Zsolt Tuza

Theorem proving through depth-first test

Acta Universitatis Carolinae. Mathematica et Physica, Vol. 33 (1992), No. 2, 135--141

Persistent URL: `http://dml.cz/dmlcz/701986`

1992 ACTA UNIVERSITATIS CAROLINAE—MATHEMATICA ET PHYSICA VOL. 33. NO. 2

Theorem Proving Through Depth-First Test

ZSOLT TUZA

Hungary*)

Received 10 May 1992

We point out that the basic concept of depth-first search is not only an efficient tool in algorithmic graph theory but also it provides a powerful approach for proving theorems of a non-algorithmic nature. The main application in this note is a sharp extremal result concerning the maximum number of clauses that can be satisfied simultaneously in a Boolean formula in conjunctive normal form, with at most two literals per clause. We prove this theorem by designing a linear-time algorithm whose worst-case performance is at least as good as the general lower bound. Some results on graph k-colorability are recalled, too. Applying depth-first search or its variations, linear or polynomial algorithms can also be obtained for solving some subproblems of those NP-complete problems.

0. Introduction

Depth-first search (DFS, for short) is one of the basic graph-search techniques in computer science, because it can easily be implemented in linear time. It has found many applications in algorithmic problems, see e.g. [7, 14]. In graph theory, a typical area where DFS is efficient is to test various sorts of connectivity (to find a spanning tree if the graph is connected, to list the connected components and the 2-connected blocks of an undirected graph or the strong components of a directed graph). References to further applications (3-connectivity, planarity, planar graph isomorphism, dominators) can be found in [14, 15].

For a long time, DFS was applied to design fast algorithms only. Recently it has turned out, however, that this algorithmic approach is powerful in solving problems of a purely theoretical nature as well. In the present note we briefly describe two areas of such applications. The basic problems are:

(a) Satisfiability of clauses of a Boolean formula in conjunctive normal form — estimate the number of clauses that can be satisfied simultaneously.

*) Computer and Automation Institute, Hungarian Academy of Sciences, H-1111 Budapest, Kende u. 13—17, Hungary.

(b) Chromatic number of graphs – find necessary and/or sufficient conditions ensuring that a graph is k-colorable, for a given natural number $k \geqq 3$.

The problems Maximum Satisfiability and Graph k-Colorability both are NP--complete. Using DFS, however, one can obtain fast algorithms that provide estimates for either of them when the formulae or the graphs in question satisfy some further requirements.

We mention a third field of application of DFS, namely that one can also find fairly large k-colorable subgraphs in a graph by a linear-time algorithm. Details concerning this subject are given in the paper [11].

Graph theoretic notions not defined here can be found e.g. in [1].

1. Satisfiability of clauses

Let Φ be a Boolean formula on n variables $x_1, \ldots, x_n$, in conjunctive form, i.e. $\Phi = \bigwedge_{i \in I} \phi_i$ where $\Phi_i = \bigvee_{j \in J(i)} y_{i,j}$ ($|I|$ finite, $|J(i)| \leqq n$ for all $i \in I$, and $y_{i,j} \in \in \bigcup_{1 \leqq k \leqq n} \{x_k, \neg x_k\}$ for $i \in I$ and $j \in J(i)$, each x_k occurring at most once in each ϕ_i). If there are no restrictions on the clauses ϕ_i, then it is NP-complete to decide whether or not a formula Φ is satisfiable; what is more, the satisfiability problem is NP-complete even on the class of those formulae in which evety clause has at most three variables [2]. For this reason we shall concentrate on the class Φ_2 of formulae Φ with $|\phi_i| := := |J(i)| \leqq 2$ for every $i \in I$.

For the restricted class of Φ_2, it is well-known (see e.g. [3]) that the satısfiability problem is polynomially solvable; in contrast, for a $\Phi \in \Phi_2$ it is NP-complete to determine the maximum number $m'(\Phi)$ of clauses that can simultaneously be satisfied in Φ, as proved by Garey *et al.* [6]. (The problem of finding $m'(\Phi)$ is sometimes referred to as *Maximum Satisfiability.*)

For $\Phi \in \Phi_2$ we shall denote by $m = m(\Phi)$, $t = t(\Phi)$, and $s = s(\Phi)$ the number $|I|$ of clauses, the number of clauses with precisely two variables, and the number of clauses with a single variable, respectively ($t + s = m$). Throughout we shall assume that each pair of distinct variables defines at most one clause in Φ (i.e., for $i \neq j$ at most one of $x_i \vee x_j$, $x_i \vee \neg x_j$, $\neg x_i \vee x_j$, and $\neg x_i \vee \neg x_j$ may be a clause in Φ); it is not necessary, however, to exclude clauses occurring more than once.

