Automatic Generation of Starting Positions in Board Games

Umair Ahmed
IIT Kanpur
umair.z.ahmed@gmail.com

Krishnendu Chatterjee
IST Austria
Krishnendu.Chatterjee@ist.ac.at

Sumit Gulwani
MSR Redmond
sumitg@microsoft.com

Abstract

Board games, like Tic-Tac-Toe and CONNECT-4, play an important role not only in development of mathematical and logical skills, but also in emotional and social development. In this paper, we motivate and address the problem of generating interesting start states for such games. Our algorithm generates starting states of varying difficulty level for player 1, given the rules of a board game, the number of steps required for player 1 to win, and the expertise levels of the two players. Our algorithm leverages symbolic methods and iterative simulation to efficiently search the humongous state space. We present experimental results that discover for the first time such interesting states for several games and demonstrate the feasibility of finding them in an offline mode. The presence of such states for a game like Tic-Tac-Toe 4×4 that was previously thought to be trivial, opens up new games to be played that have been believed to be useless for ages.

Introduction

A board game, such as Tic-Tac-Toe or CONNECT-4, involves placing pieces on a pre-marked surface or board according to a set of rules. Studies show that such board games can significantly improve a child's mathematical ability (Ramani and Siegler 2008). Giving children an early maths boost is significant because studies also show that differences in mathematical ability between children in the first year at school persist into secondary education (Duncan et al. 2007).

Board games play a vital role in the emotional and social development of a child. They instill a competitive urge and desire to master new skills in order to win, to be better than others. Winning gives a boost to their self confidence and enjoyment. Playing a game within a set of rules help them to adhere to discipline in life. They learn social etiquette; taking turns, and being patient. Strategy is another huge component of board games. Children should quickly grasp that decisions they make in the beginning of the game have consequences later on. Cause and effect is elegantly displayed in several board games.

Playing board games helps elderly people stay mentally sharp and less likely to develop Alzheimer (Gottlieb 2003). They also hold a great importance in today's digital society by strengthening family ties. It bridges the gap between young and old. It bolsters the self-esteem of children who take great pride and pleasure when an elder spends playing time with them.

Board games are typically played with a default start state. For example, in case of Tic-Tac-Toe and CONNECT-4, it is the empty board. However, there are several drawbacks associated with starting from a fixed starting state.

- The starting state may be heavily biased towards one player and hence that game has never been popular. For example, Tic-Tac-Toe (3,4,4), where the goal is to make a straight line of 3 pieces, but on a 4×4 board. In this game, the person who plays first invariably almost always wins.
- Even if the default starting state is not heavily biased, it might have a well-known conclusion. For example, both players can enforce a draw in Tic-Tac-Toe while the first player can enforce a win in CONNECT-4 (Allis 1988).
- The starting state for commonly played games yields an enjoyable game between opponents of similar expertise. The flexibility to start from some other starting state that is more biased towards the weaker player can allow for an enjoyable game among players of different expertise.
- Players can memorize certain moves and strategies from the starting state, and hence the game may lose charm quickly. Hence, it is not surprising that grown-ups find playing the standard Tic-Tac-Toe game to be quite boring.
- Certain challenges manifest only in states that are typically not easily reachable from the start state, or might require too many steps. The flexibility to start from such states might have greater educational value.
- People sometimes might be disinterested in playing a game if it takes too much time to finish. However, selecting non-default starting positions allow the potential of a shorter game play.

We address the problem of automatically generating interesting starting states for a given board game. Our algorithm takes as input the rules of a board game and the number of steps required for player 1 to win. It then generates starting states that are labelled as easy, medium, or hard for player 1 for various expertise level combinations of the two players. The difficulty for a player is defined with respect to the fraction of times player 1 will win, playing a strategy of depth k

Copyright © 2013, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

against an opponent who plays a strategy of depth k'.

Generating such states automatically from game rules also has the advantage of experimenting with new games or their variants. While people might be hesitant to learn a game with different new rules, it is quite convenient to change the rules slightly. For example, instead of allowing for straight-line matches in each of row, column, or diagonal (RCD) in Tic-Tac-Toe or CONNECT-4, one may restrict the matches to say only row or diagonal (RD).

Our algorithm is a novel combination of symbolic methods and iterative simulation to efficiently identify desired states. Symbolic methods are used to compute the winning set for player 1 and they work particularly well for navigating a state space where the transition relation forms a sparse DAG (as is the case for those board games in which a piece once placed on the board does not move, as in Tic-Tac-Toe and CONNECT-4). Simulation is used to identify the difficulty of a given winning state. Instead of randomly sampling the winning set to identify a state of a certain difficulty level, we identify states of varying difficulty level in order of increasing values of k and k'. The key observation is that hard states are much smaller in number than easy states, and given a value of k' interesting states for higher values of k are a subset of the hard states for smaller values of k.

This paper makes the following contributions.

- We motivate and formalize the problem of generating initial states for board games.
- We present an algorithm for generating starting states of varying difficulty level parameterized by the expertise levels of players, given a graph game description and the number of steps required for winning. Our algorithm uses a novel combination of symbolic methods and iterative simulation to efficiently search a huge state space.
- We present experimental results that illustrate the effectiveness of our search algorithm. During this process, we produced a large database of initial states of varying hardness level for standard games as well as some interesting variations of those games (thereby discovering some interesting variations of the standard games).

