ON DISTRIBUTED SEARCH

TED HERMAN AND K. MANI CHANDY

Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712

TR-84-24 August 1984

On Distributed Search

Ted Herman and K. Mani Chandy
Department of Computer Science
University of Texas at Austin

Keywords: Distributed computation, graph algorithms

* This research was supported by a grant from the Air Force Office of Scientific Research under grant AFOSR 81-0205.

1.0 *Introduction*

Many optimization problems have the following structure: A problem may admit many feasible solutions, and of these, one seeks the solution of lowest possible cost. In this paper, we consider the task of finding a feasible solution whose cost is bounded below some given threshold. As the threshold is lowered, an optimum solution is approximated. This work is based, in part, on the pioneering work of Karp and Held [3] on dynamic programming.

Our distributed program is a parallel search for the approximate solution. The underlying distributed system is an asynchronous network of processes, with indeterminant timings between computations. The distributed search therefore has this non-deterministic flavor: The computation advances whenever and wherever possible in the distributed system.

2.0 *The Distributed System*

The distributed system consists of a fixed set of processes that communicate solely by passing messages. A process may directly send a message to another process only if there exists a channel between the two. A message may contain arbitrary data. The channels are loss-less: If process B sends a message to process D, then we can assert that process D will receive the message (perhaps after some finite delay). The number and locations of channels is fixed in the distributed system. It is useful to think of processes as vertices and to channels as edges of a finite graph. The set of processes in the system is

$V = \{v_i\}$, $i = 1,\ldots,n$

and (v_i,v_j) denotes a channel/edge. The set E is the set of channels defined for the distributed system. If there is a channel between a pair of processes, then we call these processes *neighbors*.

3.0 *The Approximation Problem*

Let P be the problem to be approximated. The solution to P will be an item called a *policy*. Policies that satisfy the basic contraints of P are *feasible* policies. Define POLICY as the set of all possible policies. Then for $x \in$POLICY, the predicate function FEAS determines feasibility: FEAS(x) is true iff x is a feasible policy. Messages between processes will represent policies. Processes receive policies, amend them, and transmit them during the computation.

Policies are ordered by a cost function. COST(x) yields a number that is the cost of policy x. Policy x is *acceptable* if it satisfies

FEAS(x) and COST(x)<R,

where R is a number defining the level of approximation desired for problem P. Further restrictions on the COST function are revealed in Section 4.

There is a bijection from the set of processes V to a set of functions Z. Elements of Z are functions that map policies to policies. For convenience, let z_i correspond to process v_i.

The class of approximation problems suited to our analysis must have solutions expressible as compositions of functions in Z. Let $W(v_j)$ represent some finite walk, originating at v_j, over edges of the graph defined by the distributed system. $W(v_j)$ can be written in the form

$v_j v_{i(1)} v_{i(2)} \ldots v_{i(k-1)} v_{i(k)}$

where the sequence i(1), i(2), ..., i(k) designates the order of vertices in the walk. Corresponding to $W(v_j)$, we write the composition

$S(W(v_j),x) = z_{i(k)} \circ \ldots \circ z_{i(2)} \circ z_{i(1)}(x)$,

which is a generation sequence based on x generated by v_j. The *empty* walk contains no edges. $S(W(v_j),x) = x$ for the empty walk.

3.1 *Solution Characterization*: A solution to P is an item y satisfying:

(a) FEAS(y) and COST(y)<R, or

(b) FEAS(y) and
{For all w∈POLICY, FEAS(w) implies COST(y)≤COST(w)}, or

(c) y=⊥ and {For all w∈POLICY, FEAS(w) is *false*}.

The first case provides an acceptable policy. In the second case no acceptable policy can be found, so a minimum cost policy is the result. In the last case, the output is a special symbol ⊥, which indicates that no feasible solution to P is possible. By convention COST(⊥) = ∞. When a process v_i sends a message y to some other process, then v_i is said to *generate* y. We say a distributed computation generates y if any v_i in the system generates y during the computation. The class of approximation problems to be considered can now be precisely characterized.

3.2 *Proposition*: A solution to approximation problem P, P = [V,E,Z,FEAS,COST,R], can be generated by a distributed computation, provided that a solution to P is expressible as $S(W(v_*),\delta)$, for some selection of $W(v_*)$, where v_* is the label of a vertex called the *initiator*, and δ is some initial policy.

4.0 *Distributed Algorithm*

We expose the distributed algorithm in successive refinements. First, a simple procedure will suffice to generate an acceptable policy. Subsequent procedures generate solutions with greater economy. Then termination criteria are introduced to complete the algorithm.

The computation will begin at the initiator v_*. The initiator will originate the distributed computation by sending policy δ to its neighbors, following proposition (3.2). Let MESSAGE(i,j,x) denote the event: v_j receives policy x, sent by v_i. The behavior of v_j following this event is described by the following procedure.

