The complexity of natural extensions of efficiently solvable problems

Andrei Lapets

March 15, 2010

Abstract

A problem is in the class NP when it is possible to compute in polynomial time that a given solution corresponds to a given problem instance. Those problems for which it is possible to compute in polynomial time a solution for any problem instance are also in the class P. We consider a natural conjunction operation for problems that can be computed in polynomial time. We introduce the notion of an *abundant* problem in P, and specify conditions under which the conjunction of two abundant problems in P produces a problem that is NP-complete. We discuss how this is related to multi-dimensional variants of common, efficiently computable graph problems.

1   Common Definitions and Conventions

Definition 1.1. We define a *problem* to be any Turing machine v (the *verifier* corresponding to the problem) that takes inputs from some problem instance space X and some solution space S and returns a value in $\{0, 1\}$. Thus, we can view v as a computable function $v : X \times S \to \{0, 1\}$. We will assume that $X \subset \{0, 1\}^*$ and $S \subset \{0, 1\}^*$. For any x, if $v(x, s) = 1$ we call s a *solution* for x.

We review variants of the definitions for the classes NP and P.

Proposition 1.2. *For any problem v, if v terminates in time $O(n^k)$ on all inputs of size n, then v is in the class* NP.

Proposition 1.3. *For any problem v in the class* NP, *if there exists a Turing machine $m : X \to S$ (the solver) such that for every $x \in X$, $v(x, m(x)) = 1$ and m terminates in time $O(n^k)$ on all inputs of size n, then v is in the class* P.

We consider a natural notion of conjunction for problems. This conjunction operation can be computed in polynomial (in fact, constant) time given the outputs of the verifiers for any two problems.

Definition 1.4. Given two problems v, v', we define $v \wedge v'$ to be a Turing machine such that for all $((x, x'), s) \in (X \times X) \times S$,

$$(v \wedge v')((x, x'), s) = 1 \quad \text{if and only if} \quad v(x, s) = 1 \ \text{ and } \ v'(x, s) = 1.$$

As a convention, we use $\bar{1}$ to represent a sequence of 1 bits, and $\bar{0}$ to represent a sequence of 0 bits.

2   Examples

2.1   A Variant of the Path Finding Algorithm

We present a simple example of a problem in P that is NP-complete under conjunction.

1

Example 2.1. Consider the following graph problem v. Given a weighted graph G, source and destination nodes s, t, a number n, and a bit b, we must produce a a path from s to t within the graph, and return either the number n or the aggregate path weight (depending on whether the supplied bit b is 0 or 1).

$$(G, s, t, b, n) \mapsto \begin{cases} (\text{ path } p \text{ from } s \text{ to } t, & \text{weight of path p}) & \text{if } b = 1 \\ (\text{ path } p \text{ from } s \text{ to } t, & \text{the number } n) & \text{if } b = 0 \end{cases}$$

The predicate v is polynomial-time computable, and there exists a polynomial-time solver m for this problem, so it is in P.

Proposition 2.2. *The problem $v \wedge v$ is* NP-*hard.*

Proof. Suppose we have a polynomial-time solver m that produces solutions satisfying $v \wedge v$. We reduce the classic SubsetSum problem, which is NP-complete, to the problem v.

For a pair of inputs x, x', the problem $v \wedge v$ requires a solution s such that $v(x, s) = 1$ and $v(x', s) = 1$.

Given a collection of numbers $c_1, \ldots, c_n$ and a target sum k, all possible combinations of numbers can be represented by a graph G of n nodes, with two edges between each node (one with weight c_i, and one with weight 0). Any path from the first node s to the last node t of aggregate weight k would represent a solution to SubsetSum. It is then sufficient to supply the input $((G, s, t, 1, k), (G, s, t, 0, k))$ to m. The 0 bit in the second input will ensure that any solution to both subproblems must contain a path of weight exactly k. $\square$

Corollary 2.3. *The problem $v \wedge v$ is in* NP *because $v \wedge v$ is a polynomial-time computable predicate, so it is also* NP-*complete.*

It is the large solution space of the problem in this example that leads to its intractability under conjunction. This characteristic can be generalized for arbitrary polynomial-time algorithms.