Graph Games and Problem Description

We present the mathematical model of graph games, recall some basic results on graph games, and then describe the notion of hardness and finally the description of the problem we consider for graph games. We start with the background.

Graph games. An alternating graph game (for short, graph game) $G = ((V, E), (V_1, V_2))$ consists of a finite graph G with vertex set V, a partition of the vertex set into player-1 vertices V_1 and player-2 vertices V_2, and edge set $E \subseteq ((V_1 \times V_2) \cup (V_2 \times V_1))$. The game is alternating in the sense that the edges of player-1 vertices go to player-2 vertices and vice-versa. The game is played as follows: the game starts at a starting vertex v_0; if the current vertex is a player-1 vertex, then player 1 chooses an outgoing edge to move to a new vertex; if the current vertex is a player-2 vertex, then player 2 does likewise. The winning condition is given by a target set $T_1 \subseteq V$ for player 1; and similarly a target set $T_2 \subseteq V$ for player 2. If the target set T_1 is reached, then player 1 wins; if T_2 is reached, then player 2 wins; else we have a draw.

Examples. The class of graph games provide the mathematical framework to study many interesting games, such as Chess or Tic-Tac-Toe. For example, in Tic-Tac-Toe the vertices of the graph represent the board configurations and whether it is player 1 ($\times$) or player 2 ($\circ$) to play next. The set T_1 (resp. T_2) is the set of board configurations where there is three consecutive $\times$ (resp. $\circ$) in a row, column, or diagonal.

Classical game theory result. A classic result in the theory of graph games shows that for every graph game, from every starting vertex one of the following three conditions hold: (1) player 1 can enforce a win no matter how player 2 plays (i.e., there is a way for player 1 to play to ensure winning against all possible strategies of the opponent); (2) player 2 can enforce a win no matter how player 1 plays; or (3) both players can enforce a draw. In the mathematical study of game theory, the theoretical question is: given a designated starting vertex v_0 determine whether case (1), case (2), or case (3) holds. In other words, the mathematical game theoretic question concerns the best possible way for a player to play to ensure the best possible outcome. The set W_j is defined as the set of vertices such that player 1 can ensure to win within j-moves; and the winning set W^1 of vertices of player 1 is the set $\bigcup_{j \geq 0} W_j$ where player 1 can win in any number of moves. Analogously, we define W^2; and then the classical game theory question is formally stated as follows: given a designated starting vertex v_0 decide whether v_0 belongs to W^1 (player-1 winning set) or to W^2 (player-2 winning set) or to $V \setminus (W^1 \cup W^2)$ (both players draw set).

Our problem. We now present our problem description.

Notion of hardness. The mathematical game theoretic question ignores two aspects. (1) The notion of hardness: It is always concerned with optimal strategies irrespective of its hardness, and it is not concerned with when can sub-optimal strategies perform well, and when do sub-optimal strategies fail; and (2) the problem of generating different starting vertices. In this work we are interested in generating starting vertices of different hardness level. The notion of hardness we consider is the depth of the tree a player can explore which is standard in artificial intelligence. We first explain the notion of tree exploration.

Tree exploration in graph games. Consider a vertex u_0 that belongs to player 1. The search tree of depth 1 is as follows: we consider a tree rooted at u_0 such that children of u_0 are the vertices u_1 of player 2 such that $(u_0, u_1) \in E$ (there is an edge from u_0 to u_1); and for every vertex u_1 (that is a children of u_0) the children of u_1 are the vertices u_2 such that $(u_1, u_2) \in E$, and they are the leaves of the tree. This gives us the search tree of depth 1, which intuitively corresponds to exploring one round of the play. The search tree of depth $k + 1$ is defined inductively from the search tree of depth k, where we first consider the search tree of depth 1 and replace every leaf by a search tree of depth k. The depth of the search tree denotes the depth of reasoning (analysis depth) of a player and corresponds to the optimality (or competence or maturity) of the player to play the game. The search tree for player 2 is defined analogously.

Strategy from tree exploration. A depth-k strategy of a player that does a tree exploration of depth k is obtained by

the classical min-max reasoning (or backward induction) on the search tree. First, for every vertex v of the game we associate a number (or reward) $r(v)$ that denotes how favorable is the vertex for a player to win. Given the current vertex u, a depth-k strategy is as defined as follows: first construct the search tree of depth k and evaluate the tree bottom-up with min-max reasoning, i.e., a leaf vertex v is assigned reward $r(v)$, and for a vertex in the tree if it is a player-1 (resp. player-2) vertex we consider its reward as the maximum (resp. minimum) of its children, and finally, for vertex u (the root) the strategy chooses uniformly at random among its children with the highest reward.

