

The NP-Completeness Column: An Ongoing Guide

DAVID S. JOHNSON

Bell Laboratories, Murray Hill, New Jersey 07974

This is the seventh edition of a quarterly column the purpose of which is to provide continuing coverage of new developments in the theory of NP-completeness. The presentation is modeled on that used by M. R. Garey and myself in our book "Computers and Intractability: A Guide to the Theory of NP-Completeness," W. H. Freeman & Co., San Francisco, 1979 (hereinafter referred to as "[G&J]"; previous columns will be referred to by their dates). A background equivalent to that provided by [G&J] is assumed, and, when appropriate, cross-references will be given to that book and the list of problems (NP-complete and harder) presented there. Readers who have results they would like mentioned (NP-hardness, PSPACE-hardness, polynomial-time-solvability, etc.), or open problems they would like publicized, should send them to David S. Johnson, Room 2C-355, Bell Laboratories, Murray Hill, NJ 07974, including details, or at least sketches, of any new proofs (full papers are preferred). In the case of unpublished results, please state explicitly that you would like the results to be mentioned in the column. Comments and corrections are also welcome. For more details on the nature of the column and the form of desired submissions, see the December 1981 issue of this Journal.

1. INTRODUCTION

The subject of this column is parallel computing, a very popular topic these days. The literature is full of algorithms designed to be implemented by collections of processors, operating in parallel. One of the exciting aspects of VLSI technology is the potential it has for obtaining the speedups offered by parallelism in an inexpensive fashion.

Parallelism impinges on complexity theory in at least two ways. First, as alluded to above, one can ask how to use parallelism to reduce the running times of algorithms for a given problem. Section 2 asks the question "What can I do with a polynomial number of processors that I can't do with one?" and points out interesting connections between this question and some complexity classes,

old and new. Secondly, there are optimization and feasibility questions concerning the operation of multiprocessor systems (how are they designed, scheduled, linked together, tested for reliability, etc.), and one can ask whether such questions are NP-hard or solvable in polynomial time. I have already addressed many questions of design in earlier columns. In Section 3 I consider questions of scheduling, a discipline that has generated so many results that a computer is currently required to keep track of them. I discuss this project for computerized complexity classification in Section 4, and use it to explain the difficulty I am having in finding the right candidates for “Open Problem of the Month.”

2. TRADING HARDWARE FOR TIME

Anyone who has trouble keeping straight all the different models proposed for sequential computation (RAM'S, RASP'S, and Turing Machines with various bells and whistles) is in for a real treat when it comes to parallel computation. Here we have vector machines [52], alternating Turing machines [11,12], PRAM's [22,55], SIMDAG's [26], uniform circuits [9,53], conglomerates [26], aggregates [21], and hardware modification machines [21], to name just a few, and to restrict ourselves to “synchronous” models (in which the computation is performed by interconnected sequential processors whose individual steps are assumed to be synchronized). I do not have the space here to go into the details of each, and so will fix on one, the PRAM (Parallel Random Access Machine) of [22], which I can define quickly if I wave my hands fast enough. I assume that the reader is already familiar with ordinary (sequential) RAM's (e.g., see [2]).

In a PRAM we have a (potentially infinite) sequence of global *registers*, each capable of holding an integer (as in a sequential RAM), plus a (potentially infinite) set of identically-programmed processors, each with its own sequence of local registers. Computation begins with the input x loaded into the first $|x|$ global registers, one bit per register, and only the first processor p_1 activated. At each step, a processor can do one of the standard operations, or it can activate a new processor, to run in parallel with those already in operation. Each processor can refer to either its local registers or the global ones, but if ever two processors try to write into the same global register at the same time, the whole system crashes (simultaneous *reads* are permitted, however). The computation concludes when the initial processor halts.

The first thing to observe about this model is that it is quite powerful. The class of languages that it can recognize in polynomial time is precisely PSPACE, the class of languages that can be recognized by sequential machines in polynomial *space*, and so it can solve both NP-complete and PSPACE-complete problems in polynomial time (quite a feat, given the general opinion about the difficulty of these problems). This power is shared by the other parallel models and is a special case of the “parallel computation thesis” [12,26]

that, for any well-behaved time bound $T(n)$, the languages recognizable in a “reasonable” model of parallel computation in time $T(n)^{O(1)}$, i.e., polynomial in $T(n)$, are precisely those recognized by a deterministic Turing machine in *space* $T(n)^{O(1)}$.

Unfortunately, in the case of polynomial time this power is not a “realistic” power. It is easy to see that, if one restricts oneself to a number of processors that is bounded by a polynomial in the input size (about the *most* one could reasonably allow) then the set of languages that can be recognized in parallel polynomial time shrinks from PSPACE to P, the set of languages that can be recognized in *sequential* polynomial time. The real hope for applications of parallelism would thus seem to lie in speeding up the solution of problems in P, rather than in making feasible the solution of harder problems. Let us consider the quest for such speed-ups.

