A technique for proving intrinsic limitations of algorithm designs*

Karl Winklmann

CU-CS-315-85 October 1985

* Work supported in part by National Science Foundation Grants MCS-8202964 and DCR-8500741

ABSTRACT

A technique is introduced for proving intrinsic limitations of some very powerful algorithm design techniques. Specifically, it is shown how this technique can be used to prove that a natural class of algorithms, which contains, for example, standard algorithms for 2—SAT, SHORTEST PATH, and CARDINALITY MATCHING, does not contain algorithms for a number of NP-complete problems.

This report is intended to provide a brief and informal description of the proof technique and to raise some related questions. It does not give formal definitions or detailed proofs.

1. Introduction

This report outlines a technique for proving that a certain class of algorithms is too weak to solve a number of NP-complete problems. The class of algorithms contains, for example, standard algorithms for 2–SAT, SHORTEST PATH, CARDINALITY MATCHING, and a number of other combinatorial problems.

What these algorithms have in common, and what defines the class, is that they "run well" on two-processor machines, where "running well" means that the algorithms do not require much inter-processor communication. More precisely, the required amount of communication is bounded from above by a polynomial in the size of the cut of the problem instance that is made to provide each processor with part of the input. For graph problems vertex cutsets provide a natural way to cut a problem instance in two, with the size of the cut being the number of vertices in the cutset. (See Figure 1.) For Boolean expressions in conjunctive normal form, a natural way to cut a problem instance is to partition the set of clauses between the two processors, with a "cutset" of variables consisting of those variables that occur in both sets of clauses.

If we can show that some problem Π requires an amount of communication between two processors that is exponential in the size of the cut, then it follows that this class of algorithms, i.e. the class of algorithms that run well on two processors in the above sense, is too weak to solve this problem Π. Such exponential lower bounds can be proven by considering problem instances whose overall size is exponential in the size of the cut, and then borrowing adversary arguments from the theory of distributed computing [Yao 1979, Papadimitriou and Sipser 1984].

