MIT/LCS/TM-147

A $T = O(2^{n/2})$, $S = O(2^{n/4})$ ALGORITHM FOR CERTAIN NP-COMPLETE PROBLEMS

Richard Schroeppel
Adi Shamir

January 1980

A $T = O(2^{n/2})$, $S = O(2^{n/4})$ Algorithm for Certain NP-Complete Problems

by

Richard Schroeppel
Information International
Culver City, California

and

Adi Shamir*
Department of Mathematics
Massachusetts Institute of Technology

* This research was partially supported by the Office of Naval Research under contract No. N00014-76-C-0366.

Abstract

In this paper we develop a general purpose algorithm that can solve a number of NP-complete problems in time $T = O(2^{n/2})$ and space $S = O(2^{n/4})$. The algorithm can be generalized to a family of algorithms whose time and space complexities are related by $T \cdot S^2 = O(2^n)$. The problems it can handle are characterized by a few decomposition axioms, and they include knapsack problems, exact satisfiability problems, set covering problems, etc. The new algorithm has a considerable cryptanalytic significance, since it can break knapsack-based cryptosystems with up to n = 100 generators.

Key words and phrases: NP-complete problems, time/space tradeoffs, knapsack problems, Merkle-Hellman cryptosystems.

1. Introduction

Every NP-complete problem can be solved in $O(2^n)$* time by exhaustive search, but this complexity becomes prohibitive when n exceeds 40 or 50. Assuming that NP $\neq$ P, we cannot hope to find algorithms whose worst-case complexity is polynomial, but it is both theoretically interesting and practically important to determine whether substantially faster algorithms exist. Researchers have so far discovered a few special-purpose algorithms (most notably a $T = S = O(2^{n/2})$ algorithm for knapsacks by Horowitz and Sahni [1974] and a $T = O(2^{n/3})$, $S = O(n)$ algorithm for cliques by Tarjan and Trojanowski [1977]), but no comprehensive theory of such subexponential algorithms has been developed. In this paper we describe a general-purpose algorithm which can solve a fair number of NP-complete problems (including knapsack, partition, exact satisfiability, set covering, hitting set, disjoint domination in graphs, etc.) in time and space complexities which are related by the tradeoff curve $T \cdot S^2 = O(2^n)$ for $O(2^{n/2}) \leq T \leq O(2^n)$. The novel properties of this algorithm are:

(i) The time/space complexities of the algorithm are considerably better than those of all the algorithms published so far for these problems. Furthermore, the algorithm is completely practical in the sense that it is easy to program and its overhead is small, and thus it can handle problems which are almost twice as big as those handled by previous algorithms.

---

*Throughout this paper, we ignore polynomial multiplicative factors in the O-notation of exponential functions. These factors are usually of degree 0 or 1, and their practical effect is very small.

(ii) The algorithm demonstrates an interesting tradeoff between time and space — in order to decrease time by a factor c, it is enough to increase space by a factor of sqrt(c). Since space is much more expensive than time, this tradeoff has a very favorable economics.

(iii) The problems to which the algorithm can be applied are characterized axiomatically by their behavior under composition. This approach introduces a natural subclassification of NP-complete problems and indicates how a problem-independent theory of subexponential algorithms may be constructed.

One of the most important applications of the new algorithm is in cryptanalysis, since many of the newer public-key cryptosystems are based on large NP-complete problems (Diffie and Hellman [1976]). With current technology, the practical limit on the number of operations a cryptanalyst can perform is between 2^{50} and 2^{60} (a parallel computer with 1000 processors whose cycle time is one microsecond performs 2^{50} operations in about two weeks), and the practical limit on the number of memory cells he can use is between 2^{25} and 2^{30}. By choosing the point $T = O(2^{n/2})$ and $S = O(2^{n/4})$ on the time/space tradeoff curve, instances with up to $n = 100$ are within reach, and thus the new algorithm can break all the knapsack-based cryptosystems recommended so far in the literature (e.g., Merkle and Hellman [1978]). This cryptanalytic attack can be foiled by increasing the minimum recommended size from $n = 100$ to $n = 200$ (at the expense of tripling the key size and the encryption time), but it is a clear warning against overconfidence and narrow safety margins in public-key cryptosystems.