Suppose one has a problem that seems to require $\Theta(n^2)$ time to be solved by a sequential algorithm. A first observation is that we might hope to speed this up to $O(n)$ by using n processors in parallel (although such a speed-up is not guaranteed to be possible, of course). A much more intriguing possibility arises when we consider using n^2 processors: could we then solve the problem in constant time? Not in the PRAM model, since $\Omega(\log n)$ would be required just to activate the n processors. To be realistic, $\Omega(\log n)$ would be required in any “reasonable” model, because of the communication delays introduced in practice by fan-out limitations. Often, however, this (or something like it) may be enough. It turns out that many problems in P can be solved in parallel time $(\log n)^{O(1)}$, i.e., time that is polynomially bounded in the logarithm of the problem size (*polylog* time). It has long been known that such results hold for arithmetical operations like multiplication, division, etc., and in fact algorithms of this form (restricted to fixed input size) have been carried out by circuits in the arithmetical units of computers for decades. Similarly, sorting networks of polylog depth (and hence polylog parallel time) have been around for quite a while (I talked about them in the [Sep. 1982] column). More recently, algorithms of this type have been devised for such nontrivial problems as maximum flow in planar graphs [30], context-free language recognition [53], and even linear programming with a fixed number of variables [47].

Unfortunately, parallel algorithms of this type have not yet been found for all interesting problems in P. For instance, no such algorithms have been found for general linear programming, or for the maximum flow problem in arbitrary graphs. Here another theoretical result becomes relevant. These latter two problems have been proved [18,27] to be “log-space complete for P.” Recall (e.g., from [G&J], Chapter 7) that a problem X is log-space complete for P if (a) $X \in P$ and (b) each problem in P can be reduced to X by a transformation computable using logarithmic work space. By standard arguments, if any such problem can be recognized in polylog (sequential) space, then all languages in P can be so recognized ($P \subseteq \text{POLYLOGSPACE}$). Since this inclusion is not expected

to be true, this means that such problems are unlikely to be in POLYLOGSPACE, and hence, by the parallel computation thesis, unlikely to be solvable in polylog parallel time.

As a sidelight, it is interesting to note that, although we don't know if there is an inclusion relation between P and POLYLOGSPACE, or even if they are comparable, we do know that they are not equal [8] (it can be shown that POLYLOGSPACE has no log-space complete problems, whereas, as we have just seen, P does). However, if it were the case that $P \subseteq \text{POLYLOGSPACE}$, this would still not necessarily imply that everything in P could be solved in polylog parallel time with only a polynomial number of processors. For instance, a $\log^2 n$ parallel time algorithm might need (and would be able to access) $n^{\log n}$ processors.

In order to simplify matters when talking about such issues, researchers have adopted the name "NC" for the class of problems solvable by algorithms that use polylog parallel time *and* only a polynomial number of processors on a PRAM. The name NC was first proposed by Steve Cook [14]. It is not, as one might first guess, based on some tortuous analogy with "NP," but rather on the legalistic notion that, since Nick Pippenger had been the first to suggest this set of problems, it was "Nick's Class." Retaliation was swift: the class that Cook was introducing when he christened NC rapidly evolved into what is now known as "SC," for "Steve's Class." This class, at first identified by the picturesque acronym "PLOPS" [53], consists of those problems that can be solved by polynomial time sequential algorithms that use just polylog space. An interesting open question in complexity theory concerns the relation between NC and SC.

There are superficial reasons why one might think the two classes are equal, despite the fact that one deals with parallel computation and one with sequential computation. Consider the *Boolean circuit* model of parallel computation. Here an instance of size n is solved by an n -input 1-output Boolean circuit (without feedback), with a different circuit for each n . The "number of processors" is the number of Boolean gates in the circuit (the *size* of the circuit), and the "time" is the length of the longest path from an input to the output (the *depth*). For the model to be "reasonable," we also require that there be some "uniformity condition" placed on the structure of the circuits, i.e., that the circuits be in some sense "easily constructible," given n . It turns out that under a variety of uniformity conditions, those problems that can be solved in polylog depth in this model are precisely those that can be solved sequentially in polylog space [9,53] (another instance of the parallel computation thesis) and those that can be solved in polynomial size are precisely those that can be solved sequentially in polynomial time [9,53]. Thus it is appealing to think that simultaneous polynomial size and polylog depth (NC) would be the same as simultaneous polynomial time and polylog space (SC).

Unfortunately, classes based on simultaneous resource bounds are not the

same as intersections of classes; neither NC nor SC need equal $P \cap \text{POLYLOG-SPACE}$, and they need not equal each other. In [14], Cook suggested “transitive closure” as a potential member of NC – SC and, to provide a candidate for SC – NC, proved that deterministic context-free languages could be recognized in polynomial time using just $O(\log^2(n))$ space. This candidate didn’t last long, however, for just six months later, Ruzzo [53] showed that *general* context-free language recognition was in NC, putting the deterministic case in $\text{NC} \cap \text{SC}$ and making the general case a candidate for NC – SC. One can still concoct artificial candidates for SC – NC however [15], and so the relation between NC and SC remains unresolved, although new theoretical results have helped clarify the issues. Pippenger has shown that NC has a sequential interpretation as those problems solvable by deterministic Turing machines in polynomial time using polylog “reversals,” and that SC has a parallel interpretation as those problems that can be solved by circuits with polynomial size and polylog “width” [51]. Further characterizations are provided in [15,21,53].