Define a graph $G = G(\Phi) = (V, E)$ as follows. The vertex set V of G is $[n] := := \{1, 2, \ldots, n\}$; the edge set E consists of the unordered pairs $\{i, j\}$ such that $i, j \in [n]$, $i \neq j$, and the variables x_i and x_j occur together in some clause of Φ (hence, $|V| = n$ and $|E| = t$). We denote by $k = k(\Phi)$ the number of connected components in the graph $G(\Phi)$.

In [12], Poljak and Turzík proved the following result which is sharp in infinitely many cases. (For a real number x, $\lceil x \rceil$ denotes the least integer not smaller than x.)

Theorem 1. ([12]) *For every formula* $\Phi \in \Phi_2$,

$$m'(\Phi) \geqq 3t(\Phi)/4 + s(\Phi)/2 + \lceil (n(\Phi) - k(\Phi))/2 \rceil \sqrt{}/4\,.$$

Moreover, in [12] an algorithm of running time $O(n^3)$ is given that finds a $0-1$ assignment of the variables x_i (representing *TRUE* by 1 and *FALSE* by 0) which satisfies at least as many clauses as claimed in Theorem 1.

In Section 3 we prove the extremal result of Theorem 1 by a method that leads to an algorithm whose running time is proportional to the length of Φ, i.e. it is optimal apart from a multiplicative constant.

Theorem 2. *There is an algortihm that finds a* $0-1$ *assignment of variables in* $O(m + n)$ *time for an arbitrary function* $\Phi \in \Phi_2$, *such that at least* $3t/4 + s/2 + \lceil (n - k)/2 \rceil /4$ *clauses of* Φ *are satisfied.*

Let us formulate a related result more general than Theorem 1 (valid for all Boolean functions, not only those in Φ_2) that can be proved by different methods.

Theorem 1. [8, 9]) *If* $\Phi = \bigwedge_{1 \leqq j \leqq m} \phi_j$ *is a Boolean function in conjunctive form with m clauses, then at least* $m - \sum_{1 \leqq j \leqq m} 2^{-|\phi_j|}$ *clauses of* Φ *can be satisfied simultaneously.*

Also, a satisfying truth assignment (whose existence is guaranteed by Theorem 3) can be found in polynomial time. For a $\Phi \in \Phi_2$, however, Theorems 1 and 2 yield a slightly stronger sufficient condition and a faster algorithm.

2. Graph colorings

In this section, recalling some results from [16], we illustrate the power of DFS in connection with graph colorings. The *chromatic number* $\chi(G)$ of a finite undirected graph $G = (V, E)$ (with vertex set V and edge set E) is the minimum number of independent sets (= sets of pairwise non-adjacent vertices) whose union is V. Call G *k-colorable* if $\chi(G) \leqq k$. Although it is NP-complete to decide whether or not G is k-colorable (for any $k \geqq 3$), there are some necessary and sufficient conditions for k-colorability in terms of *orientations* of G. (A directed graph $D = (V, A)$ with arc set A is an orientation of G if for each edge $e \in E$ there is precisely one arc in A with the same two vertices as e, and vice versa.)

Relating the chromatic number with the orientations of a graph, the following results were established in the 1960s. Throughout, $k \geqq 2$ is an arbitrary integer.
(1) A graph is k-colorable if and only if it has an orientation containing no directed path on more than k vertices. (Gallai [5] and Roy [13])
(2) A graph G is k-colorable if and only if it has an orientation D in which every cycle C of G has at least $|C|/k$ arcs in each direction. (Minty [10])

These two results are complementary in the sense that the assumption in (2) excludes directed cycles, while (1) is fairly obvious for acyclic orientations and the more interesting part of its proof is when the orientation of the graph providing minimum length for the longest directed path is not acyclic. Those classic results have the following common generalization.

Theorem 4. ([16]) *A graph G is k-colorable if and only if it has an orientation in which every cycle C of G with $|C| \equiv 1 \pmod k$ has at least $|C|/k$ arcs in each direction.*

To show how DFS can be applied to prove results of this kind, we present the proof of an „undirected version" of Theorem 4. (It was verified independently by Dean and Toft [4] by another method not leading to a coloring algorithm but proving the existence of many cycles in a non-k-colorable graph.)

