Automatic Generation of Alternative Starting Positions for Simple Traditional Board Games

Umair Z. Ahmed
IIT Kanpur
umair@iitk.ac.in

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

Sumit Gulwani
Microsoft Research, Redmond
sumitg@microsoft.com

Abstract

Simple board games, like Tic-Tac-Toe and CONNECT-4, play an important role not only in the development of mathematical and logical skills, but also in the emotional and social development. In this paper, we address the problem of generating targeted starting positions for such games. This can facilitate new approaches for bringing novice players to mastery, and also leads to discovery of interesting game variants. We present an approach that generates starting states of varying hardness levels for player 1 in a two-player board game, given rules of the board game, the desired number of steps required for player 1 to win, and the expertise levels of the two players. Our approach leverages symbolic methods and iterative simulation to efficiently search the extremely large state space. We present experimental results that include discovery of states of varying hardness levels for several simple grid-based board games. The presence of such states for standard game variants like 4×4 Tic-Tac-Toe opens up new games to be played that have never been played as the default start state is heavily biased.

1 Introduction

Board games involve placing pieces on a pre-marked surface or board according to a set of rules by taking turns. Some of these grid-based two-player games like Tic-Tac-Toe and CONNECT-4 have a relatively simple set of rules, yet, they are decently challenging for certain age groups. Such games have been immensely popular across centuries.

Studies show that board games can significantly improve a child's mathematical ability (Ramani and Siegler 2008). Such early differences in mathematical ability persist into secondary education (Duncan et al. 2007). Board games also assist with emotional and social development of a child. They instill a competitive desire to master new skills in order to win. Winning gives a boost to their self confidence. Playing a game within a set of rules helps 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 learn cause and effect by observing that decisions they make in the beginning of the game have consequences later on.

Board games help 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. They bridge the gap between young and old. They bolster the self-esteem of children who take great pride and pleasure when an elder spends playing time with them.

Significance of Generating Fresh Starting States

Board games are typically played with a default start state (e.g., empty board in case of Tic-Tac-Toe and CONNECT-4). However, there are following drawbacks in starting from the default starting state, which we use to motivate our goals.

Customizing hardness level of a start state. The default starting state for a certain game, while being unbiased, *might not be conducive for a novice player to enjoy and master the game*. Traditional board games in particular are easy to learn but difficult to master because these games have intertwined mechanics and force the player to consider far too many possibilities from the standard starting configurations. Players can achieve mastery most effectively if complex mechanics can be simplified and learned in isolation. Csikszentmihalyi's theory of flow (Csikszentmihalyi 1991) suggests that we can keep the learner in a state of maximal engagement by continually increasing difficulty to match the learner's increasing skill. Hence, we need an approach that allows generating start states of a specified hardness level. This capability can be used to generate a progression of starting states of increasing hardness. This is similar to how students are taught educational concepts like addition through a progression of increasingly hard problems (Andersen, Gulwani, and Popovic 2013).

Leveling the playing field. The starting state for commonly played games is mostly unbiased, and hence *does not offer a fair experience for players of different skills.* The flexibility to start from other starting states that is more biased towards the weaker player can allow for leveling the playing field and hence a more enjoyable game. Hence, we need an approach that takes as input the expertise levels of players and uses that information to associate a hardness level with a state.

Generating multiple fresh start states. A fixed starting state 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), starting from the default empty starting state. Players can *memorize certain moves* from a fixed starting state and gain undue advantage. Hence, we need an approach that generates multiple start states (of a specified hardness level). This observation has also inspired the design of Chess960 (Wikipedia 2014) (or Fischer Random Chess), which is a variant of chess that employs the same board and pieces as standard chess; however, the starting position of the pieces on the players' home ranks is randomized. The random setup renders the prospect of obtaining an advantage through memorization of opening lines impracticable, compelling players to rely on their talent and creativity.

