Discrete Applied Mathematics 43 (1993) 115-129
North-Holland

On the depth of combinatorial optimization problems

W. Kern

Faculty of Applied Mathematics, University of Twente, P.O. Box 217, 7500 AE Enschede, Netherlands

Received 23 January 1992

Abstract

Kern, W., On the depth of combinatorial optimization problems, Discrete Applied Mathematics 43 (1993) 115–129.

This paper studies the notion of "depth" of a combinatorial optimization problem, which has already been touched on by several other authors (cf. Weber or Hajek). This notion can be defined as follows: Consider a discrete optimization problem $\min\{c(x) \mid x \in X\}$. Usually hill climbing algorithms work as follows: First, a neighbourhood structure has to be specified on X. Then, starting from an arbitrary solution, say x_0, one constructs a sequence $x_0, x_1, x_2, \ldots$ (hopefully leading to an optimum) such that x_i is a neighbour of x_{i-1} for all i. For $x_0 \in X$, the depth $d(x_0)$ is defined to be the smallest $d \geq 0$ such that there exists a sequence $x_0, x_1, \ldots, x_n$ with $c(x_n) < c(x_0)$ and all intermediate solutions x_i having objective value $c(x_i) \leq c(x_0) + d$. The depth of the problem is defined to be the maximum depth of a solution $x_0 \in X$. This notion plays an important role in the theory of Simulated Annealing. Here we want to show that the depth is of some interest in its own right. We will prove some upper bounds and investigate the computational complexity of the depth function for some selected examples.

Keywords. Hill climbing, depth, computational complexity, Simulated Annealing.

1. Introduction

Suppose we are given a discrete optimization problem with finite set of feasible solutions X and objective function $c: X \to \mathbb{R}$. Suppose w.l.o.g. that we want to minimize $c(x)$, $x \in X$. A *local search* algorithm works as follows: First, specify a neighbourhood function $N: X \to 2^X$ and then apply the following procedure:

Local Search
Choose $x_0 \in X_0$, let $x := x_0$
LOOP: **if** $\exists x' \in N(x)$ s.t. $c(x') < c(x)$, let $x := x'$ and **goto** LOOP
else STOP

Correspondence to: Dr. W. Kern, Faculty of Applied Mathematics, University of Twente, P.O. Box 217, 7500 AE Enschede, Netherlands.

After a finite number of steps, this algorithm will run into STOP, thereby producing a solution x, which is called a local optimum. In general, this will not be a *global* one unless the neighbourhood function N satisfies very restrictive conditions (cf. [13,14] for a discussion of such neighbourhoods). A natural way to enable the above procedure to escape from local optima is to allow also replacement of a current solution x by another solution $x' \in N(x)$, which is worse, i.e., $c(x') > c(x)$. Such an approach is known as *hill climbing*. Let us consider a concrete version, called *Simulated Annealing* or *Metropolis Algorithm* (cf. e.g. [4,12]).

Simulated Annealing
Choose $x_0 \in X$, let $k := -1$
LOOP: $k = k+1$
choose randomly $x \in N(x_k)$, say each with probability $|N(x_k)|^{-1}$
if $c(x) < c(x_k)$ **then** $x_{k+1} := x$
else let $x_{k+1} := x$ with probability $p_k(x_k, x)$
and $x_{k+1} := x_k$ with probability $1 - p_k(x_k, x)$
goto LOOP

The behaviour of this method heavily depends on the choice of the sequence of transition probabilities. Intuitively, it seems appropriate to choose $p_k(x_k, x)$ such that x is accepted more likely, if either x is not too much worse or k is small (in order to achieve stability at the end). In analogy to a concept from statistical mechanics (cf. [5]), one usually defines p_k as follows: Choose a sequence of temperatures $T_k > 0$, such that $\lim_{k \to \infty} T_k = 0$. Then, given x and $x' \in N(x)$ such that $\Delta := c(x') - c(x) \geq 0$, let $p_k(x, x') := e^{-\Delta / T_k}$.

The following is proved in [4].

Theorem 1.1 (Hajek). *Suppose that the neighbourhood structure N on X is "weakly reversable", i.e., for any $c \in \mathbb{R}$, the set of all $x \in X$ with $c(x) \leq c$, provided with the induced neighbourhood structure, splits into strongly connected components.*

Then the Simulated Annealing process with transition probabilities (p_k) defined by the sequence of temperatures $T_k \downarrow 0$ as above converges to a global optimum (i.e., $\lim_{k \to \infty} \text{Prob}(x_k \text{ optimal}) = 1$) if and only if

$$\sum_{k \geq 0} e^{-d/T_k} = \infty$$

where d denotes the depth of the discrete optimization problem (as defined below).

Definition. For a nonoptimal $x \in X$, let $d(x)$ denote the smallest number $d \geq 0$ such that there exists a sequence $x = x_0, x_1, \ldots, x_n = x'$ in X satisfying:

(i) $x_i \in N(x_{i-1})$ for all i,
(ii) $c(x') < c(x)$,
(iii) $c(x_i) \leq c(x) + d$ for all i.

