Solving 4x5 Dots-And-Boxes

Joseph K. Barker and **Richard E Korf**
{*jbarker,korf*}*@cs.ucla.edu* Tel: (541) 805-2292
Department of Computer Science
University of California, Los Angeles
4732 Boelter Hall
Los Angeles, CA 90095
Full-Length Paper: http://cs.ucla.edu/~jbarker/AAAI˙2011.pdf

Abstract

Dots-And-Boxes is a well-known and widely-played combinatorial game. While the rules of play are very simple, the state space for even small games is extremely large, and finding the outcome under optimal play is correspondingly hard. In this paper we introduce a Dots-And-Boxes solver which is significantly faster than the current state-of-the-art: over an order-of-magnitude faster on several large problems. We describe our approach, which uses Alpha-Beta search and applies a number of techniques—both problem-specific and general—to reduce the number of duplicate states explored and reduce the search space to a manageable size. Using these techniques, we have determined for the first time that Dots-And-Boxes on a board of 4x5 boxes is a tie given optimal play. This is the largest game solved to date.

Introduction

Dots-And-Boxes is a combinatorial game popular among children and adults around the world. A rectangular grid of dots is drawn on a piece of paper and each player takes turns drawing lines between pairs of horizontally- or vertically-adjacent dots, forming boxes. The size of a game is defined in terms of boxes, so a 3x3 game has nine boxes. A player captures a box by completing its fourth line and initialing it, and must then draw another line. The player who has captured the most boxes when all edges have been filled wins. A player is *not* required to complete a box if they are able to do so. Figure 1 shows an example game in progress.

We introduce an algorithm to *solve* Dots-And-Boxes: it determines the outcome given optimal play—that is, if players always make an optimal response to their opponent's move. We solve the winner's margin of victory (how many more boxes they can capture than their opponent given perfect play), rather than a simple win/loss (or draw) value.

Techniques

The previous state-of-the-art solver (Wilson 2010) uses retrograde analysis (Ströhlein 1970; Thompson 1986) to find the value of every board configuration (the number of remaining boxes capturable through optimal play) by working backwards from the final game state, in which all edges are filled in. The value for a state with n edges is determined by looking at its possible successors (with $n+1$ edges) and picking the best. Once the initial state (in which no edges are filled) is evaluated, the overall value of the game is known.

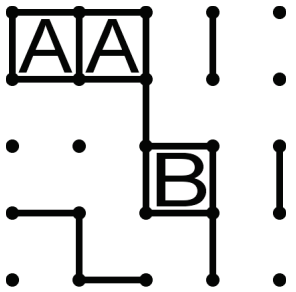

Figure 1: A 4x4 game of Dots-And-Boxes in progress. A has captured two boxes to B's one and has the lead.

As the solver is working backwards, it cannot know *a priori* if a given state is part of an optimal strategy and thus must evaluate every unique state in the problem space. An mxn game has $p = m * (n+1) + (m+1) * n$ edges and thus 2^p unique states (as any configuration of filled-in edges constitutes a legal state). This is better than the $p!$ states encountered in a naïve, depth-first exploration of the search space, but does not scale to larger problems.

As such, we use Alpha-Beta search (Knuth and Moore 1975) as our algorithm's basis. Alpha-Beta does a depth-first search, using local bounds to avoid provably-irrelevant parts of the search space. This allows it to avoid exploring all 2^p unique states; however, as a depth-first algorithm it cannot easily detect if a newly-generated state has been previously seen and may do redundant work to determine the duplicate state's value. A complete Alpha-Beta search of the problem space lets us determine the overall value of the game.

Transposition Tables

Transposition tables are a standard technique for reducing duplicate work in Alpha-Beta search; they cache explored states, associating with each its minimax value. If a newly-generated state has been previously explored, its stored value is looked up and returned, avoiding the duplicate work necessary to determine its value a subsequent time.

There are interesting implications of implementing trans-

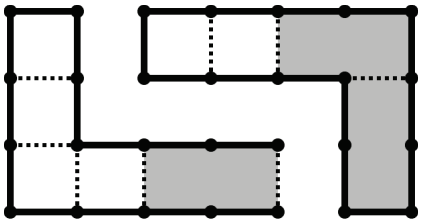

Figure 2: A half-open chain (left) and a closed chain (right) of length six. The dotted lines show the moves that leave hard-hearted handouts (shown as gray boxes) in both cases.

position tables in Dots-And-Boxes worth mentioning. For example, stored entries match the state being explored even if they do not share the same player to move or current score. An individual transposition-table entry in Dots-And-Boxes thus matches many more states than in other domains.