Example description of tree exploration. Consider the example of the Tic-Tac-Toe game. We first describe how to assign reward r to board positions. Recall that in the game of Tic-Tac-Toe the goal is to form a line of three consecutive positions in a row, column, or diagonal. Given a board position, (i) if it is winning for player 1, then it is assigned reward $+\infty$; (ii) else if it is winning for player 2, then it is assigned reward $-\infty$; (iii) otherwise it is assigned the score as follows: let n_1 be the number of two consecutive positions of marks for player 1 and n_2 be the number of two consecutive positions of marks of player 2, then the reward is the difference $n_1 - n_2$. If we consider the depth-1 strategy, then the strategy chooses all board positions uniformly at random; a depth-2 strategy chooses the center and considers all other positions to be equal; a depth-3 strategy chooses the center and also recognizes that the next best choice is one of the four corners. This example illustrates that as the depth increases, the strategies become more intelligent for the game.

Outcomes and probabilities given strategies. Given a starting vertex v, a depth-k strategy σ_1 for player 1, and depth-k' strategy σ_2 for player 2, let O be the possible outcomes, i.e., the set of possible plays given σ_1 and σ_2 from v. The strategies and the starting vertex define a probability distribution over the outcomes which we denote as $\Pr_v^{\sigma_1,\sigma_2}$, i.e., for a play ρ in the outcome O we have $\Pr_v^{\sigma_1,\sigma_2}(\rho)$ is the probability of ρ given the strategies. Note that strategies are randomized (because strategies choose distributions over the children with highest value after the search tree exploration), and hence defines a probability distribution over outcomes.

Problem description. In this work we will consider several board games (such as Tic-Tac-Toe, CONNECT-4, and variants), and the goal is to obtain starting positions that are of different hardness level, where our hardness is characterized by strategies of different depths. In other words, our goal is to obtain starting positions that are of different hardness levels: i.e., hard for depth-1 strategies, but easy for depth-2 strategies; and hard for depth-2 strategies, but easy for depth-3 strategies, and so on. More precisely, consider a depth-k strategy for player 1, and depth-k' strategy for player 2, and a starting vertex $v \in W_j$ that is winning for player 1 within j-moves. We classify the starting vertex into three types as follows: if player 1 wins (i) at least $\frac{2}{3}$ times, then we call it easy (E); (ii) at most $\frac{1}{3}$ times, then we call it hard (H); (iii) otherwise medium (M).

Definition 1 ((j,k,k')-Hardness). *Consider a vertex $v \in W_j$ that is winning for player 1 within j-moves. Let σ_1 and σ_2 be a depth-k strategy for player 1 and depth-k' strategy for player 2, respectively. Let $O_1 \subseteq O$ be the set of plays that belong to the outcome and is winning for player 1. Let $\Pr_v^{\sigma_1,\sigma_2}(O_1) = \sum_{\rho \in O_1} \Pr_v^{\sigma_1,\sigma_2}(\rho)$ be the probability of the winning plays. The (k,k')-classification of v is as follows: (i) if $\Pr_v^{\sigma_1,\sigma_2}(O_1) \geq \frac{2}{3}$, then v is easy (E); (ii) if $\Pr_v^{\sigma_1,\sigma_2}(O_1) \leq \frac{1}{3}$, then v is hard (H); (iii) otherwise it is medium (M).*

Our goal is to consider various classes of games and identify states of different categories (e.g., hard for depth-k against depth-k', but easy for depth-$(k+1)$ against depth-k', for small values of k and k').

Methodology

We describe the methodology we use for the problem of generating starting positions of different hardness levels.

Overall methodology

We start with a description of the overall methodology.

Generation of j-steps win set. Given a game graph $G = ((V,E),(V_1,V_2))$ along with target sets T_1 and T_2 for player 1 and player 2, respectively, our first goal is to compute the set of vertices W_j such that player 1 can win within j-moves. For this we define two kinds of predecessor operators. Given a set of vertices X, let $\mathsf{EPre}(X)$ (called existential predecessor) denote the set of player 1 vertices that has an edge to X; i.e., $\mathsf{EPre}(X) = \{u \in V_1 \mid \text{ there exists } v \in X \text{ such that } (u,v) \in E\}$ (i.e., player 1 can ensure to reach X from $\mathsf{EPre}(X)$ in one step); and $\mathsf{APre}(X)$ (called universal predecessor) denote the set of player 2 vertices that has all its outgoing edges to X; i.e., $\mathsf{APre}(X) = \{u \in V_2 \mid \text{ for all } (u,v) \in E \text{ we have } v \in X\}$ (i.e., irrespective of the choice of player 2 the set X is reached from $\mathsf{APre}(X)$ in one step). The computation of the set W_j is defined inductively as follows: $W_0 = \mathsf{EPre}(T_1)$ (i.e., player 1 wins with the next move to reach T_1); and $W_{i+1} = \mathsf{EPre}(\mathsf{APre}(W_i))$. In other words, from W_i player 1 can win within i-moves, and from $\mathsf{APre}(W_i)$ irrespective of the choice of player 2 the next vertex is in W_i; and hence $\mathsf{EPre}(\mathsf{APre}(W_i))$ is the set of vertices such that player 1 can win within $(i+1)$-moves.