The reader interested in pursuing results about the various parallel computation models and resource trade-offs therein is referred to the above references, and also to [29], which summarizes many of these results and also considers classes defined by *three* simultaneous resource bounds. For more on speed-ups attainable for particular problems, including problems that are log-space complete for P, see for instance [3,4,10,16,54,58,59,60,61]. Some of this work concerns “unreasonable” parallel models in which running times better than $O(\log n)$ are possible, such as “unbounded fan-in circuits” [13] and WRAM’s: parallel random access machines in which all processors start at time 0 and simultaneous writes into global memory are allowed, subject to some rule for resolving conflicts [10]. As to another “unreasonable” model, the class of problems solvable by circuits with polynomially bounded size but *no* uniformity condition turns out to be quite interesting. It contains P, it even contains R, the class of problems solvable in “random” polynomial time [1], but if it contains an NP-complete problem then the polynomial hierarchy collapses [31].

3. SCHEDULING PARALLEL PROCESSORS

If one is to speed up the operation of an algorithm (or a Computer Center) by using parallel processors, one must inevitably face the problem of scheduling the execution of tasks by the processors. Scheduling theory has turned out to be one of the most fertile territories for those wishing to explore the boundary between the NP-hard and the polynomial time solvable, and the outpouring of results has continued unabated in the four years since the publication of [G&J]. In preparing this column, I found myself confronted with well over 75 new papers on the subject (including one [17] on polylog parallel time algorithms for scheduling parallel processors).

In hopes of leaving some room in this issue of *J. Algorithms* for technical

papers, I have therefore had to be selective. Results that primarily concern the scheduling of a single processor will be postponed to a later column, on grounds of insufficient parallelism. I shall also postpone the discussion of the various *shop* scheduling problems (job shop, flow shop, open shop), although on perhaps more debatable grounds: Since these problems were originally devised to model industrial applications, the “processors” they speak of are more likely to be the Mechanic and the Electrician than they are to be the CRAY® and the VAX®. In addition, some of my updates on problems covered in [G&J] will be sketchier than usual, and I shall frequently send the reader to the references for further details. I begin with some results about *new* problems.

[1] MULTIPLE CHOICE SCHEDULING WITHIN INTERVALS

INSTANCE: Number m of processors, a set T of tasks, a common task length l , and, for each task $t \in T$, a collection $C(t) = \{[a_i(t), b_i(t)]: 1 \leq i \leq k(t)\}$ of permissible scheduling intervals, with integer endpoints and $0 \leq a_i(t) \leq b_i(t) - l$, for $1 \leq i \leq k(t)$.

QUESTION: Is there an m -processor nonpreemptive schedule for T that places each task in one of its permissible intervals, i.e., a function $s: T \rightarrow Z$ that assigns each task a non-negative starting time such that (a) for all times $u > 0$, $|\{t: s(t) \leq u < s(t)+l\}| \leq m$ and (b) for each $t \in T$ there is an i , $1 \leq i \leq k(t)$, such that $a_i(t) \leq s(t) \leq b_i(t) - l$?

Reference. Simons and Sipser [57]. Transformation from 3SAT.

Comment. NP-complete in the strong sense for any fixed value of $m \geq 1$, even if there are at most 3 intervals in each collection $C(t)$ and all intervals are of length exactly l , or if there are at most 2 intervals per collection but differing interval lengths are allowed [57]. Solvable in polynomial time for m arbitrary if there is exactly one interval per collection [56], or if $l = 1$ (and arbitrary collections of intervals are allowed) [57]. Suppose that arbitrary integer task processing times $l(t)$ are allowed, but that all intervals in $C(t)$ must have length $l(t)$. Then the values $\{a_i(t): 1 \leq i \leq k(t)\}$ correspond to the possible starting times for task t . In this case the problem is solvable in polynomial time if, when a task has more than one possible starting time, there are just two and they differ by 1 [49], or if $m = 1$ and, for each t , the maximum difference between possible starting times is less than $l(t)$ [49]. Slight generalizations of both these cases yield strong NP-completeness. For details, see [48,49]. Returning to the case of intervals, suppose that there is just one interval per task, but that different task lengths are allowed. There are three polynomial time solvable subcases when just two integer lengths, 1 and l , are allowed: (1) $m = 1$, (2) $m = 2$ and $l = 2$, and (3) m and l are arbitrary but all release times equal 0 [19]. If three lengths are allowed, all these problems become NP-complete in the strong sense [19].

[2] SCHEDULING TO MINIMIZE MACHINE COST

INSTANCE: Set T of tasks, each task t having an integer length $l(t)$ and a starting time $s(t)$, integer machine purchase prices $C_1 > C_2$ and running time factors $f_1 < f_2$, bound B .

