Alcom-FT Technical Report Series
ALCOMFT-TR-02-107

Parallel Controlled Conspiracy Number Search

Ulf Lorenz*
Department of Mathematics and Computer Science
University of Paderborn
Germany
flulo@uni-paderborn.de

ABSTRACT

Tree search algorithms play an important role in many applications in the field of artificial intelligence. When playing board games like chess etc., computers use game tree search algorithms to evaluate a position. In this paper, we present a procedure that we call Parallel Controlled Conspiracy Number Search (Parallel CCNS). Shortly, we describe the principles of the sequential CCNS algorithm, which bases its approximation-results on irregular subtrees of the entire game tree. We have parallelized CCNS and implemented it in our chess program P.ConNerS, which now is the first in the world, that could win a high-ranked Grandmaster chess-tournament. We add experiments that show a speedup of about 50 on 159 processors running on an SCI-workstation-cluster.

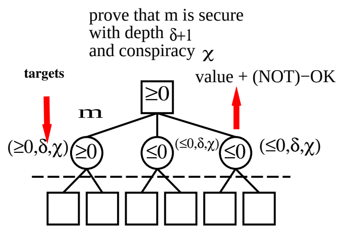
1. INTRODUCTION

For most of the interesting games like chess we are forced to base our decisions on heuristic or vague knowledge. An approximation for the best decision is mostly done by the following method: Firstly, a partial game tree, rooted at the starting position, is chosen for examination. We call such a preselected partial tree an *envelope*. A search algorithm assigns heuristic values to the leaves of the envelope and propagates these values up the tree by the minimax principle. An envelope is usually examined by the help of the $\alpha\beta$-algorithm [2][1].

Although the approximation of the real root value by the help of fixed-depth envelopes leads to good results, it is widely accepted that the form of the preselected tree strongly determines the quality of the search result. In addition to pure evaluation algorithms (like the $\alpha\beta$-algorithm), there is the class of the so called *incremental* searching algorithms [4], which 'grow' the search tree one step a time. At each step a leaf of the current tree is chosen (selection), and the successors of that leaf are added to the tree (expansion). The new leaves are evaluated and the new heuristic values are updated bottom up (update). These algorithms are best-first proof procedures which need linear space in the number of searched nodes. Their advantage, however, is that the grown trees need not be of uniform depth and the envelopes need not be determined before the search is finished. An example is that of Conspiracy Number Search, introduced in 1988, [4][5]. The aim of CNS is to distribute the available resources in a way that it is guaranteed that a certain number of arbitrary changes of leaf-values does not change the mimimax-value, resp. the best move at the root. Lorenz [3] has presented an efficiently working, carefully directed search procedure, which is called Controlled Conspiracy-2 Search (CC2S), and which properly works.

*This work was supported by the DFG research project "Selektive Suchverfahren" under grant Mo 285/12-3.
Partially supported by the Future and Emerging Technologies programme of the EU under contract number IST-1999-14186 (ALCOM-FT).

2. DESCRIPTION OF CCNS



The left figure shows the data flow in the sequential CCNS-algorithm. We try to separate a best move from the others, by proving that there exists a number x so that the minimax value of the successor with the highest payoff is at least x, and the payoffs of the other successors are less or equal to x.

As we work with heuristic values which are available for all nodes, the searched tree offers such an x and a best move m, at any point of time. As long as m is not *secure* enough, we take x and m as a hypothesis only, and we commission the successors of the root either to show that new estimations make the hypothesis fail, or to verify it. The terms of 'failing' and 'verifying' are used in a weak sense: they are related to the best possible knowledge at a specific point of time, not to absolute truth. New findings can cancel former 'verifications'. The verification is handled by the help of *targets*, which are split and spread over the search tree in a top down fashion. A target t expresses a demand of the following kind to a node v: Prove that your value is smaller (greater) than a certain bound, where the leaves of the proof-tree have a distance of at least t to v, and the result is stable against c arbitrary changes of leaf-values! Each successor of a node, which is supplied by a target, takes its own value as an expected outcome of a search below itself, and commissions its successors to examine some sub-hypotheses, etc. A target t will be *fulfilled* when t demands a leaf, or when the targets of all of v's successors are fulfilled. When a target is fulfilled at a node v, the result 'OK' is given to the father of v. If the value of v changes in a way that it contradicts the value component of t the result will be 'NOT-OK'.

For a detailed description, we refer to [3].

2.1 The Distributed Algorithm

In the following, let tuples (v, t_v) represent subproblems, where v is a root of a game tree $G = ((V, E), f)$ and t_v a target, belonging to v. The task is to find a subtree of G, rooted at v, which fulfills all demands that are described by t_v.

