Algorithmica (1989) 4: 511–517

Algorithmica

Extending NC and RNC Algorithms

Nimrod Megiddo[1]

Abstract. A technique is presented by which NC and RNC algorithms for some problems can be extended into NC and RNC algorithms, respectively, that solve more general parametric problems. The technique is demonstrated on explicit bounded degree circuits. Applications include parametric extensions of the shortest-path and spanning-tree problems and, in particular, the minimum-ratio-cycle problem, showing all these problems are in NC.

Key Words. Circuits, Probabilistic circuits, Poly-log depth, Parametric problems.

1. Introduction. A parametric technique presented in [M1] and [M2] has been applied in the design of many serial algorithms. It was shown in [M2] that sometimes a good parallel algorithm for one problem helps via this technique in the design of a serial algorithm for another problem. We show here that essentially the same idea can be used to prove that if one problem can be solved by a circuit of polynomial size and poly-log depth (that is, the problem is in the class NC), then a certain extension of the problem is also in NC. A similar result holds for probabilistic circuits, that is, if a problem is in the class RNC, then so is a certain extension of it.

The technique also sheds more light on the differences between strongly polynomial time and polynomial time vis-à-vis parallel computation. A polynomial-time algorithm for a problem with numerical inputs is said to run in strongly polynomial time if the number of arithmetic operations and comparisons is bounded by a polynomial in the number of inputs and is independent of the magnitudes of the inputs. In the context of sequential computation it seems that strongly polynomial time is merely an aesthetic advantage. However, this is not so in the context of parallel computation. To explain the difference, let us first introduce an example which is referred to several times in this paper.

Consider a parametric version of the shortest-path problem as follows. Suppose the edges of a graph on n vertices have lengths $d_{ij} = d_{ij}(t) = a_{ij}t + b_{ij}$ (assuming $a_{ij} > 0$), where t represents time. Let t^* denote the time at which the length $\delta(t)$ of the shortest path, connecting two distinguished vertices, exceeds a given integral length L (assuming, for simplicity, $\delta(0) < L$). Here, without knowing t^* explicitly we can compare it with a value $t = t'$ by computing the length of the shortest path at time t'. Thus, we can approximate t^* using a binary search. Furthermore, if all the a_{ij}'s and b_{ij}'s are rational then we can stop the search at a certain stage

[1] IBM Almaden Research Center, 650 Harry Road, San Jose, CA 95120-6099, USA, and School of Mathematical Sciences, Tel Aviv University, Tel Aviv, Israel.

Received October 31, 1986; revised November 16, 1987. Communicated by F. Thomson Leighton.

and compute t^* directly. This can be seen as follows. For simplicity, assume all the coefficients are integers of absolute value less than M. Then the binary search is for a positive rational number whose numerator and denominator are bounded by L and nM, respectively. This implies that $O(\log L+\log M+\log n)$ queries suffice. Such an approach yields a polynomial algorithm which is not strongly polynomial. The naive binary search is an "inherently sequential" operation. It does not seem that this algorithm can be parallelized to run in less than $p_1(\log n, \log\log L, \log\log M)$ time with less than $p_2(n, \log L, \log M)$ processors, where p_1 and p_2 are polynomials. However, we develop NC algorithms for problems of this kind. These are usually NC algorithms in a strong sense that the number of processors does not depend on the numerical values (provided we do not have to parallelize the execution of the arithmetic operations). These algorithms are also strongly polynomial as sequential algorithms.

2. Preliminaries. We find it convenient to work with the model of computation of a circuit (that is, a directed acyclic graph) consisting of gates which perform the operations $+$, $-$, $\times$, $\div$, MAX and MIN, and gates which tell whether a number is positive. The treatment of the latter in the extension of the circuit is analogous to that of the MAX and MIN gates. Henceforth we omit this type of a gate from the discussion. We call these circuits *arithmetic* circuits.

We say that input I of the circuit is *involved* in gate G (resp. wire w) if there is a path in the circuit leading from I into G (resp. w). A *set* of inputs is involved in a gate or a wire if at least one of the members of the set is involved in that gate or wire.

The input gates are usually fed with rational numbers but the basic idea of our technique applies to computation over any ordered field. The extension of the algorithm corresponds to feeding some of the input gates with linear functions of a single parameter, and then searching for some well-defined value of the parameter.

A concrete example of a problem to which the technique applies is the parametric shortest-path problem discussed in Section 1. Suppose the input gates of a circuit correspond to edges of a graph, and they are fed with the lengths of these edges. The circuit computes a shortest path between two designated vertices. Suppose the lengths of the edges are monotone increasing with time and we would like to compute the time at which the length of the shortest path exceeds a certain given threshold L. We show that since the shortest-path problem can be solved by a polynomial-size circuit of poly-log depth, this parametric problem can also be solved by such a circuit.

