OPTIMAL SNAPSHOTs AND THE MAXIMUM FLOW IN PRECEDENCE GRAPHS

ELI GAFNI† and VALMIR C. BARBOSA† ‡
Computer Science Department
University of California
Los Angeles, CA 90024

ABSTRACT

We consider a distributed asynchronous system in which each processor-state is associated with a value, and each system-state is associated with the sum of the values of the constituent processor-states. A run of the system reveals a set of consistent system-states (or snapshots). These are the system-states that appear in all the total-order extensions of the partial-order of events implied by the run. Given a run we are to identify a consistent system-state of maximum (or minimum) value. This problem arises in instances of Stochastic Relaxation, like Simulated Annealing, and Evidential Reasoning over Bayes Nets. In this paper we transform the problem into a maximum-flow problem over a network which is closely related to the graph of the partial-order on the events of the run. Furthermore, a particular scheduling mechanism especially suitable for Stochastic Relaxation gives rise to runs that can be processed by breaking them into segments and processing each segment in isolation. This lends the problem to be solved "on-the-fly" while the run is in progress.

1. INTRODUCTION AND MOTIVATION

Let \( X=(X_1,\ldots,X_n) \) be a vector and let \( f:R^n\to R^+ \) be a positive function. Starting with an instantiation \( x^0 \) of \( X \), consider the following probabilistic process:

Given \( x^k \), we set \( x^{k+1}:=\bar{x} \), where \( \bar{x} \) is obtained by choosing an arbitrary index \( i \), and randomly selecting a value \( \bar{x} \) for \( X_i \). The random selection is governed by the probability distribution \( P(X_i=\bar{x})=\alpha_i\frac{f(\bar{x})}{f(x)} \), where \( \alpha_i \) is a normalization constant, and \( y=(x_1,\ldots,x_{i-1},\bar{x},x_{i+1},\ldots,x_n) \). That is, \( \bar{x} \) and \( x^k \) might differ only in the \( i^\text{th} \) entry. We then set \( x^k=y \).

It was essentially proved by [Gema84], that if each index \( i, 1\leq i \leq n \), is chosen infinitely often in the probabilistic process above, then the distribution of \( X \) converges to

\[ P(X=\bar{x})=\alpha f(x) \]

where \( \alpha \) is a normalization constant.

† Supported by NSF grant DCR 84-51396 and by XEROX grant W850813.
‡ Supported by CAPES 3638/81-4, Ministry of Education, Brazil.
The case of interest for distributed systems is when \( f(x) \) is of the special form 
\[
f(x) = \prod_{i=1}^{l} g_i(x),
\]
where \( l \) is some positive constant and \( g_i(x) \), \( 1 \leq i \leq l \), is a function that depends only on few entries of the vector \( X \). In this case, in order to carry out the probabilistic step with respect to entry \( i \), it is sufficient to evaluate only these \( g_i \)'s that depend on \( X_i \). All the other functions are divided out. Hopefully, since each of the functions depends only on few entries, the probabilistic step may depend only on few entries of the vector \( X \).

This suggests immediately the possibility of parallelizing the probabilistic process. The probabilistic step with respect to two entries \( i \) and \( j \) can be carried out concurrently, provided no function \( g_i(x) \), \( 1 \leq i \leq l \), depends both on \( X_i \) and \( X_j \). Concurrent execution of these two steps is easily seen to be computationally equivalent to a serial execution of one step after the other.

An architecture that takes advantage of this parallelism consists of a network of processors \( P_i \), \( 1 \leq i \leq l \). Each processor \( P_i \) is responsible for carrying out the probabilistic step with respect to entry \( i \). We connect \( P_i \) and \( P_j \) by a bidirectional communication link if and only if the probabilistic step with respect to \( X_i \) and \( X_j \) cannot be carried out concurrently. In this network, independent sets of processors can carry the probabilistic step concurrently, and each processor is to be aware only of the current values of entries that correspond to its neighbors in the network. A mechanism to insure that no two neighbors carry out the probabilistic step simultaneously, and that each entry is involved in a step infinitely often, is described in this paper.

