Efficient belief-state AND–OR search, with application to Kriegspiel

Stuart Russell and Jason Wolfe
Computer Science Division
University of California, Berkeley, CA 94720
russell@cs.berkeley.edu, jawolfe@berkeley.edu

Content areas: Game-playing, uncertainty

Abstract

The paper reports on new algorithms for solving partially observable games. Whereas existing algorithms apply AND-OR search to a tree of black-box belief states, our "incremental" versions treat uncertainty as a new search dimension, examining the physical states within a belief state to construct solution trees incrementally. On a newly created database of checkmate problems for Kriegspiel (a partially observable form of chess), incrementalization yields speedups of two or more orders of magnitude on hard instances.

1 Introduction

"Classical" games, such as chess and backgammon, are fully observable. Partially observable games, despite their greater similarity to the real world, have received less attention in AI. The overall computational task in such games can be divided conceptually into two parts. First, *state estimation* is the process of generating and updating the *belief state*—a representation of the possible states of the world, given the observations to date. Second, *move selection* is the process of choosing a move given the belief state.

Partially observable games are intrinsically more complicated than their fully observable counterparts. At any given point, the set of logically possible states may be very large, rendering both state estimation and move selection intractable. Furthermore, move selection must consider one's own information state (gathering information is helpful) and the opponent's information state (revealing information is harmful). Finally, game-theoretically optimal strategies are *randomized* because restricting oneself to deterministic strategies provides additional information to the opponent. The paper by Koller and Pfeffer [1997] provides an excellent introduction to these issues.

In this paper, we focus on a particular subproblem: deciding whether a *guaranteed win* exists—i.e., a strategy that guarantees the optimal payoff within some finite depth, regardless of the true state, for any play by the opponent. This is simpler than the general problem, for two reasons. First, the state estimate need represent only the *logically possible* states, without considering their probabilities. Second, we have the following:

Theorem 1 *A guaranteed win exists from some initial belief state iff it exists against an opponent with full observation.*

This follows from the fact that an opponent, by simply *choosing moves at random*, has a nonzero probability of duplicating any finite behavior of an optimal opponent with full observation. Thus, the *logical possibility* that the solver can guarantee a win within some bounded depth depends only upon the *solver's* own information about the true state. This implies that neither the opponent's information state nor randomized strategies need be considered. We will see shortly that the theorem does *not* apply to certain other strategies, which win *with probability 1* but are not *guaranteed*.

Despite these restrictions, the problem is quite general. Indeed, it is isomorphic to that of finding guaranteed plans in nondeterministic, partially observable environments. Our particular focus in this paper is on *Kriegspiel*, a variant of chess in which the opponent's pieces are *completely invisible*. As the game progresses, partial information is supplied by a referee who has access to both players' positions. The players, White[1] and Black, both hear every referee announcement. There is no universally accepted set of rules; we adopt the following:

- White may propose to the referee any move that would be a legal chess move on a board containing just White's pieces; White may also propose any pawn capture.
- If the proposed move is illegal on the board containing both White and Black pieces, the referee announces "Illegal."[2] White may then try another move.
- If the move is legal, it is made. Announcements are:
  - If a piece is captured on square X: "Capture on X."
  - If Black is now in check: "Check by D," where D is one or two of the following directions (from the perspective of Black's king): Knight, Rank, File, Long Diagonal, and Short Diagonal.
  - If Black has no legal moves: "Checkmate" if Black is in check and "Stalemate" otherwise.
- Finally: "Black to move."

(Examples of play are given in Section 4. In addition, more details are available online at our Kriegspiel website, `www.cs.berkeley.edu/~jawolfe/kriegspiel/`.)

One important aspect of Kriegspiel and many other partially observable games and planning domains is the "try-until-feasible" property—a player may attempt any number

[1] W.l.o.g., we assume it is White's turn to move.

[2] If the move is not legal for White's pieces alone, or if it has already been rejected on this turn, the referee says "Nonsense."

of actions until a legal one is found. Because the order of attempts matters, a move choice is actually a plan to try a given *sequence* of potentially legal moves. Hence, the branching factor *per actual move* is the number of such sequences, which can be superexponential in the number of potentially legal moves. We will see that try-until-feasible domains admit certain simplifications that mitigate this branching factor.

Kriegspiel is very challenging for humans. Even for experts, "announced" checkmates are extremely rare except for so-called *material wins*—such as KQR-vs.-K—when one side is reduced to just a king and the other has sufficient (safe) material to force a win. Checkmate is usually "accidental" in the sense that one player mates the other without knowing it in advance.