Corollary 5. *If an undirected graph $G = (V, E)$ contains no cycle of length $\equiv 1 \pmod k$, then G is k-coloraable, and a k-coloring of G can be found in $O(|V| + |E|)$ time.*

Proof. Let T be a spanning tree in G (or in one of its connected components), with root r, found by a DFS algorithm. For $v \in V$ denote by $d(v, r)$ the length of the (unique) $v - r$ path in T. It is known (see e.g. [14]) that if $v, v' \in V$ are two adjacent vertices and $d(v,r) \geqq d(v', r)$, then in fact $d(v, r) > d(v', r')$ and v' is an internal vertex of the path joining v and r in T. Consequently, the assumption on cycle lengths implies that $\{v, v'\} \notin E$ whenever $d(v, r) - d(v', r)$ is a multiple of k. Thus, assigning color $d(v, r) \pmod k$ to each vertex $v \in V$, the monochromatic sets of vertices are independent, proving k-colorability of (each component of) G. Such a coloring can be found on-line, during the DFS algorithm, assigning the corresponding color to each vertex when it is reached by the procedure for the very first time.

Variants of DFS in directed graphs provide us with a linear-time algorithmic proof of the Gallai-Roy theorem and a $O(k \,.\, |E[\cdot |V|^2)$ algorithm for Theorem 4 (and hence also for Minty's theorem). One reason why such results are of interest is that the (generally NP-complete) problem of finding the chromatic number of a graph becomes linearly solvable when a „good" orientation is available on the edge set. Of course, the time bound of $O(|V| + |E|)$ is optimal, but most probably $O(k \cdot |E| \cdot |V|^2)$ is far from being best possible.

3. A linear-time algorithm

In this last section we sketch the proof of Theorem 2. The algorithm, applying several ideas also from [12] and [11], consists of the following basic parts.

1. Construct the graph $G = G(\Phi) = (V, E)$ and define a $0-1$ assignment f on its edges as follows. Say, $e = \{i, j\} \in E$. If the clause ϕ_t corresponding to e is $x_i \vee x_j$ or $\neg x_i \vee \neg x_j$ then let $f(e) = 0$, and if $\phi_t = \neg x_i \vee x_j$ or $x_i \vee \neg x_j$ then let $f(e) = 1$.
2. In the connected components of G, find spanning trees $T_1, \ldots, T_k$ with arbitrarily chosen roots $r_1, \ldots, r_k$, using DFS. In the record of each vertex v of T_i store the

distance of v from r_i, the list of its „sons" (immediate successors in T_i), and the number $n(v)$ of sons that are *not* leafs (= endvertices distinct from the root) of T_i. If this number is 0 and v is not a leaf of T_i, then v will be called a *semi-leaf.*

3. Compile a two-way list L_i of semi-leafs for each T_i, containing those v in increasing order of their distances from r_i.

4. Find a decomposition of the T_i into vertex-disjoint stars by choosing the last element v of L_i as the center of the next star whose other vertices will be the sons of v. Delete this small branch from T_i, and decrease $n(v')$ by 1 at the father v' of v; if v' becomes a leaf (i.e. it had just one son v), then $n(v'')$ should be decreased by 1 at v'', the father of v'. Of course, v has to be deleted from L_i and if $n(v')$ or $n(v'')$ becomes zero then the corresponding vertex has to be inserted in L_i.

The stars (some of which possibly consist of just one vertex) obtained by this procedure will be denoted by $S_1, S_2, \ldots$

5. In each S_j split the vertices into two sets X_j and X'_j (some of them may be empty) in such a way that the following property is satisfied; if $f(e) = 0$, then e meets both of X_j and X'_j; if $f(e) = 1$, then e is contained in X_j or in X'_j.

6. Define a preliminary $0-1$ assignment c on the vertices as follows. In S_1, let $c(v) = 0$ for $v \in X_1$ and $c(v) = 1$ for $v \in X'_1$. Having fixed the assignment of S_{i-1}, try a c' in S_i with $c'(v') = 0$ or 1 according as $v' \in X_i$ or $v' \in X'_i$. Scan the edges joining S_i with $V \setminus S_i$. If the other endpoint of an edge is in $\bigcup_{j>i} S_j$, then count it with weight 0; otherwise, if an edge $e = \{v, v'\}$ $(v' \in S_i, v \in \bigcup_{j<i} S_j)$ has $c(v) + c'(v') + f(e) \equiv 1 \pmod 2$ then count e with weight $+1$ and if $c(v) + c'(v') + f(e) \equiv 0 \pmod 2$ then count e with weight -1. If the total sum of those weights is nonnegative, then we put $c(v) = c'(v)$ for all $v \in S_i$; if the total is negative, then we put $c(v) = 1 - c'(v)$ for all of those vertices. Having fixed the values of c on S_i, we mark the edges of weiht $+1$, as well as the edges of S_i. Denote by Φ' the set of clauses belonging to the edges marked, and by Φ'' the clauses of $\Phi \setminus \Phi'$ (hence, the s clauses with a single variable are in Φ'').

7. Check whether or not the assignment c satisfies at least half of the clauses in Φ''; if it does, then the algortihm assigns $c(v)$ to each vertex v; otherwise it assigns $1 - c(v)$ to each v.