For an optimal $x \in X$, let $d(x) = 0$. $d(x)$ is called the *depth* of x (with respect to N). The depth of the optimization problem is defined to be

$$d := \max\{d(x) \mid x \in X\}.$$

Of course, there is an analogous definition for maximization problems. In the following sections we will investigate the depth for some selected examples of combinatorial optimization problems. Section 2 is about upper bounds for d and Section 3 deals with the computational complexity of the depth function.

2. General bounds

Suppose we are given a combinatorial optimization problem P and want to apply the Simulated Annealing method to a particular instance I of P. Then Theorem 1.1 tells us that we may choose $T_k \geq d/\ln k$, where $d = d(I)$ is the depth of the instance I. Since (at least for some classes of problems) computing the depth will turn out to be at least as difficult as solving the problem (cf. Section 3), we will usually have to content ourselves with a general upper bound, depending only on the size of the problem instances. Of course, this in turn depends heavily on the neighbourhood structure we choose. However, as with encodings, there usually exists (at least one) natural neighbourhood structure. In the following, we will consider some selected examples.

Max Cut. An instance of Max Cut is defined by a graph $G = (V, E)$. The problem is to find $U \subseteq V$ such that the set $\delta(U)$ of edges leaving U is of maximum cardinality. There is a well-known local search algorithm due to Kernighan and Lin, cf. [9], using the following natural neighbourhood structure N:

$$\text{For } U \subseteq V, \text{ let } U' \in N(U) \text{ if } |U \triangle U'| = 1.$$

Thus, if we want to determine the depth of a solution $U \subseteq V$, we have to consider sequences $U_0, U_1, \ldots$, obtained by successively adding or deleting vertices one by one. The worst intermediate solution, which can appear in such a sequence has objective function value $c(U_i) = |\delta(U_i)| = 0$. Hence the depth of a particular solution, $d(U)$, is bounded by $c(U) = |\delta(U)|$. Thus the depth of the instance defined by $G = (V, E)$ is bounded by $|E|$, which is of order n^2. The following simple example [6] shows that essentially no better bound exists:

Let $G = (V, E)$ be a disjoint union of two complete bipartite graphs $K_{n,n}$, i.e., $V = V_1 \cup V_1' \cup V_2 \cup V_2'$ and $E = V_1 \times V_1' \cup V_2 \times V_2'$. The instance of the Max Cut problem defined by G has precisely four optimal solutions. These are $V_1 \cup V_2$, $V_1 \cup V_2'$, $V_1' \cup V_2$ and $V_1' \cup V_2'$. It is obvious, that, starting from any solution $U \subseteq V$, we can find a sequence $U = U_0, U_1, \ldots, U_k$ with (strictly) increasing objective function values $c(U_i) = |\delta(U_i)|$ and U_k optimal. Hence this particular instance of Max Cut has depth equal to zero.

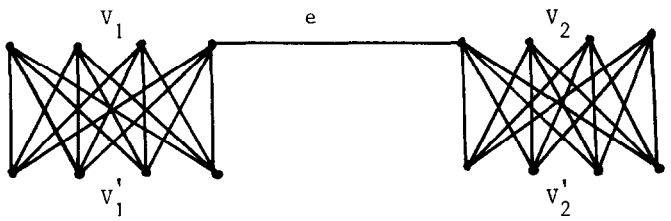

Fig. 1.

Now look at what happens if we insert an edge e between a vertex $v_1 \in V_1$ and $v_2 \in V_2$. Let $\tilde{G}$ denote the graph obtained in this way (cf. Fig. 1). It is not difficult to see, that $\tilde{G}$ has depth of order n^2: Note that in $\tilde{G}$ there are only two global optima, namely $V_1 \cup V_2'$ and $V_1' \cup V_2$. The two other solutions $V_1 \cup V_2$ and $V_1' \cup V_2'$ have become local optima. Furthermore, if $U_0, U_1, \ldots, U_k$ is a sequence joining one of the local optima, say $V_1 \cup V_2$, to a global one, say $V_1' \cup V_2$, then in one of the intermediate solutions U_i about half of all the edges joining $V_1 \cup V_1'$ must be "missing". This shows that each of the local optima has depth of order n^2, and hence so does the problem instance defined by $\tilde{G}$. Thus we have proved:

Proposition 2.1. *If $\bar{d}(n)$ is an upper bound for the depth of Max Cut instances of size n, then $\bar{d}$ is of order n^2.*

Upper bounds are not always as bad as the Max Cut example might suggest:

Max Matching. An instance of Max Matching is defined by a graph $G=(V,E)$ and the problem is to find a matching $M \subseteq E$ (i.e., a subset of pairwise disjoint edges) of maximum cardinality. It has been observed in [3] that for the natural neighbourhood, defined in the same way as above, the depth of Max Matching instances is bounded by 1. We include the simple proof for completeness, since we will be concerned with Max Matching in Section 3 again.