Customizing length of play. 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. Certain interesting situations might manifest only in states that are typically not easily reachable from the start state, or require *too many steps*. The flexibility to start from such states might lead to more opportunities for practice of specific targeted strategies. Thus, we need an approach that can take as input a parameter for the number of steps that can lead to a win for a given player.

Experimenting with game variants. While people might be hesitant to learn a game with completely 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). However, *the default starting state of a new game may be heavily biased towards a player*; as a result that specific game might not have been popular. For example, consider the game of 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 with a naive strategy. Hence, such a game has never been popular. However, there can be non-default unbiased states for such games and starting from those states can make playing such games interesting. Hence, we need an approach that is parameterized by the rules of a game. This also has the advantage of experimenting with new games or variants of existing games.

Problem Definition and Search Strategy

We address the problem of automatically generating *interesting* starting states (i.e., states of desired hardness levels) for a given two-player board game. Our approach takes as input the *rules of a board game* (for game variants) and *the desired number of steps required for player 1 to win* (for controlling the length of play). It then generates multiple starting states of varying hardness levels (in particular, *easy, medium, or hard*) for player 1 for various *expertise level combinations of the two players*. We formalize the exploration of a game as a strategy tree and the expertise level of a player as depth of the strategy tree. The hardness of a state is defined w.r.t. the fraction of times player 1 will win, while playing a strategy of depth k_1 against an opponent who plays a strategy of depth k_2.

Our solution employs a novel combination of symbolic methods and iterative simulation to efficiently search for desired states. Symbolic methods are used to compute the winning set for player 1. These methods work particularly well for navigating a state space where the transition relation forms a sparse directed acyclic graph (DAG). Such is the case for those board games in which a piece once placed on the board doesn't move, as in Tic-Tac-Toe and CONNECT-4. Minimax simulation is used to identify the hardness of a given winning state. Instead of randomly sampling the winning set to identify a state of a certain hardness level, we identify states of varying hardness levels in order of increasing values of k_1 and k_2. The key observation is that hard states are much fewer than easy states, and for a given k_2, interesting states for higher values of k_1 are a subset of hard states for smaller values of k_1.

Contributions

- We introduce and study a novel aspect of graph games, namely generation of starting states. In particular, we address the problem of generating starting states of varying hardness levels parameterized by look-ahead depth of the strategies of the two players, the graph game description, and the number of steps required for winning (§2).
- We present a novel search methodology for generating desired initial states. It involves combination of symbolic methods and iterative simulation to efficiently search a huge state space (§3).
- We present experimental results that illustrate the effectiveness of our search methodology (§5). We produce a collection of initial states of varying hardness levels for standard games as well as their variants (thereby discovering some interesting variants of the standard games in the first place).

While our search methodology applies to any graph game; in our experiments we focus on generating starting states in *simple* board games and their variants as opposed to games with complicated rules.

Variants of traditional simple games are easier to adopt compared to games with complicated rules, which are hard to learn in the first place. The problem of automated generation of starting states should also be experimented for complex games as future work; however, our experimental results, which are focused on simple games, make a useful contribution since they make a valuable novel discovery for simple games which have not been studied in the past.

2 Problem Definition

2.1 Background on Graph Games

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 provides the mathematical framework to study many board games like 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 with three consecutive $\times$ (resp. $\circ$) in a row, column, or diagonal.

Classical game theory result. A classic result in the theory of graph games (Gale and Stewart 1953) shows that for every graph game with respective target sets for both players, 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 strategy 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 (player 1 can enforce a draw no matter how player 2 plays, and player 2 can enforce a draw no matter how player 1 plays). The classic result (aka determinacy) rules out the following possibility: against every player-1 strategy, player 2 can win; and against every player-2 strategy, player 1 can win. In the mathematical study of game theory, the theoretical question (which ignores the notion of hardness) is as follows: 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 result. 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 the classical game theory question is 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 ensuring set).