The problems our algorithm can handle are described in Section 2. A $T = O(2^{n/2})$ $S = O(2^{n/4})$ algorithm is presented in Section 3. In Section 4 we generalize this algorithm to a family of algorithms whose time and space complexities are related by the $T \cdot S^2 = O(2^n)$ tradeoff curve.

2. A Calculus of Problems

To make our basic algorithm as versatile as possible and to expose the minimum conditions that guarantee its correctness, we define the notion of a problem in a fairly abstract way:

Definition: A problem of size n is a predicate P over n-bit binary strings. A string x is a solution (or a witness) of the problem if P(x) is true. The goal is to find one such x, if it exists.

Example: The predicate of the knapsack problem "is there a bit string $x_1x_2x_3x_4x_5$ such that $x_1 \cdot 7 + x_2 \cdot 3 + x_3 \cdot 9 + x_4 \cdot 6 + x_5 \cdot 2 = 11$?" is of size 5, and its solutions are 01011 and 00101. /_/

Remark: The size $|P|$ of a problem P is defined as the number of bits in its solution rather than the number of bits in its description, since the $O(2^n)$ complexity of exhaustive search (upon which we want to improve) is determined primarily by the size of the solution space. However, to make our results strictly correct we have to assume that these two measures are polynomially related, i.e., that we are not given huge descriptions of problems with very few bits of unknowns.

One of the most useful algorithmic techniques for solving problems is divide-and-conquer. Given a problem P, we decompose it into a number of subproblems (usually two of half size each), solve them separately, and then combine their solutions. To simplify the mathematical analysis of this process, we introduce the following operator:

Definition: A binary operator $\oplus$ on problems is a composition operator if

(i) it is additive: for all P' and P", $|P' \oplus P"| = |P'| + |P"|$;

(ii) it is sound: for any two solutions x' of P' and x" of P", the string concatenation* x'x" is a solution of $P' \oplus P"$;

(iii) it is complete: for any solution x of P and for any representation of x as x = x'x", there are problems P' and P" such that x' solves P', x" solves P", and $P = P' \oplus P"$;

(iv) it is polynomial: the problem $P' \oplus P"$ can be calculated in time which is polynomial in the sizes of P' and P".

Intuitively, $\oplus$ is sound if any two solutions of the subproblems P' and P" can be easily conbined in order to get a solution for the original problem P, and complete if any solution of P can be obtained in such a way:

$$\{x|P(x)\} = \bigcup_{P'\oplus P"=P} \{x'x"|P'(x') \text{ and } P"(x")\} .$$

A pair of problems P',P" is said to be a decomposition of P if $P'\oplus P" = P$, and in general a problem can have many possible decompositions. To solve P, we can try out all its possible P',P" decompositions until we find a

*The string concatenation can be replaced by any other simple operation which is length-additive.

pair of solvable subproblems. If P is solvable and $\oplus$ is complete, these subproblems must exist, while if P is unsolvable and $\oplus$ is sound, these subproblems cannot exist (otherwise their concatenated solutions would have solved P). There are many NP-complete problems for which composition operators exist, and the following examples are typical:

Example: Let $(b,a_1,\ldots,a_n)$ be the knapsack problem in which the target value b is to be represented as a sum of a subset of the generators a_i. For any two problems $P' = (b',a'_1,\ldots,a'_\ell)$ and $P'' = (b'',a''_1,\ldots,a''_m)$ we define $P'\oplus P'' = (b'+b'',a'_1,\ldots,a'_\ell,a''_1,\ldots,a''_m)$ (i.e., we add the b's and concatenate the a_i's). We claim that this $\oplus$ is a composition operator:

(i) $\oplus$ is additive since the number of generators in $P'\oplus P''$ is by definition the sum of the corresponding numbers in P' and P".