QUESTION: Are there integers k_1 and k_2 such that $k_1 C_1 + k_2 C_2 \leq B$ and such that the tasks of T can be scheduled nonpreemptively on a set of k_1 processors of Type 1 and k_2 processors of Type 2, where a task t scheduled on a processor of Type i starts at time $s(t)$ and takes processing time $l(t) \cdot f_i$?

Reference. Nakajima, Hakimi and Lenstra [50]. Transformation from NUMERICAL MATCHING WITH TARGET SUMS [SP17].

Comment. NP-complete in the strong sense. Basically, this is a question of finding the right mix of fast expensive computers and slow cheap ones. The problem is solvable in polynomial time if all the starting times are equal, or if the Type 1 machines are not just faster and more expensive, but in fact can execute each task in just one time unit, irrespective of its processing time on a Type 2 machine [50]. In the variant where a task t assigned to Type 1 machines may start later than time $s(t)$, so long as it finishes by the time $s(t) + l(t) \cdot f_2$ that it would have finished on the slower machine, the problem is NP-hard in the strong sense even if all values of $s(t)$ are equal (and even if $C_1/C_2 = f_2/f_1 = 3$). Here we get polynomial time solvability only if all starting times are equal *and* the fast machine can execute any task in time 1 [50]. For more detailed results, see [48,50]. Note that if only machines of Type 1 are available, the problem is solvable in polynomial time for arbitrary starting and processing times, as it then becomes simply a question of interval graph coloring.

[3] PREEMPTIVE PERIODIC SCHEDULING

INSTANCE: Number m of processors, a set T of tasks, and for each task t a release time $r(t)$, a deadline $d(t)$, a length $l(t)$, and a period $p(t)$, all integers, with $r(t) < d(t)$ and $0 < l(t) < d(t) < p(t)$.

QUESTION: Is there an infinite periodic preemptive schedule for the tasks in T , i.e., an assignment of an infinite sequence of disjoint open intervals $I_1(t), I_2(t), \dots$ to each task so that (a) each interval is a subinterval of an interval of the type $[r(t) + k \cdot p(t), d(t) + k \cdot p(t)]$, for k a nonnegative integer, (b) the sum of the lengths of the subintervals corresponding to t in each interval of the above form is $l(t)$, and (c) for any $u > 0$, u is contained in at most m intervals $I_i(t)$, $i \geq 1$ and $t \in T$?

Reference. Leung and Merrill [42]. Transformation from CLIQUE.

Comment. Not known to be in NP, but is in PSPACE. The transformation is

an indirect one via a variant on SIMULTANEOUS INCONGRUENCES [AN2], proved NP-complete in [43] and to be described in a later column. The current problem is NP-hard for any fixed number $m \geq 1$ of processors, although there is a very simple scheduling rule (always work on the available task with the earliest deadline) that is guaranteed to generate a feasible 1-processor schedule if one exists. The difficulty is that infeasibility may not show up until one has been scheduling for an exponential amount of time (the least common multiple N of the periods). For $m > 1$, there exist algorithms for testing feasibility in time polynomial in n and N , even for uniform (different speed) processors or “unrelated” processors (ones having different rates for different tasks) [40]. Note, however, that these algorithms need not even be pseudo-polynomial, due to the potential size of N (the NP-hardness result is not “in the strong sense,” however, so a pseudo-polynomial time algorithm may still be possible). Strong NP-hardness results for the multiprocessor case are presented in [43], which considers the question of whether one can assign priorities to the tasks (and possibly partition them among processors) so that a simple priority-based scheduling heuristic will generate a valid schedule.

We now turn to updates on old problems from [G&J], with one paragraph for each problem being updated. Formal definitions from [G&J] will not be repeated, but readers who have made it through the above three new problems should not find the remarks too cryptic. (Readers who have not made it through will probably be just as happy to skip over these updates anyway.)

MULTIPROCESSOR SCHEDULING [SS8]. In this problem one wishes to schedule independent tasks on m processors to meet a given deadline D . Leung [41] has recently shown that if one restricts the task lengths to a finite set of k elements, the problem can be solved in polynomial time by a clever application of dynamic programming. Unfortunately, the degree of the polynomial depends linearly on k .

PRECEDENCE CONSTRAINED SCHEDULING [SS9]. In this problem one wishes to schedule unit-length tasks on m processors to meet a deadline D , subject to a precedence relation $<$, where $t' < t$ implies that t cannot start until t' finishes. The question of whether this problem is NP-complete when the value of m is fixed and greater than 2 remains open, but researchers continue to exercise great ingenuity in devising other special cases and variants to investigate. The set of classes of precedence graphs for which the problem is NP-complete when m is arbitrary has grown to include “hierarchical parallel graphs,” a variant on series-parallel graphs that is supposed to reflect the control flowgraphs for programs in Concurrent Pascal and Algol 68 [46]. On the algorithmic front, the case of precedence graphs with bounded “depth” has received recent emphasis. If m is arbitrary, even a depth bound of 3 doesn’t help, but for any fixed bound and fixed value of m , the problem is solvable in polynomial time [20]. In

the 2-processor case, these results can be extended to processors of different speeds, so long as the slower one takes $1 + 1/k$ time units for each one taken by the faster one, for some integer k [24]. More significantly, the running time for solving the 2-processor (equal speed, arbitrary precedence) problem has been reduced from the time for performing matrix multiplication to linear time, using an algorithm from [23] and a data structure from [25].