Exploring vertices from W_j. The second step is to explore vertices from W_j, for increasing values of j starting with small values of j, for strategies of different depth for player 1 against strategies of different depth for player 2, and then obtain starting vertices of different hardness levels. That is, we consider a vertex v from W_j, consider a depth-k strategy for player 1 and a depth-k' strategy for player 2, and play the game multiple times with starting vertex v to find out the hardness level with respect to (k,k')-strategies, i.e., the (k,k')-classification of the vertex $v \in W_j$. Note that from W_j player 1 can win within j-moves. Thus the approach has the benefit that player 1 has a winning strategy with a small number of moves and the game need not be played for long.

Two key issues. There are two main computational issues associated with the above approach in practice. The first issue is related to the size of the state space (number of vertices) of the game which makes enumerative approach to analyze the game graph explicitly computationally infeasible.

For example, the size of the state space of a Tic-Tac-Toe 3×3 game is 8,532; Tic-Tac-Toe 4×4 game is 7,848,848; and a CONNECT-4 5×5 game is 114,130,912 (more than 100 million). Hence any enumerative method would not work for such large game graphs. The second issue is related to exploring the vertices from W_j. If W_j has a lot of witness vertices, then playing the game multiple times from all of them will be computationally expensive. In other words, we need an initial metric to guide the search of the vertices from W_j such that the computation of the metric is not computationally expensive. We solve the first issue with symbolic approach, and the second one by iterative simulation.

Symbolic methods

In this section we discuss the symbolic methods that allow to analyze games with large state spaces. The key idea is to represent the games symbolically (not with explicit state space) using variables, and operate on the symbolic representation. The key object used in symbolic representation are called BDDs (boolean decision diagrams) (**?**) that can efficiently represent a set of states using a dag representation of a boolean formula representing the set of states. The tool CUDD supports many symbolic representation of state space using BDDs and supports many operations on symbolic representation on graphs using BDDs (**?**).

Symbolic representation of vertices. In symbolic methods, a game graph is represented by a set of variables $x_1, x_2, \ldots, x_n$ such that each of them takes values from a finite set (e.g., $\times, \circ$, and blank symbol); and each vertex of the game represents a valuation assigned to the variables. For example, the symbolic representation of the game of Tic-Tac-Toe of board size 3×3 consists of ten variables $x_{1,1}, x_{1,2}, x_{1,3}, x_{2,1} \ldots, x_{3,3}, x_{10}$, where the first nine variables $x_{i,\ell}$ denote the symbols in the board position (i, ℓ) and the symbol is either $\times$, $\circ$, or blank; and the last variable x_{10} denote whether it is player 1 or player 2's turn to play.

Symbolic encoding of transition function. The transition function (or the edges) are also encoded in a symbolic fashion: instead of specifying every edge, the symbolic encoding allows to write a simple program over the variables to specify the transitions. The tool CUDD takes such a symbolic description written as a program over the variables and constructs a BDD representation of the transition function. For example, for the game of Tic-Tac-Toe, a simple program to describe the symbolic transition is as follows: the program maintains a set U of positions of the board that are already marked; and at every point receives an input (i, ℓ) from the set $\{(a, b) \mid 1 \leq a, b \leq 3\} \setminus U$ of remaining board positions from the player of the current turn; then adds (i, ℓ) to the set U and sets the variable $x_{i,\ell}$ as $\times$ or $\circ$ (depending on whether it was player 1 or player 2). This represents the symbolic description of the transition function. The CUDD tool accepts such symbolic description and outputs a BDD representation of the game.

Symbolic encoding of target states. The set of target states is encoded as a boolean formula that represents a set of states. For example, in Tic-Tac-Toe the set of target vertices for player 1 is given by the following boolean formula:

$$\exists i, \ell.\ 1 \leq i, \ell \leq 3.\ (x_{i,\ell} = \times \wedge x_{i+1,\ell} = \times \wedge x_{i+2,\ell} = \times)$$
$$\vee\ (x_{i,\ell} = \times \wedge x_{i,\ell+1} = \times \wedge x_{i,\ell+2} = \times)$$
$$\vee\ (x_{2,2} = \times \wedge ((x_{1,1} = \times \wedge x_{3,3} = \times) \vee (x_{3,1} = \times \wedge x_{1,3} = \times)))$$
$$\wedge \text{ Negation of above with } \circ \text{ to specify player 2 not winning}$$

The above formula states that either there is some column ($x_{i,\ell}, x_{i+1,\ell}$ and $x_{i+2,\ell}$) that is winning for player 1; or a row ($x_{i,\ell}, x_{i,\ell+1}$ and $x_{i,\ell+2}$) that is winning for player 1; or there is a diagonal ($x_{1,1}, x_{2,2}$ and $x_{3,3}$; or $x_{3,1}, x_{2,2}$ and $x_{1,3}$) that is winning for player 1; and player 2 has not won already. To be precise, we also need to consider the BDD that represents all valid board configurations (reachable vertices from the empty board) and intersect the BDD of the above formula with valid board configurations to obtain the target set T_1.

Symbolic computation of W_j. The symbolic computation of W_j is as follows: given the boolean formula for the target set T_1 we obtain the BDD for T_1; and the CUDD tool supports both EPre and APre with a set of basic operations using symbolic functions; i.e., the tool takes as input a BDD representing a set X and supports the operation to return the BDD for $\mathsf{EPre}(X)$ and $\mathsf{APre}(X)$. Thus we obtain the symbolic computation of the set W_j.

