(I have some notes on this from PapaD.'s Talk at MSRI)

HOW EASY IS LOCAL SEARCH? (Extended Abstract)

David S. Johnson
AT&T Bell Laboratories
Murray Hill, New Jersey 07974

Christos H. Papadimitriou
Stanford University
National Technical University of Athens

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

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 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 that differ from it in just two edges (this is called the 2-*change* neighborhood). In the graph partitioning problem (given a graph with $2n$ nodes and weights on the edges, partition the nodes into two sets of n nodes 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 nodes.

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 lower value, until no such neighboring solutions exist, 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 [K] 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 [LK], 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 [KL], 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 [LK, PS2]). There has been a debate on the *quality* of the optima [LK, Pa1, PS1], but the fact that local optima are easy to obtain has never been challenged. It has been shown [Lu] that in the 2-change neighborhood for the TSP we can have an exponentially long sequence of successive improvements, but this only with repeated unfortunate choices in a contrived example. Even for this simple case, the example is rather complex, and there are no such 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 simplex and the ellipsoid algorithm 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 interesting, because 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 [Kn] (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 non-singular $m \times m$ 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 [Pa2]. However, it is easy to see that, according to Cramer's rule, all we need is a *local optimum* 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 local optimum) has been observed always to converge after very 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 of functions. This class, called PLS for polynomial-time local search, is made up of functions that map instances of a combinatorial optimization problem (with a given neighborhood structure), to local optima. To make this class meaningful, we must make certain assumptions on the problem and the neighborhood structure: First,

0272-5428/85/0000/0039$01.00 © 1985 IEEE

given an instance (e.g., an $n \times n$ distance matrix) 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 total, multivalued functions, lies somewhere between the functional analogues of P and NP. (Any polynomial-time computable function can be formalized as a restricted local search problem, and any local search problem obeying the above restrictions can be solved nondeterministically in polynomial time by guessing a solution and verifying its local optimality.)

Where exactly is PLS? In particular, is it true that PLS = P (and thus we can always find local optima in polynomial time), or is it true that *PLS = NP* (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 and NP? Practically all the empirical evidence would lead us to conclude that finding locally optimal solutions is much easier than solving NP-hard problems. Moreover, it is easy to show that no problem in PLS can be NP-hard unless NP = co-NP. Thus it seems unlikely that PLS = NP. On the other hand, if PLS = P, 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 =? PLS =? NP thus has no obvious answers and suggests no natural conjectures, a property it shares with the more general open question of whether the computation of partial functions is in general sped up by knowledge that an image exists. (In our case, an image, i.e., a locally optimal solution, must exist by definition, since 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 ones 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, and 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. The reason is that, 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 3 we exhibit our first such discovery, a natural and well-studied local search problem that is complete for PLS: the Kernighan-Lin neighborhood structure for the graph partitioning problem [KL]. The proof is rather subtle, in that it must deal with delicate issues not present in ordinary reductions.

Showing more such problems complete is possible, but certainly not automatic. We can show that the corresponding local search problem for independent sets is complete, and so are several variants of the basic Kernighan-Lin local search problem. We have not, however, been able to prove PLS-completeness for the TSP with the λ-change neighborhood. In addition, it seems unlikely that certain local search problems, such as the TSP under the 2- and 3-change neighborhood structures and the problem of subdeterminant maximization, are complete. The reason is that these problems are defined in terms of algorithms for specifying the neighborhood and computing the cost which run in polylogarithmic 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.)

Notice finally that, for a PLS problem not to be trivially polynomial, the range of its possible solution costs must be exponential in size. For example, one can in polynomial time find a truth assignment such that flipping any single variable's truth value does not increase the number of clauses satisfied, and similarly for the unweighted version of the graph partitioning problem to be discussed in Section 3.

