A Distributed Algorithm Solving CSPs with a Low Communication Cost

Nicolas Prcovic
CERMICS-INRIA Sophia Antipolis
2004, Route des Lucioles, B.P. 93
06902 Sophia Antipolis Cedex
nprcovic@sophia.inria.fr

Abstract

We present a new distributed algorithm which finds all solutions of Constraint Satisfaction Problems. Based on the Backtrack algorithm, it spreads subtrees of the search tree over processes running in parallel. The work is equitably shared among the processes while the communication cost remains low. We show that the speedup of the resolution is asymptotically linear as the number of variables increases.

1. Introduction

This paper presents a distributed algorithm which finds all the solutions of CSPs (total exploration of the search tree) and splits different parts of the tree to allocate them to processes in a network of computers. There has been a lot of researches on parallel processing of backtracking search for general tree search problem[3, 2, 6, 4]. Good experimental results have been reported and theoretical works about the load balancing among processes are numerous but we have found very few theoretical work about the minimization of the communication cost and have decided to focus our interest on it.

2. The Distributed Algorithm

The goal of the distribution is to increase the performances linearly, i.e. to achieve the same work as a single process while dividing the resolution time by the number of engaged processes. To get the nearest to this aim, we have to minimize the main overheads due to additional operations brought by distribution : the number of interprocess messages and the idle-time of processes (due to a bad sharing of the work).

Distributed tree search is achieved by spreading the root nodes of subtrees among all the processes. Each process searches for its own subtree the same way as if it was alone. The tree structure generally cannot be split into a given number of equal parts so we must adopt a finer grain than just the number of processes to divide the tree, i.e. each process will search several subtrees, but we have to take care of minimizing the message traffic between processes to keep the communication cost as low as possible.

2.1. Basic Scheme of the Algorithm

All the processes have the same function: running a depth-first search of their own subtree. One process has a special status, the master (the others are simple workers) and has the additional function to share the work. At the beginning, the master owns the root node of the tree whereas the workers don’t own any node. Whenever a worker has no work to do (at the beginning or when it has finished its subtree), it asks the master for a new subtree root node. During the resolution, the master shall give : first, all the sons but the leftmost one of the root node of the search tree, then all the sons but the leftmost one of the remaining node, and so on, until it meets the leaf of the left branch of the tree. When the master has no more work to do, it asks all the workers for the depth of their root node and exchange its master status with the worker which has one of the shallowest root nodes. If no worker has any node then it sends a “stop” message to all the workers and stops itself.

2.2. Theoretical Performance of the Algorithm

Two kinds of messages occur several times during the resolution: some for changing the master and others for asking for a node to the master. Let n be the number of variables, let d be the cardinal of the largest domain, let p be the number of processes and let \( \pi_i \) be the depth of the root node of a process i.

Theorem : The total number of messages is lower than

\[ dpn^2 + pn. \]
Proof: First, we calculate the maximal number of changes of master. Let's consider $II = \sum_{i=1}^{p} \pi_i$. The master has at first the shallowest root node (depth: $\pi_m$). When it exchanges its status, it becomes a worker and ask the new master for a node of depth $\pi_m < \pi$ and the others $\pi_i$ can't have decreased so $II$ strictly grows. At the beginning $II > 0$, and as $\forall i \pi_i \leq n$, we finally have $II \leq pn$. Thus, the total number of changes of master is lower than $pn$. Now, we calculate the maximal number of asked nodes. As long as the master remains the same, it will distribute the brothers located at the right of the node of his left branch, so a maximum of $\sum_{i=1}^{\pi_m} (|D_i| - 1)$ times ($D_i$ denotes a domain) which is lower than $n!d$. This will be done for each new master so, for all, less than $d!n^2$ times.

We sum up the main characteristics of the algorithm: (1) it shares equitably the work among the processes: the work is shared again each time a process has finished its subtree and a process stays idle only when waiting for an answer to its message, (2) the number of messages is bounded by a polynomial function $(d!n^2 + pn)$, (3) the number of generated nodes is the same as the one for the sequential Backtrack algorithm (the difference lies in the stack which is split between the process: a node created from a local stack may be pushed on another stack).

All this leads us to the following conclusion: the communication overhead will become insignificant compared to the tree search time (which is in $O(d^n)$) when the size of a problem is big. So, we have an asymptotic linear speedup when $n$ or $d$ are large enough for any fixed $p$.

3. Experimental Results

The algorithm was implemented as a function library intended to take part of PROSE [1], a functional tool-box for constraint interpretation, and using CHOOE [5], a distributed environment manager.

The experiments were done on a network of Sun stations. We tested our algorithm on several instances of the classical $n$-queen problem (see figure 1) and of the Golomb problem using Forward-Checking. The results confirm what we expected: (1) the speedup becomes nearly linear as $n$ gets large for any fixed $p$, (2) the higher $p$ is, the lower the speedup efficiency is for any fixed $n$. We can notice that when the problems are small, our algorithm is not performant. This bad behavior of the algorithm can be explained by the following reason: we observed that during the last seconds of the resolution, when the processes only possessed a few nodes to generate, there were some "useless" and time-expensive efforts to share the remaining work. Therefore, it would be better to fix a threshold to the depth of the nodes used to make the change of mastership and the node donations because, at a too deep level, the communications cost more than the rest of the resolution.

4. Conclusion

We managed to build a simple distribution mechanism which is a low time consumer while preserving a balanced work sharing. We proved and experimented that the speedup converges to linearity when the size of a problem is large. Meanwhile, a naive and direct implementation of the algorithm doesn't produce good results when the problem is small and we proposed some minor changes to enhance its efficiency. However, it seems clear that this algorithm is particularly well suited for large problems to distribute over a network of distant computers.

![Figure 1. Speedup: single process time/time with $p$ processes](image)

References