Iterative simulation

We now describe a computationally inexpensive (but approximate) way to aid sampling of vertices as candidates for starting positions of a given hardness level. Given a starting vertex v, a depth-k strategy for player 1, and a depth-k' strategy for player 2, we need to consider the tree exploration of depth $\max\{k, k'\}$ to obtain the hardness of v. Hence if the strategy of the opponent is of high depth, then it is computationally expensive. Thus we need a preliminary metric that can be computed relatively easily for small values of k as a guide for vertices to be explored in depth. We use a very simple metric for this purpose. For the vertices in W_j, we fix a depth-k strategy, and fix a small depth strategy for the opponent and assign the vertex a number based on the performance of a depth-k strategy and the small depth strategy of the opponent. The vertices that have the lowest performance according to this metric are then iteratively simulated against depth-k' strategies of the opponent to obtain vertices of different hardness level.

Framework for Board Games

We now consider the specific problem of board games. We describe a framework to specify several variants of board games such as Tic-Tac-Toe, CONNECT-4, and several new games. Note that though our implementation of symbolic methods works for this class of board games, our methodology is applicable to the general class of graph games.

Parameter to generate different games. In our framework we consider three different types of parameters to generate different variants of board games.

1. The first parameter is the board size; e.g., the board size could be 3×3; or 4×4; or 4×5 and so on.
2. The second parameter is the way to specify the winning condition; and we consider the cases where a player wins

if a sequence of the moves of the player are in a line but the line could be in a row (R), in a column (C), or along the diagonal (D). The user can specify any combination, i.e., the winning condition could be (i) RCD (denoting the player wins if the moves are in a line along a row, column or diagonal); (ii) RC (line must be along a row or column, but diagonal lines are not winning); (iii) RD (row or diagonal, but not column); or (iv) CD (column or diagonal, but not row).

3. The third parameter is related to the allowed moves of the player. At any point the players can choose a column (if it is available, i.e., there is at least one empty position in the column) but can be restricted according to the following parameters: (i) Full gravity (once a player chooses a column, the move is chosen as the lowest available position in that column); (ii) partial gravity-ℓ (once a player chooses the column the move can be chosen as the one of the bottom-ℓ available positions in the column); or (iii) no gravity (the player can choose any of the available positions in the column).

Observe that Tic-Tac-Toe is given as board size (i) board size 3×3; (ii) winning condition RCD; and (iii) no-gravity; whereas in CONNECT-4 the winning condition is still RCD but moves are with full gravity. But in our framework there are many new variants of the previous classical games, e.g., Tic-Tac-Toe in a board of size 4×4 but diagonal lines are not winning (RC). Tic-Tac-Toe, Bottom-2 and CONNECT-3 require 3 consecutive positions to be marked for a player to win, while CONNECT-4 requires 4 consecutive positions.

Features of our implementation. We have implemented our symbolic approach to solve the problem of generation of starting vertices (or board positions) of different hardness levels (if they exist) for the class of board games described above. The main features that our implementation supports are as follows: (1) Generation of starting vertices of different hardness level if they exist. (2) Playing against opponents of different levels. We have implemented the depth-k' strategy of the opponent for $k' = 1, 2$ and 3 (typically in all the above games depth-3 strategies are quite intelligent). Thus, a learner (beginner) can consider starting with board positions of various hardness levels and play with opponents of different skill level and thus hone her ability to play the game and be exposed to new combinatorial challenges of the game.

Experimental Results

We now present our experimental results, which in our opinion are the most interesting aspect and finding of our paper.

CONNECT games. In Table 1 we present the experimental results for CONNECT-3 and CONNECT-4 games. The first column represents the type of the game and the board size (CONNECT-3 or CONNECT-4, in either 4×4 or 5×5) The second column denotes the size of the state space of the game. We explore vertices from W_2 and W_3 only as the set W_4 is almost always empty (i.e., if there is a winning starting position it belongs to either W_1, W_2 and W_3). The third column $j = 2, 3$ denotes whether we explore from W_2 or W_3. The fourth column denotes the winning condition (RCD, CD, RC, CD, respectively). The fifth column denotes the number of states in W_j. The sixth column denotes sampling to select starting vertices if the number of states in W_j is large. In this column "All" denotes that we explore all states in W_j, and if there is a large number of states, then we first sample 5000 states randomly and then use our metric on the 5000 states and select bottom 100 (B100), i.e., hundred states with the least score according to our iterative simulation metric. This describes the sampling of states from W_j. The next three columns describes the analysis of the chosen states with respect to depth-k strategies for player 1, and depth-3 strategy of the opponent, for $k = 1, 2$, and 3. Given a board position, we run the game between the depth-k vs the depth-3 strategy for 30 times. If player 1 (i) wins more than $\frac{2}{3}$ times (20 times), then the position is identified as easy (E); (ii) wins less than $\frac{1}{3}$ times (10 times), then it is identified as hard (H); (iii) else as medium (M). The next three columns presents the result (classification of the states as E, M, and H) and the * denotes the number of remaining states.