2.2 Formalization of Problem Definition

Notion of hardness. The game theoretic question ignores two aspects. (1) The notion of hardness: It is concerned with optimal strategies irrespective of hardness; and (2) the problem of generating different starting vertices. We are interested in generating starting vertices of different hardness. The hardness notion we consider is the depth of the tree a player explores, which is standard in artificial intelligence.

Tree exploration in graph games. Consider a player-1 vertex u_0. 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. 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 defined as follows: first construct the search tree of depth k and evaluate the tree bottom-up with min-max reasoning. In other words, a leaf vertex v is assigned reward $r(v)$, where the reward function r is game specific, and intuitively, $r(v)$ denotes how "close" the vertex v is to a winning vertex (see the following paragraph for an example). 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. Note that the rewards are assigned to vertices only based on the vertex itself without any look-ahead, and the exploration is captured by the classical min-max tree exploration.

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 (resp. n_2) be the number of two consecutive positions of marks for player 1 (resp. player 2) that can be extended to satisfy the winning condition. Then the reward is the difference $n_1 - n_2$. Intuitively, the number n_1 represents the number of possibilities for player 1 to win, and n_2 represents the number of possibilities for player 2, and their difference represents how favorable the board position is for player 1. 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 is illustrated in the appendix of full version available at (Ahmed, Chatterjee, and Gulwani 2015). 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_1 strategy σ_1 for player 1, and depth-k_2 strategy σ_2 for player 2, let O be the set of possible outcomes, i.e., the set of possible plays given σ_1 and σ_2 from v, where a play is a sequence of vertices. The strategies and the starting vertex define a probability distribution over the set of outcomes which we denote as $\mathrm{Pr}_v^{\sigma_1,\sigma_2}$, i.e., for a play ρ in the set of outcomes O we have $\mathrm{Pr}_v^{\sigma_1,\sigma_2}(\rho)$ is the probability of ρ given the strategies. Note that strategies are randomized (because strategies choose distributions over the children in the search tree exploration), and hence define a probability distribution over the set of outcomes. This probability distribution is used to formally define the notion of hardness we consider.

Problem definition. We consider several board games (such as Tic-Tac-Toe, CONNECT-4, and variants), and our goal is to obtain starting positions that are of different hardness levels, where our hardness is characterized by strategies of different depths. Precisely, consider a depth-k_1 strategy for player 1, and depth-k_2 strategy for player 2, and a starting vertex $v \in W_j$ that is winning for player 1 within j-moves and a winning move (i.e., $j+1$ moves for player 1 and j moves of player 2). We classify the starting vertex 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_1, k_2)-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_1 strategy for player 1 and depth-k_2 strategy for player 2, respectively. Let $O_1 \subseteq O$ be the set of plays that belong to the set of outcomes 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_1, k_2)-classification of v is: (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).*

Remark 1. In the definition above we chose the probabilities $\frac{1}{3}$ and $\frac{2}{3}$, however, the probabilities in the definition could be easily changed and experimented. We chose $\frac{1}{3}$ and $\frac{2}{3}$ to divide the interval $[0, 1]$ symmetrically in regions of E, M, and H. Here we present results based on the above definition.

Our goal is to consider various games and identify vertices of different categories (hard for depth-k_1 vs. depth-k_2, but easy for depth-(k_1+1) vs. depth-k_2, for small k_1 and k_2).

Remark 2. In this work we consider classical min-max reasoning for tree exploration. A related notion is Monte Carlo Tree Search (MCTS) which in general converges to min-max exploration, but can take a long time. However, this convergence is much faster in our setting, since we consider simple games that have great symmetry, and explore only small-depth strategies.