(ii) $\oplus$ is sound since $\sum_{i=1}^{\ell} x'_i a'_i = b'$ and $\sum_{i=1}^{m} x''_i a''_i = b''$ imply that $\sum_{i=1}^{\ell+m} x_i a_i = b'+b''$ (where the x_i are the bits of x'x" and the a_i are the generators in $P'\oplus P''$).

(iii) $\oplus$ is complete since for each $x = x'x''$ such that $|x'| = \ell$, $|x''| = m$, and $\sum_{i=1}^{\ell+m} x_i a_i = b$, there are b' and b" such that x' satisfies $\sum_{i=1}^{\ell} x'_i a_i = b'$, x" satisfies $\sum_{i=\ell+1}^{\ell+m} x''_i a_i = b''$, and the sum of these two subproblems is the original problem $(b,a_1,\ldots,a_{\ell+m})$.

(iv) $\oplus$ is polynomial since the only operations involved are numeric addition and list concatenation. /_/

Example: Let F be a formula in CNF (i.e., a conjunction of clauses which are disjunctions of literals which are variables x_i or their negations $\bar{x}_i$). The satisfiability problem is to find a truth-value assignment to the

variables which makes at least one literal in each clause true. To decompose this problem, we can partition the list of variables into two complementary sublists, and try to satisfy by the two partial assignments two sets of clauses whose union contains all the clauses. In this generalized formulation, each subproblem corresponds to a sublist of variables and a subset of clauses in F, and the $\oplus$ operator concatenates the sublists (if they are consecutive) and unions the subsets in P' and P". This operator is clearly additive and polynomial. It is sound since by definition P'$\oplus$P" is satisfied by the concatenation of any pair of assignments that satisfy P' and P", and it is complete since for any x that solves P we can use the clauses which are actually satisfied by the prefix and suffix of x in order to define the appropriate P',P" decomposition. /_/

The relationship between problems (especially NP-complete problems) and their solutions is often asymmetric, since it may be much harder to find a solution for a given problem than to find a problem which is solved by a given solution. This motivates the following definition:

Definition: A set of problems is polynomially enumerable if there is a polynomial time algorithm which finds for each bit string x the subset of problems which are solved by x.

Example: (i) The set of all the knapsack problems is not polynomially enumerable since for each x there are infinitely many knapsack problems which are solved by x.

(ii) The set of knapsack prlblems with a fixed set of a_i generators (but

varying target values b) is polynomially enumerable, since for each x there is exactly one b such that $b = \sum_{i=1}^{n} x_i a_i$, and this b can be easily calculated.

(iii) The set of (generalized) satisfiability problems with a fixed formula F is polynomially enumerable, since the subset of clauses which are satisfied by the truth-value assignment x is uniquely defined and can be found by simple evaluation. /_/

If a set of problems is polynomially enumerable, then all its solvable instances of size n can be tabulated (as problem/solution pairs) in $O(2^n)$ time and space. Again, there are many NP-complete problems whose sets of subproblems are polynomially enumerable, and they have the curious property that it is almost as difficult to solve a single instance of size n as it is to solve all the instances of size n — in both cases we have to enumerate all the possible n-bit solutions.

The most restrictive and least intuitive condition we impose on problems is:

Definition: A composition operator $\oplus$ is monotonic if the problems of each size can be totally ordered in such a way that $\oplus$ behaves monotonically: $|P'| = |P''|$ and $P' < P''$ imply that $P' \oplus P < P'' \oplus P$ and $P \oplus P' < P \oplus P''$.