As above, define the neighbourhood structure N by saying that two matchings M and M' are neighbours, if $|M \triangle M'| = 1$. Thus, again, if we want to determine the depth of a particular matching M, we have to consider all sequences $M = M_0, M_1, \ldots$, obtained by adding and removing edges one by one, finally reaching a better solution M'. It is well known, that given an arbitrary suboptimal matching M, we can

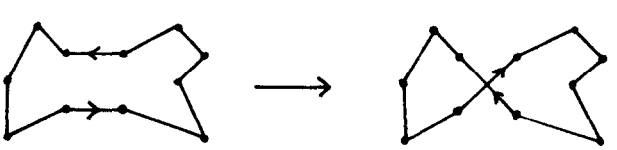

Fig. 2.

construct a larger one by successively removing one edge and inserting another, along a so-called alternating path. All intermediate matchings, obtained in this way, have cardinality at least $|M|-1$, so the depth of M is at most 1.

Proposition 2.2. *Any instance of Max Matching has depth at most* 1.

Finally, let us consider at least one example of a problem involving weights.

Euclidean Traveling Salesman Problem (ETSP). An instance of ETSP is specified by a set $x_1, \ldots, x_n$ of points in the unit square and the problem is to find a tour (polygon) through $x_1, \ldots, x_n$ of minimum (total) length. A tour T through $x_1, \ldots, x_n$ may be considered as a finite set of line segments $[x_i, x_j]$, which we call edges. Again there is a quite natural notion of neighbourhood: Say that two tours T and T' are neighbours if $T \triangle T'$ consists of precisely four edges (cf. Fig. 2). This is the neighbourhood structure used by the so-called Switching Algorithm for TSP (which is again the obvious local search method, due to Kernigham and Lin, cf. [9]).

Proposition 2.3. *There are constants $\alpha, \beta > 0$ such that any instance $x_1, \ldots, x_n$ of ETSP has depth bounded by $\alpha\sqrt{n} + \beta$.*

This is an immediate consequence of the following two results:

Lemma 2.4. *There are constants $\alpha, \beta > 0$, such that any instance $x_1, \ldots, x_n$ of ETSP has an optimal solution of length $\leq \alpha\sqrt{n} + \beta$.*

Proof. A rough estimate follows easily from subdividing the unit square into small squares of side length $\approx 1/\sqrt{n}$ (cf. [1,8], for tighter bounds). □

Lemma 2.5. *Any instance $x_1, \ldots, x_n$ of ETSP has depth at most $2L^*$, where L^* denotes the length of a shortest tour through $x_1, \ldots, x_n$.*

Proof. Let T be a tour and let T^* be an optimal tour. We will show, that $d(T) \leq 2L^*$ by constructing a sequence $T = T_0, T_1, \ldots, T_k = T^*$ such that every T_i is a neighbour

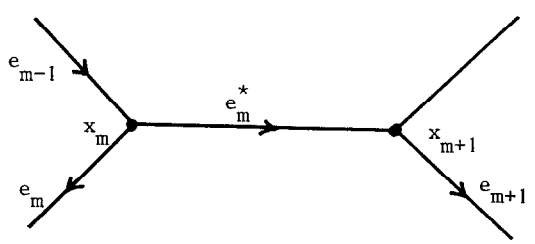

Fig. 3.

of T_{i-1}, and its length does not exceed the length of T by more than $2 \cdot L^*$. This can be achieved by starting with $T_0 = T$ and successively "switching in" edges from T^*:

Suppose that we have already constructed T_i, $i \geq 0$, and $T_i \neq T^*$. We may fix an orientation of T_i and T^*. Then the statement "edge e leaves (enters) point x_j" is understood to have the obvious meaning. Suppose that the points are ordered in such a way that T^* visits $x_1, \ldots, x_n$ in that order. Let e_j^* denote the edge of T^* going from x_j to x_{j+1} for all j (with indices mod n) and let e_m^* denote the first edge in $T^* \setminus T_i$. Then $m < n$, since $T_i \neq T^*$ implies that $|T^* \setminus T_i| \geq 2$. Furthermore the (possibly empty) path $e_1^*, \ldots, e_{m-1}^*$ is a subset of T_i and we assume that the orientation of T_i agrees with that of T^* on $e_1^*, \ldots, e_{m-1}^*$. Since $e_m^* = (x_m, x_{m+1})$ is not in T_i, there is an edge e_m of T_i, leaving x_m and not contained in T^* (cf. Fig. 3).

Let e_{m+1} denote the edge of T_i, leaving x_{m+1} (this may be equal to e_{m+1}^* or not). Then it is obvious that we can "switch in" e_m^*, i.e., we can replace e_m and e_{m+1} by the two edges joining the heads and the tails of e_m and e_{m+1} respectively, thus obtaining a new tour T_{i+1}. Note that T_{i+1} contains $e_1^*, \ldots, e_m^*$ (thus we will be finished after a while). Furthermore, switching from T_i to T_{i+1} causes an increase in length of at most twice the length of e_m^* (this is an easy application of the triangle inequality). Thus none of the "intermediate solutions" T_i will exceed the length of T by more than $2 \cdot L^*$. □

A similar analysis can be carried out, e.g. for the so-called *Minimum Euclidean Matching Problem*. An instance of that is given by an even set $x_1, \ldots, x_{2n}$ of points in the unit square and the task is to find n pairwise disjoint "edges" of minimum total length. Arguments similar to the ones above show that the depth of such an instance is $O(\sqrt{n})$. (This disproves a conjecture of [15], where it is claimed that any bound on the depth would be of order n.) However, we do not know, whether there exist nontrivial bounds of those problems in general graphs (i.e., in case the edge weighting does not satisfy the triangle inequality).