RESOURCE CONSTRAINED SCHEDULING [SS10]. This is another scheduling problem for unit-length tasks, this time with tasks having integer resource requirements rather than precedence relations, and with a bound on the amount of each resource available at any one time. This was known to be NP-complete in the strong sense for 3 processors and a single arbitrary-valued resource; strong NP-completeness also holds for an arbitrary number of 0-1 valued resources with bounds of 1 [7] both in the case of 3 (identical) processors and of 2 different speed uniform processors [7] (the 2 uniform processor problem is solvable in polynomial time, however, if there is just one resource, even if it has arbitrary values [7]). For an arbitrary number of processors, the different speed uniform processor problem is solvable in polynomial time if there is just one 0-1 valued resource, no matter what its bound [7]. The identical processor problem is solvable in polynomial time if the number r of resources is fixed and each obeys a fixed bound, although the algorithm is exponential in r and the resource bound [6]. If each task has an integer release time and deadline, and must be executed within the interval they define, the identical processor problem can still be solved in polynomial time for an arbitrary number of processors, so long as the only resource is an arbitrarily-bounded 0-1 resource [5]. If precedence constraints are added, the 2-processor problem is NP-complete even if the precedence graph is a union of chains, so long as there is a single, arbitrary-valued resource [7]. Still with me? We're almost through. The 2-processor problem is solvable in polynomial time for arbitrary precedence constraints if the resource requirement can be either 1 or 2 and the bound is 2 (and hence the resource can be viewed as the number of processors that need to be operated concurrently in order to execute a given task) [44]. This latter problem becomes NP-complete, however, if the resource bound (and number of processors) is increased to 3 [44]. The above NP-completeness results all carry over if one is interested in bounding the sum of task finishing times instead of their maximum, as do many of the algorithms (see [7]).

PREEMPTIVE SCHEDULING [SS12]. In this problem we are given arbitrary length tasks and ask whether they can be scheduled preemptively to meet a given deadline (this is like the above **PREEMPTIVE PERIODIC SCHEDULING** problem when all the periods are equal and hence we need only figure out how to schedule within one of them). The complexity of the following new subcases has been determined: The case with m "unrelated" processors and an individual release time and deadline for each task can be solved in polynomial

time by linear programming [37]. If the processors are uniform, a more straightforward approach is possible using network flow techniques [45], and if $m = 2$, the problem can be solved in polynomial time even in the presence of arbitrary precedence constraints [35], as can the m identical processor problem in the presence of “intree” precedence constraints, so long as all release times are 0 [35]. Minimizing the number of tardy tasks is NP-hard if m is arbitrary, even if processors are identical, there are no precedence constraints, and all release times are 0 [36]. For any *fixed* value of m , this problem can be solved in polynomial time, even for different speed processors, and the corresponding *weighted* case can be solved in pseudo-polynomial time (the latter is NP-complete, however, even for $m = 1$) [36]. Polynomial time algorithms are given in [34] for situations where processors have different sized memories and each task has an individual memory requirement.

4. DIRECTORY ASSISTANCE FOR SCHEDULING PROBLEMS

On more than one occasion in the last section, the reader interested in “further details” was directed to technical reports or papers originating at the Mathematisch Centrum in Amsterdam. One reason for this is that some of the best researchers in deterministic scheduling theory are associated with that institution, either as permanent staff or as regular visitors. There is, however, an even more important reason: the Mathematisch Centrum, in the persons of B. J. Lageweg, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan, has become *the* clearing house for complexity results about scheduling, much more so than this column could ever hope to be. I gladly yield them primacy, as they have developed some very useful tools for dealing with the sheer bulk and detail of the results in scheduling and, as we shall see, there are in a sense more complexity results known for scheduling problems than for all other problems combined!

One of their tools is an elaborate shorthand scheme for specifying scheduling problems, described in [7,28,32,33,38]. Using this scheme, the specific results mentioned in the update for PREEMPTIVE SCHEDULING in Section 3 become simply:

$R \mid pmtn, r_j \mid L_{\max}$, $Q \mid pmtn, r_j \mid L_{\max}$, $Q2 \mid pmtn, prec, r_j \mid L_{\max}$, and $P \mid pmtn, intree \mid L_{\max}$ are solvable in polynomial time. $P \mid pmtn \mid \Sigma U_j$ and $I \mid pmtn \mid \Sigma w_j U_j$ are NP-hard. For fixed m , $Qm \mid pmtn \mid \Sigma U_j$ is solvable in polynomial time and $Qm \mid pmtn \mid \Sigma w_j U_j$ is solvable in pseudo-polynomial time.

A bit cryptic, perhaps, but definitely more concise and, to those who know the system, less likely to be misunderstood than the discursive presentation of Section 3. (Unfortunately, I could not have taken advantage of this concision without first spending 5 or 10 pages explaining the notation.)

