those corresponding to the x_i will have their outputs set to 0, and so the assignment of vertices to A and B is automatic.) By the proof of Theorem 2, this consistent partition will be locally optimal for WLOKL if and only if the given inputs were locally optimal for MAXFLIP, which is true if and only if the output of C was 0, so P-completeness holds for the WLOKL verification problem. The result can be extended to LOKL by a simple modification to the weights that break initial ties in favor of the swap of the vertices $x_{n+1}, \bar{x}_{n+1}$ corresponding to the latch input.

Note that neither of the above P-completeness results for local optimality verification followed immediately from the corresponding PLS-completeness results; both required modification of the proofs. However, at present we do not see how one could prove PLS-completeness for a problem L *without* using a proof that could be modified to show that the verification problem for L is P-complete. It thus seems unlikely to us that a neighborhood structure based on a bounded number of "swaps," like TSP 2-changes or graph partitioning single-swaps, could be PLS-complete. (Their verification problems are solvable in logspace and hence P-complete only if P = logspace.)

On the other hand, it can be shown that the single-swap graph partitioning PLS-problem is *itself* P-hard, even if its verification problem is not. This holds even if we restrict costs to be polynomially bounded in the input size (although it then requires a new proof of equivalent complexity to that of Theorem 2). Under such a restriction the PLS-problem is of course in P, since the standard local search algorithm will run in polynomial time. Thus the result essentially says that, under this restriction, there is no better way to find a local optimum than simply running the standard algorithm. If such a conclusion were to hold when there is *no* bound on costs, it would imply that the general single-swap graph partitioning PLS-problem is not in P. One might obtain additional support for such a conclusion by identifying a natural subclass of PLS for which the unbounded-cost problem is complete. Unfortunately, we have been unable to do so and must leave this as an open problem for the reader.

REFERENCES

1. M. R. GAREY AND D. S. JOHNSON, "Computers and Intractability: A Guide to the Theory of NP-Completeness," Freeman, San Francisco, 1979.
2. L. M. GOLDSCHLAGER, The monotone and planar circuit value problems are log space complete for P, *SIGACT News* **9**, No. 2 (1977), 25–29.
3. D. S. JOHNSON, L. A. MCGEOCH, AND E. E. ROTHBERG, Near-optimal solutions to very large traveling salesman problems, to appear.
4. B. KERNIGHAN AND S. LIN, An efficient heuristic procedure for partitioning graphs, *Bell Systems Tech. J.* **49** (1972), 291–307.
5. S. KIRKPATRICK, C. GELAT, AND M. VECCHI, Optimization by simulated annealing, *Science* **220** (1983), 671–680.
6. D. E. KNUTH, manuscript, Stanford University, 1984.
7. S. LIN AND B. KERNIGHAN, An effective heuristic for the traveling saleman problem, *Oper. Res.* **21** (1973), 498–516.
8. G. LUEKER, manuscript, Princeton University, 1976.
9. C. H. PAPADIMITRIOU, "The Complexity of Combinatorial Optimization Problems," Ph. D. thesis, Princeton University, 1976.
10. C. H. PAPADIMITRIOU, The largest subdeterminant of a matrix, *Bull. Math. Soc. Greece* **15** (1984), 96–105.
11. C. H. PAPADIMITRIOU AND K. STEIGLITZ, Some examples of difficult traveling salesman problems, *Oper. Res.* **26** (1978), 434–443.
12. C. H. PAPADIMITRIOU AND K. STEIGLITZ, "Combinatorial Optimization: Algorithms and Complexity, Prentice-Hall, Englewood Cliffs, NJ., 1982.