A Hard Problem That is Almost Always Easy

George Havas and B.S. Majewski

Key Centre for Software Technology, Department of Computer Science, University of Queensland, Queensland 4072, Australia

Abstract. NP-completeness is, in a well-defined sense, a worst case notion. Thus, 3–colorability of a graph, for a randomly generated graph, can be determined in constant expected time even though the general problem is NP-complete. The reason for this is that some hard problems exhibit a structure where only a small (perhaps exponentially small) fraction of all possible instances is intractable, while the remaining large fraction has a polynomial time solution algorithm. We add a new problem to the list of NP-complete problems that are solvable in average polynomial time.

1 Introduction

Some provably intractable tasks, including NP-complete problems, may be solvable in acceptable time in practice. Different possibilities exist. Firstly, the instances that actually occur may always be small, hence even an exponential time solution runs sufficiently fast. This seems to be the case with input-output decomposition, a decomposition technique for acyclic graphs, proved to be NP-complete by Tarjan (1984). Pichai, Sezer & Šiljak (1984) have argued, however, that this NP-completeness result is not relevant to their situation since large instances are not considered in practice.

A stronger result arises when a given difficult problem has a large class of instances that are solvable in polynomial time. For example, Hamiltonian circuit has a polynomial time algorithm due to Karp (1975) and subsequently improved by Angluin & Valiant (1977) that succeeds in $O(n^2 \log n)$ time with probability $1 - O(n^{-c})$ for a constant c. This may yield a method that runs fast in practice, but still does not result in average polynomial time complexity. The reason is as follows. A dynamic programming routine that guarantees finding a Hamiltonian circuit, if one exists, has complexity $O(n^2 2^n)$. If we want to have an algorithm that always answers correctly the question about the existence of a Hamiltonian cycle for a given graph we may start with the $O(n^2 \log n)$ algorithm and, when it fails to provide an answer, call the exponential time dynamic programming routine. As the latter is executed with probability as high as $\Omega(n^{-c})$, this yields a contribution as high as $2^n n^{2-c}$ to the expected running time, still leaving this approach exponential.

Finally, there exists a class of problems which, although NP-complete, have average polynomial time complexity. Wilf (1984) has shown that for graphs where each edge is present with however small but fixed nonzero probability, 3-colorability can be solved in expected constant time. The reason is that such

graphs are almost certain to contain a large number of 4-cliques, and the standard backtrack search algorithm can be expected to detect one in constant time.

We consider the extended gcd problem which, for a given vector of positive numbers $a = [a_i]_{i=1}^n$, asks for an integer vector $x = [x_i]_{i=1}^n$, such that $\sum_{i=1}^n x_i a_i = \gcd(a_1, a_2, \ldots, a_n)$. (Implementations, applications and the importance of the problem are discussed by Havas & Majewski (1995).) Solving this problem for any vector x is easy; the challenge arises when we want to minimize some measure of x. In this paper we concentrate on just one measure, the L_0 metric, which is equal to the number of nonzero elements in x. By finding an x optimal with respect to this metric we obtain a sparsest possible solution to the extended gcd problem.

Recent results of Majewski & Havas (1994, Theorem 1) show that finding a solution to the extended gcd problem which is optimal with respect to the L_0 metric is NP-complete. The proof relies on a polynomial time transformation from MINIMUM COVER (see Garey & Johnson (1979, problem SP5)). This contradicts practical experience in that the gcd's of large sets of randomish numbers can be found with the use of 2 or 3 numbers. Indeed, as indicated by a result of E. Cesàro (see Theorem 3), we have a substantial chance that just two of these numbers will have gcd equal to 1. Thus on the one hand we expect to be able to find a short x by simply inspecting a few pairs of numbers, while on the other hand we have a proof that sometimes it may take exponential time to find a short solution x. These seemingly contradictory statements can be easily explained. The majority of extended gcd problems can be solved in polynomial time. However, the polynomial transformation from MINIMUM COVER targets only a small subset of difficult instances of the problem, as illustrated in Fig. 1.

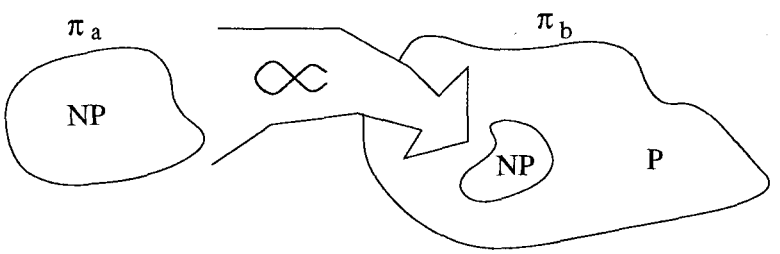

Fig. 1. NP-completeness of π_b by polynomial time transformation from a known NP-complete problem π_a.