Several research groups have studied the Kriegspiel checkmate problem. Ferguson [1992] exhibited a randomized strategy for the KBN-vs.-K endgame that wins *with probability 1*; the lone Black king can escape checkmate only by guessing White's moves correctly infinitely often. Subsequently [1995], he derived a strategy for the KBB-vs.-K endgame that wins with probability $1-\epsilon$ for any $\epsilon > 0$. In our terminology, these mates are not *guaranteed*, and they do not work if the opponent can see the board. Additional material-win strategies, all deterministic, have been developed [Ciancarini *et al.*, 1997; Bolognesi and Ciancarini, 2003; 2004].

As we explain in Section 4, algorithms for *finding* a checkmate in Kriegspiel involve searching an AND–OR tree whose nodes correspond to belief states. This idea is common enough in the AI literature on partially observable planning search (see, for example, Chapters 3 and 12 of [Russell and Norvig, 2003]). It was proposed for Kriegspiel, and applied to the analogous partially observable variant of Shogi (Japanese chess), by Sakuta and Iida; results for several search algorithms are summarized in Sakuta's PhD thesis [2001]. Bolognesi and Ciancarini [2004] add heuristic measures of "progress" to guide the tree search, but consider only positions in which the opponent has a lone king. In all of these papers, the initial belief state is determined externally.

The search algorithm developed by Ginsberg [1999] for (partially observable) Bridge play works by sampling the initial complete deal and then solving each deal as a fully observable game. This approach gives a substantial speedup over solving the true game tree, but never acts to gather or hide information (which is essential for domains such as Kriegspiel). The only previous Kriegspiel-playing agent we know of, developed by Parker *et al.* [2005], uses this approach. It keeps track of a sample of its true belief state, and at each point selects the move that would be best if the remainder of the game were played as fully observable chess.

Solving Kriegspiel is an instance of nondeterministic partially observable planning, and could therefore be carried out by symbolic techniques such as the ordered binary decision diagram (OBDD) methods developed by Bertoli *et al.* [2001] or by quantified Boolean formulae (QBF) solvers. Unfortunately, the computational penalty for generating chess moves by symbolic inference methods appears to be around four orders of magnitude [Selman, personal communication].

This paper's contributions are as follows. Section 2 addresses the problem of state estimation; for the purposes of this paper, we focus on exact estimation using straightforward methods. Section 3 describes a simple but complete Kriegspiel player, combining both state estimation and move selection, and explains how self-play was used to generate the first database of Kriegspiel checkmate problems with observation histories.

Section 4 develops the basic AND–OR tree structure for solving Kriegspiel-like games, allowing for possibly-illegal moves. Section 5 defines two baseline algorithms—versions of depth-first search (DFS) and proof-number search (PNS)—for solving such trees, and then presents some basic improvements for these algorithms and analyzes their performance on our checkmate database.

Section 6 develops a new family of *incremental* search algorithms that treat uncertainty as a new search dimension in addition to depth and breadth, incrementally proving belief states by adding a single constituent "physical state" to a solution at each step. This leads to a further speedup of one or more orders of magnitude. Finally, Section 7 shows how state estimation and incremental checkmate search may be interleaved.

2 State estimation

If we are to find a guaranteed win for White, state estimation must identify the (logical) *belief state*—the set of all *physical states* (configurations for White and Black pieces) that are consistent with White's history of moves and observations. A naive algorithm for exact state estimation looks like this:

- The initial belief state is a singleton, because Black's pieces start the game in their normal positions.
- For each White move attempt, apply the move to every physical state and remove those states that are inconsistent with the subsequent percept.
- For each Black turn,
  - For each sequence of k "Illegal" percepts for the unobserved Black move attempts, remove any physical state for which there are fewer than k distinct illegal moves.
  - Replace each remaining physical state by the set of updated states corresponding to all Black moves that are legal and yield the given percept when made in that state.
  - Remove duplicate states using a transposition table.

In Kriegspiel, the belief state can grow very large—10^{10} states or more—so this algorithm is not always practical. We have found, however, that with certain aggressive styles of play, the belief state remains no larger than a few thousand states throughout the game.

The naive algorithm given above can be viewed as a breadth-first expansion of a physical-state tree with branching at Black's moves and pruning by percepts. An alternative method, which we adopt, performs a depth-first search of the tree instead. This has the advantage of generating a stream of consistent current states with only a moderate amount of effort per state. Furthermore, if the set of states found so far does not admit a checkmate, then the whole belief state does not admit a checkmate and both state estimation and move selection can be terminated early (see Section 7).

A randomized depth-first search can generate a randomly selected sample of consistent states that can be used for approximate decision making. We explore approximate state estimation in a subsequent paper; for now, we assume that the exact belief state is available to the checkmate algorithm.