4.1 *Procedure*: Upon MESSAGE(i,j,x) v_j sends $z_j(x)$ to all neighbors.

4.2 *Lemma*: The procedure 4.1 generates a solution to problem P.
Proof. By definition, the specification to problem P includes a set of functions Z such that a solution to P is in the form $S(W(v_*),\delta)$. Since procedure 4.1 generates all possible $W(v_*)$, a solution to P must be generated. ■

Note that procedure 4.1 does not terminate, nor does it recognize a solution to P. Lemma 4.2 only states that some process v_i in the distributed system will, at some point, send a message $z_i(x)$, where $z_i(x)$ is a solution to P.

It is important to consider the effect that an individual message has on the course of the distributed computation. For example, if a message x can be removed from a computation and P is solved anyway, then message x should not be generated for reasons of efficiency. The next results develop apparatus needed to decide when a policy x should be discarded.

4.3 *Procedure*: Upon MESSAGE(i,j,x), if there is no solution to P of the form $S(W(v_j),z_j(x))$, for any $W(v_j)$, then v_j sends no messages. Otherwise procedure 4.1 is invoked.

Temporarily we focus on P where $R=\infty$. P is therefore a search for any feasible policy. Let $\mathscr{L}_j$ be a relation over policies such that

$x\mathscr{L}_j y$ *IFF* For all $W(v_j)$, $FEAS(S(W(v_j),x)) = FEAS(S(W(v_j),y))$.

The reader can verify that $\mathscr{L}_j$ is an equivalence relation. Informally, $x\mathscr{L}_j y$ means that policy x and policy y behave equivalently (with respect to feasibility) under any sequence of applications of functions in Z. Let $\mathscr{L}_j(x)$ denote the equivalence class of x. The following procedure is suited to the search for any feasible policy.

4.4 *Procedure*: Upon MESSAGE(i,j,x), v_j computes $z_j(x)$ and determines $\mathscr{L}_j(z_j(x))$. If v_j has previously sent $z_j(y)$ to its neighbors, for some $z_j(x)\mathscr{L}_j z_j(y)$, then v_j sends no messages. Otherwise procedure 4.3 is invoked.

4.5 *Lemma*: Procedure 4.4 generates a solution to P ($R=\infty$).
Proof. Let z be a solution to P, $z = S(W_1(v_*),\delta)$. Consider tracing W_1 versus an execution of procedure 4.4. Notice that any execution of 4.4 generates at least one prefix of W_1 because the initiator v_* sends δ to all its neighbors. Let W_2 be the longest prefix of W_1 generated by some execution of 4.4, where W_2 terminates at $v_{i(1)}$ Since W_2 is the longest prefix we infer that $v_{i(1)}$ did not send $x = S(W_2(v_*),\delta)$ to its neighbors. It follows that $v_{i(1)}$ previously sent y, for some $y\mathscr{L}_j x$. If x can be extended to feasibility, so can y, and we therefore continue tracing W_1 starting at $v_{i(1)}$ with policy y. This argument can repeated to exhaust W_1 and obtain a feasible solution. ■

Returning to the case R finite, cost is of importance. We now define a cost-sensitive equivalence relation similar to one defined in [3]. Let $\equiv_j$ be a relation over policies.
$x\equiv_j y$ *IFF*
(a) $\equiv_j$ is an equivalence relation,
(b) $x\mathscr{L}_j y$ holds, and
(c) for all $W(v_j)$, $COST(S(W(v_j),x))\leq COST(S(W(v_j),y))$.

4.6 *Procedure*: Upon MESSAGE(i,j,x), v_j computes $z_j(x)$ and determines $\equiv_j(z_j(x))$. If v_j has previously sent $z_j(y)$ to its neighbors, for some $z_j(x)\equiv_j z_j(y)$, and $COST(z_j(y))\leq COST(z_j(x))$, then v_j sends no messages. Otherwise procedure 4.3 is invoked.

4.7 *Lemma*: Procedure 4.6 generates a solution to P.
The proof is similar to the proof of lemma 4.5. Given some execution of procedure 4.6 and some optimum policy z, we can trace the walk of z and show that procedure 4.6 either generates z or another feasible policy of equal cost. ■

4.8 *Theorem*: There is an algorithm to solve P if

(a) Proposition (3.2) is satisfied, and
(b) $\equiv_j$ has finite rank for each v_j, and
(c) For all $v_j \in V$ and $x \in$POLICY and for all $W(v_j)$, $COST(x) \leq COST(S(W(v_j),x))$.

Proof. There are three parts to the proof. First we show that any policy generated by procedure 4.6 is finite under conditions (a-c). This result will show termination of procedure 4.6. Finally, we appeal to previous work on diffusing computation [1,2] to detect termination and output a result.