Variants of the probabilistic process outlined above occur in distributed Simulated Annealing [Kirk83] and Evidential Reasoning [Pear86]. In both cases a value of interest is \( x^* \) that minimizes (or maximizes) \( f(x) \). In general, in the absence of any simplifying properties of \( f(x) \) this is a difficult problem. Thus, one may settle for an approximation, by identifying the instantiation of \( X \) in the probabilistic process that minimizes (or maximizes) \( f \) over all instantiations that occurred in the run. For a sequential implementation of the probabilistic process this is quite simple. At time \( k \) one retains the minimum (or maximum) of the function seen up to then, and upon determination of \( x^* \) one evaluates \( f(x^*) \) and updates the minimum (or maximum).

Although the parallel execution is correct as a result of its equivalence to a serial one, two problems arise in trying to import the sequential mechanism to the parallel setting. First, the parallel execution is equivalent to many serial executions. Second, in the parallel execution no processor is aware of the value of \( f(x) \) at any point of time. Each processor is only aware of the multiplicative factor (or additive amount) by which it increases or decreases (the logarithm of) \( f(x) \) each time its variable changes.

In this paper we propose to identify the instantiation of \( X \) that minimizes (or maximizes) \( f \) over all the serial runs that are equivalent to the parallel run. In Section 2 we abstract the problem, pose it formally, and propose solutions via min- and max-flow techniques. In Section 3 we explain the ramifications of a particular scheduling proposed in [Barb86], and show that as a result, "on-the-fly" solution is possible.

2. PROBLEM FORMULATION AND SOLUTION

A partial-order \( E \) on events of a distributed system is the transitive closure of total orders \( E_i \) on the events that occurred at \( P_i \), \( i=1,\ldots,n \), and of a relation \( <_m \) on events at different processors. Without loss of generality we assume that the sequence of events \( e_{i,j}, j=0,1,\ldots,j_i \) that occurred at each given \( P_i \), \( i=1,\ldots,n \), are totally ordered according to the second index. The relation \( <_m \) between events that occurred at different processors, is such that \( e_{i,j} <_m e_{p,q} \) if and only if the reception of a message that was sent as direct result of event \( e_{i,j} \) triggered the event \( e_{p,q} \).

Let \( \Theta \) denote the set of total orders which are extensions of \( E \). Let \( v_{i,j} \) be a value associated with event \( e_{i,j}, i=1,\ldots,n, j=0,1,\ldots,j_i \). Also, let \( <_T \) be the relation between events corresponding to the total order \( T \). Problem 1 is to find an index \( i,j \) that solves

\[
\min_{i,j} \sum_{l=1}^{n} v_{i,j} \text{subject to } e_{i,j} <_T e_{l,i}.
\]

To see how Problem 1 is associated with the problem outlined in the Introduction, let \( v_{i,j} \) be the logarithm of the factor by which \( f \) is changed in the \( j \)-th time \( P_i \) carries out the probabilistic step. Had the process been carried out serially according to the order \( T \), then at the time that \( e_{i,j} \) is executed, the sum in Problem 1 is the logarithm of the factor by which the initial value of \( f \) differs from its current value.

Transformation of Problem 1 into an Independent-Set Problem

Without loss of generality, by possibly adding initial events of zero value, we assume that the sum in Problem 1 includes at least one value from each processor. We now relax \( E \) into \( E' \) by eliminating from \( E \) the relations between events that are solely related by "message-relation" \( <_m \). Formally, \( E' \) is the transitive-closure of \( E_m \), \( i=1,\ldots,n \), and the relation \( <_m \) between events that occurred at different processors, where \( e_{i,j} <_m e_{p,q} \) if and only if \( e_{i,j} < e_{p,q} \). From the formal definition of \( E' \) it is obvious that \( E' \) is a relaxation of \( E \). That is, every relation in \( E' \) is a relation in \( E \).

Pictorially, Figure 1 depicts the graph whose transitive-closure captures \( E \). This graph is a directed acyclic graph on the set of nodes that correspond to the events of the run. The partial order \( E' \) can be obtained by considering relations among arcs in Figure 1, where event \( e_{i,j} \) is associated with the arc leading from node \( e_{i,j} \) to node \( e_{i,j+1} \), and one arc precedes another if the former's head node precedes the latter tail node.