We prove that, under the assumption of a uniform and random distribution for the input numbers, the optimization version of the extended gcd problem for the L_0 metric is polynomial in n and $\log(\max_i\{a_i\})$.

Before we do so we clarify some definitions, following Johnson (1984). For a given decision problem D we denote the set of instances of size n by I and the probability that a random such instance is x by $\mu(x)$, and we let $T(x)$ be the

running time of the algorithm under consideration for input x. In one view, an algorithm is considered to have average polynomial running time if

$$E(T) = \sum_{x \in I} \mu(x) T(x) = O\left(n^{O(1)}\right).$$

As pointed out by Johnson, such a definition has a serious drawback: unlike the standard notion of worst case polynomial time, it is not machine independent. Instead the following approach should be used. An algorithm has average polynomial time complexity if for some $c > 0$

$$E^*(T) = \sum_{x \in I} \mu(x) \sqrt[c]{T(x)} = O(n).$$

A randomized decision problem D together with a probability distribution μ on problem instances is considered easy if it has an algorithm with average polynomial time complexity for distribution μ. Then AP (Average P) is the class of all easy randomized decision problems.

It is worth mentioning that Levin (1986) considered the question whether all decision problems in NP are easy, i.e., if they belong to AP. The answer seems to be no. For a number of problems it has been shown that whole classes of methods almost always run in exponential time: INDEPENDENT SET (Chvátal, 1977); GRAPH k–COLORABILITY (McDiarmid, 1979); and KNAPSACK (Chvátal, 1980), to mention just a few. Thus, these problems seem to be hard on average. To capture this notion Levin introduced a class called "random NP-complete", subsequently renamed DNP (Distributional NP), and proved that RANDOM TILING belongs to this class. This means that RANDOM TILING, and any other problem belonging to DNP, can be solved in expected polynomial time if and only if all problems in DNP can. A major open question is whether DNP $\subseteq$ AP.

2 Complexity Results

In this section we use the RAM model of computation with uniform cost measure as defined by Aho, Hopcroft & Ullman (1974). In particular we assume that basic arithmetic operations (such as addition, multiplication, modulo, and accessing an array) take constant time. Carrying out the analysis for other computational models, such as Turing Machines or Markov algorithms, although more laborious, leads to the same results, with different constants.

Consider the time complexity of a simple exhaustive search through all possible k–tuples of numbers, $2 \leq k \leq n$. We assume that $a_i \neq a_j$ for $i \neq j$. (If this is not the case, all duplicates can be easily eliminated in linear time, once the a_i's are sorted.) Furthermore, we stipulate that $\gcd(a_1, a_2, \ldots, a_n) = 1$. (Again, if this is not the case we can guarantee this condition in $O(n + \log(\max_i\{a_i\}))$ time by first computing the gcd then dividing through by it.)

The algorithm, presented in Fig. 3, calls the procedure *kSearch* (Fig. 2) for increasing k's, starting with $k = 2$. As soon as a subset with the gcd 1 is discovered, the search is abandoned and the subset is returned. This allows the main procedure to provide the user with a proof that the found subset indeed gives a solution. In the implementation presented in Fig. 3 the subset is discarded and its cardinality, representing the size of the optimum with respect to the L_0 metric solution, is returned. In the following paragraphs we show that the expected complexity of this superficially naïve approach is polynomial in n and $\log(\max_i\{a_i\})$, even though the worst case complexity is clearly exponential in those parameters.

```
procedure kSearch(k)
    p := k;
    for i := 1,2 .. k do
        s_i := i;
    end for;
    loop
        if gcd(a_{s_1}, a_{s_2}, ..., a_{s_k}) = 1 then
            return {a_{s_1}, a_{s_2}, ..., a_{s_k}};
        end if;
        if s_k = n
            then p := p - 1;
            else  p := k;
        end if;
        exit if p < 1;
        for i := k, k - 1 .. p do
            s_i := s_p + i - p + 1;
        end for;
    end loop;
    return ∅;
end kSearch;
```

Fig. 2. Lexicographic search through all k–subsets

Lemma 1. *The time complexity of a single execution of the procedure kSearch is* $O(\binom{n+1}{k}(k + \log a_n))$.

Proof. Consider initially the number of times elements of the index array s are modified. Initially, the algorithm executes k assignments to s_i, $1 \leq i \leq k$. In the inner **for** loop s_k is modified for each of $\binom{n}{k}$ combinations, s_{k-1} is modified once for each combination with $s_k = n$, ie, $\binom{n-1}{k-1}$ times, s_{k-2} is modified once for each combination with $s_{k-1} = n - 1$ and $s_k = n$, ie, $\binom{n-2}{k-2}$ times, and so on.