Using this notion, we can state the main result of this paper (which is proved in the next section):

Theorem 1: If a set of problems is polynomially enumerable and has a monotonic composition operator, then its instances of size n can be solved in time $T = O(2^{n/2})$ and space $S = O(2^{n/4})$.

Example: The $\oplus$ operator on knapsack problems is monotonic if we order them lexicographically, since $(b',a'_1,\ldots,a'_\ell) < (b'',a''_1,\ldots,a''_\ell)$ implies that $(b'+b,a'_1,\ldots,a'_\ell,a_1,\ldots,a_m) < (b''+b,a''_1,\ldots,a''_\ell,a_1,\ldots,a_m)$ and $(b+b',a_1,\ldots,a_m,a'_1,\ldots,a'_\ell) < (b+b'',a_1,\ldots,a_m,a''_1,\ldots,a''_\ell)$. (Note that this $\oplus$ operator is not monotonic if P' and P" are allowed to be of different sizes). Consequently, our algorithm can solve knapsack problems. /_/

Composition operators based on set unions are usually non-monotonic, but they become monotonic if we replace the set unions by multiset unions:

Lemma 2: (i) If $|S| \geq 3$, then the subsets of S cannot be totally ordered in a way that makes the set union operator monotonic.

(ii) If S is denumerable, then the multisubsets of S with finite multiplicities can be totally ordered in a way that makes the multiset union operator monotonic.

Proof: (i) Suppose that such an order exists. Without loss of generality, we can assume that for $a,b,c \in S$, $\{a\} < \{b\} < \{c\}$. By taking the set unions of these singletons with $\{a,c\}$ and by using the monotonicity of U, we get

$$\{a\} \cup \{a,c\} < \{b\} \cup \{a,c\} < \{c\} \cup \{a,c\} ;$$

this evaluates to

$$\{a,c\} < \{a,b,c\} < \{a,c\}$$

which is a contradiction.

(ii) Let S be $\{a_1,a_2,\ldots\}$. The multisubsets of S with finite multiplicities can be represented by semi-infinite vectors of multiplicities $(n_1,n_2,\ldots)$ in which each n_i represents the number of occurrences of a_i. In this

representation, multiset union is simply a componentwise addition of multiplicity vectors, and it is clearly a monotonic operator if we order the vectors lexicographically. Q.E.D.

Example: By part (i) of the lemma, the (generalized) satisfiability problems cannot be totally ordered in a way that makes the $\oplus$ operator monotonic, and thus our algorithm cannot be used in order to solve them. /_/

Example: The exact satisfiability problem is similar to the satisfiability problem, except that we want to satisfy exactly one literal in each clause. Its subproblems consist of sublists of variables and multisets of clauses, and the multiplicity of each clause indicates how many literals are satisfied in it (e.g., the original problem corresponds to the multiset $(1,1,\ldots,1,0,0,\ldots)$). By part (ii) of the lemma, the $\oplus$ composition operator that concatenates the sublists and adds the multiplicities is monotonic, and thus we can apply our algorithm to this variant of the satisfiability problem. /_/

We leave it as an exercise for the reader to verify that all the NP-complete problems listed in the introduction have monotonic composition operators. This list is not exhaustive, and it is easy to come up with additional examples.

3. The Algorithm

The algorithm uses the soundness and completeness of $\oplus$ in order to reduce the general problem to the following combinatorial search problem:

Definition: Given k problem/solution tables T_i with $O(2^{n/k})$ solvable

problems each, a monotonic composition operator $\oplus$, and a problem P, the k-table problem is to determine whether there are k representatives $P_i \in T_i$ such that $P = P_1 \oplus P_2 \oplus \ldots \oplus P_k$ (under a given parenthesization).