3 A Kriegspiel checkmate database

In order to evaluate checkmate-finding algorithms for Kriegspiel, we require a database of test positions. Prior to this work, no such database existed.[3] Our first database consists of White's move and percept histories for 1000 Kriegspiel games, up to the point where a 3-ply checkmate might exist for White.[4] 500 of these are actual mate instances; the other 500 are *near-miss* instances, which "almost" admit a guaranteed checkmate within 3 ply. For each near-miss instance, there is a checkmate plan that works in at least half of the possible physical states, but not in all of them.

This database was created by analyzing games between two different Kriegspiel programs. The first program, playing White, performs exact state estimation and makes a complex static approximation to 2-ply lookahead; it plays well but can be defeated easily by a skilled human.[5] The second program, playing Black, is much weaker: it computes a limited subset of its true belief state and attempts moves that are most likely to be captures and checks first. Whenever White's belief state has 100 Black positions or fewer, we determine if the belief state describes a mate or near-miss instance. If so, the move and percept history for the game-in-progress is saved. Games in which White's belief state grows above 10,000 positions are excluded. With these two programs, White's belief state generally remains fairly small, and about half the games played result in problem instances.[6]

Obviously, the checkmate problems we generate by this method have belief states that never exceed 10,000 physical states throughout the move history and have at most 100 physical states in the final position (the average is 11). Furthermore, the solution is at most 3-ply, but may include branches with illegal move attempts in addition to 3 actual moves.

By simply re-analyzing our 3-ply near-miss problems at 5-ply, we have also constructed a more difficult database of 258 5-ply mate instances and 242 5-ply near-miss instances. Both databases are available at our website (URL in Section 1).

As better state estimation, search, and evaluation methods are developed, it will be possible to construct more difficult problems that better reflect the kinds of positions reached in expert play. Nonetheless, our problems are far from trivial; for example, we will see that on 97% of the 5-ply near-miss instances, basic depth-first search requires more than 2000 CPU seconds to determine that no mate exists (within 5 ply).

4 Guaranteed Kriegspiel checkmates

Thanks to Theorem 1, our search problem involves a tree whose nodes correspond to White's belief states. Figure 1 shows a simple example: a miniature (4x4) 3-ply Kriegspiel checkmate. In the root belief-state node (1) there are three possible physical states, which differ in the locations and types of Black's pieces (White will always know the *number* of remaining Black pieces). The figure depicts a minimal proof tree for the problem instance, with other possible moves by White omitted; it describes the following strategy:

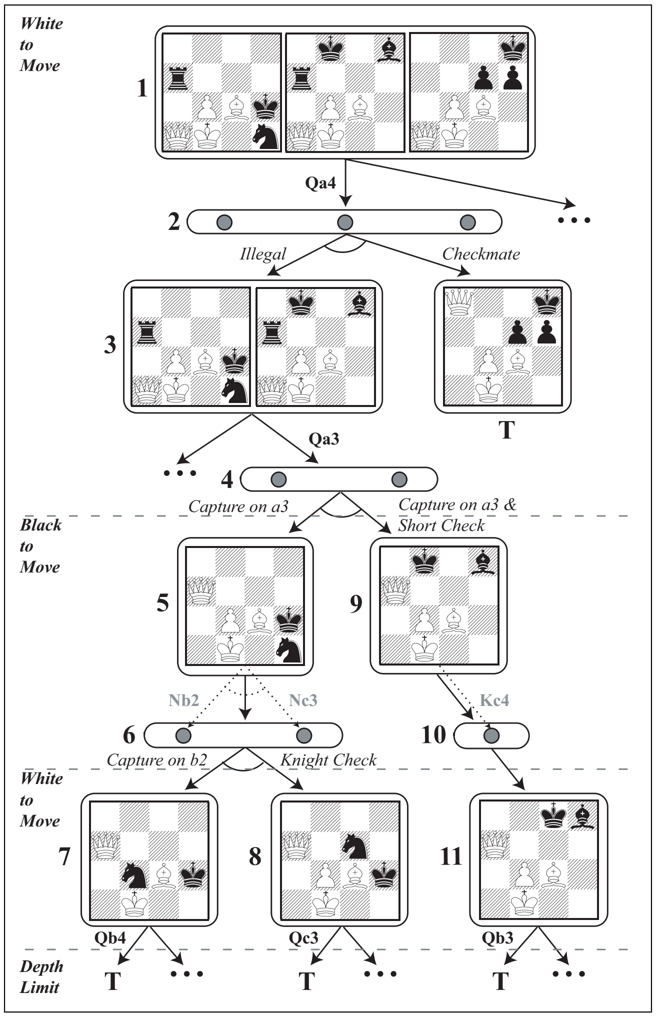

Figure 1: A minimal AND–OR proof tree for a 4x4 Kriegspiel 3-ply checkmate problem. The grayed moves in the Black-to-Move section are hidden from White.