3. Some remarks on the computational complexity

At the first glance, computing the depth of a problem instance I seems to be a really hard problem: According to the definition of the depth of I, we have to compute the maximum over all depths of feasible solutions (and there are exponentially many of them), and even to compute the depth of just one particular feasible solution, one has to consider all sequences starting at that particular feasible solution and leading to a better one. Nevertheless, it may happen, that the depth can be computed efficiently:

Proposition 3.1. *The depth of an instance $G = (V, E)$ of the Max Matching problem and the depth of any particular feasible solution can be computed in polynomial time.*

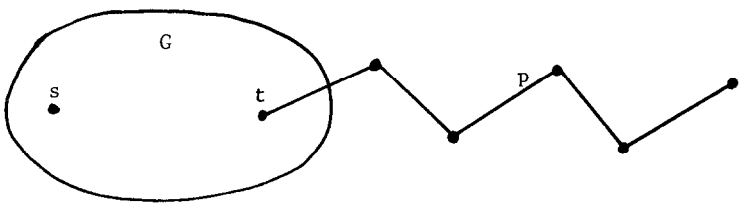

Fig. 4.

Proof. Let $G=(V,E)$ be an instance of the Max Matching problem and let M be a matching. If we can add an edge e to M, such that $M_1:=M+e$ is a matching, then the depth of M is zero. If no edge can be added to M, then either M is optimal (which can be checked in polynomial time) and hence has depth 0, or M is not optimal and then must have depth equal to 1. Hence $d(M)$ can be computed in polynomial time. Now let us turn to the depth of the problem instance $G=(V,E)$. We know, that it is either 0 or 1 (cf. Section 2), according to whether every matching M in G can be augmented to an optimal one by simply adding some edges, or not. In other words, the depth of G is 0 if and only if every *maximal* matching in G is also *maximum*. Such graphs are called *equimatchable*, and a polynomial algorithm for testing whether a graph is equimatchable or not has recently been found by Lesk, Plummer and Pullyblank (cf. [10] or [11, Theorem 3.2.8]). □

Next let us consider a typical example of an NP-complete problem, the Longest Path problem.

An instance of Longest Path is given by a graph $G=(V,E)$, a distinguished vertex $s\in V$ and an integer $K\geq 0$. The problem is to decide whether there exists a (simple) path in G, starting at s and having length at least K.

This is easily seen to be NP-complete: Reduce it to the so-called Longest s-t Path problem [2] by adjoining a sufficiently long path p to G, starting at t (cf. Fig. 4).

Again, if p or p' are two paths in G, both starting at s, let us say that they are neighbours, if $|p\triangle p'|=1$, i.e., if p' arises from p by adjoining an edge, or deleting the last edge of p. If we agree that this kind of neighbourhood is natural for the problem, we can easily prove the following:

Proposition 3.2. *There is no polynomial algorithm for computing the depth of an instance of Longest Path (or a particular feasible solution), unless* P = NP.

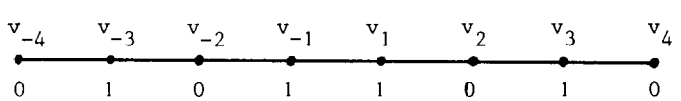

Fig. 5.

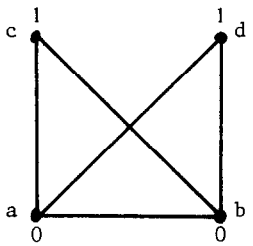

Fig. 6.

Proof. Let $G=(V,E)$ and $s \in V$. Construct a graph $\tilde{G}$ by adjoining a path p^* to G, starting at s and having length $n=|V|$. Then it is obvious that p^* is the unique optimum for the Longest Path problem in $\tilde{G}$. Furthermore, if p is a longest path in G, starting at s, then the depth of p (in the instance defined by $\tilde{G}$) is equal to the length of p, since in order to go from p to p^* by successively adding and removing edges, all edges of p have to be removed. Thus the depth of the problem instance $\tilde{G}$ equals the length of a longest path in G. This shows that the depth of a problem instance in general cannot be computed efficiently unless P = NP. A similar argument can be used to prove the same result for the depth of particular solutions: Let G be as above. For $i=1,\ldots,|V|$, let G_i denote the graph obtained by adjoining to G a path p_i^* of length i, starting at s. Then it is obvious that the length of a longest path in G, starting at s equals

$$\min\{i \mid \text{depth}(p_i^*)=0 \text{ (in the problem instance } G_i)\}.$$

Thus the depth of a particular solution can in general not be computed efficiently, unless P = NP. □

The crucial point in this example seems to be that any oracle for computing the depth of a given solution immediately yields a trivial polynomial algorithm for finding a better solution: If our current solution has depth d, then simply delete the last d edges to obtain a solution of depth 0. From there we proceed by adding edges, one by one in such a way that the depth remains equal to 0 until this is no longer possible.

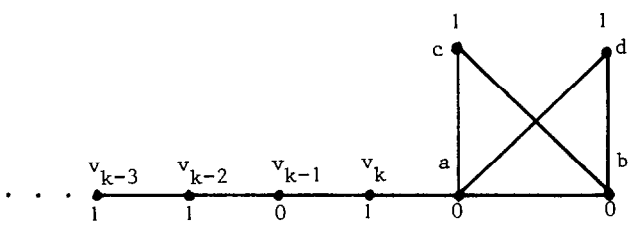

Fig. 7.