Many algorithms in combinatorial optimization can be carried out without multiplications or divisions. If a circuit has neither multiplication nor division gates then we can parametrize the entire set of inputs. In general, we use the notion of a parametrizable set of input gates defined as follows.

DEFINITION 2.1. A set S of inputs is *parametrizable* if it is not involved in any division gate, and for every multiplication gate G there is at most one wire w going into G such that S is involved in w.

The idea behind this definition is as follows. If we feed members of S with linear functions of a certain parameter t then, since we neither divide by a function of t nor multiply two functions of t, all the wires carry piecewise linear functions t.

The sense of the extension of the problem is that we can feed the members of a parametrizable set of inputs with linear functions of a parameter, and solve for the value of the parameter at which a certain value of the output is attained. This extension unifies many optimization problems, in particular max-min and min-ratio problems arising in location and scheduling (see [M2] and the references thereof).

3. Extensions of NC Algorithms. In this section we explain the extension operation. Consider an arithmetic circuit C with a certain parametrizable set S of inputs. For each input gate I_i, let $x_i = x_i(t) = a_i t + b_i$ denote some linear function (of an indeterminate t representing, say, time) associated with I_i, such that $a_i = 0$ for $I_i \notin S$. Suppose we feed each I_i with the value $x_i(t_0)$ where t_0 is any value of t, the same for all i. Obviously, the number carried by any wire can be described by a piecewise linear function of t.

The functions carried by the wires may in general have a large number of breakpoints, so a simultaneous implementation of the circuit at all the values of t may require a large number of gates. However, there are many applications where we are interested in implementing the circuit at some well-defined value $t = t^*$ which is *not given explicitly*. Furthermore, we are actually interested in computing t^*.

It is easy to operate on linear (rather than piecewise linear) functions. Suppose G is an ADD gate with inputs u and v and output $w = u + v$. If u and v are linear functions of t, $u = at + b$, $v = ct + d$, then it is trivial to replace the gate G by two ADD gates G_1 and G_2 that add $a + c$ and $b + d$, respectively. Similarly, if H multiplies u and z, where z is a constant, then H can be replaced by two gates H_1 and H_2 that multiply $a \times z$ and $b \times z$, respectively. The more interesting case is of course that of MAX gates.

A MAX gate receives two numbers as input. It outputs the maximum. We first explain the basic idea of the algorithm and then show how to carry it out in a circuit. We use for a while the terminology of processors. Suppose P processors attempt to run a parallel algorithm with numerical inputs which are given as linear functions of t. Suppose t is involved in the algorithm only in additions, subtractions, multiplications by constants, and comparisons. Also, suppose one of the outputs of the algorithm, $F = F(t)$ is a monotone function of t, and the value of $F(t^*)$ is known even though t^* is not known. We would like to compute t^* and all the other outputs at t^*. The extended algorithm maintains throughout the execution an interval $[\underline{t}, \bar{t}]$ that is guaranteed to contain t^*. The current values of all the "program variables" are linear functions of t over the interval $[\underline{t}, \bar{t}]$.

When a processor has to compare two such functions, the outcome may not be uniform on the entire interval $[\underline{t}, \bar{t}]$. In such a case the processor can at least compute a point t', such that the outcome of the comparison would be uniform over each of the intervals $[\underline{t}, t']$ and $[t', \bar{t}]$. Now, by setting t to t' and running the original algorithm, we can tell whether $t' < t^*$, $t' = t^*$, or $t' > t^*$. If this is done then the interval $[\underline{t}, \bar{t}]$ can be replaced by one of the intervals $[\underline{t}, t']$ and $[t', \bar{t}]$, and the processor can continue, so that all the variables remain linear functions of t over the revised interval. At the end all the outputs are linear functions of t, over some interval that contains t^*, so that t^* and all the outputs at t^* can be computed directly.

We now explain how to extend a circuit to carry out the idea of the preceding paragraph. Assume we are given an arithmetic circuit C with a parametrizable set S of input gates. Also, assume there exists an output gate whose value would be a monotone function of t if the gates in S were fed with linear functions of t (and the other ones with constants). We specify how to replace each gate of C by some arithmetic circuit. First, all the input and output gates that have to carry linear functions of t are each replaced by a pair of gates, so that they can carry the two coefficients of the function. Then ADD, SUBTRACT, and MULTIPLY gates are replaced by suitable circuits of fixed small size that (respectively) add or subtract linear functions of t, or multiply linear functions of t by constants. DIVIDE gates stay as they are since they operate on constants. Also note that, by assumption, MULTIPLY gates never multiply two linear functions of t. The more interesting part of the construction is the case of MAX gates.