1. White attempts move Qa4 from belief state 1. If the right-most state (1.c) is the true state, White wins.
2. Otherwise, Qa4 was illegal and White now attempts move Qa3 from belief state 3.
   (a) If the subsequent percept is "Capture on a3," Black has two legal moves: Nb2 and Nc3.
      i. If Black makes Nb2, the referee announces "Capture on b2" and White mates with Qb4.
      ii. If Black makes Nc3, the referee announces "Knight Check" and White mates with Qc3.
   (b) If the subsequent percept is "Capture on a3 & Short (Diagonal) Check," Black has only one legal move: Kc4. White mates with Qb3.

In general, belief-state AND–OR trees consist of three types of nodes:

- OR-nodes: In Figure 1, OR-nodes appear in the *White*

[3]The Internet Chess Club has a database of several thousand Kriegspiel games, but guaranteed wins cannot be identified because the database omits the history of attempted moves.

[4]In fact, we find deeper mates through the simple expedient of classifying leaf nodes as wins if Black is checkmated or if White has a known material win, as defined above.

[5]For ordinary play, we combine this move selection algorithm with an online version of our approximate depth-first state estimation algorithm.

[6]To achieve this high efficiency, we add extra illegal move attempts to White's move history as "hints"; without these hints, fewer games satisfy our belief-state size criteria.

to Move sections (e.g., nodes 1, 3, 7). An OR-node represents a choice between possible moves for White, and is proven iff at least one of its children is proven. Its children are AND-nodes, each containing the results of applying a single move in every possible physical state.

- EXPAND-nodes: EXPAND-nodes appear in the *Black to Move* sections, representing Black's moves (e.g., nodes 5, 9). Since Black's moves are invisible to White, each EXPAND-node has only a single child, an AND-node containing the union (eliminating duplicates) of the legal successors of its possible physical states. An EXPAND-node is proven iff its only child is proven.
- AND-nodes: AND-nodes are the thin nodes that appear at every other level in the tree (e.g., nodes 2, 4, 6). Physical states within AND-nodes are abbreviated as circles. An AND-node represents the arrival of a percept from the referee, and can be terminal or non-terminal:
  - If every physical state in an AND-node is a terminal win for White, the node is terminal with value $true$. If *any* physical state is a terminal draw or loss for White, the node is terminal with value $false$.
  - Otherwise, the AND-node is nonterminal, and has children that form a partition of its nonterminal physical states (percepts do not change the underlying physical states—see, e.g., nodes 7 and 11).

  Thus, an AND-node is proven iff all of its belief-state-tree children *and* its terminal physical states are proven.

In Kriegspiel, the referee makes an announcement after each move attempt. Thus, Kriegspiel belief-state trees have AND-nodes at every other level. The intervening nodes alternate between EXPAND-nodes (Black moves) and sequences of OR-nodes (White move attempts).[7] Because one turn for White may involve several move attempts, White's entire turn has a worst-case branching factor equal to the factorial of the number of possible moves.

5 Searching belief-state AND–OR trees

This section describes two common algorithms—depth-first search and proof-number search—for searching belief-state AND–OR trees. Like other existing algorithms, both solve a belief-state tree as an ordinary AND–OR tree with black-box belief-state nodes. After introducing the algorithms, we evaluate their performance on our 5-ply checkmate database, with and without some basic improvements.

5.1 DFS and PNS

The pseudocode for **DFS** (depth-first search) is shown in Figure 2.[8] **DFS** operates using the EXPAND method, which constructs and evaluates the children of a belief-state node (as described in Section 4); as an example, Figure 3 shows EXPAND's OR-node instance. To use **DFS**, we simply initialize an OR-node with the root belief state and remaining depth, and pass it to SOLVE-TOP. In Figure 1, the numbers beside the nodes indicate an order in which **DFS** might expand them when searching the tree.

```
function SOLVE-TOP($b$) returns $true$ or $false$
  inputs: $b$, a belief-state node

  EXPAND($b$)
  return SOLVE($b$)
--------------------------------------------------
method SOLVE($b$ an OR-node) returns $true$ or $false$
  while CHILDREN($b$) is not empty do
    if SOLVE-TOP(FIRST(CHILDREN($b$))) then return $true$
    POP(CHILDREN($b$))
  return $false$
--------------------------------------------------
method SOLVE($b$ an EXPAND-node) returns $true$ or $false$
  return SOLVE-TOP(CHILD($b$))
--------------------------------------------------
method SOLVE($b$ an AND-node) returns $true$ or $false$
  if TERMINAL($b$) then return VALUE($b$)
  while CHILDREN($b$) is not empty do
    if not SOLVE-TOP(FIRST(CHILDREN($b$))) then return $false$
    POP(CHILDREN($b$))
  return $true$
```

