Theoretical Computer Science 644 (2016) 127–142

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Conspiracy number search with relative sibling scores ☆

Jakub Pawlewicz [a,*], Ryan B. Hayward [b,*]

[a] Institute of Informatics, University of Warsaw, Poland
[b] Computing Science, University of Alberta, Canada

ARTICLE INFO

Article history:
Received 22 January 2016
Received in revised form 17 May 2016
Accepted 20 June 2016
Available online 27 June 2016

Keywords:
Conspiracy number search
Search algorithm
Hex

ABSTRACT

For some two-player games (e.g. Go), no accurate and inexpensive heuristic is known for evaluating leaves of a search tree. For other games (e.g. chess), a heuristic is known (sum of piece values). For other games (e.g. Hex), only a local heuristic — one that compares children reliably, but non-siblings poorly — is known (cell voltage drop in the Shannon/Anshelevich electric circuit model). In this paper we introduce a search algorithm for a two-player perfect information game with a reasonable local heuristic.

Sibling Conspiracy Number Search (SCNS) is an anytime best-first version of Conspiracy Number Search based not on evaluation of leaf states of the search tree, but — for each node — on relative evaluation scores of all children of that node. SCNS refines CNS search value intervals, converging to Proof Number Search. SCNS is a good framework for a game player.

We tested SCNS in the domain of Hex, with promising results. We implemented an 11-by-11 SCNS Hex bot, DeepHex. We competed DeepHex against current Hex bot champion MoHex, a Monte Carlo Tree Search player, and previous Hex bot champion Wolve, an Alpha–Beta Search player. DeepHex widely outperforms Wolve at all time levels, and narrowly outperforms MoHex once time reaches 4 min/move.

We tested the strength of SCNS features: most critical is to initialize leaves via a multi-step process. Also, we show a simple parallel version of SCNS: it scales well for 2 threads but less efficiently for 4 or 8 threads.

1. Introduction

Consider a 2-player perfect information game with no known global heuristic, but with a reasonable *local* heuristic evaluation (good at relative scoring of children of a node, but bad at comparing non-sibling nodes). Suppose you want to build a bot for this game. What algorithm would you use?

The usual algorithms have drawbacks for a game with only a local heuristic. $\alpha\beta$ Search [20] needs a globally reliable heuristic. Monte Carlo Tree Search[1] [9,8], which uses random simulations, needs no heuristic but can be slow to converge.

☆ Research supported by Natural Sciences and Engineering Research Council of Canada Discovery Grant 137764.

* Corresponding authors.
E-mail addresses: pan@mimuw.edu.pl (J. Pawlewicz), hayward@ualberta.ca (R.B. Hayward).

[1] MCTS is a non-uniform best-first search that uses random simulations to evaluate leaves. Strong moves are exploited; weak moves are explored only if a visit threshold — based on an exploitation/exploration formula such as Upper Confidence Bound [21] — is crossed.

http://dx.doi.org/10.1016/j.tcs.2016.06.027
0304-3975/

Proof-Number Search[2] [2] performs well as a solver, particularly on search trees with non-uniform branching, but can be weak as a player, especially early in games with search trees with almost uniform branching.

In this paper we introduce Sibling Conspiracy Number Search, an algorithm for a two-player perfect information game with a reasonable local heuristic. SCNS is based on Conspiracy-Number Search [25,26], which generalizes PNS: in PNS, each search tree leaf score is -1 or 1, while in CNS a leaf score can have any value — e.g. any floating point value in the range from -1 to 1 — that indicates an associated final game score. For a node in a search tree and a target minimax value, the conspiracy number is the minimum number of leaves whose evaluations must change in order for the node's minimax score to reach the target. CNS expands leaves in an order that is based on conspiracy numbers. SCNS combines features of MCTS (anytime, best-first) and PNS (strong tactically, approaching perfect play near the end of a game). We will explain CNS and SCNS in further detail later.

Hex has a reliable local heuristic,[3] so we pick 11×11 Hex as our test domain. We ran DeepHex, our SCNS Hex bot, against an MCTS player (current champion MoHex) and an $\alpha\beta$ player (previous champion Wolve) [4,14]. DeepHex outperforms Wolve at all time levels, and outperforms MoHex once time reaches 4 min/move.

Next, we measure the relative contribution of the feature enhancements of our SCNS Hex bot, and measure the performance of a parallel implementation.

2. Conspiracy number search

In 2-player game search, CNS has shown promise in chess [35,34,19,24,23,27] and shogi [17]. CNS can be viewed as a generalization of PNS, which is how we will describe our implementation.

2.1. Proof number search

Definition 1. Each node n has a *proof number* (pn) p_n and *disproof number* (dn) d_n. A node's (dis)proof number is the smallest number of descendant leaves that, if all true (false), would make the node true (false).[4]

Fact 1. *If a node n is a leaf then*

$$\begin{aligned} &p_n = 1 \quad d_n = 1 \quad \text{if } n \text{ is non-terminal} \\ &p_n = 0 \quad d_n = +\infty \quad \text{if } n \text{ is true} \\ &p_n = +\infty \quad d_n = 0 \quad \text{if } n \text{ is false,} \end{aligned} \tag{1}$$

otherwise

$$\begin{aligned} &p_n = \min_{s\in\text{children}(n)} p_s, \quad d_n = \sum_{s\in\text{children}(n)} d_s \quad \text{if } n \text{ is or-node} \\ &p_n = \sum_{s\in\text{children}(n)} p_s, \quad d_n = \min_{s\in\text{children}(n)} d_s \quad \text{if } n \text{ is and-node.} \end{aligned} \tag{2}$$

Definition 2. *A* most proving node (mpn) is a leaf whose proof will reduce the root's proof number and whose disproof will reduce the root's disproof number.

PNS iteratively selects a most proving leaf and expands it. See Algorithms 1 and 2.

Algorithm 1 Proof number search.

```
1: function PNS(root)
2:     while not root solved do
3:         n ← SelectMPN(root)
4:         Expand n and initiate new children by (1)
5:         Update nodes along path to the root using (2)
```

[2] PNS is used in *and/or* trees (i.e. each leaf has minimax value ±1) and is guided by proof and disproof numbers (for each node, the smallest number of descendant leaves that need be 1, resp. -1, for the node to have value 1, resp. -1).

[3] Shannon built an analogue circuit to play the connection game Bridg-it, with moves scored by voltage drop [12]. Adding links between virtual connected cells [3] improves the heuristic, which although erratic between non-sibling states is reliable among siblings [15]. So we use this heuristic for our Hex SCNS bot.

[4] In PNS, a leaf node with value true indicates that the search goal is reached. Usually the search goal is to determine the game win/loss value, but it could be any desired search goal, e.g. in chess indicating the capture of a queen, or that the game ends in a win or draw but not a loss.