Table 1: CONNECT-3 & -4 against depth-3 strategy of opponent.

Game	State Space	j	Win Cond	No. States	Sampling	k = 1			k = 2			k = 3		
						E	M	H	E	M	H	E	M	H
CONNECT-3	7.2×10^4	2	RCD	110	All	*	18	4	*	2	0	*	0	0
4x4	1.0×10^5		RC	292	All	*	44	4	*	17	3	*	0	0
	1.2×10^5		RD	444	All	*	34	16	*	15	3	*	0	0
	1.1×10^5		CD	323	All	*	39	23	*	30	15	*	0	0
CONNECT-3		3	RCD, RC	0	-									
4x4			RD	18	All	*	0	0	*	0	0	*	0	0
			CD	2	All	*	0	0	*	0	0	*	0	0
CONNECT-4	1.1×10^8	2	RCD	>5000	Rand + B100	*	4	8	*	1	4	*	0	0
5x5	1.3×10^8		RC	>5000	Rand + B100	*	7	24	*	1	24	*	0	0
	1.5×10^8		RD	>5000	Rand + B100	*	8	8	*	4	2	*	0	0
	1.5×10^8		CD	>5000	Rand + B100	*	5	11	*	2	8	*	0	0
CONNECT-4		3	RCD	>5000	Rand + B100	*	11	28	*	8	21	*	7	12
5x5			RC	>5000	Rand + B100	*	8	79	*	10	23	*	5	20
			RD	>5000	Rand + B100	*	23	42	*	2	14	*	6	8
			CD	>5000	Rand + B100	*	1	15	*	7	7	*	3	6

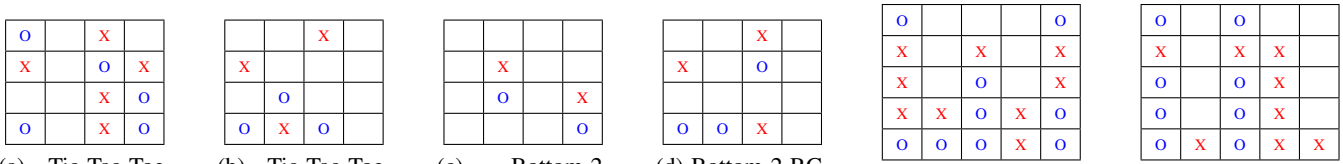
Table 2: Bottom-2 against depth-3 strategy of opponent.

Board Size	State Space	j	Win Cond	No. States	Sampling	k = 1			k = 2			k = 3		
						E	M	H	E	M	H	E	M	H
3x3	6.5×10^3	2	RCD	22	All	*	7	1	*	1	0	*	0	0
	6.6×10^3		RC	0	-									
	6.7×10^3		RD	11	All	*	2	4	*	3	2	*	0	0
	6.7×10^3		CD	1	All	*	0	0	*	0	0	*	0	0
3x3		3	Any	0	-									
4x4	3.1×10^6	2	RCD	209	All	*	17	24	*	0	2	*	0	0
	4.1×10^6		RC	2735	B100	*	18	10	*	1	3	*	0	0
	3.9×10^6		RD	2175	B100	*	7	3	*	0	0	*	0	0
	4.1×10^6		CD	1507	B100	*	10	14	*	1	0	*	0	0
4x4		3	RCD	0	-									
			RC	90	All	*	40	24	*	0	0	*	0	0
			RD	26	All	*	0	2	*	0	0	*	0	0
			CD	20	All	*	7	3	*	1	0	*	0	0

Experimental results for Bottom-2 and Tic-Tac-Toe. The results for Bottom-2 (partial gravity-2) and Tic-Tac-Toe

Table 3: Tic-Tac-Toe against depth-3 strategy of opponent.

Board Size	State Space	j	Win Cond	No. States	Sampling	k = 1			k = 2			k = 3		
						E	M	H	E	M	H	E	M	H
3x3	8.5×10^3	2	RCD	36	All	*	16	2	*	0	0	*	0	0
	8.6×10^3		RC	0	-									
	8.7×10^3		RD	1	All	*	0	0	*	0	0	*	0	0
	8.7×10^3		CD	1	All	*	0	0	*	0	0	*	0	0
3x3		3	Any	0	-									
4x4	7.8×10^6	2	RCD	128	All	*	8	0	*	0	0	*	0	0
	1.1×10^7		RC	3272	B100	*	46	19	*	0	0	*	0	0
	1.1×10^7		RD	4627	B100	*	3	2	*	0	0	*	0	0
	1.1×10^7		CD	4627	B100	*	3	2	*	0	0	*	0	0
4x4		3	RCD, RC	0	-									
			RD	4	All	*	0	0	*	0	0	*	0	0
			CD	4	All	*	0	0	*	0	0	*	0	0



(a) Tic-Tac-Toe RC Hard for k=1 (b) Tic-Tac-Toe CD Hard for k=1 (c) Bottom-2 RCD hard for k=2 (d) Bottom-2 RC Hard for k=2 (e) CONNECT-4 RCD Hard for k=3 (f) CONNECT-4 RD Hard for k=3

Figure 1: Example starting board positions.

games are shown in Table 2 and Table 3, respectively. The meaning of the entries are as described above for Table 1. In many cases, the number of states in the winning set W_i was less than 5000 but more than 1000, and in all these cases we use the bottom 100 sampling using our metric.