Figure 2: The **DFS** algorithm.

```
method EXPAND($b$ an OR-node)
  for each $m$ in MOVES(FIRST(STATES($b$))) do
    $b' \leftarrow$ a new AND-node with TERMINAL($b'$)= $false$,
        VALUE($b'$)= $true$, DEPTH($b'$)= DEPTH($b$),
        CHILDREN($b'$)= an empty list, and
        STATES($b'$)= MAP(SUCCESSOR(*,$m$),STATES($b$))
    for each $s$ in STATES($b'$) do
      if $s$ is a win for White then remove $s$ from STATES($b'$)
      else if $s$ is terminal or DEPTH($b$)= 1 then
        $b' \leftarrow false$; break
    if $b' \neq false$ then
      PUSH($b'$,CHILDREN($b$))
      if STATES($b'$) is empty then
        TERMINAL($b'$) $\leftarrow true$; break
```

Figure 3: The OR-node instance of the EXPAND method, which constructs and evaluates the children of b.

PNS (proof-number search) is a best-first search algorithm for AND–OR trees, and is commonly believed to be superior to **DFS**. At each step **PNS** expands a "most-proving" node, which can make the largest contribution to proving or disproving the entire tree. A most-proving node is defined as any node that is a member of *both* a *minimal proof set* and a *minimal disproof set* of the tree, where a minimal proof/disproof set is a minimal-cardinality set of unexpanded nodes that, if proved/disproved, would be sufficient to prove/disprove the root. Every tree has at least one most-proving node; if there are multiple most-proving nodes, the **PNS** algorithm chooses one arbitrarily [Allis, 1994].[9]

5.2 Analysis and Improvements

Figure 4 shows the solving ability of our search algorithms on the 500 problems in our 5-ply database (for readability, we show only a subset of the algorithms tested). We will

[7]Thanks to Theorem 1, illegal Black moves are not considered.

[8]The pseudocode we present in this paper was written for simplicity, and does not include modifications necessary for handling possibly-illegal moves. Our actual implementations are also more efficient (for instance, they construct only one child at a time at OR-nodes), and thus differ significantly from the pseudocode shown.

[9]We alter the initialization of **PNS**'s tree to reflect the fact that wins occur only after White moves, but do not attempt to take the depth limit [Allis, 1994] or the amount of uncertainty [Sakuta, 2001] into account.

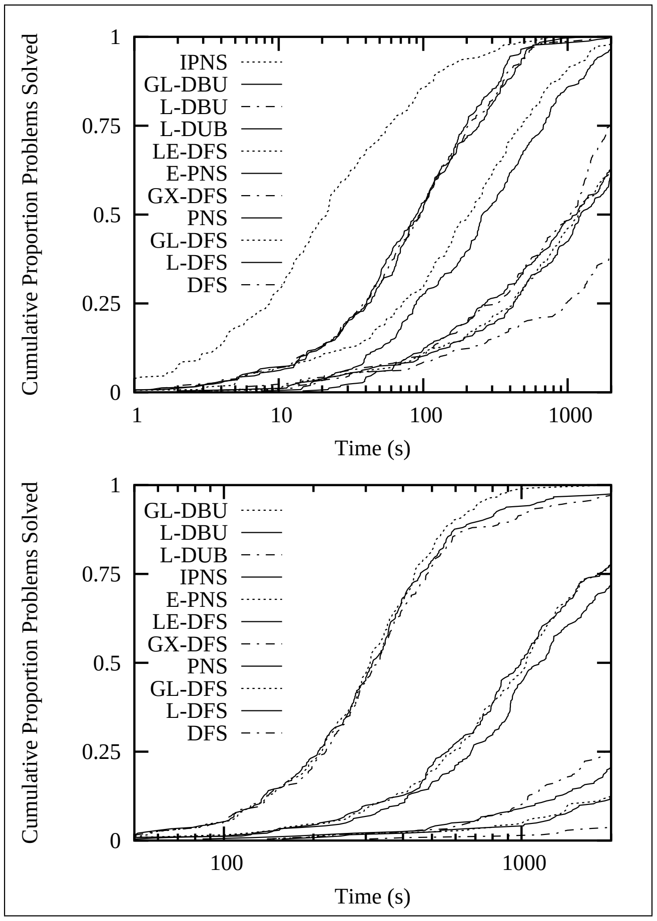

Figure 4: Performance of search algorithms on our 5-ply Kriegspiel checkmate database. Top: mate instances; Bottom: near-miss instances. The y-axes show the fraction of problems solvable within a given amount of CPU time (in Lisp, on a 550 MHz machine). The algorithms are ranked in decreasing order of efficiency.

introduce the **DBU**, **DUB**, and **IPNS** algorithms later, in Section 6. Performance on our 3-ply database (not shown) is qualitatively similar, but does not allow for accurate discrimination between our improved algorithms.