Thus the total number of times we access the array s is

$$k + \sum_{j=0}^{k} \binom{n-j}{k-j} = \binom{n+1}{k} + k.$$

For each of these combinations the gcd of the elements $a_{s_1}, a_{s_2}, \ldots, a_{s_k}$ has to be computed. By Theorem 1 of Bradley (1970) it takes at most $O(k+\log(\min_i\{a_i\}))$ constant time steps. If we assume that the sequence is sorted, as the k–subsets are generated in lexicographic order, the last expression takes the form $O(k + \log(a_{n-k+1})) = O(k + \log(a_n))$. In fact we can make the algorithm somewhat more clever, computing the gcd's at a minimum possible cost by maintaining another array, g. However, as efficiency is not the main issue and in order to simplify the analysis, we split the computations into two separate operations. Thus the time complexity can be estimated as $O\left(\binom{n+1}{k} + k\right) + \binom{n}{k}(k + \log a_n)) = O\left(\binom{n+1}{k}(k + \log a_n)\right)$. □

Observe that this estimate is suitable for small k. For k approaching n the estimate is unnecessarily high by a factor of n. For $k > n/2$ we can use a similar algorithm to generate complements of $(n-k)$–subsets, thus maintaining average constant time per subset.

```
    Sort the sequence a: a_1 ≤ a_2 ≤ ··· ≤ a_n;
    if a_1 = 1 then
        return 1;
    end if;
Step 1:
    if kSearch(2) ≠ ∅ then
        return 2;
    end if;
Step 2:
    for k := 3 .. n − 1 do
        if kSearch(k) ≠ ∅ then
            return k;
        end if;
    end for;
    return n;
```

Fig. 3. Optimum zero metric solver

Lemma 2. *Let $a = \langle a_1, a_2, \ldots, a_n \rangle$ be a sequence of random integers. The probability that there exist a pair a_i, a_j with $\gcd(a_i, a_j) = 1$ is $1 - (1 - 6/\pi^2)^{\binom{n}{2}}$.*

Proof. To prove the lemma we use the following theorem, due to E. Cesàro, 1881 (Knuth, 1973, p. 301):

Theorem 3. *For two integers chosen at random the probability that their gcd is 1 is* $6/\pi^2 = 1/\zeta(2)$.

From Theorem 3, for each pair (a_i, a_j) the probability of their gcd being 1 is $1/\zeta(2)$. There are $\binom{n}{2}$ pairs among n numbers, thus the probability that one or more pairs have gcd 1 is equal to the complement of the probability that none of the pairs has gcd 1. The latter probability is $(1 - 1/\zeta(2))^{\binom{n}{2}}$, which yields the desired result. □

Theorem 4. *Under the assumption of a uniform and random distribution for the input numbers, the algorithm shown in Fig. 3 belongs to AP.*

Proof. The algorithm starts by sorting the sequence and testing if the smallest number in it is equal to 1, at the cost of $O(n \log n)$ constant time operations. Step 1 of the algorithm is executed with probability $1 - \Pr(a_1 = 1)$ which may be assumed to be 1 if numbers are selected from a sufficiently large universe. Step 2, however, is activated only if Step 1 fails. By Lemma 2 this occurs with probability $(1 - 1/\zeta(2))^{\binom{n}{2}}$, which is less than $\left(\frac{2}{5}\right)^{n^2/3}$ for $n \geq 3$. Notice that we are interested only in cases with $n \geq 3$, thus for all practical purposes we may assume that the inequality always holds. The complexity of Step 1 is $O\left(\binom{n+1}{2}(2 + \log(a_n)) = O(n^2 \log(a_n))\right)$. The complexity of Step 2 is

$$\begin{aligned}\sum_{k=3}^{n} \binom{n+1}{k} (k + \log(a_n)) &= (n+1)(2^n - 2(n+1)) \\ &\quad + \left(2^{n+1} - \frac{n(n-3)}{2} - 3\right) \log(a_n) \\ &< 2^{n+1} (n + \log(a_n)) .\end{aligned}$$

From this it follows that the expected time complexity of the algorithm of Fig. 3, for $n \geq 3$, is bounded from above by

$$\begin{aligned}E(T) &\leq n \log n + \left(1 - \left(1 - \tfrac{1}{\zeta(2)}\right)^{\binom{n}{2}}\right) O(n^2 \log a_n) \\ &\quad + \left(1 - \tfrac{1}{\zeta(2)}\right)^{\binom{n}{2}} O\left(2^{n+1} (n + \log(a_n))\right) \\ &< O(n^2 \log a_n) + 2 (n + \log(a_n)) \left(\tfrac{2}{5}\right)^{n^2/3} 8^{n/3}.\end{aligned}$$