Another advantage of having a shorthand for specifying problems is that it allows one to systematize one's knowledge of the field. One is more likely to recognize suggestive patterns in the results, and to identify open problems. Indeed, if one generates a list of all the allowed combinations of parameters in the system (not counting those referring to resource constraints) [33,39], one obtains 4,536 different scheduling problems, many of them open. Factoring in the various possibilities for resource constraints could well have increased this by an order of magnitude.

Having such a large collection of problems, however, might not be considered an advantage in all circles. Just because a problem can be defined does not make it important. How does one tell which of the thousands of problems are the truly significant ones, and which are mere syntactic artifacts? What does one *do* with 4,536 problems?

Being both mathematicians and computer scientists, the Centrum researchers have chosen an interdisciplinary approach: First, they defined a partial order, and then they wrote a computer program. The partial order is the natural one, with $A < B$ meaning that problem A is a special case of problem B , as 2-processor scheduling is a special case of m -processor scheduling. It allows us to propagate complexity results, since if $A < B$ then $B \in P$ implies $A \in P$ and the NP-hardness of A implies the NP-hardness of B .

The computer program was written by Lageweg as a graduation present to Rinnooy Kan (for a detailed description of the whole history of the project see [39]). It incorporates the partial order and hence, given a collection of polynomial time solvability and NP-hardness results for problems in the class, can quickly deduce all the immediate consequences. The status of the 4,536 problems as of 1982 was 3730 (82%) NP-hard, 416 (9%) polynomial time solvable, and 390 (9%) open, with similar percentages if one considers the parallel distinction between strong NP-completeness and pseudo-polynomial time solvability [32].

The program also summarizes the results by providing lists of the most general polynomial time solvable problems (a set of 40 dominated all others in 1981 [33]) and the most restricted NP-complete ones (here a set of 73 dominated), as well as lists of maximal and minimal open problems (27 and 40 respectively). Although such knowledge does not tell us which problems are important from a practical point of view, it can at least provide an objective measure of the theoretical significance of a result *within the class*.

This approach can of course be applied to other types of combinatorial problems. The authors of [32] suggest vehicle routing, facilities location, and network design as possibilities. They also suggest using the information generated by the program to help choose which open problems to study. When there are 432 open problems in a class, it seems a bit presumptuous to identify just one as the "most important." What one would really like is an optimal *collection* of open problems. Unfortunately, the following NP-completeness result holds,

which I present in lieu of an “Open Problem of the Month.”

[4] MINIMUM COMPLETE RESEARCH PROGRAM

INSTANCE: Set X of open problems, partial order $<$ on X , bound B .

QUESTION: Is there a subset $X' \subseteq X$, $|X'| \leq B$, and a complexity classification function $c: X' \rightarrow \{EASY, HARD\}$ that would resolve all the open problems, i.e., such that c can be extended to a well-defined total function on X by applying the rules (a) $x < y$ and $c(x) = HARD$ implies $c(y) = HARD$ and (b) $x < y$ and $c(y) = EASY$ implies $c(x) = EASY$.

Reference. Lageweg, Lawler, Lenstra, and Rinnooy Kan [32]. Transformation from VERTEX COVER.

Comment. Variant in which one asks if there is a complexity classification c for X , such that B or fewer well-chosen NP-completeness proofs would imply all the “hardness” results, is simply a minimum cut problem and hence solvable in polynomial time [32].