3 Search Strategy

3.1 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: one predecessor operator for player 1, which uses existential quantification over successors, and one for player 2, which uses universal quantification over successors. 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: $W_0 = \mathsf{EPre}(T_1)$ (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. Formally, we consider a vertex v from W_j, consider a depth-k_1 strategy for player 1 and a depth-k_2 strategy for player 2, and play the game multiple times with starting vertex v to find out the hardness level with respect to (k_1, k_2)-strategies, i.e., the (k_1, k_2)-classification of v. 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 Tic-Tac-Toe 4×4 game is 6,036,001; and a CONNECT-4 5×5 game is 69,763,700 (above 69 million). Thus 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. So we need an initial metric to guide the search of vertices from W_j such that the metric computation is inexpensive. We solve the first issue with *symbolic* methods, and the second one by *iterative simulation*.

3.2 Symbolic methods

We discuss the symbolic methods 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) (Bryant 1986) that can efficiently represent a set of vertices using a DAG representation of a boolean formula representing the set of vertices. The tool CUDD supports many symbolic representation of state space using BDDs and supports many operations on symbolic representation on graphs using BDDs (Somenzi 1998).

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} denotes whether it is player 1 or player 2's turn to play. Note that the vertices of the game graph not only contains the information about the board configuration, but also additional information such as the turn of the players. To illustrate how a symbolic representation is efficient, consider the set of all valuations to boolean variables $y_1, y_2, \ldots, y_n$ where the first variable is true, and the second variable is false: an explicit enumeration requires to list 2^{n-2} valuations, where as a boolean formula representation is very succinct. Symbolic representation with BDDs exploit such succinct representation for sets of vertices, and are used in many applications, e.g. hardware verification (Bryant 1986).

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 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 Tic-Tac-Toe, a program to describe the symbolic transition is: 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's turn). This gives the symbolic description of the transition function.

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

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

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 as 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 W_j.

3.3 Iterative simulation

We now describe a computationally inexpensive way to aid sampling of vertices as candidates for starting positions of a given hardness level. Given a starting vertex v, a depth-k_1 strategy for player 1, and a depth-k_2 strategy for player 2, we need to consider the tree exploration of depth $\max\{k_1, k_2\}$ to obtain the hardness of v. Hence if either of the strategy 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_1 and k_2 as a guide for vertices to be explored in depth. We use a very *simple metric* for this purpose. The hard vertices are rarer than the easy vertices, and thus we rule out easy ones quickly using the following approach:

If k_1 is large: Given a strategy of depth k_2, the set of hard vertices for higher values of k_1 are a subset of the hard vertices for smaller values of k_1. Thus we iteratively start with smaller values and proceed to higher values of k_1 only for vertices that are already hard for smaller values of k_1.

If k_2 is large: Here we exploit the following intuition. Given a strategy of depth k_1, a vertex which is hard for high value of k_2 is likely to show indication of hardness already in small values of k_2. Hence we consider the following approach. For the vertices in W_j, we fix a depth-k_1 strategy, and fix a small depth strategy for the opponent and assign the vertex a number (called *score*) based on the performance of the depth-k_1 strategy and the small depth strategy of the opponent. The score indicates the fraction of games won by the depth-k_1 strategy against the opponent strategy of small depth. The vertices that have low score are then iteratively simulated against depth-k_2 strategies of the opponent to obtain vertices of different hardness level. This heuristic serves as a simple metric to explore vertices for large value of k_2 starting with small values of k_2.

4 Framework for Board Games

We now consider the specific problem of board games. We describe a framework to specify several variants of two-player grid-based board games such as Tic-Tac-Toe, CONNECT-4.