Now, given a total order \( T \in \Theta \) and an index \( i,j \), the set \( A \) of events that precede and include \( e_{i,j} \) in \( T \), has the property that no event in \( A \), the complement of \( A \), precedes (in \( E \)) an.
event in \( A \). Similarly, given a set \( A \) with this property there exist a total-order \( T \) and and index \( i,j \) such that \( A \) is the set of events that precedes and includes \( e_{i,j} \).

This property of \( A \) implies that it contains an independent set \( I \) of \( n \) events in \( E' \), one from each processor, such that the set of events in \( E' \) that precedes and includes \( I \) is \( A \). The set \( I \) is obtained by choosing the “latest” event in \( A \) that occurred at each \( P_i \), \( i=0,\ldots,n \). This set of events defines a snapshot - the collection of states in which each processor is left after executing the corresponding event. To see that this is an independent set, notice that if \( e_{i,j} < e_{p,q} \) and \( e_{p,q} \) belong to \( I \) and both are members of \( I \), then \( e_{i,j+1} < e_{p,q} \) and \( e_{p,q} \) does not belong to \( A \), which means that \( e_{i,j+1} > e_{p,q} \), contradicting the requirement that \( T \) is an extension of \( E' \). Thus, with each set \( A \) we identify an independent set \( I \) in \( E' \).

Similarly, with each independent set \( I \) in \( E' \) of cardinality \( n \) we can associate a set \( A \) of the type above by considering the set \( I \) and all the events preceding it in \( E' \). Let \( e_{i,j} \) and \( e_{p,q} \) belong to \( I \). If there is an event \( e_{i,j} \) in \( A \), preceding an event \( e_{p,q} \) in \( A \), then \( j\geq j+1 \), and \( q\geq q+1 \), and as a result \( e_{i,j} \) precedes \( e_{p,q} \) in \( E' \). This is a contradiction to the independence of the set \( I \). Thus, with each independent set in \( E' \) we identify a set of indices for the sum in Problem 1.

As a result, we can reformulate Problem 1 in terms of independent sets in \( E' \). With each event \( e_{i,j} \) we associate a new value \( v'_{i,j} \) as follows:
\[
v'_{i,j} = \sum_{i=0}^{n} v'_{i,j},
\]
Problem 2 is to solve
\[
\text{MIN} \sum_{\text{independent in } E} v'_{i,j},
\]
We have estableshed that Problem 1 and Problem 2 are equivalent.

Solution of Problem 2 via Min-Flow

Notice that by adding a constant to all values \( v'_{i,j} \) of events that occurred at \( P_i \), we are not altering the independent set that solves Problem 2, since \( I \) includes a single event from each processor. Thus we can consider Problem 2 as a maximization problem with positive event-values, by replacing each value with its negative, and then adding a constant large enough to make all values positive. Thus w.l.o.g Problem 2 is a maximization problem, and all \( v'_{i,j} \) are positive.

We now make the graph of Figure 1 into an \((s,t)\) capacitated graph \( G' \), depicted in Figure 2, on which we will consider a min-flow problem [Even79]. With each arc from \( e_{i,j} \) to \( e_{i,j+1} \) we associate a lower bound \( b_{i,j} = v'_{i,j} \), and an upper bound \( c_{i,j} = \infty \). With each message arc we associate lower bound of zero and upper bound of infinity. We connect a new node \( s \) to all nodes \( e_{i,0} \), \( i=1,\ldots,n \), and connect each node \( e_{i,j} \), \( i=1,\ldots,n \), to a new node \( t \). All the new arcs have lower bound of zero and upper bound of infinity. We now consider the minimum \((s,t)\) flow in \( G' \).

Proposition 1. The minimum \((s,t)\) flow in \( G' \) is the value of the solution to Problem 2. Moreover, the arcs in the maximum \((t,s)\) cut of \( G' \) is the independent set in \( E' \) that solves 2.

It is enough to prove the following Lemma. The rest follows directly from the min-flow max-cut theorem [Even79, Law976].

Lemma 1. There is a one-to-one correspondence between finite \((t,s)\) cuts and independent sets in \( E' \).