2. PLS and FLIP.

A *polynomial-time local search (PLS) problem* L is defined as follows: As with all computational problems, L has a set of instances, which can be taken to be a 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 length $p(|x|)$, polynomially bounded. For each solution $s \in F_L(x)$ we have a nonnegative integer *cost* $c_L(s,x)$ and also a 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, produces a particular standard solution $A(x) \in F_L(x)$. 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 $s' \in N(s,x)$ such that $c_L(s',x) < c_L(s,x)$, C produces such a solution. Otherwise it reports that no such solutions exist and hence that s is *locally optimal*. Now, the computational problem L is the following: *Given x, find a locally optimal solution $s \in F_L(x)$*. Note that such solutions must exist because $F_L(x)$ is finite.

We call PLS the class of problems of function computation that can be posed as local search problems for appropriate functions A,B,C. It is easy to see that the class P of polynomially computable functions is a subset of PLS, with $F_L(x)$ a singleton and all $N(s,x)$ empty; and the class NP of non-deterministic polynomial-time computable functions is a superset of PLS, in which the algorithms are allowed to be non-deterministic.

We say that a problem L in PLS is reducible to another, K, if there are polynomially computable functions f and g such that (a) f maps instances of L to instances of K, (b) g maps (solution, instance) pairs for instances of K generated by f back to solutions of L, and (c) for all instances x of L, if s is a local optimum for instance $f(x)$ of K, then $g(s,f(x))$ is a local optimum for x. We say that a problem L in PLS is *PLS-complete* if any problem in PLS is reducible to L.

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 $F(x)$ is any bit vector with m components. 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. Finally, there are m neighbors of each solution s, the m strings of length m with Hamming distance one from s, i.e., the strings that can be obtained from s by changing exactly one bit. Intuitively, this PLS problem asks for an input such that the output cannot be improved lexicographically by flipping a single input bit. It should be clear from this description that the required polynomial-time algorithms A_{FLIP}, B_{FLIP}, and C_{FLIP} all exist.

THEOREM 1. *FLIP is PLS-complete.*

Proof Sketch: Consider a PLS problem L. 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 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 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 M are of length $2p+1$. 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). $uu0$, in which case $c_Q(uu0,x) = (2p+2)c_L(u,x)$.

(b). $uv0$, 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 $c_Q(uv0,x)$ is $(2p+2)c_L(w,x)+(p+2)+h$, where h is the Hamming distance between v and w ($h=0$ is allowed).

(c). $vu1$, where v is any string of length p. The cost $c_Q(vu1,x)$ is $(2p+2)c_L(u,x)+h+1$, where h is the Hamming distance between v and u.

All "non-candidates" s will have cost exceeding

$$Z = (4p+2)(\max\{c_L(u,x): u \text{ is a solution for } x\}).$$

In particular, if we let a be the standard solution returned by algorithm A_L, and h be the Hamming distance between s and aao, then $c_Q(s,x) = Z+h$. Thus there is a downhill path from any non-candidate solution to $aa0$. Furthermore, by (a), (b), and (c), the only candidate solutions of Q that can be locally optimal must have the form "$vv0$." Since all solutions to L are at least Hamming distance 2 apart by assumption, the candidate solutions of the form "$vv0$" that *are* locally optimal are precisely those for which v is locally optimal for L. (This one-to-one correspondence between local optima is actually a stronger property than we need for a reduction from M to Q, but will certainly suffice. To complete the definition of Q, we let algorithm A_Q return the "standard" solution 0^{2p+1}.)

The last step of our proof is a 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 $(2p(|x|)+1)$ 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 hypothesis). □

3. 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 "Kernighan-Lin" 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 1. It is a *greedy* swap if $c(A,B)-c(C,D)$ is maximized 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 swap of (A,B). Let (A_i,B_i) be a sequence of partitions, of which each one 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 node is switched back to its original set). Finally, we say that a partition (C,D) is a neighbor of (A,B) in the Kernighan-Lin neighborhood structure if it occurs in the (unique) maximal monotonic sequence of lexicographic swaps starting with (A,B). Thus each partition has $O(|V|)$ 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 original paper [KL] does not specify what is to be done when there are more than one neighbors offering the best improvement, and so various implementations differ on this point. 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 can be shown to hold, however, for any reasonable variant on lexicographic tie-breaking.)