Example: To reduce a given knapsack problem P with $n = 3m$ generators a_i to the 3-table problem, we

(i) divide the generators into three sublists $(a_1,\ldots,a_m)$, $(a_{m+1},\ldots,a_{2m})$ and $(a_{2m+1},\ldots,a_{3m})$;

(ii) tabulate in T_i $(i = 1,2,3)$ all the $O(2^{n/3})$ target values b_i which can be generated by summing a subset of the n/3 generators in the i-th third of the problem;

(iii) check whether the original target value b can be represented as $b = b_1 + b_2 + b_3$ for some $b_1 \in T_1$, $b_2 \in T_2$, $b_3 \in T_3$;

(iv) concatenate the three solutions x_i tabulated for these b_i target values (if they exist) in order to get a solution $x = x_1x_2x_3$ for the original problem. /_/

Example: To reduce an exact satisfiability problem to the 4-table problem, we divide the variable list into four quarters, enumerate for each quarter all the $O(2^{n/4})$ possible truth-value assignments, tabulate for each assignment the multiset of satisfied clauses, and determine whether there are four multiplicity vectors in the four tables whose sum is $(1,1,\ldots,1,0,0,\ldots)$. /_/

This general technique is a mixture of divide-and-conquer and dynamic programming — we repeatedly divide problems into pairs of subproblems until we get k problems of size n/k each, and then finish by searching k problem/solution tables. Since we do not assume that $\oplus$ is associative, we

have to fully parenthesize the $P_1 \oplus P_2 \oplus \ldots \oplus P_k$ sum to make it meaningful, but the completeness of $\oplus$ implies that the solvability of this k-table problem does not depend on the parenthesis structure (i.e., we can choose the order that makes the search most efficient).

The obvious algorithm for the k-table problem is to try out all the $O(2^n)$ combinations of representatives from the k tables, and it is clearly optimal for $k = 1$. However, for $k \geq 2$, better algorithms exist:

Theorem 3: The 2-table problem can be solved in $O(2^{n/2})$ time and space.

Proof: Consider the following algorithm:

(1) Sort T_1 into increasing problem order;
sort T_2 into decreasing problem order.

(2) Repeat until either T_1 or T_2 become empty (in which case print "unsolvable" and halt):

$S \leftarrow$ first $(T_1) \oplus$ first (T_2);
if $S = P$ print "solvable" and halt;
if $S < P$ delete first (T_1) from T_1;
if $S > P$ delete first (T_2) from T_2.

To prove the correctness of this algorithm, we have to show that whenever a problem is deleted from T_1 or T_2, it cannot possibly participate in any sum which equals P (and thus the deletion cannot affect the correctness of the rest of the algorithm). Since T_2 is decreasing and $\oplus$ is monotonic, first $(T_1) \oplus P_2 \leq$ first $(T_1) \oplus$ first (T_2) for any $P_2 \in T_2$. Consequently, the left-hand side cannot be equal to P if the right hand side is smaller than P, and the deletion of first (T_1) from T_1 is justified. Similarly,

$P_1 \oplus \text{first}(T_2) \preceq \text{first}(T_1) \oplus \text{first}(T_2) = S \cdot P$ justifies the deletion of first (T_2) from T_2.

The time complexity of the sorting step is $O(2^{n/2}(n/2)) = O(2^{n/2})$, and the time complexity of the search step is $O(|T_1| + |T_2|) = O(2^{n/2})$ since we delete at least one element at each iteration. Q.E.D.

Remark: This 2-table problem has been posed and solved in a number of papers under various disguises (e.g., Knuth [1973, page 9], Horowitz and Sahni [1974]). In the rest of this paper we refer to this algorithm as the basic algorithm.

The basic algorithm can be easily extended to other values of k:

Lemma 4: The 3-table problem can be solved in $O(2^{2n/3})$ time and $O(2^{n/3})$ space.

Proof: For each one of the $O(2^{n/3})$ problems $P_1 \varepsilon T_1$, use the basic algorithm on the T_2, T_3 tables in order to find a solution for $P = P_1 \oplus (P_2 \oplus P_3)$ in time $O(|T_i|) = O(2^{n/3})$. Q.E.D.