Proof. Let \((S,\bar{S})\) be a partition of the node set of \( G' \) such that \( S \) includes \( s \) and \( \bar{S} \) includes \( t \). Denote the set of directed arcs from \( S \) to \( \bar{S} \) and from \( \bar{S} \) to \( S \) by \((S,S)\) and \((\bar{S},S)\), respectively. The value of a \((t,s)\) cut defined by a partition \((S,\bar{S})\) is the sum of the upper bounds of arcs in \((S,S)\), minus the lower bounds of arcs in \((\bar{S},S)\). Thus in order to have a finite cut, the set \((S,S)\) must be empty. Consider arcs in \((S,\bar{S})\). Since for each processor \( P_i \), the sequence of node \( s, e_{i,0}, e_{i,1}, \ldots, e_{i,j}, \ldots \) is a directed path in \( G' \), there must be an arc leading from \( e_{i,j} \) to \( e_{i,j+1} \) such that node \( e_{i,j} \) belongs to \( S \) and node \( e_{i,j+1} \) belongs to \( \bar{S} \) (we tacitly allow the first of the two to be \( s \), and the last to be \( t \)). Consider all the arcs of this type. Suppose that they do not constitute an independent set of arcs in \( G' \). Let \( e_{i,j} \) and \( e_{i,j+1} \) be in the set and \( e_{i,j} < e_{p,q} \). Then there exists a directed path in \( e_{i,j}, e_{i,j+1}, \ldots, e_{p,q} \) that leads to \( t \) and from \( s \). Thus we have a contradiction to our hypothesis. But this implies that somewhere along the path there is an arc that belongs to \((S,S)\), contradicting the finiteness of the cut.

Thus a finite \((t,s)\) cut identifies an independent set of events-arc in \( G' \). The partial order implied by \( G' \) is the same as \( E' \) only that it is enlarged to include the set of new arcs, added in going from \( G \) to \( G' \). The assumption on the positivity of the lower bounds, will preclude them from participation in the optimal cut. On the rest of the arcs, \( G' \) and \( E' \) agree, and we have identified an independent set in \( E' \).

The reverse direction, namely that a set of arcs, one from each processor, such that the events corresponding to the arcs are independent in \( E' \), defines a finite cut in \( G' \) is left to the reader. 

Solution of Problem 2 via Max-Flow

As before, we assume that all \( v'_{i,j} \) are positive, but in contrast we consider the minimization problem. We build a graph \( G' \) as before only that “message-arcs” are of reversed directionality, and in addition we also add arcs \( e_{i,j+1} \) to \( e_{i,j} \). We use the following capacity assignment. All lower bounds are zero. The capacity of arc \( e_{i,j} \) to \( e_{i,j+1} \) is \( v'_{i,j} \). All other capacities are infinity. This construction is depicted in Figure 3. We have the following proposition.

Proposition 2. The maximum \((s,t)\) flow in \( G' \), is the value of the solution to Problem 2. Moreover, the forward arcs in the min-cut correspond to the independent set of events that
solves Problem 2.

A Lemma and an argument similar to the min-flow case holds here. Essentially it boils down to showing that if all \((S, \bar{S})\) arcs are of finite capacity then the events that correspond to the arcs in \((S, \bar{S})\) are independent in \(E'.\) If they are not, we again consider the sequence \(e_{i_1,1}, \ldots, e_{p,q}, e_{i_2,1}, \ldots \) in \(E'.\) Along this sequence there must be a switch from \(\bar{S}\) to \(S.\) If this switch corresponds to a message, then in \(G'\) since "message-arcs" have been reversed we have identified an infinite capacity arc in \((S, \bar{S}).\) Similarly, if the switch is an "event-arc", by adding infinite capacity arcs in the backward direction in \(G',\) we have identified again an infinite capacity arc in \((S, \bar{S}),\) and the result follows.

3. SCHEDULING BY EDGE REVERSAL

In [Barb86] a distributed scheduling mechanism especially suitable for Stochastic Relaxation is proposed. The mechanism, called edge-reversal, is based on [Gafn81, Chan84]. Basically it works as follows. Initially all the bidirectional communication links in the network that implements the relaxation are assigned a conceptual direction in such a manner that the underlying graph is acyclic. Each processor which is a sink in this conceptual graph carries out the probabilistic step. It then reverses the direction of all its adjacent links. It implements the reversal by sending its new value to its neighbors, whereby the neighbors interpret the direction of the link as incoming. The reversal of the links create new sinks, etc, and the process continues ad infinitum.