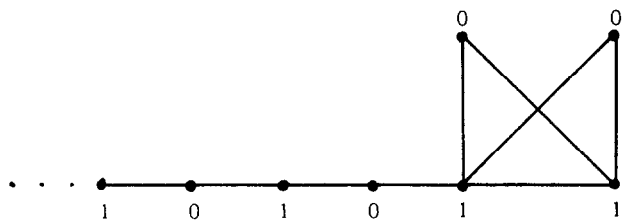

Fig. 8.

According to the above discussion, the Longest Path problem may appear to be of a very special kind with respect to the complexity of its depth function, due to the fact that an oracle for computing the depth function may be used as a guide to an optimal solution in a trivial way. The situation is somewhat different in the context of Max Cut problems, as we will see below. There, an oracle for computing the depth function does *not* help us to find a better solution in such an obvious way, but still the problem of computing the depth function is NP-complete! More precisely, we will see that it is possible to construct instances of Max Cut problems together with given feasible solutions which, by construction, have depth equal to 0 or 1 and such that

(1) computing their depth amounts to solving a satisfiability problem for an associated Boolean function $f(x_1, \ldots, x_n)$;

(2) finding an optimal solution (and a path from our current nonoptimal solution to the optimal one) amounts to explicitly solving the equation $f(x_1, \ldots, x_n) = 1$ by exhibiting an appropriate assignment of 0's and 1's to the variables $x_1, \ldots, x_n$.

Thus in particular we get:

Proposition 3.3. *Computing the depth function for Max Cut problems is* NP*-hard.*

Proof. Suppose we are given a graph $G = (V, G)$ and a cut defined by a bipartition $V = V_0 \cup V_1$ of the vertices. A vertex $v \in V_0$ will simply be called a 0-*vertex* in the following and similarly a vertex $v \in V_1$ will be called a 1-*vertex*.

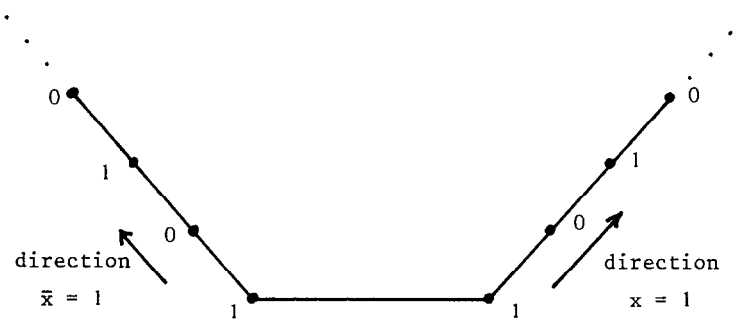

Fig. 9.

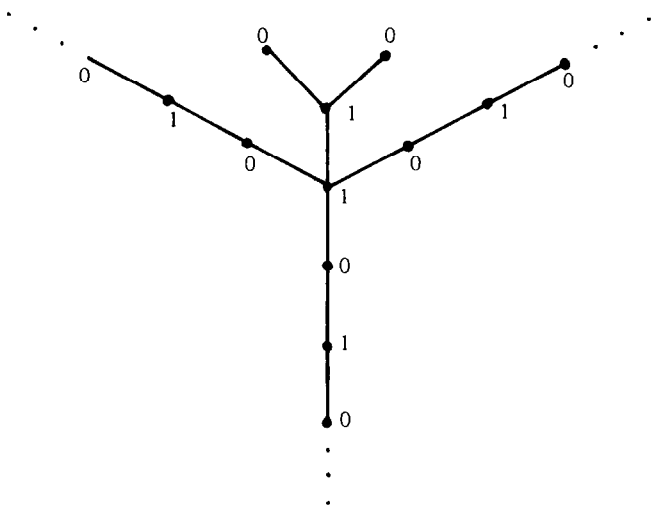

Fig. 10.

Of course, the above decision is trivial in case one of the vertices $v \in V$ can be switched, i.e., a 0-vertex can be switched to a 1-vertex or the other way around, thereby increasing the cut. Thus assume that for every 0-vertex v its neighbourhood $N(v)$ contains at least as many 1-vertices as it contains 0-vertices. Similarly, assume that every 1-vertex v has at least as many 0-neighbours as there are 1-neighbours.

Say that a vertex is *flexible*, if it has equally many 0- and 1-neighbours, and call it *fixed* otherwise. Thus flexible vertices may be switched from V_0 to V_1 or the other way around without changing the value of the cut, while switching a fixed vertex results in decreasing the value of the cut. For example, consider the path with vertices $\ldots, v_{-4}, v_{-3}, v_{-2}, v_{-1}, v_1, v_2, v_3, v_4, \ldots$ being 0- or 1-vertices as indicated in Fig. 5. In this case v_{-1} and v_1 are flexible, all other vertices are fixed. Thus, for example, v_1 may be switched to a 0-vertex. As a consequence, v_{-1} would become fixed, v_2 would become flexible and v_1 would stay flexible. Now v_2 can be switched, resulting in v_1 fixed and v_3 flexible and so on. Consider this process as a signal running along the path $v_1, v_2, v_3, \ldots$ without any energy (value of the cut) being lost.