Different parameters. Our framework allows three different parameters to generate 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, where a player wins if a sequence of the player's moves are in a line, which could be along a row (R), a column (C), or the diagonal (D). The user can specify any combination: (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, not column); or (iv) CD (column or diagonal, not row). (3) The third parameter is related to the allowed moves of the player. At any point the players can choose any available column (i.e., column with at least one empty position)

Table 1: CONNECT-3 & -4 against depth-3 strategy of opponent; (C-3 (resp. C-4) stands for CONNECT-3 (resp. CONNECT-4)). The third column (j) denotes whether we explore from W_2 or W_3. The sixth column denotes sampling to select starting vertices if $|W_j|$ is large: "All" denotes that we explore all vertices in W_j, and Rand denotes first sampling 5000 vertices randomly from W_j and exploring them. The E, M, and H columns give the number of easy, medium, or hard vertices among the sampled vertices. For each $k_1 = 1, 2$, and 3 the sum of E, M, and H columns is equal to the number of sampled vertices, and * denotes the number of remaining vertices. Observe that $|W_j|$ is small fraction of $|V|$ (this illustrates the significance of our use of symbolic methods as opposed to the prohibitive explicit enumerative search). Also, observe that vertices labeled medium and hard are a small fraction of the sampled vertices (this illustrates the significance of our efficient iterative sampling strategy).

Game	State Space $\|V\|$	j	Win Cond	No. of States $\|W_j\|$	Sampling	$k_2 = 3$								
						$k_1 = 1$			$k_1 = 2$			$k_1 = 3$		
						E	M	H	E	M	H	E	M	H
C-3	4.1×10^4	2	RCD	110	All	*	24	5	*	3	0	*	0	0
4x4	6.5×10^4		RC	200	All	*	39	9	*	23	5	*	0	0
	7.6×10^4		RD	418	All	*	36	17	*	25	4	*	0	0
	6.5×10^4		CD	277	All	*	41	24	*	27	21	*	0	0
C-3		3	RCD, RC, CD	0	-									
4x4			RD	18	All	*	0	0	*	0	0	*	0	0
C-4	6.9×10^7	2	RCD	1.2×10^6	Rand	*	184	215	*	141	129	*	0	0
5x5	8.7×10^7		RC	1.6×10^6	Rand	*	81	239	*	70	186	*	0	0
	1.0×10^8		RD	1.1×10^6	Rand	*	106	285	*	151	82	*	0	0
	9.5×10^7		CD	5.3×10^5	Rand	*	364	173	*	209	96	*	0	0
C-4		3	RCD	2.8×10^5	Rand	*	445	832	*	397	506	*	208	211
5x5			RC	7.7×10^5	Rand	*	328	969	*	340	508	*	111	208
			RD	8.0×10^5	Rand	*	398	1206	*	464	538	*	179	111
			CD	1.5×10^5	Rand	*	146	73	*	171	110	*	120	72

but can be restricted according to the following parameters: (i) Full gravity (once a player chooses a column, the move is fixed to be the lowest available position in that column); (ii) partial gravity-ℓ (once a player chooses a column, the move can be 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 (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); and Bottom-2 (partial gravity-2) which is between Tic-Tac-Toe and CONNECT games in terms of moves allowed.

Features of our implementation. We have implemented our approach and the main features that our implementation supports are: (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_2 strategy of the opponent for $k_2 = 1, 2$ and 3 (typically in all the above games depth-3 strategies are quite intelligent, and hence we do not explore larger values of k_2). 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.

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

Game	State Space $\|V\|$	j	Win Cond	No. of States $\|W_j\|$	Sampling	$k_2 = 3$								
						$k_1 = 1$			$k_1 = 2$			$k_1 = 3$		
						E	M	H	E	M	H	E	M	H
3x3	4.1×10^3	2	RCD	20	All	*	5	0	*	1	0	*	0	0
	4.3×10^3		RC	0	-									
	4.3×10^3		RD	9	All	*	2	1	*	3	0	*	0	0
	4.3×10^3		CD	1	All	*	0	0	*	0	0	*	0	0
3x3		3	Any	0	-									
4x4	1.8×10^6	2	RCD	193	All	*	12	26	*	0	2	*	0	0
	2.4×10^6		RC	2709	All	*	586	297	*	98	249	*	0	0
	2.3×10^6		RD	2132	All	*	111	50	*	18	16	*	0	0
	2.4×10^6		CD	1469	All	*	123	53	*	25	8	*	0	0
4x4		3	RCD	0	-									
			RC	90	All	*	37	31	*	0	0	*	0	0
			RD	24	All	*	1	2	*	0	0	*	0	0
			CD	16	All	*	6	4	*	1	0	*	0	0