Lemma 5: The 4-table problem with a non-balanced parenthesis structure $P = P_1 \oplus (P_2 \oplus (P_3 \oplus P_4))$ can be solved in $O(2^{3n/4})$ time and $O(2^{n/4})$ space.

Proof: For each one of the $O(2^{n/4})$ problems $P_1 \varepsilon T_1$, solve the remaining 3-table problem in $O(|T_i|^2) = O(2^{n/2})$ time. Q.E.D.

All the time and space complexities considered so far satisfy the invariant relation $T \cdot S = O(2^n)$, and thus improvements in the space complexity

make the time complexity worse by a similar factor. This trend is broken by the unexpected behavior of the following case:

Theorem 6: The 4-table problem with balanced parenthesis structure $P = (P_1 \oplus P_2) \oplus (P_3 \oplus P_4)$ can be solved in $O(2^{n/2})$ time and $O(2^{n/4})$ space.

A direct application of the basic algorithm to the two $O(2^{n/2})$ supertables generated by the $(P_1 \oplus P_2)$ and $(P_3 \oplus P_4)$ combinations leads to a $T = S = O(2^{n/2})$ algorithm. However, the basic algorithm accesses the elements of the sorted supertables sequentially, and thus there is no need to store all the possible combinations simultaneously in memory — all we need is the ability to generate them quickly (on-line, upon request) in sorted order. To implement this key idea, we use two priority queues:

(i) Q' stores pairs of problems from T_1 and T_2, enables arbitrary insertions and deletions to be done in logarithmic time, and makes the pair with the smallest $P_1 \oplus P_2$ sum accessible in constant time.

(ii) Q" stores pairs of problems from T_3 and T_4, enables arbitrary insertions and deletions to be done in logarithmic time, and makes the pairs with the largest $P_3 \oplus P_4$ sum accessible in constant time.

Efficient heap implementations of priority queues are described in Aho, Hopcroft, Ullman [1974].

The balanced 4-table algorithm

(1) Sort T_2 into increasing problem order;
sort T_4 into decreasing problem order;
insert into Q' all the pairs (P_1, first (T_2)) for $P_1 \in T_1$;
insert into Q" all the pairs (P_3, first (T_4)) for $P_3 \in T_3$.

(2) Repeat until either Q' or Q" become empty (in which case print "unsolvable" and halt):

$(P_1,P_2) \leftarrow$ pair with smallest $P_1 \oplus P_2$ sum in Q';

$(P_3,P_4) \leftarrow$ pair with largest $P_3 \oplus P_4$ sum in Q";

$S \leftarrow (P_1 \oplus P_2) \oplus (P_3 \oplus P_4)$

if $S = P$ print "solvable" and stop;

if $S < P$ do

delete (P_1,P_2) from Q';

if the successor P_2' of P_2 in T_2 is defined,

insert (P_1,P_2') into Q';

if $S > P$ do

delete (P_3,P_4) from Q";

if the successor P_4' of P_4 in T_4 is defined,

insert (P_3,P_4') into Q".

Lemma 7: The space complexity of this algorithm is $O(2^{n/4})$.

Proof: It is easy to show by induction that at each stage a $P_1 \varepsilon T_1$ can participate in at most one pair in Q', and a $P_3 \varepsilon T_3$ can participate in at most one pair in Q" (the number of occurrences of $P_2 \varepsilon T_2$ and $P_4 \varepsilon T_4$ in Q' and Q" can be higher). The space complexity of the priority queues is thus bounded by $O(|T_i|) = O(2^{n/4})$. Q.E.D.

Lemma 8: The time complexity of this algorithm is $O(2^{n/2})$.

Proof: Each (P_1,P_2) pair can be deleted from Q' at most once, since it is never reinserted into Q'. Similarly, each (P_3,P_4) pair can be deleted

from Q" at most once. At each iteration of step 2, one pair is deleted from Q' or Q", and thus the number of iterations cannot exceed the number of possible pairs, which is $O(2^{n/2})$. Q.E.D.