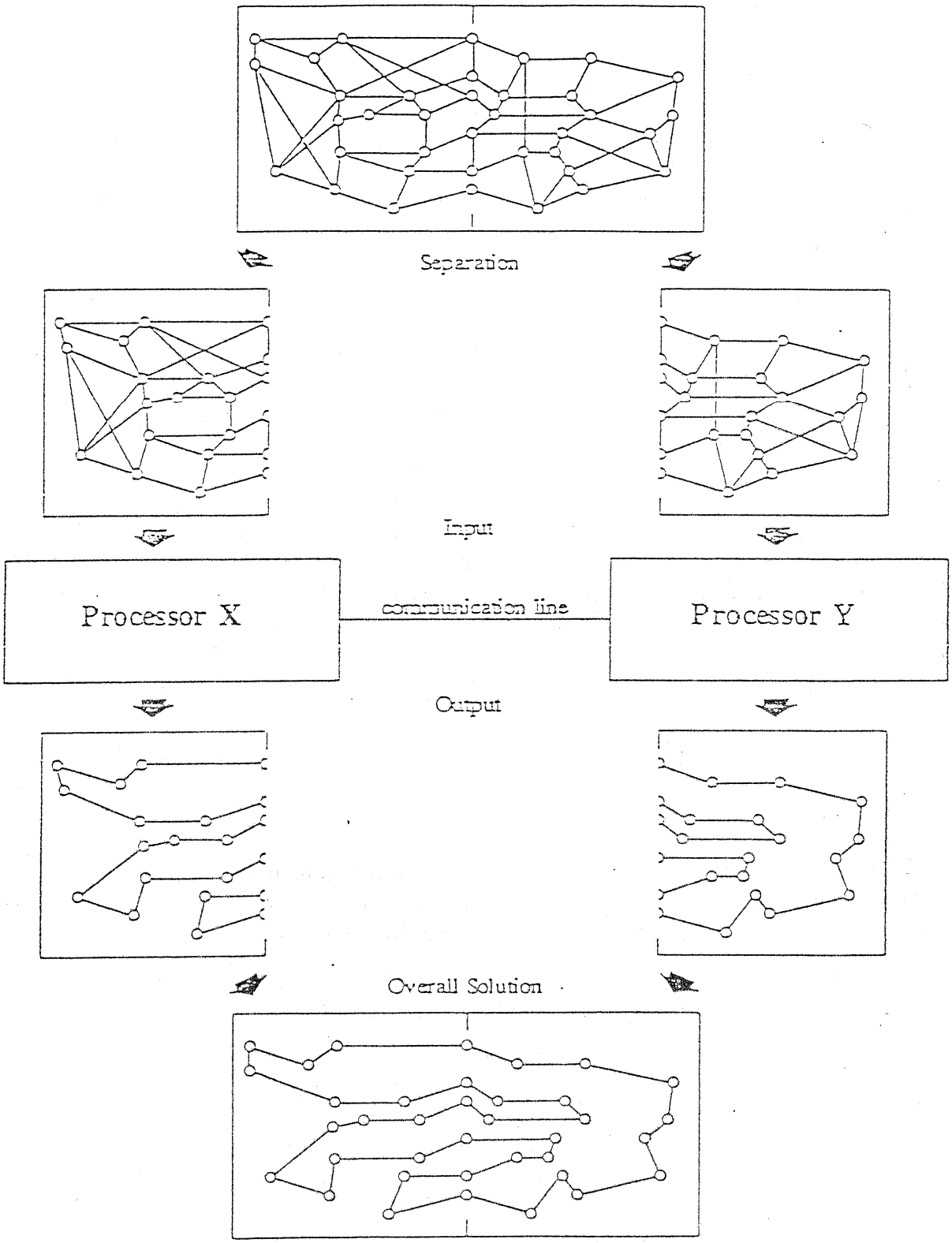

Figure 1. Two processors, solving a graph problem. The cut size is 6. The problem solved here is the HAMILTONIAN CIRCUIT problem.

2. Adapting algorithms to run on two-processor machines

2-SAT [Even, et. al. 1976]. Let E be a Boolean expression in conjunctive normal form with at most two literals per clause. Assigning a specific value to some variable in E may force other variables to take on specific values. This comes from the simple observation that if literal l is *false* and there is a clause $(l+l')$, then literal l' in any satisfying assignment must be *true*. This leads to the following greedy algorithm for 2-SAT:

REPEAT

choose any variable x that has not been assigned a value yet;

IF assigning x the value *true* does not force any clause to become *false*
THEN assign x the value *true*
ELSE assign x the value ~~*true*~~ false;

replace x with its assigned value and simplify the expression

UNTIL

the expression is *true* or *false*

A greedy algorithm for 2-SAT (after [Even, et. al. 1976]).

Now consider an instance of 2-SAT that is split between two processors. Each processor knows some of the clauses and also knows which variables are the "cutset variables", i.e. which variables occur in both sets of clauses. The above greedy algorithm can then be executed by each processor on its own set of clauses except that the assignment of values to cutset variables requires some coordination, which is accomplished as follows. Whenever a processor is ready to assign a value to a cutset variable it informs the other processor about this, awaiting a response that could be "no objection", or "would cause one of my clauses to become *false*", or "would force the following assignments of values to other cutset variables".

It is fairly easy to see that such a protocol does properly solve 2-SAT, and that it does so with an amount of communication that is $O(s^2 \log s)$, where s is the number of cutset variables.

SHORTEST PATH [Dijkstra 1959]. Dijkstra's single-source shortest path algorithm maintains a set S of vertices to which a shortest path from the source vertex has been found. In each iteration a new vertex gets added to that set — a vertex not yet in S whose distance to S is minimal. With two processors, rather than maintaining a single set S each processor maintains its own set S_i, i=1,2. Each processor grows its own set S_i by applying Dijkstra's algorithm to that part of the network that is visible to it. Communication becomes necessary when each processor has determined which cutset vertex v_i it would add to its set S_i, and what the distance d_i of that vertex v_i from the source vertex is. The processor that offers the smaller distance value "wins the competition", meaning that it is allowed to add the cutset vertex v_i to its set S_i. The losing processor has to reset its set S_i to what it was the last time this competition took place and then add the cutset vertex that the winning processor proposed to add. Clearly this "competition" takes place a number of times equal to the number of cutset vertices. The amount of information exchanged

each time is $O(\log s)$ bits to identify the cutset vertices in contention, plus two distance values.

CARDINALITY MATCHING [Edmonds 1956]. We assume that the reader is somewhat familiar with Edmonds' algorithm and we will only discuss its adaptation to two processors.

The two processors first look for augmenting paths that do not involve cutset vertices, constructing a matching as large as possible in this fashion. After this has been accomplished, augmenting paths that do involve cutset vertices are considered — using communication as needed to find such paths. In this fashion at most s augmenting paths involving cutset vertices have to be dealt with and each such path can be established with $O(s \log s)$ bits of communication. Neither part of this claim is obvious, although both are straightforward to verify. For more details we refer the reader to [Lakshmipathy and Winklmann, to appear].

3. Proving exponential lower bounds for NP-complete problems

3—SAT. Let E be a Boolean expression in conjunctive normal form with at most three literals per clause. Assume that E is partitioned into two sets of clauses, E_1 and E_2 and that $S=\{x_1, \cdots, x_s\}$ is the set of "cutset variables", i.e. the set of those variables that occur in both E_1 and E_2. E_1 and S are known to processor P_1, E_2 and S are known to processor P_2. Thus each processor can determine, without communicating with the other processor, the set A_i of those truth value assignments to S which can be extended to satisfy E_i. The whole expression E is then satisfiable *iff* $A_1 \cap A_2 \neq \emptyset$. Each set A_i is a subset of the set of all truth value assignments to S, hence a subset of a set of size 2^s, s being the size of S. In [Papadimitriou and Sipser 1984] it was shown

that such a set intersection problem requires 2^s bits of communication in the worst case. This lower bound carries over to 3-SAT if we can show that for any set A_1 of truth value assignments to S, there exists a Boolean expression E_1 in 3-conjunctive normal form with the property that an assignment α of truth values to S can be extended to an assignment that satisfies E_1 *iff* $\alpha \in A_1$. This is easily shown, e.g. by first constructing E_1 with the desired properties except that it may not be in 3-conjunctive normal form (and most easily would be constructed in disjunctive normal form), and then converting E_1 into 3-conjunctive normal form without changing its satisfiability properties using the method from [Cook 1971, or see Hopcroft and Ullman 1979, Theorem 13.3]. (The size of E_1 may well be exponential in the size of S. This is of no concern here.)