As an intermediary in our proof, we consider a variant on LOKL with a considerably larger neighborhood structure, called the *Weak* LOKL problem. 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.

THEOREM 2. *WLOKL is PLS-complete.*

Very Sketchy Proof Sketch: We start from FLIP. We first transform the circuit of the FLIP instance to a set of clauses of *not-all-equal-SAT* persuasion, each with a *level number*, corresponding to the level of the corresponding gate in the circuit, going from 1 (leaves) to d. Each **or** gate "$a = b$ **or** c" gives three clauses: $(\bar{a},b,c),(a,\bar{b},0),(a,\bar{c},0)$, and similarly for **and** gates (remember, these are "not all equal" clauses). Also, we assume that the input x_i is involved in only two clauses (x_i,y_i), $(\bar{x}_i,\bar{y}_i)$, where y_i, the negation of x_i, propagates the value upwards.

The graph has two "one nodes" and two "zero nodes," connected by edges of weight M^2. For each variable (input, gate, or output) a we have two nodes a and $\bar{a}$, connected by an edge of weight M. For each clause of level i, we connect the (three) literals with edges of weight N^{d-i}, where N is the circuit size. We also connect the jth output with a zero node, by an edge of weight $2^j/M$. Finally, we have two new nodes $a',\bar{a}'$ for each non-input variable a, an edge $(a',\bar{a}')$ of weight M, and edges $(a,\bar{a}')$ and $(\bar{a},a')$ with weight $N^{2d_1} + \frac{1}{M^2}$.

It can be shown that any local optimum of the resulting instance of WLOKL is such that all variables are separated, and thus a truth assignment is induced. Furthermore, the truth values must be consistent with the circuit, because otherwise Kernighan-Lin would improve the partition, starting from the variable corresponding to the lowest inconsistent gate. Finally, these values correspond to a local optimum input, otherwise Kernighan-Lin would discover the improvement. □

By more complex arguments, we can show that several other PLS problems are PLS-complete. First, we can extend the above proof to LOKL. Then, we can show a similar result for variants of LOKL in which the first swap is completely exhaustive (not greedy), or exhausts all nodes of A and then chooses the node to leave B greedily. Also, we can prove along the same lines completeness of the independent set problem (of course weighted), in which the analog of a swap consists of inserting a node in the independent set, and removing its neighbors. The first swap is exhaustive, and the remaining ones lexicographic. There seems to be no easy way to extend these results to other PLS problems; the salient open question here is the λ-change heuristic for the TSP.

REFERENCES

[KL] B. Kernighan and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs," *Bell Syst. Tech. J.*, 49, 2, pp. 291-307, 1972.

[KGV] S. Kirkpatrick, C. Gelat, and M. Vecchi, "Optimization by Simulated Annealing," *Science 220*, pp. 671-680, 1983.

[Kn] D. E. Knuth, manuscript, Stanford University, 1984.

[LK] S. Lin and B. Kernighan, "An Effective Heuristic for the Traveling Salesman Problem," *Operations Res.*, 21, pp. 498-516, 1973.

[Lu] G. Lueker, manuscript, Princeton University, 1976.

[Pa1] C. H. Papadimitriou, *The Complexity of Combinatorial Optimization Problems*, Ph.D. Thesis, Princeton University, 1976.

[Pa2] C. H. Papadimitriou, "The Largest Subdeterminant of a Matrix," *Bulletin of the Math. Society of Greece*, 15, pp. 96-105, 1984.

[PS1] C. H. Papadimitriou and K. Steiglitz, "Some Examples of Difficult Traveling Salesman Problems," *Operations Res.* 26, 3, pp. 434-443, 1978.

[PS2] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, Englewood Cliffs, N.J., 1982.