Lemma 9: Q' can become empty only after we consider all the possible (P_1,P_2) pairs of problems from T_1,T_2 (similarly for Q" and T_3,T_4).

Proof: Initially P_1 shares a pair in Q' with the first element of T_2. After each deletion of a (P_1,P_2) pair we reinsert P_1 together with the next larger element of T_2, and thus the only way Q' can become empty is if each P_1 runs out of companions after a complete first-to-last scan of T_2. Q.E.D.

Lemma 10: The sums of the pairs extracted from Q' are in non-decreasing sorted order, and the sums of the pairs extracted from Q" are in non-increasing sorted order.

Proof: The smallest (P_1,P_2) pair in Q' is replaced by a (P_1,P_2') pair whose sum is larger (since T_2 is sorted and $\oplus$ is monotonic), and thus the sum of the next pair extracted from Q' cannot be smaller than $P_1 \oplus P_2$. The proof for Q" is similar. Q.E.D.

Proof of Theorem 6: Lemmas 7, 8, 9 and 10 reduce the 4-table algorithm to the 2-table algorithm whose correctness was proved in Theorem 3.

Q.E.D.

4. Time/Space Tradeoffs

Lemma 11: If a set of problems has a monotonic composition operator, then for any P and P' there is at most one P" such that $P = P' \oplus P"$.

Proof: If $P"_1 < P"_2$ are two different solutions, we get $P' \oplus P"_1 = P' \oplus P"_2$ which contradicts the monotonicity of $\oplus$ (note that $|P"_1| = |P"_2| = |P| - |P'|$).

Q.E.D.

Definition: The complementation operator $\ominus$ is the partial binary operator defined by:

$$P" = P \ominus P' \quad \text{iff} \quad P = P' \oplus P" .$$

The problem P" is the complement of P' with respect to P.

Example: Given two knapsack problems $P = (b, a_1, \ldots, a_n)$ and $P' = (b', a'_1, \ldots, a'_\ell)$, $P \ominus P'$ is defined (as $(b-b', a_{\ell+1}, \ldots, a_n)$) iff $\ell < n$ and for all $1 \leq i \leq \ell$, $a_i = a'_i$.

Given two exact satisfiability problems P and P', P-P' is defined iff they have the same CNF formula, the list of variables in P' is a prefix of the list of variables in P, and the componentwise difference between their multiplicity vectors is non-negative. /_/

In all examples of monotonic $\oplus$ operators considered so far, the $\ominus$ operator is easy to compute in polynomial time (either directly or by a quick binary search on candidate problems).

Theorem 12: Let Q be a polynomially enumerable set of problems with a monotonic composition operator and a polynomial complementation operator, and let A be an algorithm that solves these problems in $O(2^{\alpha n})$ time and $O(2^{\beta n})$

space (for some $0<\alpha,\beta<1$). Then the problems in Q can be solved in any time/space combination along the tradeoff curve $T \cdot S^{(1-\alpha)/\beta} = O(2^n)$, $O(2^{\alpha n}) \leq t \leq O(2^n)$.

Proof: For each $0 \leq \gamma \leq 1$, let A_j be the following algorithm:

(1) Enumerate all the bit strings x' of size $(1-\gamma)\cdot n$.

(2) For each x' enumerate all the problems P' which are solved by x'.

(3) For each P', find its complement P" with respect to P; if it exists, use algorithm A to solve it; if it is solvable, concatenate x' with its solution x", print it out and halt.

Note that for $\gamma = 0$, A_0 reduces to a simple exhaustive search, while for $\gamma = 1$, A_1 reduces to A. A slight technical difficulty is that for each n the set of usable values of γ is discrete. However, for large values of n this set becomes essentially continuous.