Let G be a MAX gate. We replace G by a circuit $\bar{G}(C)$ as follows. First, let C_p (see Figure 1) denote a circuit, which is essentially the same as C except that it contains a "front-end"; this portion of the circuit amounts to two input gates per each input gate of C in the set S, and also one more input gate, through which it receives a value of t. The circuit C_p receives as inputs linear functions of t and a specific value of t. It then computes specific inputs for the copy of C contained therein. The input gates of each copy of C_p are appropriately linked to the input gates of the extended circuit $\bar{C}$ (that is, two gates per input gate of the original circuit which is in S). To keep the fan-out bounded, this linkage is indirect, using a standard binary tree structure. The circuit $\bar{G}(C)$ (see Figure 2) contains a copy of the circuit C_p. Besides the inputs of C_p, it also has four other input gates through which it receives the coefficients of the linear functions of t

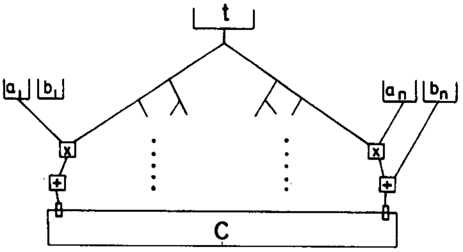

Fig. 1

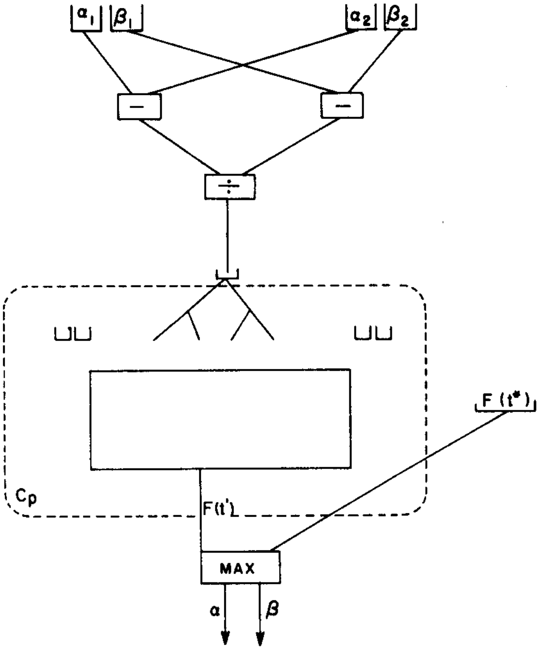

Fig. 2

it has to compare. The circuit $\bar{G}(C)$ first computes the intersection point t' of the two linear functions it receives (the cases of parallel or identical lines are trivial and hence omitted from our discussion). This value t' is sent to the copy of C_p contained in $\bar{G}(C)$ so that the problem is solved at t'. The resulting value $F(t')$ is compared with $F(t^*)$ (which is given as input to the grand circuit). Finally, $\bar{G}(C)$ has two outputs carrying the coefficients of the input function which is identified as the maximum (relative to function values at t^*). Thus, in this construction each MAX gate solves the entire problem at some value t', which it first computes from its inputs, and then carries out one comparison and forwards the result.

The dimensions of the extended circuit are related to the original one as follows. Let m and d denote the total number of gates and the depth of C, respectively. The extended circuit $\bar{C}$ is obtained by replacing each gate G of C by a circuit $\bar{G}(C)$ of size $O(m)$ and depth $O(d)$, and also adding a communication network (that is, wires and joints) of size $O(m^2)$ and depth $O(\log m)$ for linking every $\bar{G}(C)$ to the inputs. Thus, the extended circuit has size $O(m^2)$ and depth $O(d^2 + \log m)$. Hence, if C is an NC circuit then so is the extended circuit. We can state the result as follows.

THEOREM 3.1. *Let C be an arithmetic circuit of size m and depth d. Suppose C has a certain output gate whose result is a monotone function $F(t)$ of the parameter t if the inputs (in a parametrizable set of input gates) are values of linear functions* *of t. Under these conditions, there exists an arithmetic circuit of size $O(m^2)$ and depth $O(d^2+\log m)$ that solves the equation $F(t)=L$ for any L.*