Basic **DFS** is by far the slowest of the algorithms tested, primarily because of the factorial branching factor for White (which subsequent algorithms avoid, to a large extent); basic **PNS** is much faster. Notice that the near-miss instances are generally more difficult to solve than the mate instances.

Heuristic ordering

When searching a belief-state AND–OR tree using a black-box algorithm such as **DFS**, there are two possible opportunities for heuristic ordering: White moves at OR-nodes, and percepts at AND-nodes. In this paper we focus on the underlying search algorithms; we do not investigate heuristic orderings for the White moves, and test only a simple but effective ordering for the percepts.

At AND-nodes, the *legal children* (children in which the last move was legal) are generally much cheaper for **DFS** to explore than the *illegal child*, since they have lower remaining depth. This suggests a simple heuristic: investigate the legal children first. As shown in Figure 4, **L-DFS** (Legal-first DFS) is considerably faster than **DFS**. On the other hand, **L-PNS** (not shown) performs almost identically to **PNS** (which naturally allocates its efforts efficiently).

Future work may investigate the effects of ordering the White moves (e.g., information-gathering and likely checking moves first) and the legal percepts (e.g., checks and captures first for Black and last for White).

Pruning

Because a proof of guaranteed checkmate is a single branching plan that must succeed in every physical state of a belief state, we can make the following observation:

Theorem 2 *If a belief state does not admit a guaranteed checkmate, no superset of that belief state admits a guaranteed checkmate.*

A straightforward implementation of the EXPAND method (e.g., Figure 3) constructs all elements of a belief state before evaluating any of them. Theorem 2 suggests a more efficient strategy: evaluate each physical state as soon as it is constructed. If a terminal physical state with value $false$ (or a nonterminal physical state at the depth limit) is found, the construction of the belief state can be halted early. In the best case, this reduces the effective search depth by one level (since only a single element of each belief state at the depth limit will be constructed). As shown in Figure 4 (indicated by **E-** for "early termination"), this simple idea is the most effective of the improvements we consider in this section.

Theorem 2 also suggests another pruning, which is specific to try-until-feasible trees. Consider the situation in which White is in belief state b, attempts a possibly-legal move, and is told that the move is illegal. White's new belief state $b' \subset b$. Theorem 2 implies that if b' does not admit a guaranteed checkmate, then neither does b. In other words, when the illegal child of an AND-node is disproved, this is sufficient to disprove the AND-node's parent OR-node as well. For example, if node 3 in Figure 1 were disproved, that would show not only that trying Qa4 first fails to ensure checkmate, but also that *no other White move from node 1 gives checkmate*. Clearly, this is a useful pruning rule.

We call this pruning *greedy*, since when combined with the legal-first heuristic it allows White's turns to be solved without backtracking, by adding moves to the plan iff they lead to checkmate when legal. Because a move plan cannot include repetitions, a greedy algorithm such as **GL-DFS** (Greedy Legal-first DFS) has a worst-case branching factor per White's turn that is only quadratic in the number of possible moves. However, Figure 4 shows that **GL-DFS** only slightly outperforms **L-DFS**. This is because the pruning only applies when there are moves that lead to checkmate if legal but not if illegal.

Perhaps surprisingly, our experiments show that a **G-DFS** algorithm performs better when it tries the *illegal child* first instead (even though the resulting algorithm is *not* actually greedy); this algorithm, shown as **GX-DFS** in Figure 4, outperforms even **PNS**. The power of **GX-DFS** stems from its ability to test a subset of its belief state using possibly-legal moves, and terminate early if it disproves the subset.

We did not implement a "**G-PNS**" algorithm, because the greedy pruning could force **PNS** to choose between the goals of proving and disproving the root (it always does *both simultaneously*). Future work may explore this issue further.

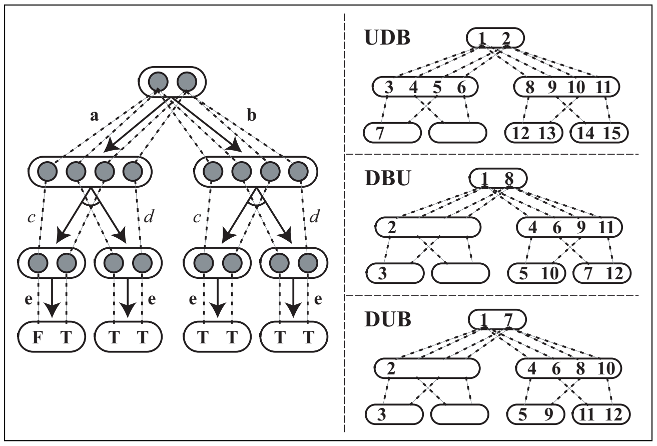

Figure 5: Left: a simple belief-state tree for a planning domain with nondeterministic transitions. **a**, **b**, and **e** are moves; *c* and *d* are percepts. Right: for each incremental algorithm, the order in which it would expand the nonterminal physical states in the tree.