The correctness of each A_γ follows from the soundness and completeness of $\oplus$ in the usual way. The only new element is the unbalanced decomposition of P into problems of sizes $(1-\gamma)\cdot n$ and $\gamma \cdot n$, but our definition of completeness is general enough to handle this case.

Algorithm A is applied at step 3 to a problem of size $\gamma \cdot n$ and thus its time complexity is $O(2^{\alpha\gamma n})$ and its space complexity is $O(2^{\beta\gamma n})$. Step 2 multiplies the time complexity by a polynomial factor, and step 1 multiplies it further by $O(2^{(1-\gamma)n})$. The overall time complexity of A_γ is thus $T_\gamma = O(2^{(\alpha\gamma-\gamma+1)n})$ and its space complexity is $S_\gamma = O(2^{\beta\gamma n})$.

To find the invariant relation satisfied by the time/space complexities of all the A_γ algorithms, we use linear algebra in order to eliminate γ:

$$T \cdot S^{(1-\alpha)/\beta} = O(2^{(\alpha\gamma-\gamma+1)n}) \cdot O(2^{\beta\gamma n(1-\alpha)/\beta})$$

$$= O(2^{(\alpha\gamma-\gamma+1)n} \cdot 2^{(\gamma-\gamma\alpha)n})$$

$$= O(2^n) \ . \qquad \text{Q.E.D.}$$

Theorem 13: If a polynomially enumerable set of problems has a monotonic composition operator and a polynomial complementation operator, then its instances of size n can be solved in any time/space combination along the tradeoff curve $T \cdot S^2 = O(2^n)$, $O(2^{n/2}) \leq T \leq O(2^n)$.

Proof: By Theorem 6, there exists an algorithm A with time complexity $T = O(2^{n/2})$ and space complexity $S = O(2^{n/4})$. By substituting $\alpha = 1/2$ and $\beta = 1/4$ into the general formula, we get the tradeoff curve $T \cdot S^2 = O(2^n)$. Q.E.D.

While we conjecture that $T = O(2^{n/2})$ is optimal for all the k-table problems, we do not have any reason to believe that $S = O(2^{n/4})$ is optimal. If S can be reduced to $S = O(2^{n/6})$ or $S = O(2^{n/8})$ without worsening T, we can get even better tradeoff curves such as $T \cdot S^3 = O(2^n)$ or $T \cdot S^4 = O(2^n)$.

5. Open Problems for Further Research

(i) Are there other axiomatically characterizable subsets of NP-complete problems which can be solved in less than $O(2^n)$ time?

(ii) Can we use other properties of $\oplus$ (besides monotonicity) in order to reduce the complexity of the k-table problem?

(iii) What are the best search strategies for $k > 4$ tables?

(iv) Is $T = O(2^{n/2})$ a lower bound for all k?

(v) Is there an algorithm with $T = O(2^{n/2})$ but $S = o(2^{n/4})$?

(vi) Are there easy ways to determine whether a set of problems has a monotonic composition operator?

Acknowledgements: We would like to thank Martin Hellman and Ron Rivest for many fruitful discussions.

Bibliography

1. A. Aho, J. Hopcroft and J. Ullman, "The Design and Analysis of Computer Algorithms", Addison-Wesley, 1974.

2. W. Diffie and M. Hellman, "New Directions in Cryptography", IEEE Trans. Information Theory, November 1976.

3. E. Horowitz and S. Sahni, "Computing Partitions With Applications to the Knapsack Problem", JACM, April 1974.

4. D. Knuth, "The Art of Computer Programming", Vol. 3, Addison-Wesley, 1973.

5. R. Merkle and M. Hellman, "Hiding Information and Receipts in Trap Door Knapsacks", IEEE Trans. Information Theory, September 1978.

6. R. Tarjan and A. Trojanowski, "Finding a Maximum Independent Set", SIAM J. Computing, September 1977.