Usually a more efficient extended circuit can be constructed. Obviously, it is not necessary that each gate will have its own copy of the original circuit. As argued in [M2], it can also be shown here that critical values produced by different gates can be tested in a binary search fashion. This would reduce the size of the circuit at the expense of increasing its depth (which remains poly-log though). The idea is as follows. We say that gate G is at depth k if k is the length of the longest path connecting any input to G. Let $C(k)$ and m_k denote the set of all MAX gates of C at depth k and the number of these gates, respectively $(k=1,\ldots,d)$. It suffices to use only $O(\log m_k)$ copies of C for all the gates in $C(k)$. These copies (connected in series) can be used to search for t^* in a set of m_k values. More precisely, let $\bar{C}(k)$ denote a circuit that receives m_k inputs, $t_1,\ldots,t_{m_k}$, and then outputs two values $a, b\in\{t_1,\ldots,t_{m_k},-\infty,\infty\}$, such that $a\le t^*\le b$, and there is no t_l such that $a<t_l<b$. The circuit $\bar{C}(k)$ can be assumed to have size $O(m\log m_k)$ and depth $O(d\log m_k)$ (these figures include a sorter for the inputs of $\bar{C}(k)$). Now, each MAX gate G at depth k is extended into a circuit that computes a critical value t_G, sends it to $\bar{C}(k)$, and then receives from $\bar{C}(k)$ the outcome of the comparison between t_G and t^*. With this information, G can perform the comparison for which it is responsible. The new construction has size $O(m\sum_k\log m_k)$ and depth $O(d\sum_k\log m_k)$. In any case we have the following theorem:

THEOREM 3.2. *Under the conditions of Theorem* 3.1, *there exists an arithmetic circuit of size $O(md\log m)$ and depth $O(d^2\log m)$ that does the same.*

Depending on the relative importance of depth and size, we can construct hybrids of the approaches that led to Theorems 3.1 and 3.2, so that even more efficient circuits can be designed for accomplishing the same task.

Theorems 3.1 and 3.2 can be used to show that many parametric versions of problems in NC are also in NC. An interesting example is the minimum cost-to-time ratio cycle in a network with edge-costs c_{ij} and edge-time $\tau_{ij}>0$. The problem is to find a simple cycle over which the ratio of total cost to total time is minimized relative to all simple cycles. It is well known that this problem is equivalent to finding the smallest t for which the network with edge-lengths $d_{ij}(t)=c_{ij}-\tau_{ij}t$ contains a cycle of total length less than or equal to zero. The problem of deciding whether a network has a cycle of total nonpositive length is in NC, since it can be solved by the all-pair shortest-path algorithm. This algorithm can easily be realized in a circuit with only ADD and MIN gates. Thus we have

COROLLARY 3.3. *The minimum-ratio-cycle problem is in NC.*

Other applications are parametric versions of spanning-tree problems, certain problems of computational geometry, linear programming in bounded dimension, and specializations of hard combinatorial optimization problems to trees (see the references of [M2] for more detail).

4. Extensions of RNC Algorithms. A probabilistic circuit is one that, besides the usual gates, also has gates that "toss coins." An RNC algorithm for a problem is a probabilistic circuit of polynomial size and poly-log depth, where one of the output gates indicates "success" or "failure"; the circuit computes the "correct" solution to the problem (and indicates success) with probability of at least $\frac{1}{2}$. Obviously, if the problem is processed by the circuit k times independently, the probability of (at least one) success is at least $1-2^{-k}$.

Consider an extension of such an RNC algorithm where each MAX gate G is replaced by a copy of the original circuit C, as described in the preceding section. The probability that all the copies of C succeed may be as small as 2^{-m}, if there are m copies of C. If one copy fails then the entire circuit may fail. However, this is not a serious difficulty. All we have to do is run each such copy K times, that is, use K copies instead of one for each MAX gate. The gate succeeds if at least one of these copies succeeds. Thus the entire circuit succeeds with probability of at least $(1-2^{-K})^m$. We choose K so that this probability is at least $\frac{1}{2}$. Obviously this results in depth $O(Kd^2)$ and size $O(Km^2)$. It is easy to see that a suitable K is $O(\log m)$ so the depth remains poly-log and the size remains polynomial.

References

[M1] N. Megiddo, Combinatorial optimization with rational objective functions, *Math. Oper. Res.* **4** (1979) 414-424.

[M2] N. Megiddo, Applying parallel computation algorithms in the design of serial algorithms, *J. Assoc. Comput. Mach.* **30**, 4 (1983) 337-341.

[MT1] N. Megiddo and A. Tamir, Finding least-distances lines, *SIAM J. Algebraic Discrete Methods* **4**, 2 (1983) 207-211.

[MT2] N. Megiddo and A. Tamir, New results on the complexity of p-center problems, *SIAM J. Comput.* **12**, 4 (1983) 751-758.

[MZ] N. Megiddo and E. Zemel, A randomizing $O(n \log n)$ algorithm for the weighted 1-center problem, *J. Algorithms* 7 (1986) 358-368.