6 Incremental belief-state AND–OR search

As we saw in the previous section, early termination via interleaved belief-state construction and evaluation can lead to large improvements in performance. This section develops this *incremental* idea into a novel framework for belief-state AND–OR tree search, which treats uncertainty as a new search dimension in addition to depth and breadth. After introducing this framework, we present results and theoretical analysis for our new algorithms.

6.1 Introduction

Ordinary AND–OR trees have two dimensions: depth and breadth. This leads to two "directional" search algorithms, depth- and breadth-first search, as well as numerous "best-first" algorithms (e.g., **PNS**). In addition to depth and breadth, belief-state AND–OR trees have uncertainty over physical states. By recognizing uncertainty as a new possible dimension for search, we can construct a new class of directional belief-state AND–OR search algorithms, as well as new best-first algorithms that balance all three factors efficiently.

In this paper, of the possible incremental directional algorithms, we consider only the three that put depth before breadth, which we will call **UDB**, **DBU**, and **DUB**. Figure 5 shows a simple belief-state tree for a domain with nondeterministic transitions, as well as the order that each of these algorithms would expand the physical states in the tree. The first algorithm, **UDB** (uncertainty-then-depth-then-breadth), is in fact just the **E-DFS** algorithm discussed in Section 5. In the figure, the difference between **UDB** and the other new algorithms should be immediately apparent; whereas **UDB** expands all physical states at a node before moving to the next node, the other algorithms begin by exploring the first *physical-state tree* in a depth-first manner. Thus, unlike existing algorithms, **DBU** and **DUB** can construct *minimal disproofs* that consider only a single element of each belief state.

In the tree, the difference between **DBU** and **DUB** first arises when selecting the seventh node for expansion. After establishing a proof on a single physical-state branch (i.e., non-branching path from the root to a leaf), **DBU** gives precedence to verifying the proof on the current *physical-state tree*, whereas **DUB** gives precedence to verifying it on the current *belief-state branch*. Thus, all three algorithms construct proofs by "looking inside" the belief state; they differ in that **UDB** incrementally constructs belief-state *nodes*, whereas **DUB** incrementally constructs *branches* and **DBU** incrementally constructs entire *proof trees*.

```
function SOLVE-TOP($b$) returns $true$ or $false$
  inputs: $b$, a belief-state node

  while STATES($b$) is not empty do
    INCREMENTAL-EXPAND($b$,POP(STATES($b$)))
    if not SOLVE($b$) then return $false$
  return $true$
-----------------------------------------------
method SOLVE($b$ an AND-node) returns $true$ or $false$
  if TERMINAL($b$) then return VALUE($b$)
  return $(\forall b' \in$ CHILDREN($b$)) SOLVE-TOP($b'$)
```

Figure 6: The **DBU** algorithm (which builds upon **DFS**).

At each point, **DBU** expands the deepest unexpanded physical state within the current proof tree. **DUB** does the same, except limited to a *single belief-state branch* at a time. Thus, in a pure OR-tree with no percept branching, **DUB** and **DBU** act identically. This brings us to an important point: the breadth that our **B** refers to is only the breadth of a proof, the AND-branching (percepts).

Whereas the algorithms differ significantly with respect to establishing disproofs, when exploring a proof tree such as the right branch of Figure 5, all three algorithms expand the *same* physical states, just in a different order. Since **UDB** and **DUB** both put breadth last, they explore the same sequence of belief-state branches, with different orderings for physical states within each branch. Likewise, **DUB** and **DBU** explore the same first physical-state branch.

In addition to these directional algorithms, we have implemented a best-first **IPNS** (incremental **PNS**) algorithm that operates on a single physical state at a time. This algorithm uses the above tree model, allowing AND-nodes to store unexpanded physical states. By simply redefining a most-proving node as a *physical state* that, if expanded, could contribute most to the proof/disproof of the entire tree, the proof-number idea naturally generalizes over uncertainty as well as depth and breadth. Among other things, this allows **IPNS** to naturally consider the relative ease of proving and disproving its belief-state nodes based on their sizes, an ability which other researchers have attempted to artificially introduce into a **PNS**-type algorithm [Sakuta, 2001].