REFERENCES

1. L. ADLEMAN, Two theorems on random polynomial time, in "Proceedings 19th Ann. Symp. on Foundations of Computer Science," pp. 75-83, IEEE Computer Society, Los Angeles, 1978.
2. A. V. AHO, J. E. HOPCROFT, AND J. D. ULLMAN, "The Design and Analysis of Computer Algorithms," Addison-Wesley, Reading, Mass., 1974.
3. M. J. ATALLAH AND S. R. KOSARAJU, Graph problems on a mesh-connected processor array, in "Proceedings 14th Ann. ACM Symp. on Theory of Computing," pp. 345-353, Association for Computing Machinery, New York, 1982.
4. J. L. BENTLEY, A parallel algorithm for constructing minimum spanning trees, *J. Algorithms* **1** (1980), 51-59.
5. J. BLAZEWICS, Deadline scheduling of tasks with ready times and resource constraints, *Inform. Process. Lett.* **8** (1979), 60-63.
6. J. BLAZEWICS AND K. ECKER, "A polynomial in time algorithm for task scheduling under fixed resource constraints," Institute of Control Engineering, Technical University of Poznan, Poznan, Poland, 1980.
7. J. BLAZEWICZ, J. K. LENSTRA, AND A. H. G. RINNOOY KAN, Scheduling subject to resource constraints: Classification and complexity, *Disc. Applied Math.* **5** (1983), 11-24.
8. R. V. BOOK, Translational lemmas, polynomial time, and $(\log n)^j$ -space, *Theor. Comput. Sci.* **1** (1976), 215-226.
9. A. BORODIN, On relating time and space to size and depth, *SIAM J. Comput.* **6** (1977), 733-744.
10. A. BORODIN AND J. E. HOPCROFT, Routing, merging and sorting on parallel models of computation, in "Proceedings 14th Ann. ACM Symp. on Theory of Computing," pp. 338-344, Association for Computing Machinery, New York, 1982.
11. A. K. CHANDRA, D. C. KOZEN, AND L. J. STOCKMEYER, Alternation, *J. Assoc. Comput. Mach.* **28** (1981), 114-133.
12. A. K. CHANDRA AND L. J. STOCKMEYER, Alternation, in "Proceedings 17th Ann. Symp. on Foundations of Computer Science," pp. 98-108, IEEE Computer Society, Los Angeles, 1976.
13. A. K. CHANDRA, L. J. STOCKMEYER, AND U. VISHKIN, A complexity theory for unbounded fan-in parallelism, in "Proceedings 23rd Ann. Symp. on Foundations of Computer Science," pp. 1-13, IEEE Computer Society, Los Angeles, 1982.
14. S. A. COOK, Deterministic CFL's are accepted simultaneously in polynomial time and log squared space, in "Proceedings 11th Ann. ACM Symp. on Theory of Computing," pp. 338-345, Association for Computing Machinery, New York, 1979.
15. S. A. COOK, "Towards a complexity theory of synchronous parallel computation," Report No. 141/80, Department of Computer Science, University of Toronto, Toronto, Canada, 1980.
16. E. DEKEL, D. NASSIMI, AND S. SAHNI, Parallel matrix and graph algorithms, *SIAM J. Comput.* **10** (1981), 657-675.
17. E. DEKEL AND S. SAHNI, Parallel scheduling algorithms, *Operations Res.* **31** (1983), 24-49.
18. D. DOBKIN, R. J. LIPTON, AND S. REISS, Linear programming is log-space hard for P, *Inform. Process. Lett.* **8** (1979), 96-97.
19. D. DOLEV, B. SIMONS, AND M. WARMUTH, Procrastination pays: A mixed scheduling strategy, manuscript (1982).
20. D. DOLEV AND M. WARMUTH, "Scheduling precedence graphs of bounded height," Report No. RJ3399, IBM Research Laboratory, San Jose, Calif., 1982.
21. P. W. DYMOND AND S. A. COOK, Hardware complexity and parallel computation, in "Proceedings 21st Ann. Symp. on Foundations of Computer Science," pp. 360-372, IEEE Computer Society, Los Angeles, 1980.
22. S. FORTUNE AND J. WYLLIE, Parallelism in random access machines, in "Proceedings 10th Ann. ACM Symp. on Theory of Computing," pp. 114-118, Association for Computing Machinery, New York, 1978.

23. H. N. GABOW, An almost-linear algorithm for two-processor scheduling, *J. Assoc. Comput. Mach.* **29** (1982), 766-780.
24. H. N. GABOW, "Exact and approximate algorithms for scheduling UET systems on two uniform processors," Report No. CU-CS-225-82, Department of Computer Science, University of Colorado, Boulder, Colo., 1982.
25. H. N. GABOW AND R. E. TARJAN, A linear time algorithm for a special case of disjoint set union, in "Proceedings 15th Ann. ACM Symp. on Theory of Computing," pp. 244-249, Association for Computing Machinery, New York, 1983.
26. L. M. GOLDSCHLAGER, A universal connection pattern for parallel computers, *J. Assoc. Comput. Mach.* **29** (1982), 1073-1086.
27. L. GOLDSCHLAGER, R. SHAW, AND J. STAPLES, The maximum flow problem is log-space complete for P, *Theor. Comput. Sci.* **21** (1982), 105-111.
28. R. L. GRAHAM, E. L. LAWLER, J. K. LENSTRA, AND A. H. G. RINNOOY KAN, Optimization and approximation in deterministic sequencing and scheduling: A survey, *Ann. Discrete Math.* **5** (1979), 287-326.
29. J. W. HONG, On similarity and duality of computation, in "Proceedings 21st Ann. Symp. on Foundations of Computer Science," pp. 348-359, IEEE Computer Society, Los Angeles, 1980.
30. D. B. JOHNSON AND S. M. VENKATESAN, Parallel algorithms for minimum cuts and maximum flows in planar networks, in "Proceedings 23rd Ann. Symp. on Foundations of Computer Science," pp. 244-254, IEEE Computer Society, Los Angeles, 1982.
31. R. M. KARP AND R. J. LIPTON, Some connections between nonuniform and uniform complexity classes, in "Proceedings 12th Ann. ACM Symp. on Theory of Computing," pp. 302-309, Association for Computing Machinery, New York, 1980.
32. B. J. LAGEWEG, E. L. LAWLER, J. K. LENSTRA, AND A. H. G. RINNOOY KAN, Computer aided complexity classification of combinatorial problems, *Comm. ACM* **25** (1982), 817-822.
33. B. J. LAGEWEG, E. L. LAWLER, J. K. LENSTRA, AND A. H. G. RINNOOY KAN, "Computer aided complexity classification of deterministic scheduling problems," Report No. BW 138/81, Mathematisch Centrum, Amsterdam, the Netherlands, 1981.
34. T.-H. LAI AND S. SAHNI, "Preemptive scheduling of uniform processors with memory," Report No. 82-5, Computer Science Department, University of Minnesota, Minneapolis, Minn., 1982.
35. E. L. LAWLER, "Preemptive scheduling of precedence-constrained jobs on parallel machines," Report No. BW 132/81, Mathematisch Centrum, Amsterdam, the Netherlands, 1981.
36. E. L. LAWLER, Preemptive scheduling of uniform parallel machines to minimize the number of late jobs, manuscript (1981).
37. E. L. LAWLER AND J. LABETOULLE, On preemptive scheduling of unrelated parallel processors by linear programming, *J. Assoc. Comput. Mach.* **25** (1978), 612-619.
38. E. L. LAWLER, J. K. LENSTRA, AND A. H. G. RINNOOY KAN, "Recent developments in deterministic sequencing and scheduling: A survey," Report No. BW 146/81, Mathematisch Centrum, Amsterdam, the Netherlands, 1981.
39. E. L. LAWLER, J. K. LENSTRA, AND A. H. G. RINNOOY KAN, A gift for Alexander: At play in the fields of scheduling theory, *Optima: Mathematical Programming Society Newsletter* **7** (1982), 1-3.
40. E. L. LAWLER AND C. U. MARTEL, Scheduling periodically occurring tasks on multiple processors, *Inform. Process. Lett.* **12** (1981), 9-12.
41. J. Y.-T. LEUNG, On scheduling independent tasks with restricted execution times, *Operations Res.* **30** (1982), 163-171.
42. J. Y.-T. LEUNG AND M. L. MERRILL, A note on preemptive scheduling of periodic, real-time tasks, *Inform. Process. Lett.* **11** (1980), 115-118.
43. J. Y.-T. LEUNG AND J. WHITEHEAD, On the complexity of fixed-priority scheduling of periodic, real-time tasks, manuscript (1981).