5 Experimental Results

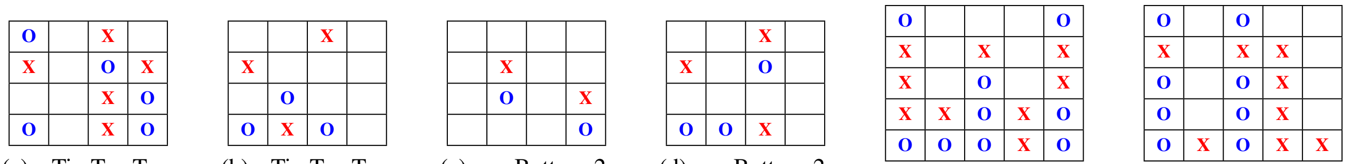
Our experiments reveal useful discoveries. The main aim is to investigate the existence of interesting starting vertices and their abundance in CONNECT, Tic-Tac-Toe, and Bottom-2 games, for various combinations of expertise levels and winning rules (RCD, RC, RD, and CD), for small lengths of plays. Moreover, the computation time should be reasonable.

Description of tables. The caption of Table 1 describes the column headings used in Tables 1-3, which describe the experimental results. In our experiments, we explore vertices from W_2 and W_3 only as the set W_4 is almost always empty. The third column $j = 2, 3$ denotes whether we explore from W_2 or W_3. For the classification of a board position, we run the game between the depth-k_1 vs the depth-k_2 strategy 30 times. If player 1 wins (i) more than $\frac{2}{3}$ times (20 times), then it is identified as easy (E); (ii) less than $\frac{1}{3}$ times (10 times), then it is identified as hard (H); (iii) else as medium (M).

Experimental results for CONNECT games. Table 1 presents results for CONNECT-3 and CONNECT-4 games, against depth-3 strategies of the opponent. An interesting finding is that in CONNECT-4 games with board size 5×5, for all winning conditions (RCD, RD, CD, RC), there are easy, medium, and hard vertices, for k_1=1,2, and 3, when j=3. That is, even in much smaller board size (5×5 as compared to the traditional 7×7) we discover interesting starting positions for CONNECT-4 games and its simple variants.

Experimental results for Bottom-2 games. Table 2 shows the results for Bottom-2 (partial gravity-2) against depth-3 strategies of the opponent. In contrast to CONNECT games, medium or hard vertices do not exist for depth-3 strategies.

Experimental results for Tic-Tac-Toe games. The results for Tic-Tac-Toe games are shown in Table 3. For Tic-Tac-Toe games the strategy exploration is expensive (a tree of depth-3 for 4×4 requires exploration of 10^6 nodes). Hence using the iterative simulation techniques we first assign a score to all vertices and use exploration for bottom hundred vertices (B100), i.e., hundred vertices with the least score according to our iterative simulation metric. In contrast to



(a) Tic-Tac-Toe RC, for $k_1 = 1$ (b) Tic-Tac-Toe CD, for $k_1 = 1$ (c) Bottom-2 RCD, for $k_1 = 2$ (d) Bottom-2 RC, for $k_1 = 2$ (e) CONNECT-4 RCD, for $k_1 = 2$ (f) CONNECT-4 RD, for $k_1 = 3$

Figure 1: Some "Hard" starting board positions generated by our tool for variety of games and different expertise level k_1 of player 1. The opponent expertise level k_2 is 3. Player 1 (x) can win in 2 steps for games (a)-(e) and in 3 steps for game (f).

Table 3: Tic-Tac-Toe against depth-3 strategy of opponent. The sampling B100 denotes exploring vertices with the least scored hundred vertices according to iterative simulation score.