2.2   Decompositions of NP-complete Problems

The previous example suggests one possible means for producing problems in P which are NP-complete under conjunction. Problems that are NP-complete, such as SetCover and SubsetSum, can be decomposed into two component problems that are in P.

Suppose that given some problem instance p, S_p is always some exponentially large collection of selections (e.g. each $s \in S_p$ is a subset of a specific list of numbers, or a candidate set covering). Let T be a range of metric values for the selections in S_p (it could represent the sum of a list of numbers, or the total number of sets used in a set cover). Let $m : S_p \to T$ be a polynomial-time computable mapping from S_p to T, and let t^* always represent the solution target.

Example 2.4. Suppose that a problem v is such that for any problem instance p, the solution space is always

$$C = \{(s, m(s)) \mid s \in S_p\}.$$

Suppose also that another problem v' is such that for any problem instance p, the solution space is always

$$C' = S_p \times \{t^*\}.$$

Any solution to the problem $v \wedge v'$ is necessarily contained in

$$C \cap C' = \{(s, t^*) \mid s \in S_p\}.$$

In the cases of both SetCover and SubsetSum, such a decomposition is trivial to construct. In Section 3, we prove that any such decomposition of an NP-complete problem into two problems in P must necessarily consist of problems that are not *specific*.

3 Conjunction of Specific and Abundant Problems

3.1 Specific Problems

Definition 3.1. We call a problem v *specific* if for every problem instance x such that $|x| \leq n$ we have that

$$\left| \{s \mid v(x,s) = 1\} \right| \leq 1.$$

The notion of a *specific* problem is related to the complexity class UP: specific problems with a polynomial-time v are in the class UP [Val].

Theorem 3.2. *For any two problems v, v' in* P, *if v or v' is* specific, *the problem $v \wedge v'$ is in* P.

Proof. Given an input (x, x'), it is sufficient to compute $m(x)$ and check whether the output is a solution for v', or vice versa. □

One example of a specific problem is the minimum spanning tree problem on weighted graphs in which all the weights are distinct.

Corollary 3.3. *For any problem v that is* NP-*complete, and any two problems v', v'' in* P *such that $v = v' \wedge v''$, neither v' nor v'' are specific unless* P $=$ NP.

Proof. If either v' or v'' were specific, $v = v' \wedge v''$ would be in P by Theorem 3.2. □

3.2 Abundant Problems

Definition 3.4. We call a problem v *sparse* if for every problem instance x we have that

$$\left| \{s \mid v(x,s) = 1\} \right| \in O(n^k).$$

Definition 3.5. We call a problem v *abundant* if for every n, there exist at least $\Omega(2^n)$ distinct inputs x such that $|x| \geq n$ and

$$\left| \{s \mid v(x,s) = 1\} \right| \in \Omega(2^n).$$

In other words, abundant problems have exponentially many problem instances of every size for which there exist exponentially many solutions.

3.3 Coding-Amenable Problems

Definition 3.6. A problem v is *coding-amenable* if for any n, it is possible to find some input x such that $|x| \geq n$ and such that this x embodies one or more instances of any of the following constraints on the individual bits b_i of its solution(s) $s = b_1 \ldots b_n$:

> **constant:** for a chosen i, $b_i = c$ for a chosen $c \in \{0,1\}$;
>
> **dependent bit pair:** for a chosen i, j, $b_i = b_j$ or $b_i \neq b_j$ (it is *not* required that it be possible to "chain" such dependent bit pairs, e.g. $b_i = b_j = b_k$);
>
> **disjunction:** for a chosen i, $b_{i+2} = b_i \vee b_{i+1}$ for $c \in \{0,1\}$;

sequence: for a chosen i, k, $b_i \ldots b_{i+k} \in \{\overline{0}, \overline{1}\}$.