In addition, a table entry may store a bound on the minimax value, not the exact value itself. Most transposition tables store a single value with each entry along with a flag indicating if the value is exact or an upper or lower bound. In Dots-And-Boxes, we find it effective to instead store both an upper and lower bound with each entry; this is not worth the extra space in most domains (Schaeffer 2011).

Symmetries

Symmetric states in Dots-And-Boxes have identical minimax values with symmetric optimal strategies. All boards have horizontal and vertical symmetry, reducing the state space by a factor of four; diagonal symmetries reduce the state space of square boards by an additional factor of two. All games have an additional, less obvious symmetry: the two edges that form a corner are equivalent. Filling in either edge of such a pair results in an equivalent state; thus, we need consider only one such move on states where both corner edges are unfilled. A full proof can be found in (Berlekamp 2000).

Chains

A sequence of one or more capturable boxes is called a *chain*. If only one end of a chain is initially capturable (i.e., is a box with three edges filled in), it is *half-open*. If both ends are initially capturable, it is a *closed* chain. Most available moves on a state with chains can be provably discarded as non-optimal, significantly reducing the search space.

In a game state with a half-open chain, there are only two sequences of moves that can possibly be part of an optimal strategy: capture every available box (and then make an additional move), or capture all but two boxes and then fill in the end of the chain—leaving two capturable boxes for the opponent. The remaining configuration of two boxes capturable with a single line is called a *hard-hearted handout*.

The possibly-optimal moves in states with a closed chain are similar: capture every available box (and then make an additional move), or capture all but *four* boxes and fill in the edge that separates them into two hard-hearted handouts. Figure 2 shows an example of chains and how they can be filled in to leave hard-hearted handouts.

These rules significantly reduce the number of moves to consider in states with capturable boxes and consequently reduce the overall size of the search space. They are widely known, but are most thoroughly treated (with a proof sketch of their validity) in (Berlekamp 2000). Chain analysis cannot, however, be used in a retrograde-analysis approach such as that of Wilson's solver; this is a significant advantage of our Alpha-Beta approach.

Move Ordering

A *move-ordering heuristic* decides the order in which children of a state are explored, and has a strong effect on the performance of Alpha-Beta search. In general, a good heuristic explores the most valuable children of a state first.

All possible capturing moves occur in chains and are dealt with by the rules of the previous section; thus, our heuristic only considers non-capturing moves. Of those, it considers moves that fill in the third edge of a box last, as they leave a capturable box for the opponent. The remaining moves are explored by considering edges in an order radiating outwards from the center of the board. We have found empirically that considering center moves first is significantly more effective than a simple left-right, top-down move ordering.

Results

Combining these techniques results in a solver over an order-of-magnitude faster than the previous state-of-the-art on a number of hard benchmark problems, including the 4x4 problem (the largest problem previously solved). We find that doing chain analysis is the most consistently useful improvement: removing it increases the runtime of our solver by over an order-of-magnitude. Our transposition-table and symmetry techniques provide a smaller but consistent speedup, each reducing runtime by a factor of two to four. Our move-ordering heuristic is helpful but inconsistent, providing up to a factor of 17 speedup on empty boards but only small speedups on games with existing edges.

In addition to these experiments, we have solved the 4x5 game for the first time: it is a tie given optimal play. Our solver took nine days to complete on a 3.33 GHz Intel Xeon CPU with a 24GB transposition table. We conservatively estimate that Wilson's solver would require 130 days to solve the 4x5 game (as well as 8 terabytes of disk space).

Acknowledgments

This work was supported by NSF Grant IIS-0713178.

References

Berlekamp, E. 2000. *The Dots-and-Boxes Game*. A K Peters.

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

Schaeffer, J. 2011. Personal communication.

Ströhlein, T. 1970. *Untersuchungen ber kombinatorische Spiele*. Ph.D. Dissertation, Technischen Hochschule München.

Thompson, K. 1986. Retrograde analysis of certain endgames. *ICCA Journal*.

Wilson, D. 2010. Dots-and-boxes analysis index. http://homepages.cae.wisc.edu/~dwilson/boxes/.