Now consider the graph with four vertices, partitioned into 0- and 1-vertices as shown in Fig. 6. Obviously, each of its vertices is fixed. But look at what happens if we attach a path to vertex a along which a signal may run until it hits vertex a (cf. Fig. 7). As soon as v_k is switched to a 0-vertex, node a becomes flexible and may be switched to a 1-vertex. This causes both c and d to become flexible and thus both may be switched to a 0-vertex. This in turn allows to switch b to a 1-vertex, thereby increasing the cut value by 1. The final figuration looks like Fig. 8.

The main idea in the proof for NP-completeness consists in reducing SAT to the

above problem in the following way: Let $f=f(x_1,\ldots,x_n)$ be a Boolean function. For each variable $x=x_i$ we construct a "signal generator", which essentially is just a path $v_{-k},\ldots,v_{-1},v_1,\ldots,v_k$ as mentioned at the beginning. That is, both v_{-1} and v_1 are 1-vertices and the remaining vertices form alternating sequences of 0- and 1-vertices in both directions. A signal running through $v_1,v_2,v_3,\ldots$ shall correspond to setting variable $x=x_i$ to 1, whereas a signal running through $v_{-1},v_{-2},v_{-3},\ldots$ shall correspond to setting variable $\bar{x}=\bar{x}_i$ to 1. The next step is to build up a kind of Boolean circuit for the function $f=f(x_1,\ldots,x_n)$, using "AND" and "OR" gates as well as bifurcation elements that allow to multiply and combine signals in a way

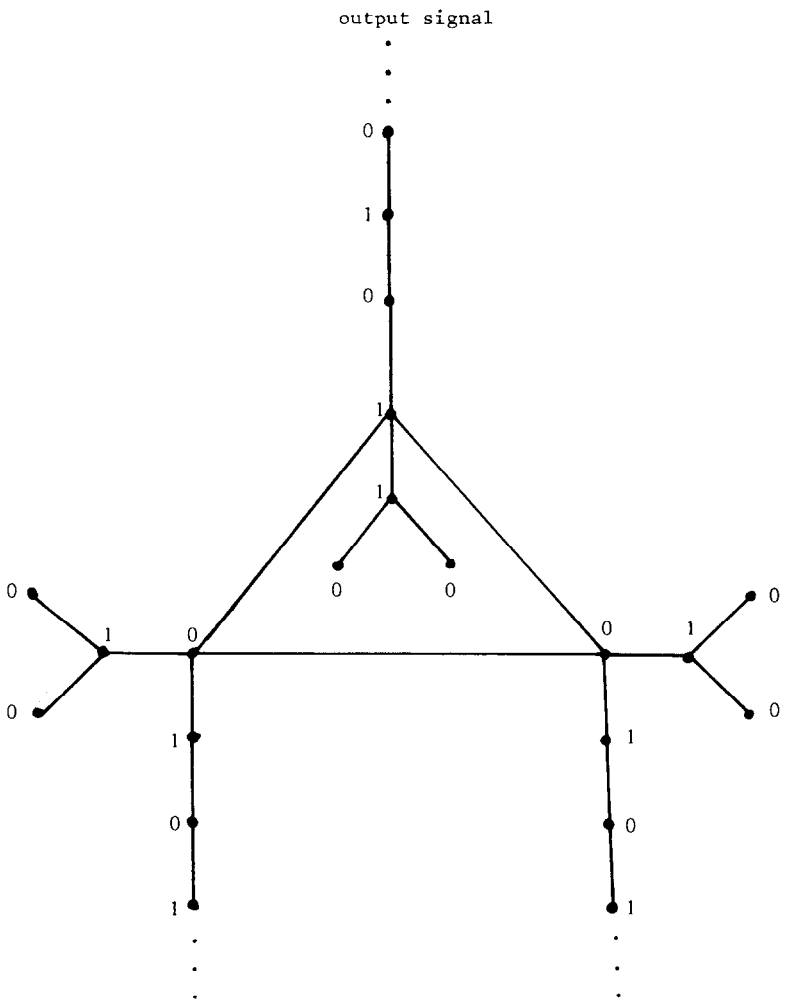

Fig. 11.

such that f is satisfiable if and only if the signal generators can generate signals that run through all the gates, yielding a signal as output of our circuit. This output signal is the one which eventually reaches our 4-vertex test graph described above by entering vertex a. Thus, f is satisfiable if and only if there exists a sequence of switchings, none of which decreases the current cut value, that finally leads to an improvement by 1. In the following we give a detailed description of the building blocks we use for constructing the Boolean circuit.

(i) *The signal generator.* This has already been described. It consists of a path as indicated in Fig. 9 with two 1-vertices neighbouring. Note that a signal can leave a signal generator only in one of the directions $x=1$ or $\bar{x}=1$, for as soon as one of the flexible 1-vertices switches, the other becomes fixed.