An equivalent requirement is to allow **dependent bit pair** constraints and the following constraint, which encompasses the other three:

sequence selection: for a chosen i, k, $b_i \ldots b_{i+k} \in S$ where S is a collection of a constant number of bit sequences.

Theorem 3.7. *For any two abundant, coding-amenable problems v, v', $v \wedge v'$ is* NP-*hard.*

Proof. We reduce the 3-SAT problem to $v \wedge v'$.

We are given an instance of 3-SAT consisting of an ℓ-clause CNF formula φ with variables drawn from $y_1, \ldots, y_n$. We can represent each disjunction in the formula as a sequence of three bits $b_1 b_2 b_3$ followed by another bit $b_4 = b_1 \vee b_2 \vee b_3$ representing the result of that disjunction:

$$D_i = b_1 b_2 b_3 b_4.$$

Taking advantage of the **disjunction** coding capacity, we construct our first input x so that any solution s for x must contain a region of such formula strings D_i. We then add a region Y_i for every variable y_i. Each region has one bit for every occurrence of the variable y_i within φ. Using the **dependent bit pair** coding capacity, we construct x so that each bit in Y_i corresponds to the bit in some D_i that represents that particular occurrence of the variable in φ (being sure to reverse the bit if the variable occurs under a negation). Finally, we use the same technique to create a region C of ℓ bits wherein each bit corresponds to the last bit of each region D_i.

Thus, the layout of the relevant portion of a solution s would be

$$\ldots \# D_1 \ldots D_\ell \# Y_1 \ldots Y_k \# C \# \ldots$$

We construct x' using the **sequence** coding capacity so that the instance sequence Y_i for each y_i is either $\overline{0}$ or $\overline{1}$. We also use the **constant** coding capacity to ensure that the region corresponding to C is of the form $\overline{1}$.

For any s that is a solution for both x and x', the conjunction sequence C must be $\overline{1}$, and each variable sequence Y_i must be either $\overline{0}$ or $\overline{1}$ (ensuring that all variable occurrences are consistent). Thus, the solution s constitutes a substitution for the CNF formula φ so that it is true. $\square$

Corollary 3.8. *For any two abundant, coding-amenable problems $v, v' \in$* P, $v \wedge v'$ *is* NP-*complete.*

Proof. We know that $v \wedge v'$ is NP-hard. It is in NP because for any solution s, it suffices to compute the polynomial-time algorithms v and v' to determine its validity. $\square$

4 Two-dimensional Variants of Efficiently Solvable Graph Problems

Many common problems in P that involve weighted graphs can be generalized to allow weights with two (or more) dimensions.

Example 4.1. We are given a weighted graph $G = (V, E, (w, w'))$ where $w : E \to \mathbb{R}$ and $w' : E \to \mathbb{R}$ are two functions that specify weights for every edge $e \in E$. We are also given two nodes $s, t \in V$. The problem is to find the shortest path P from s to t in (V, E) such that P is the shortest path under *both* weight functions w and w'.

This problem can be written as the conjunction of two abundant problems. It is also closely related to, though distinct from, the NP-complete $\overline{f}$-SPP problem [DKL10]. It is possible to make two-dimensional variants of other common graph problems, such as minimum maximal matching and minimum spanning tree. Whether any of these two-dimensional variants are NP-complete are open problems.

References

[DKL10] Kevin Donnelly, Assaf Kfoury, and Andrei Lapets. The Complexity of Restricted Variants of the Stable Paths Problem. Technical Report BUCS-TR-2010-004, CS Dept., Boston University, March 2010.

[Val] L. Valiant. The relative complexity of checking and evaluating. In *Inf. Process. Lett.*, 5:20–23, 1976.