Proof-Set Search

Martin Müller
Department of Computing Science, University of Alberta, Canada
mmueller@cs.ualberta.ca

Abstract. Victor Allis’ proof-number search is a powerful best-first tree
search method which can solve games by repeatedly expanding a most-
proving node in the game tree. A well-known problem of proof-number
search is that it does not account for the effect of transpositions. If the
search builds a directed acyclic graph instead of a tree, the same node
can be counted more than once, leading to incorrect proof and disproof
numbers. While there are exact methods for computing proof numbers
in DAGs, they are too slow to be practical.

Proof-set search (PSS) is a new search method which uses a similar value
propagation scheme as proof-number search, but backs up proof and
disproof sets instead of numbers. While the sets computed by proof-set
search are not guaranteed to be of minimal size, they do provide provably
tighter bounds than is possible with proof numbers.

The generalization proof-set search with (P,D)-truncated node sets or
PSS_{P,D} provides a well-controlled tradeoff between memory require-
ments and solution quality. Both proof-number search and proof-set
search are shown to be special cases of PSS_{P,D}. Both PSS and PSS_{P,D}
can utilize heuristic initialization of leaf node costs, as has been proposed
in the case of proof-number search by Allis.

1 Proof Sets and Proof Numbers

Victor Allis’ proof-number search (PNS) [1] is a well-known game tree search
algorithm, which has been successfully applied to games such as connect-four,
qubic and gomoku. In contrast to many other methods, PNS does not compute
a minimax value based on heuristic position evaluations; rather its aim is to find
a proof or disproof of a partial boolean predicate P defined on a subset of game
positions. The usual predicate is CanWin(p), but predicates representing other
goals such as the tactical capture of some playing piece can also be used with
proof-number search.

Proof-number search is a best-first method for expanding a game tree. It
computes proof and disproof numbers in order to find a most-proving node,
which will be expanded next in the tree search. Search continues until the root
is either proven or disproven.

There is a simple bottom-up backup scheme for computing proof numbers,
which is correct for trees. However, many game-playing programs use a trans-
position table to detect identical positions reached by different move sequences.
Such a table changes the search graph from a tree to a directed acyclic graph
(DAG) or even a directed cyclic graph (DCG). If the same backup method for proof numbers is used on a DAG, it fails to compute the correct proof and disproof numbers, since the same node may be counted more that once along different paths. The new algorithm of proof-set search (PSS) is designed to reduce this problem and thereby improve the search performance on game graphs containing many transpositions.

The outline of the paper is as follows: the introduction continues with a short description of proof-number search on game trees and on directed acyclic graphs, and with an example that illustrates the problems of proof-number computation in DAGs. Section 2 describes the new method of proof-set search, and characterizes it by a theorem establishing its dominance over PNS on the same DAG. On the other hand, counterexamples show that even PSS cannot always select a smallest proof set. Section 3 describes the algorithmic aspects of PSS in those areas where it differs from PNS. Section 4 introduces the data structure of a $K$-truncated node set, defines the generalization of PSS to PSS with $(P,D)$-truncated node sets or $\text{PSS}_{P,D}$, characterizes both PNS and PSS as special cases of $\text{PSS}_{P,D}$, and proves a generalized dominance theorem of $\text{PSS}_{P,D}$ over PNS. Section 5 describes how to use a heuristic initialization of leaf node costs in PSS and $\text{PSS}_{P,D}$. Section 6 closes with a discussion of future work, including the extension of PSS to cyclic game graphs and potential applications of PSS.

1.1 Proof-Number Search in a Tree

This introductory section describes the basic procedure of proof-number search. For detailed explanations and algorithms, see [1]. Proof-number search (PNS) grows a game tree by incrementally expanding a most-proving node at the frontier. Nodes in a proof tree can have three possible states: proven, disproven, and unproven. Search continues as long as the status of the root is unproven. After each expansion, a leaf evaluation predicate $P$ is applied to each new node, to see whether it is defined in the corresponding game position. If yes, the new node can be evaluated as proven ($P = \text{true}$) or disproven ($P = \text{false}$), while if $P$ does not apply, the node status becomes unproven. Proofs and disproofs are propagated to interior nodes by using proof numbers. Interior nodes are proven by finding a proof tree or disproven by finding a disproof tree. A proof tree $s$ for a node $r$ is a subtree of the game tree with following properties:

1. $r$ is the root of $s$.
2. In all leaf nodes of $s$, the predicate $P$ is well-defined and evaluates to true.
3. If $n$ is an AND node in $s$, then all of its successor nodes in the game tree are also contained in $s$.

Analogous properties hold for a disproof tree $s$ of $r$:

1. $r$ is the root of $s$.
2. In all leaf nodes of $s$, the predicate $P$ is well-defined and evaluates to false.
3. If $n$ is an OR node in $s$, then all of its successor nodes in the game tree are also contained in $s$. 