It is easy to see that the edge reversal guarantees no concurrent activation of neighbors, and it is free of deadlock and starvation [Chan84].

In a partial-order of events that is produced by a run that is governed by edge-reversal, the events that occurred at \(P_i\) can be partitioned into two sets of events. The set \(C_i\) of events of reception of values from neighbors that transform \(P_i\) into a sink, and the set \(\bar{C}_i\) of the remaining events. Let \(e_{i,j}\) and \(e_{m,j}\) be two consecutive events in the total order induced by \(C_i.\) Obviously we have \(v'_{i,j} = v'_{i,j}, \forall 1 \leq i \leq m.\) Moreover, given an event \(e_{p,q},\) let \(e_{p,q}\) be the first event in \(C_p\) which precedes or is \(e_{p,q}.\) It is not difficult to see that if a set \(\{ e_{i,j}\}, i=1, \ldots, n, \) is an independent set in \(E',\) then the set \((e_{i,j}), i=1, \ldots, n,\) is independent in \(E'.\) Since these two sets have the same value in Problem 2, we can w.l.o.g restrict \(E\) to be the partial order induced by the union of events in \(C_i, i=1, \ldots, n.\)

We now notice that if \(P_i\) and \(P_p\) are neighbors, then \(E\) restricted to the events that occurred at \(P_i\) and \(P_p\) is a total-order. Moreover, in this total-order the events at \(P_i\) alternate with the events at \(P_p.\) As a result, if \(e_{i,j}\) and \(e_{p,q}\) are independent in \(E',\) then \(|j−q| \leq 1\). Inductively, we assume that the network is connected, we have that if \(e_{i,j}\) and \(e_{k,l}\) are independent in \(E'\) then \(|j−f| \leq n.\) Thus, the elements of an independent set are not "far" apart.

This suggests the following way of processing the partial-order: We first compute \(v'_{i,j}\) for all \(i,j,\) and then consider a sequence of partial-orders, \(P_{r_1}, P_{r_2}, \ldots\) The partial order \(P_{r_k}\) is the one induced by the events \(e_{i,j}, i=1, \ldots, n, k−1 \leq j \leq (k−1)n,\) i.e. it is a "slice" of length \(2|p|+1\) of the original partial-order. For each partial-order we solve Problem 2. The minimum or maximum (depending whether one solves minimization or maximization problem) of all these solution is the solution to the original problem. The reason being that the subproblems overlap sufficiently so that every independent set resides entirely in at least one of the subproblems.

To implement the partial-order processing on-the-fly, we appoint a "leader" who collects (via messages) all the \(v'_{i,j}\) as the corresponding events occur. By the knowledge of the initial conceptual orientation of links, the leader can process \(P_{r_1}\) once it receives all \(v'_{i,j}, i=1, \ldots, n, j=0, \ldots, 2n,\) etc.

Various alternatives by which \(P_{r_1}\) is processed by distributed max-flow [Awer85, Gold86, Marb84] are also possible. The point is that in both cases one can discard history as the probabilistic process progresses.

We mention in passing that when one transforms the problem into a max-flow problem, one can discard the backward arcs \(e_{i,j+1}\) to \(e_{i,j}\) since the reversal of "message-edges" establishes an infinite capacity path from the first node to the second.

4. CONCLUSION

We have shown the relation between snapshots and cut-sets. Finally, for a particular application and scheduling we have proposed on-the-fly solution. A certificate to the practicality of the problem solved in this paper is that although most people "feel" what the problem is, it is rather difficult to formulate it without the precise notion of a snapshot [Chan85]. This notion was made precise only recently. If it was not around it would have to be invented just for this problem.

A point of interest is that by considering all total-order extensions of the run instead of a single one, we have exponentially (in \(n\)) more instantiations of \(X.\) The improvement to the approximation that this gets us is questionable, since the instantiations are not uniformly spread over the problem domain, but are rather very much correlated. We are now experimenting with various examples in order to be able to quantify this improvement.
REFERENCES