Example board positions. In Figure 1(a)-Figure 1(f) we present examples of several board positions that are of different hardness level for strategies of certain depth. In all the figures, player-X is the current player to play. All these board positions were discovered in our experimental results.

Important aspects. We now highlight two important aspects of our experimental results. Our first key finding is the existence of states of different hardness levels in various games. Let us consider the case where the depth of the opponent strategy is $k' = 3$. We observe that in Tic-Tac-Toe games only board positions that are hard for $k = 1$ exist, but interestingly they also exist in board of size 4×4. With the slight variation of allowable moves (Bottom-2), we obtain board positions that are hard for $k = 2$. In Connect-4 we obtain states that are hard for $k = 3$ even with small board size of 5×5. The second key aspect of our results is the fact that the number of interesting states is a negligible fraction of the huge state space. For example, in Bottom-2 RCD games with board size 4×4 the size of the state space is over three million, but has only two positions that are hard for $k = 2$. Since the interesting positions are extremely rare, a naive approach of randomly generating positions and measuring its hardness will be searching for a needle in a haystack and be completely ineffective to generate interesting positions. Thus there is need for automation which our tool provides.

Related Works and Conclusion

Tic-Tac-Toe and Connect-4. Tic-Tac-Toe has been generalized to different board size, match length (Ma), and even polyomino matches (Harary 1977) to find variants that are interesting from the default start state. Existing research has focussed on establishing which of these games have a winning strategy (Gardner 1979; 1983; Weisstein). In contrast, we show that even simpler variants can be interesting if we start from certain specific states. Existing research on Connect-4 also focussed on establishing that there is a winning strategy from the default starting state for the first player (Allis 1988). In contrast, we study how easy or difficult is to win from winning states given the expertise levels.

Level generation. The problem of level generation has been studied for specific games. Goldspinner (Williams-King et al. 2012) is an automatic level generation system for KGoldrunner, which is a puzzle-oriented platform game with dynamic elements. It uses a genetic algorithm to generate candidate levels and simulation to evaluate dynamic aspects of the game. We also use simulation to evaluate the dynamic aspect, but use symbolic methods to generate candidate states; also, our system is parameterized by game rules.

Most other work has been restricted to games without opponent and dynamic content such as Sudoku (Hunt, Pong, and Tucker 2007; XUE et al. 2009). Smith et al. used answer-set programming to generate levels for the educational puzzle game Refraction that adhered to prespecified constraints written in first-order logic (Smith et al. 2012). Similar approaches have also been used to generate levels for platform games (Smith et al. 2009). In all these approaches, designers must explicitly specify constraints that the generated content must reflect, for example, the tree needs to be near the rock and the river needs to be near the tree. In contrast, our system takes as input rules of the game and does not require any further help from the designer. (Andersen, Gulwani, and Popovic 2013) also uses a similar model and applies symbolic methods (namely, test input generation techniques) to generate various levels for DragoxBox, an algebra-learning video game that became the most purchased game in Norway on the Apple App Store (Liu 2012). In contrast, we use symbolic methods for generating valid start states, and use simulation for estimating the difficulty level.

Problem generation. Recently, there has been interesting work on generating fresh set of practice problems for various subject domains including proof problems for high-school algebra (Singh, Gulwani, and Rajamani 2012) and procedural problems for middle-school mathematics (Andersen, Gulwani, and Popovic 2013). The former uses probabilistic testing to guarantee the validity of a generated problem candidate (from abstraction of the original problem) on random inputs, but there is no guarantee of the difficulty level. Our simulation is also akin to this probabilistic testing approach, but it is used to guarantee difficulty level; whereas validity is guaranteed by symbolic methods. Interesting starting states that require few steps to play and win are often published in newspapers and magazines for sophisticated games like Chess and Bridge. These are usually obtained from database of past games. In contrast, we show how to automatically generate such states, albeit for simpler games.

Conclusion. We revisit the classic domain of simple board games that have been popular since ancient times. We define a novel problem of generating starting states of a given difficulty level, given rules of the game. We present a novel search technique that combines use of symbolic methods and iterative simulation. Our experiments identified states of varying difficulty level for various games, opening up new games to be played that were believed to be useless for ages.

References

Allis, V. 1988. A knowledge-based approach of connect-four. Vrije Universiteit, Subfaculteit Wiskunde en Informatica.

Andersen, E.; Gulwani, S.; and Popovic, Z. 2013. A trace-based framework for analyzing and synthesizing educational progressions. In CHI.

Duncan, G.; Dowsett, C.; Claessens, A.; Magnuson, K.; Huston, A.; Klebanov, P.; Pagani, L.; Feinstein, L.; Engel, M.; Brooks-Gunn, J.; et al. 2007. School readiness and later achievement. Developmental psychology 43(6):1428.