Game	State Space $\lvert V\rvert$	j	Win Cond	No. of States $\lvert W_j\rvert$	Sampling	$k_2 = 3$								
						$k_1 = 1$			$k_1 = 2$			$k_1 = 3$		
						E	M	H	E	M	H	E	M	H
3x3	5.4×10^3	2	RCD	36	All	*	14	2	*	0	0	*	0	0
	5.6×10^3		RC	0	-									
	5.6×10^3		RD,CD	1	All	*	0	0	*	0	0	*	0	0
3x3		3	Any	0	-									
4x4	6.0×10^6	2	RCD	128	All	*	6	2	*	0	0	*	0	0
	7.2×10^6		RC	3272	B100	*	47	22	*	0	0	*	0	0
	7.2×10^6		RD,CD	4627	B100	*	3	2	*	0	0	*	0	0
4x4		3	RCD, RC	0	-									
			RD,CD	4	All	*	0	0	*	0	0	*	0	0

CONNECT games, interesting vertices exist only for depth-1 strategies.

Running times. The generation of W_j for j=2,3 took between 2-4 hours per game (this is a one-time computation for each game). The time to classify a vertex as E, M, or H for depth-3 strategies of both players, playing 30 times from a board position on average varies between 12 sec. (for CONNECT-4 games) to 25 min. for Tic-Tac-Toe games. Details for depth-2 strategy of the opponent are given in the appendix of (Ahmed, Chatterjee, and Gulwani 2015).

Important findings. Our first key finding is the *existence of vertices of different hardness levels* in various games. We observe that in Tic-Tac-Toe games only board positions that are hard for $k_1 = 1$ exist; in particular, and very interestingly, they also exist in board of size 4×4. Since the default start (the blank) vertex in 4×4 Tic-Tac-Toe games is heavily biased towards the player who starts first, they have been believed to be uninteresting for ages, whereas our experiments discover interesting starting vertices for them. With the slight variation of allowable moves (Bottom-2), we obtain board positions that are hard for $k_1 = 2$. In Connect-4 we obtain vertices that are hard for $k_1 = 3$ even with small board size of 5×5. For example, the default starting vertex in Tic-Tac-Toe 3×3 and Connect-4 5×5 does not belong to the winning set W_j; in Tic-Tac-Toe 4×4 it belongs to the winning set W_j and is Easy for all depth strategies.

The second key finding of our results is that *the number of interesting vertices 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 1.8 million, but has only two positions that are hard for $k_1 = 2$; and in CONNECT-4 RCD games with board size 5×5 the state space size is around sixty nine million, but has around two hundred hard vertices for $k_1 = 3$ and $k_2 = 3$, when $j = 3$, among the five thousand vertices sampled from W_j. Since the size of W_j in this case is around 2.8×10^5, the total number of hard vertices is around twelve thousand (among sixty nine million state space size). Since the interesting positions are quite rare, a naive approach of randomly generating positions and measuring its hardness will be searching for a needle in a haystack and be ineffective to generate interesting positions. Thus there is need for a non-trivial search strategy (§3), which our tool implements.

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. Also see appendix of (Ahmed, Chatterjee, and Gulwani 2015) for an illustration. In all the figures, player-X is the current player against opponent of depth-3 strategy. All these board positions were discovered through our experiments.

6 Related Work

Tic-Tac-Toe and Connect-4. Tic-Tac-Toe has been generalized to different board sizes, match length (Ma 2014), 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 2014). In contrast, we show that even simpler variants can be interesting if we start from certain specific states. Connect-4 research has also focussed on establishing a winning strategy from the default starting state (Allis 1988). In contrast, we study how easy or difficult is to win from winning states given expertise levels.