44. E. L. LLOYD, Concurrent task systems, *Operations Res.* **29** (1981), 189-201.
45. C. MARTEL, Preemptive scheduling with release times, deadlines, and due times, *J. Assoc. Comput. Mach.* **29** (1982), 812-829.
46. E. MAYR, "Well structured parallel programs are not easier to schedule," Report No. STAN-CS-81-880, Computer Science Department, Stanford University, Stanford, Calif., 1981.
47. N. MEGIDDO, private communication (1982).
48. K. NAKAJIMA, "On Nonpreemptive Multiprocessor Scheduling with Discrete Starting Times," Ph.D. Dissertation, Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, Ill., 1982.
49. K. NAKAJIMA AND S. L. HAKIMI, Complexity results for scheduling tasks with discrete starting times, *J. Algorithms* **3** (1982), 344-361.
50. K. NAKAJIMA, S. L. HAKIMI, AND J. K. LENSTRA, Complexity results for scheduling tasks in fixed intervals on two types of machines, *SIAM J. Comput.* **11** (1982), 512-520.
51. N. PIPPENGER, On simultaneous resource bounds, in "Proceedings 20th Ann. Symp. on Foundations of Computer Science," pp. 307-311, IEEE Computer Society, Los Angeles, 1979.
52. V. PRATT AND L. STOCKMEYER, A characterization of the power of vector machines, *J. Comput. System Sci.* **12** (1978), 198-221.
53. W. L. RUZZO, On uniform circuit complexity, in "Proceedings 20th Ann. Symp. on Foundations of Computer Science," pp. 312-318, IEEE Computer Society, Los Angeles, 1979. Final publication in *J. Comput. System Sci.* **22** (1981), 365-383.
54. C. SAVAGE AND J. JA'JA', Fast, efficient parallel algorithms for some graph problems, *SIAM J. Comput.* **10** (1981), 682-691.
55. W. J. SAVITCH AND M. J. STIMSON, Time bounded random access machines with parallel computing, *J. Assoc. Comput. Mach.* **26** (1979), 103-118.
56. B. SIMONS, A fast algorithm for multiprocessor scheduling, in "Proceedings 21st Ann. Symp. on Foundations of Computer Science," pp. 50-53, IEEE Computer Society, Los Angeles, 1980.
57. B. SIMONS AND M. SIPSER, "On scheduling unit-time jobs with multiple release time/deadline intervals," Report No. RJ3236, IBM Research Laboratory, San Jose, Calif., 1981.
58. Y. SHILOACH AND U. VISHKIN, Finding the maximum, merging, and sorting in a parallel computation model, *J. Algorithms* **2** (1981), 88-102.
59. Y. SHILOACH AND U. VISHKIN, An $O(\log n)$ parallel connectivity algorithm, *J. Algorithms* **3** (1982), 57-67.
60. Y. SHILOACH AND U. VISHKIN, An $O(n^2 \log n)$ parallel MAX-FLOW algorithm, *J. Algorithms* **3** (1982), 128-146.
61. L. G. VALIANT, Parallelism in comparison problems, *SIAM J. Comput.* **4** (1975), 348-355.