Our parallelization of the CCNS-algorithm is based on a dynamic decomposition of the game tree to be searched, and on parallel evaluation of the resulting subproblems. Although the sequential CCNS-algorithm is a best-first search algorithm, which is able to jump irregularly in the searched tree, the algorithm prefers to work in a depth-first manner. To do so, a current variation is additionally stored in the memory. On the left and on the right of this current variation, there exist nodes which have got targets. These nodes, together with their targets describe subproblems. The idea of the parallelization is to make the not yet examined subproblems available for parallel evaluation. By this, several processors may start a tree search on a subtree of the whole game tree. These processors build up current variations by themselves.

Initially, all processors are idle. The host processor reads the root position and sends it to an especially marked processor. In the following this processor behaves like all other processors in the network.

A processor, which receives a subproblem, is responsible for the evaluation of this subproblem. It starts the search of this subproblem as in the sequential algorithm, i.e. it builds up a current variation of subproblems. Other subproblems are generated, which are stored for parallel evaluation. By the help of a WORK-message, these subproblems are sender-initiated posted to other processors, where a destination Q is determined by a hash-function from nodes into the set $\{1 \dots N\}$, N being the number of processors. Such a worker Q starts a search below the root of its new subproblem. If v has not yet been examined on processor Q before, Q sends an ADDRESS-message to processor P, which contains the address of the node v on processor Q. By this, processor Q later can find the node v again, for further computations.

A processor Q which has solved a subproblem by itself or with the help of other processors, sends an ANSWER-message to its employer P. This message contains a heuristic value computed for the root of the concerned subproblem, as well as OK or NOT-OK, depending on whether the target of that subproblem could be fulfilled.

An employer P can also resolve an employer-worker relationship. Let v be the root of a subproblem p, sent to Q. Let v' be the predecessor of v. If processor P must reorganize its search at the node v', it sends a STOP-message to Q, and Q cancels the subproblem which belongs to the node v.

3. EXPERIMENTAL RESULTS

All results are taken with our chess program P.ConNerS. The used hardware consists of the PSC-workstation cluster at the Paderborn Center for Parallel Computing. Every processor is a Pentium II/450 MHz running the Linux operating system. The processors are connected as a 2D-Torus by a Scali/Dolphin interconnection network. The communication is implemented on the basis of MPI.

3.1 Grandmaster Tournament

In July 2000 P.ConNerS won the 10^{th} Grandmaster tournament of Lippstadt (Germany). Thus, for the very first time, a chess program has won a strong, categorized FIDE tournament (category 11). The opponents had an average ELO-strength of 2522 points and P.ConNerS played a performance of 2660 ELO-points. (The ELO-system is a statistical measure for the strength of chess players. An International Master has about 2450 ELO, a Grandmaster about 2550, the top-ten chess players have about 2700 ELO on the average, and the human World Champion about 2800 ELO.)

3.2 BT2630 Test

Another possibility how to get an impression of the playing-strength of a program is to compare its strength by means of a set of selected positions, with other programs and human players. On the widely accepted BT2630 test, P.ConNerS achieves 2589 points, which is a result that is not reached by any other program. The test consists of 30 positions, each of which is assigned with a solution recommended by a human Grandmaster. For each of those positions, a program exactly gets 15 minutes computation time for finding the solution. The program is punished with 900 points for every position on which it does not find the 'correct' solution. Concerning the other positions, the program is punished with the number of seconds up to the latest point of time, when it finds and keeps the solution.

By this procedure, a pseudo-ELO number is assigned to the tested chess-program.

3.3 Speedups

P	SPE	SO(%)	LO(%)
159	50	45	45
79	30	45	55
39	20	35	65
19	10	40	70
9	5	30	50
3	2	30	50
2	1	15	50

The table on the left presents us with the data from the parallel evaluations (averaged over the 30 instances of the BT2630 test). As can be seen, the overall speedup (SPE) is about 50 on 159 (P) processors. The search overhead (SO) is kept in an acceptable range, so that the main disabler for even better speedups is the limited average-load (LO).

4. REFERENCES

[1] R. Feldmann, M. Mysliwietz, and B. Monien. Studying overheads in massively parallel min/max-tree evaluation. In *6th ACM Annual symposium on parallel algorithms and architectures (SPAA'94)*, pages 94–104, New York, NY, 1994. ACM.

[2] D.E. Knuth and R.W. Moore. An analysis of alpha-beta pruning. *Artificial Intelligence*, 6(4):293–326, 1975.

[3] U. Lorenz. Controlled Conspiracy-2 Search. *Proceedings of the 17th Annual Symposium on Theoretical Aspects of Computer Science (STACS), (H. Reichel, S.Tison eds), Springer LNCS*, pages 466–478, 2000.

[4] D.A. McAllester. Conspiracy Numbers for Min-Max searching. *Artificial Intelligence*, 35(1):287–310, 1988.

[5] J. Schaeffer. Conspiracy numbers. *Artificial Intelligence*, 43(1):67–84, 1990.