(ii) *The bifurcation elements.* A bifurcation element is needed for splitting a signal into two. This may be obtained as indicated in Fig. 10. Note that all vertices

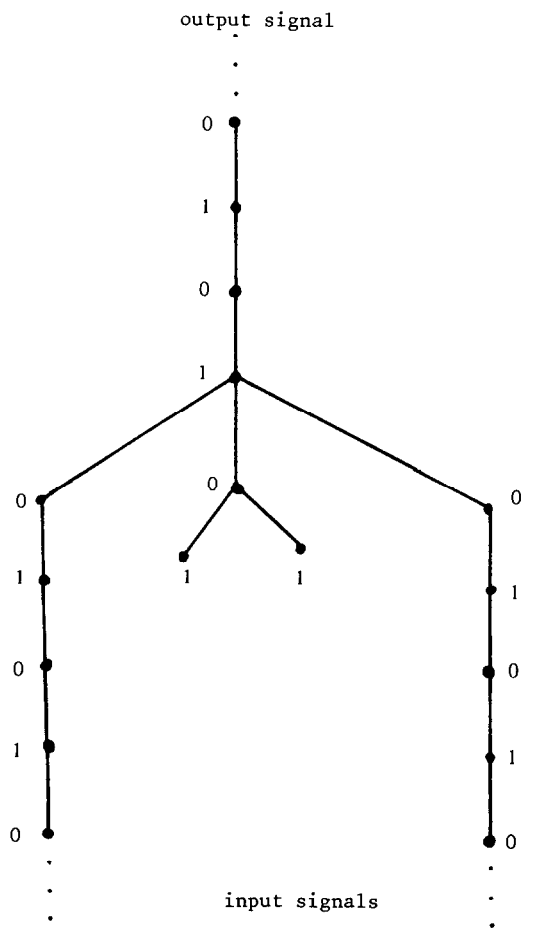

Fig. 12.

are fixed and become flexible only if a signal comes in. The 3-star appended to the branching point is only for balancing purposes.

(iii) *The* OR-*gates*. An OR-gate has two incoming signals and one output signal (see Fig. 11).

(iv) *The* AND-*gates*. An AND-gate has two incoming signals and one output signal as shown in Fig. 12. Note that the length of the signal paths in each of the

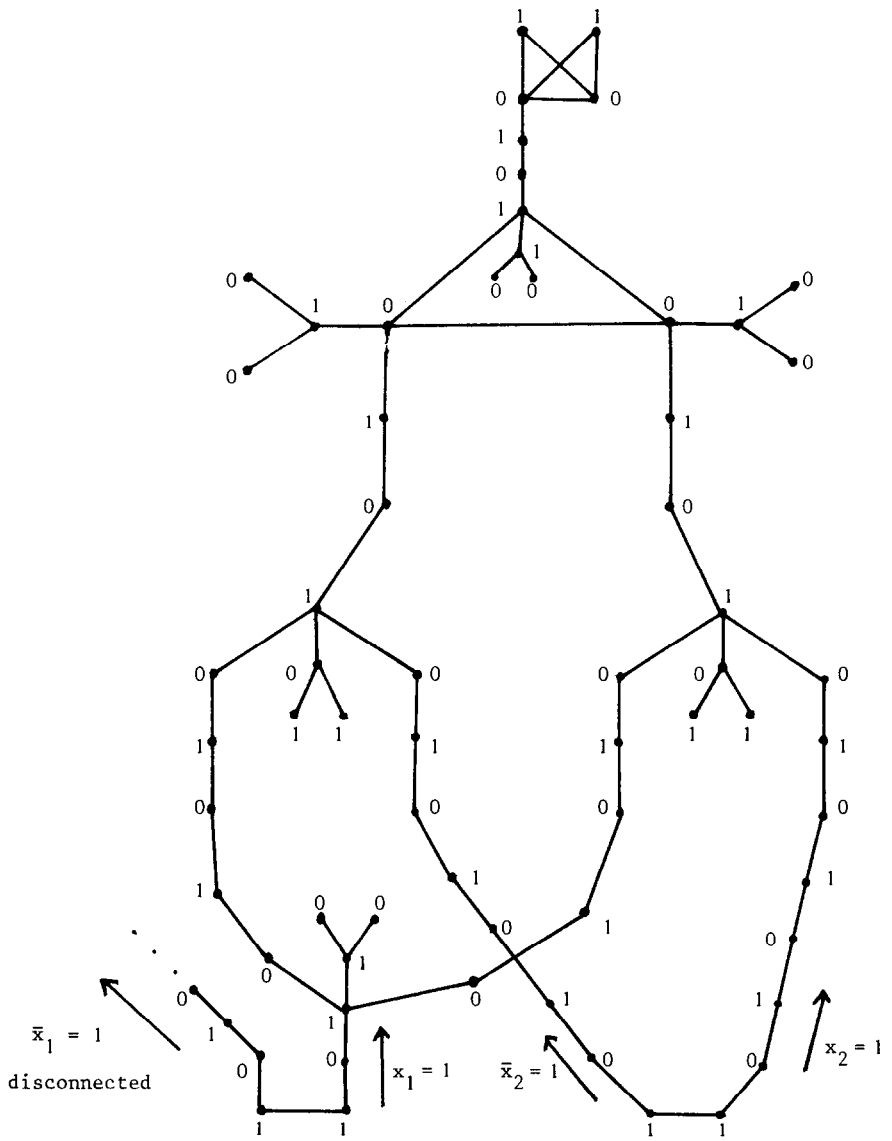

Fig. 13.