Similar arguments can be made for other NP-complete problems. For example, in [Lakshmipathy and Winklmann, to appear] it is shown that for GRAPH 3—COLORABILITY the number of bits of communication is exactly $\frac{3^s+3}{3!}$, in the worst case. More easily, the exponential lower bound of 3—SAT can be propagated to other NP-complete problems by the same transformations that are used to propagate NP-hardness. The resulting bounds are not perfectly tight, due to losses incurred in the transformations, but they are still exponential in the cutset size s.

4. Summary and open questions

Standard polynomial-time algorithms for a number of combinatorial problems can be mapped to two-processor machines, with the resulting two-processor protocols requiring little inter-processor communication. As a function of the size of the cut made to provide each processor with a piece of the problem instance, the amount of communication is bounded from above by a polynomial.

In contrast, a number of NP-complete problems require two-processor protocols with an amount of communication that is exponential in the size of this cut.

Consequently, these standard algorithms belong to a large and natural class of algorithms — algorithms that "map well to" two-processor protocols —, which does not contain any solution algorithms for a number of NP-complete problems.

This raises a number of questions.

First, "mapping well to" two-processor protocols, or "running well on two-processor machines", is a well-defined notion only after we decide on the details of such a mapping. How can we formally define these mappings and hence the class of algorithms for which our result holds? (One promising subclass of algorithms for which such a precise definition may be fairly easy to come by is the class of "traversal-based graph algorithms". These algorithms are characterized by the fact that computational activities are associated with the vertices and edges of a graph (vertices "get visited", edges "get traversed") and that activities at one location (vertex or edge) are triggered by activities at neighboring locations. But it remains to be seen if a formally defined class

remains large enough to be of interest.)

Second, the results obtained so far are all about the boundary between NP-complete problems and problems in P, in the sense that the exponential lower bounds are all about NP-complete problems and the polynomial upper bounds are all about problems in P. Can similar results be obtained about subclasses of P, showing that certain algorithm designs are too weak to solve certain classes of problems in P?

Third, in the case of Dijkstra's SHORTEST PATH algorithm the two processors communicated integer (or real) distance values. To be able to show NP-complete problems that involving numeric values require exponential amounts of communication (as always, exponential in the size of the cut, not in the overall problem size), it is necessary to prevent the (ab)use of numeric values as carriers of vast amounts of nonnumeric information. While this seems a reasonable restriction on the types of protocols we want to consider, deciding on a precise formal definition of protocols that do not abuse numeric values in this way may not be trivial.

Finally, a more general issue. Two-processor machines play no role in the statement of the result; they are merely a tool in the proof. In principle this leaves open the possibility that other machine models can be similarly useful for defining natural classes of algorithms and proving limitations of such classes. Similarly, problem instances with a small cut, where small means logarithmic in the overall size of the problem instance, serve to establish the exponential (in the cut size) lower bounds that let us drive a wedge between a class of standard algorithms and some NP-complete problems. But again, these special instances are only a tool in the proof and do not enter the statement of the result, leaving open the possibility that other types of instances, combined with other machine models, may lead to more results.

5. References

Cook, S. A. (1971), The complexity of theorem-proving procedures, *in* "Proc. 3rd Ann. ACM Symposium on Theory of Computing," Assoc. for Computing Machinery, New York, 151-158.

Dijkstra, E. W. (1959), A note on two problems in connexion with graphs. *Numerische Mathematik* **1**, 269-271.

Edmonds, J. (1965), Paths, trees and flowers, *Canad. J. Math.* **17**, 449-467.

Even, S., Itai, A., and Shamir, A. (1976), On the complexity of timetable and multicommodity flow problems, *SIAM J. Comput.* **5**, 691-703.

Hopcroft, J. E., and Ullman, J. D. (1979), "Introduction to Automata Theory, Languages, and Computation," Addison Wesley, Reading.

Lakshmipathy, N., and Winklmann, K. (1984), 'Global' graph problems tend to be intractable, Tech. Rep. 84-7, Department of Computing Science, University of Alberta.

Papadimitriou, C. H., and Sipser, M. (1984), Communication complexity, *J. Comput. System Sci.* **28**, 260-269.

Yao, A. C. (1979), Some complexity questions related to distributive computing, *in* "Proc. 11th Ann. ACM Symposium on Theory of Computing," Assoc. for Computing Machinery, New York, 209-213.