Gardner, M. 1979. Mathematical games in which players of ticktacktoe are taught to hunt bigger game. Scientific American (April 1979) 18–26.

Gardner, M. 1983. Tic-tac-toe games. In Wheels, life, and other mathematical amusements. WH Freeman. chapter 9.

Gottlieb, S. 2003. Mental activity may help prevent dementia. BMJ 326(7404):1418.

Harary, F. 1977. Generalized tic-tac-toe. http://en.wikipedia.org/wiki/Harary's_generalized_tic-tac-toe.

Hunt, M.; Pong, C.; and Tucker, G. 2007. Difficulty-driven sudoku puzzle generation. UMAPJournal 343.

Liu, J. 2012. Dragonbox: Algebra beats angry birds. Wired.

Ma, W. J. Generalized tic-tac-toe. http://www.weijima.com/index.php?option=com_content&view=article&id=11&Itemid=15.

Ramani, G., and Siegler, R. 2008. Promoting broad and stable improvements in low-income childrens numerical knowledge through playing number board games. Child development 79(2):375–394.

Singh, R.; Gulwani, S.; and Rajamani, S. 2012. Automatically generating algebra problems. In AAAI.

Smith, G.; Treanor, M.; Whitehead, J.; and Mateas, M. 2009. Rhythm-based level generation for 2D platformers. In International Conference on Foundations of Digital Games. ACM.

Smith, A. M.; Andersen, E.; Mateas, M.; and Popović, Z. 2012. A case study of expressively constrainable level design automation tools for a puzzle game. In International Conference on the Foundations of Digital Games. ACM.

Weisstein, E. W. Tic-tac-toe. From MathWorld–A Wolfram Web Resource http://mathworld.wolfram.com/Tic-Tac-Toe.html.

Williams-King, D.; Denzinger, J.; Aycock, J.; and Stephenson, B. 2012. The gold standard: Automatically generating puzzle game levels. In AIIDE.

XUE, Y.; JIANG, B.; LI, Y.; YAN, G.; and SUN, H. 2009. Sudoku puzzles generating: From easy to evil. Mathematics in Practice and Theory 21:000.

Appendix

We first present a summary of our experimental results, and then illustrate the hardness of an example.

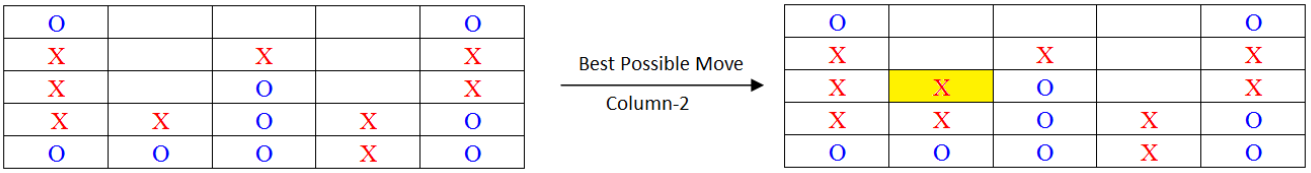
Summary. We summarize our results in Table 4. We call a state category-i state if it is not easy for depth-$(i-1)$ strategy, but it is easy for depth-i strategy. In Table 4 we summarize the different games and the existence of category i states in such games.

Illustration of hardness with an example. In Figure 2, we present how different depth-k strategies choose the best available position to mark on a Connect-4 RCD category-3 state. The example board position of Figure 2 is the same as for Figure 1 (e). The three depth-k strategies ($k = 1, 2, 3$) play as player-X and assign a score to each of the three available positions (column-$2, 3, 4$) by looking k-turns ahead. In each sub-figure, the position with yellow-background is the one chosen for exploration and the positions with grey-background are the predicted moves of how the game might turn out after k-turns.

As observed, only $k = 3$ strategy is able to foresee that marking column-2 would lead player-X to a winning state and also conclude that the other column choices will lead to a draw. Where as, $k = 1, 2$ incorrectly choose column-3 as the best position to mark hence making this starting position a category-3 state.

Table 4: Summary

Game	Category-1	Category-2	Category-3	Category-4
Tic-Tac-Toe	All variants	3x3 - only RCD 4x4 - All except RCD	-	-
Bottom-2	All variants	All variants except 3x3-CD RC for 4x4	RCD for 3x3 and 4x4	-
CONNECT-3	All variants	All i=2 variants	All i=2 variants except RCD	-
CONNECT-4	All variants	All variants	All variants	All i=3 variants



(a) $k = 1$, column-2 fetches Player-X a reward of +5

(b) $k = 1$, column-3 fetches Player-X a reward of +6

(c) $k = 1$, column-4 fetches Player-X a reward of +2

(d) $k = 2$, column-2 fetches Player-X a reward of +3

(e) $k = 2$, column-3 fetches Player-X a reward of +6

(f) $k = 2$, column-4 fetches Player-X a reward of +0

(g) $k = 3$, column-2 fetches Player-X a reward of $+\infty$

(h) $k = 3$, column-3 fetches Player-X a reward of +0

(i) $k = 3$, column-4 fetches Player-X a reward of +0

Figure 2: Depth-K strategy exploration on a CONNECT-4 RCD category-3 state