building blocks above equals 4 which is unnecessarily large. We have chosen long paths only for being more illustrative. But anyway, any constant length of the signal paths gives rise to a Boolean circuit (graph) which represents a given Boolean function $f=f(x_1, \ldots, x_n)$ such that the size of the graph is polynomially bounded by the size of the function f. Thus in particular this is a polynomial reduction from SAT to our decision problem. □

As an example, consider the instance of our problem corresponding to the Boolean function $f=(x_1 \wedge \bar{x}_2) \vee (x_1 \wedge x_2)$. This is obtained by constructing the corresponding Boolean circuit and attaching our 4-vertex test graph to the output signal path (see Fig. 13).

4. Conclusions

As indicated by the examples presented in Section 3, there seem to be some interesting relations between the complexity of optimization problems and their depth functions, which perhaps should be studied in a more rigorous manner than it is done here (by just looking at some examples, most of which were selected by chance, the remaining ones by the fact that we could prove anything about their depth functions). In particular, one is tempted to make the following:

Conjecture 1. Computing the depth function is at least as hard as solving an optimization problem.

This is true at least for the examples discussed in Section 3. In fact, for computing the depth function for matching problems by means of the Lesk–Plummer algorithm we do have to solve matching problems. In case of the Longest Path problem, we have seen that computing the depth function is as hard as solving the problem, since any oracle for computing the depth function trivially gives rise to an improvement procedure. This is different in case of the Max Cut problem, where we have seen that computing the depth function is as hard as solving the satisfiability problem for a Boolean function $f(x_1, \ldots, x_n)$, whereas solving the optimization problem amounts to exhibiting explicitly a truth assignment to the variables $x_1, \ldots, x_n$. But still: Once we have an oracle for computing the depth function, i.e., an oracle solving SAT, we can construct a polynomial time algorithm for explicitly finding truth assignments in the obvious way. (Given $f=f(x_1, \ldots, x_n)$, we successively ask our oracle whether $f \wedge x_i$ or $f \wedge \bar{x}_i$ is satisfiable.) In other words: Computing optimal solutions for the instances of Max Cut constructed in the proof of Proposition 3.3 is still NP-easy. Thus, our examples also give rise to the following.

Conjecture 2. Computing the depth function is at most as hard as solving the optimization problem.

Before dealing with these two conjectures in general, one has first to spend some thoughts on the notion of *natural* neighbourhoods. These should be polynomially equivalent, as far as the computational complexity is concerned. The very interesting recent work of Johnson, Papadimitriou and Yannakakis on so-called PLS-completeness [7] might be helpful in this context. Perhaps it is also worthwhile to study reductions among NP-complete problems with respect to their effect on depth functions.

References

[1] L. Few, The shortest path and the shortest road through n points, Mathematika 2 (1955) 141–144.
[2] M.R. Garey and D.S. Johnson, Computers and Intractability. A Guide to the Theory of NP-Completeness (Freeman, San Francisco, CA, 1979).
[3] B. Hajek, A tutorial survey of theory and applications of Simulated Annealing, Tech. Rep., Department of Electrical and Computer Engineering, University of Illinois, Urbana, IL (1985).
[4] B. Hajek, Cooling schedules for optimal annealing, Math. Oper. Res. 13 (1987) 311–329.
[5] H. Hammersley and D.C. Handscomb, Monte Carlo Methods (Wiley, New York, 1964).
[6] Ch. Hohmann, private communication, 1986.
[7] D.S. Johnson, C.H. Papadimitriou and M. Yannakakis, How easy is local search?, J. Comput. System Sci. 37 (1988) 79–100.
[8] R. Karp and M. Steele, Probabilistic analysis of heuristics, in: Lawler et al., eds., The Traveling Salesman Problem (Wiley, New York, 1985).
[9] B.W. Kernighan and S. Lin, An effective heuristic algorithm for the traveling salesman problem, Oper. Res. 21 (1973) 498–516.
[10] M. Lesk, M.D. Plummer and W.R. Pulleyblank, Equimatchable graphs, in: B. Bollobás, ed., Graph Theory and Combinatorics, Proceedings of the Cambridge Combinatorial Conference in Honour of Paul Erdös (Academic Press, London, 1984) 239–254.
[11] L. Lovasz and M.D. Plummer, Matching Theory, Annals of Discrete Mathematics 29 (North-Holland, Amsterdam, 1986).
[12] M. Lundy and A. Mees, Convergence of the annealing algorithms, Math. Programming 34 (1986) 111–124.
[13] C.H. Papadimitriou and K. Steiglitz, On the complexity of local search for the Traveling Salesman Problem, SIAM J. Comput. 6 (1977) 70–83.
[14] S.L. Savage, P. Weiner and M.J. Krone, Convergent local search, RR 14, Department of Computer Science, Yale University, New Haven, CT (1973).
[15] M. Weber and Th.M. Liebling, Euclidean matching problems and the metropolis algorithm, working paper, Département de Mathématiques, EPF-Lausanne (1985).