(1) Any policy $S(W(v_*),\delta)$, generated by procedure 4.6, must satisfy: For all $v_j \in V$, $W(v_*)$ can contain at most M_j occurrences of v_j, where M_j denotes the rank of $\equiv_j$.

Proof (by contradiction). Suppose, on the contrary, that $W(v_*)$ contains $k>M_j$ occurrences of v_j, for some j. Let W_1, ..., W_k be the prefixes of $W(v_*)$ that terminate at v_j. Clearly, there must be two distinct prefixes $W_{i(1)}$ and $W_{i(1)}$, "walks of the same equivalence class,"

$$[S(W_{i(1)}(v_*),\delta)] \equiv_j [S(W_{i(2)}(v_*),\delta)].$$

Since $W_{i(1)}$ is a prefix of $W_{i(2)}$ (or vice-versa), we must conclude that

$$COST[z_j(S(W_{i(2)}(v_*),\delta))] < COST[z_j(S(W_{i(1)}(v_*),\delta))]$$

for if the walk $W_{i(2)}$ is extended, then by the logic of procedure 4.6, it must represent a lower cost policy than that of $W_{i(1)}$. But the equivalence of these two policies and part (c) of the premise implies the contrary, hence there is a contradiction.

(2) Procedure 4.6 terminates.
Proof. We prove termination by showing that every process v_j eventually reaches a permanently inactive state. Part (1) implies that every walk induced by an execution of procedure 4.6 has finite length. Since every step of the computation of procedure 4.6 extends or terminates some walk, and every walk is finite, we conclude that computations will cease in finite time.

(3) Termination detection/answer extraction.
Our plan to base an algorithm on procedure 4.6 will entail local variables for each process: Each v_j maintains a representative policy for each class of $\equiv_j$. The value of such a local variable is initially $\perp$, and is subsequently updated whenever MESSAGE(i,j,x) reflects an improvement in cost for $\equiv_j(z_j(x))$. Then the algorithm can succeed in two ways: First, some acceptable x may be found, in which case v_j discovers x and should broadcast a message throughout the distributed system to halt further activity. In the second instance, no feasible x with $COST(x)<R$ exists, so the algorithm produces a policy of optimum cost--which will reside in a local variable. This optimum policy must be extracted when the distributed computation halts; the diffusing computation protocols [1,2] provide suitable termination detection and extraction techniques. ■

5.0 *Applications*

5.1 *Tour*: This example is an approximation to the travelling salesman's problem. The problem is to find a low-cost tour through m vertices. For instance, if $m=3$, we wish to search the set of cycles r,p,q,r where p and q are distinct vertices. A motivation for this problem is that a process r may require a communication cycle through two other processes for the purpose of soliciting votes on major issues.

A policy will be a sequence of edges corresponding to some path beginning at v_*. In terms of proposition 3.2, a policy is feasible if it represents a cycle that begins and ends at v_*, contains m vertices, and has no repeated intermediate vertices. The cost of a policy is the sum of the weights of its edges.

The function z_j will extend policy MESSAGE(i,j,x) by adding (v_i,v_j) to x. Under this scheme, equivalence $x \equiv_j y$ holds if

(a) $z_j(x)$ and $z_j(y)$ cannot be extended to feasibility, that is, they contain repeated vertices or contain more than m vertices.
(b) x and y can be extended to feasibility, and they are both permutations of T, a subset of V, where $|T| \leq m$.

Following procedure 4.3, v_j will not send $z_j(x)$ to neighbors when case (a) applies. Case (b) implies that $\mathrm{Rank}(\equiv_j) = 2^{|V|}$ in the worst case (m=n). This could lead to an exponential requirement for space, to accomodate local variables for each equivalence class. Since the travelling salesman problem is NP-complete, the exponential result is expected for a worst case.

5.2 *Shortest Walk*: Here we seek low-cost walks from the initiator to all other vertices. Edges have associated positive weights, and the cost of a walk is the sum of its edge-weights. A policy can be adequately represented by its cost and final vertex, since $x \equiv_j y$ holds for any walks x and y that terminate at v_j. Consequently $\mathrm{Rank}(\equiv_j) = 1$, so v_j simply keeps track of the best policy reaching v_j during the course of the computation. This program is extended to the case of negative cycles in [2].

References

[1] E. W. Dijkstra and C. S. Scholten, Termination Detection for Diffusing Computations. *Information Processing Letters 11* 1(Aug 1980), pp. 1-4.
[2] K. M. Chandy and J. Misra, Shortest Path Algorithms, *Comm. ACM 25* 11(Nov 1982), pp. 833-837.
[3] R. M. Karp and M. Held, Finite-State Processes and Dynamic Programming, *Siam J. Appl. Math. 15* 3(May 1967), pp. 693-718.