Proof of correctness

From the following simple claims, we shall deduce that the $0-1$ assignment provided by the above algorithm indeed satisfies at least $3t/4 + s/2 + \lceil (n-k)/2 \rceil / 4$ clauses of Φ.

Claim 1. The leafs in each star S_i are pairwise non-adjacent.

(Since each T_i has been found by DFS.)

Claim 2. Every clause is satisfied in Φ'.

(This follows from the definition of f.)

Claim 3. At least half of the clauses of Φ'' are satisfied.

(If a clause is not satisfied by $c(v)$ then it always is satisfied by $1 - c(v)$.)
Claim 4. The stars S_i have at least $\lceil (n - k)/2 \rceil$ edges in all.

(Each of the k components of G can contain at most one star of just one vertex, and in any other star S_i on s_i vertices there are precisely $s_i - 1 \geqq s_i/2$ edges.)
Claim 5. Denoting by q the number of star-edges, Φ' has at least $(t + q)/2$ clauses.

(Applying induction on i, one can prove that — in accordance with the appropriate modificalion done in c' when it was necessary — in the subgraph induced by $\bigcup_{j \leqq i} S_j$ at least half of those edges will be marked which are in G but not in any S_j. For $i = m$ this yields $(t - q)/2$ clauses for Φ', plus those q clauses from the S_j.)

By Claims 1 through 5, denoting by m'' the number of clauses in Φ', we obtain that $m'(\Phi) \geqq (m - m'')/2 + m'' \geqq m/2 + (t + q)/4 \geqq 3t/4 + s/2 + \lceil (n - k)/2 \rceil /4$ as stated.

Time analysis

Steps 1, 2, 6, and 7 require $O(m + n)$ time, and steps 3, 4, and 5 can be executed in $O(n)$ time. Hence, the total running time is a linear function of the input size.
Notes on implementation and complexity
Step 2: Distance from the root can be recorded on-line; to determine $n(v)$, add 1 each time when returning to v from a leaf son.
Step 3: In order to speed up insertions in L_i, maintain a pointer vector v_i of length n, the jth coordinate of which tells the location of the last semi-leaf contained in L_i whose distance from r_i is j.
Step 5: A list of star-edges can contain at most $n - 1$ records, yielding time complexity $O(n)$ for this step.

Step 6: During the recursive definition of c, each edge of G is visited at most three times (twice when the algorithm arrives at its two endpoints, and once when the edge is marked). This fact implies the bound of $O(m + n)$.

References

[1] Berge C.: Graphs and Hypergraphs. North-Holland, 1973.
[2] Cook S. A.: The complexity of theorem proving procedures. Proc. 3rd Ann. ACM Symp. on Theory of Computing, Association for Computing Machinery, New York, 1971, 151—158.
[3] Davis M. and Putnam H.: A computing procedure for quantification theory. J. ACM 7 (1960) 201—215.
[4] Dean N. and Toft B., private communication, 1990.
[5] Gallai T.: On directed paths and circuits. In: Theory of Graphs (P. Erdös and G. O. H. Katona, Eds.), Proc. Colloq. Math. Soc. J. Bolyai, Tihany (Hungary) 1966, Akadémiai Kiadó, Budapest, 1968, pp. 115—118.
[7] Golomb S. W. and Baumert L. D.: Backtrack programming. J. ACM 12 (1965) 516—524.
[6] Garey M. R., Johnson D. S. and Stockmayer L.: Some simplified NP-complete graph problems. Theor. Comp. Sci. 1 (1976) 237—267.

[8] Johnson D. S.: Approximation algorithms for combinatorial problems. J. Comp. Syst. Sci. 9 (1974) 256—278.

[9] Kratochvíl K., Savický P., and Tuza Zs.: One more occurrence of variables makes SATISFIABILITY jump from trivial to NP-complete. SIAM J. Comput., in print.

[10] Minty G. J.: A theorem on *n*-colouring the points of a linear graph. Amer. Math. Monthly 67 (1962) 623—624.

[11] Ngoc N. V. and Tuza Zs.: Linear-time approximation for the Max Cut problem to appear.

[12] Poljak S. and Turzík D.: A polynomial algorithm for constructing a large bipartite subgraph, with an application to a satisfiability problem. Canad. J. Math. 34 (1982) 519—524.

[13] Roy R.: Nombre chromatique et plus longs chemins d'un graphe. Revue AFIRO 1 (1967) 127—132.

[14] Tarjan R.: Depth-first search and linear graph algorithms. SIAM J. Comput. 1 (1972) 146—160.

[15] Tarjan R. E.: Complexity of combinatorial algorithms. SIAM Review 20 (1978) 457—491.

[16] Tuza Zs.: Graph coloring in linear time. J. Combin. Theory Ser. B, to appear.