BDDs have been used to represent board games (Kissmann and Edelkamp 2011) to perform MCTS run with Upper Confidence Bounds applied to Trees (UCT). In such a usage, BDDs are instantiated to find the number of states explored by a single agent. In our setting we have two players, and use BDDs to compute the winning set.

Level generation. Goldspinner (Williams-King et al. 2012) is a level generation system for KGoldrunner, a puzzle 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 Refraction that adhered to pre-specified 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 these approaches, designers must explicitly specify constraints on the generated content, e.g., 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 DragonBox, which became the most purchased game in Norway on the Apple App Store (Liu 2012). In contrast, we use symbolic methods for generating start states, and use simulation for estimating their hardness level.

Problem generation. Automatic generation of fresh problems can be a key capability in intelligent tutoring systems (Gulwani 2014). The technique for generation of algebraic proof problems (Singh, Gulwani, and Rajamani 2012) 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 hardness level. Our simulation can be linked to this probabilistic testing approach, but it is used to guarantee hardness level; whereas validity is guaranteed by symbolic methods. The technique for generation of natural deduction problems (Ahmed, Gulwani, and Karkare 2013) and (Alvin et al. 2014) involves a backward existential search over the state space of all possible proofs for all possible facts to dish out problems with a specific hardness level. In contrast, we employ a two-phased strategy of backward and forward search; backward search is necessary to identify winning states, while forward search ensures hardness levels. Furthermore, our state transitions alternate between different players, thereby necessitating alternate universal vs. existential search over transitions.

Interesting starting states that require few steps to play and win are often published in newspapers 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.

Acknowledgments. The research was partly supported by Austrian Science Fund (FWF) Grant No P23499- N23, FWF NFN Grant No S11407-N23 (RiSE), ERC Start grant (279307: Graph Games), and Microsoft faculty fellows award.

References

Ahmed, U. Z.; Chatterjee, K.; and Gulwani, S. 2015. Automatic Generation of Alternative Starting Positions for Simple Traditional Board Games. *CoRR* abs/1411.4023.

Ahmed, U. Z.; Gulwani, S.; and Karkare, A. 2013. Automatically generating problems and solutions for natural deduction. In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, 1968–1975. AAAI Press.

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

Alvin, C.; Gulwani, S.; Majumdar, R.; and Mukhopadhyay, S. 2014. Synthesis of geometry proof problems. In *AAAI*.

Andersen, E.; Gulwani, S.; and Popovic, Z. 2013. A trace-based framework for analyzing and synthesizing educational progressions. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 773–782. ACM.

Bryant, R. 1986. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers* C-35(8):677–691.

Csikszentmihalyi, M. 1991. *Flow: The Psychology of Optimal Experience*, volume 41. New York, USA: Harper & Row Pub. Inc.

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.

Gale, D., and Stewart, F. M. 1953. Infinite games with perfect information. *Annals of Math.* Studies No. 28:245–266.

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

Gardner, M. 1983. Tic-tac-toe games. In *Wheels, life, and other mathematical amusements*, volume 86. WH Freeman. chapter 9.

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

Gulwani, S. 2014. Example-based learning in computer-aided stem education. *Commun. ACM*.

Harary, F. 1977. Generalized tic-tac-toe.

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

Kissmann, P., and Edelkamp, S. 2011. Gamer, a general game playing agent. *KI-Künstliche Intelligenz* 25(1):49–52.

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

Ma, W. J. 2014. Generalized tic-tac-toe. [Online; accessed 9-September-2014].

Ramani, G., and Siegler, R. 2008. Promoting broad and stable improvements in low-income children's 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 *FDG*.

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 *FDG*.

Somenzi, F. 1998. Cudd: Cu decision diagram package release. *University of Colorado at Boulder*.

Weisstein, E. W. 2014. Tic-tac-toe. From MathWorld–A Wolfram Web Resource; [Online; accessed 9-September-2014].

Wikipedia. 2014. Chess960 – wikipedia, the free encyclopedia. [Online; accessed 9-September-2014].

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.