The last expression, again for $n \geq 3$, is bounded from above by

$$O\left(n^2 \log(a_n)\right) + 2 (n + \log(a_n)) \left(\frac{4}{5}\right)^n .$$

The above construction proves that $E(T) = O\left((n \log(a_n))^{O(1)}\right)$ which, as noted in the previous section, is a computational model dependent proof of average

time complexity. For the assumed model of computation, taking $c = 2$ yields the desired result of $E^*(T) = O(n \log(a_n))$. As noted at the beginning of this section, for other models of computation, we obtain the same result by increasing c by a constant. □

The above analysis does not take into account the fact that the algorithm, after failing to detect a pair with the gcd 1, may stop in polynomial time for a k–tuple, with $k = O(1)$. In fact, the probability that the algorithm will require scanning $(k+1)$–tuples is exponentially smaller that the probability that it will fail for j–tuples, with $1 \leq j \leq k-1$ and succeed for k–tuples.

It is also interesting to note that the expected number of pairs that need to be inspected before we find the first pair with gcd 1 is $\zeta(2) < 2$. The algorithm terminates with probability $1 - \frac{1}{n}$ after inspecting $\ln(n)/\left(2\ln(\pi) - \ln(\pi^2 - 6)\right) \approx 1.068 \ln(n)$ pairs of numbers. Thus the expected running time of the algorithm is $O(n+\log(\max\{a_i\}))$, and it terminates almost always in $O(n+\ln(n)\log(\max\{a_i\}))$ time. (These results apply to a version that does not sort the input numbers. The term n comes from a verification process that looks for $a_i = 1$.)

3 Conclusion

Under the assumption of a uniform and random distribution for the input numbers, we have shown that the extended gcd optimization problem with respect to the L_0 metric belongs to AP. Moreover, the algorithm is expected to terminate very rapidly. It would be interesting to consider the behavior of the method under different distributions, more or less commonly encountered in practice.

Another question worth investigating would be to ask if a similar result holds for any other norm. For example, as shown by Havas & Majewski (1995), applying the Fincke-Pohst algorithm to the solution returned by the LLL gcd method is certain to obtain an optimal solution with respect to the Euclidean norm, and in practice it seems to terminate reasonably quickly.

References

Aho, A.V. and Hopcroft, J.E. and Ullman, J.D. (1974), *The Design and Analysis of Computer Algorithms*. Addison-Wesley Pub. Co., Reading, Mass.

Angluin, D. and Valiant, L.G. (1977), "Fast Probabilistic Algorithms for Hamiltonian Circuits and Matchings", *Proceedings 9th Annual ACM Symposium on Theory of Computing*, pp. 30–41.

Bradley, G.H. (1970), "Algorithm and Bound for the Greatest Common Divisor of n Integers", *Communications of the ACM*, **13**, 433–436.

Chvátal, V. (1977), "Determining the Stability Number of a Graph", *SIAM J. on Computing*, **6**, 643–662.

Chvátal, V. (1980), "Hard Knapsack Problems", *Operations Research*, **28**, 1402–1411.

Garey, M.R. and Johnson, D.S. (1979), *Computers and Intractability: A Guide to the Theory of NP-completeness.* W.H. Freeman, San Francisco.

Havas, G. and Majewski, B.S (1995), "Extended Gcd Calculation", Research Report **TR0325**, The University of Queensland, Brisbane.

Johnson, D.S. (1984), "The NP-Completeness Column: An Ongoing Guide", *Journal of Algorithms*, **5**, 284–299.

Karp, R.M. (1975), "On the Complexity of Combinatorial Problems", *Networks*, **5**, 45–68.

Knuth, D.E. (1973), *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms* (2nd edition). Addison-Wesley, Reading, Mass.

Levin, L. (1986), "Average Case Complete Problems", *SIAM J. on Computing*, **15**, 285–286.

Majewski, B.S. and Havas, G. (1994), "The Complexity of Greatest Common Divisor Computations": *Algorithmic Number Theory* pp. 184–193. Springer-Verlag.

McDiarmid, C.J.H. (1979), "Determining the Chromatic Number of a Graph", *SIAM J. on Computing*, **8**, 1–14.

Pichai, V. and Sezer, M.E. and Šiljak, D.D. (1984), "Reply to "Input-output Decomposition of Dynamic Systems is NP-complete"", *IEEE Transactions on Automatic Control*, **29**(9), 864.

Tarjan, R.E. (1984), "Input-output Decomposition of Dynamic Systems is NP-complete", *IEEE Transactions on Automatic Control*, **29**(9), 863–864.

Wilf, H.S. (1984), "An $O(1)$ Expected Time Graph Coloring Algorithm", *Information Processing Letters*, **18**, 119–122.