6.2 Implementations

Our implementation of **DBU**, shown in Figure 6, uses a new INCREMENTAL-EXPAND method that expands a single physical state rather than an entire belief state at a time (its OR-node instance is shown in Figure 8, for comparison with Figure 3). When **DBU**'s SOLVE-TOP encounters uncertainty, it first constructs a proof for a single state, and then extends the proof to cover additional states one-at-a-time. To support such incremental proofs, **DFS**'s SOLVE instance for AND-nodes must also be modified to save proved children, rather than popping them; this allows **DBU** to continually refine a single proof tree that works in all physical states examined so far.

Our implementation of **DUB** uses *two* sets of recursive methods. The inner recursion is exactly that of **DBU**, ex-

```
function OUTER-TOP($b$) returns $true$ or $false$
  inputs: $b$, a belief-state node

  return (SOLVE-TOP($b$) and OUTER($b$))
--------------------------------------------------
method OUTER($b$ an OR-node) returns $true$ or $false$
  loop do
    if OUTER(FIRST(CHILDREN($b$))) then return $true$
    POP(CHILDREN($b$))
    if not SOLVE($b$) then return $false$
--------------------------------------------------
method OUTER($b$ an EXPAND-node) returns $true$ or $false$
  return OUTER(CHILD($b$))
--------------------------------------------------
method OUTER($b$ an AND-node) returns $true$ or $false$
  if TERMINAL($b$) then return VALUE($b$)
  loop do
    if not OUTER(FIRST(CHILDREN($b$))) then return $false$
    POP(CHILDREN($b$))          /* percept branching here */
    if CHILDREN($b$) is empty then return $true$
    if not SOLVE($b$) then return $false$
--------------------------------------------------
method SOLVE($b$ an AND-node) returns $true$ or $false$
  if TERMINAL($b$) then return VALUE($b$)
  return SOLVE-TOP(FIRST(CHILDREN($b$)))   /* not here */
```

Figure 7: The **DUB** algorithm (which builds upon **DBU**)

```
method INCREMENTAL-EXPAND($b$ an OR-node, $s$ a state)
  if CHILDREN($b$) is empty then    /* create b's children */
    for each $m$ in MOVES($s$) do
      $b' \leftarrow$ a new AND-node with TERMINAL($b'$)= $true$,
            VALUE($b'$)= $true$, DEPTH($b'$)= DEPTH($b$),
            CHILDREN($b'$)= an empty list, MOVE($b'$)= $m$,
            and STATES($b'$)= an empty list
      PUSH($b'$,CHILDREN($b$))
  for each $b'$ in CHILDREN($b$) do   /* integrate s's children */
    $s' \leftarrow$ SUCCESSOR($s$,MOVE($b'$))
    if $s'$ is terminal or DEPTH($b$)= 1 then
      if $s'$ is not terminal or $s'$ is not a win for White then
        remove $b'$ from CHILDREN($b$)
    else PUSH($s'$,STATES($b'$)); TERMINAL($b'$) $\leftarrow$ $false$
```

Figure 8: The OR-node instance of the INCREMENTAL-EXPAND method, which constructs and evaluates the children of s, integrating them into the children of b (which are also constructed if necessary).

cept that the SOLVE method for AND-nodes is modified to test only the first percept encountered (rather than all possible percepts); one might call this modified recursion simply **DU**. It either returns $false$, indicating a certain disproof, or $true$, representing a partial proof of a single belief-state-tree branch. The outer recursion, consisting of OUTER-TOP and OUTER, uses the inner recursion to construct a partial proof and then verify this proof on other percepts (deepest-first).

When implementing **DUB** or **DBU** in a try-until-feasible domain, a new issue arises: potential White moves that are *always illegal* are useless, but inflate the branching factor substantially; thus, it is crucial to avoid them during search. This is trivial for an uncertainty-first algorithm, since always-illegal moves can be filtered out during move generation. However, an incremental algorithm cannot use this method, because in general only a single physical state will be available when constructing a belief-state node. To avoid the large penalty associated with always-illegal moves, our actual implementations of **DUB** and **DBU** use the legal-first heuristic and skip the move in question (saving it for a later attempt) if it is not legal in any states examined so far.

With incremental search, there are also new opportunities for heuristic orderings that we have not yet investigated. For one, the physical states within a belief state can be ordered (e.g., best for Black first). One might also consider dynamic move orderings, using physical-state and/or belief-state transposition tables to cache proving moves; this could be especially effective in combination with iterative deepening.

6.3 Results

In Figure 4, we see that the directional incremental algorithms have significantly higher solving ability than their non-incremental counterparts. The true depth-first algorithms (**L-DUB** and **L-DBU**) perform at a similar level, outpacing **L-UDB** (**LE-DFS**) by a large margin. Again, greedy pruning has a small but significant effect: **GL-DBU** has the highest solving ability of the algorithms tested, solving 499 of 500 5-ply problems within the 2000-second time limit.[10]

In the figure, we see that **IPNS** is by far the most effective of our algorithms in solving the mate instances, but falls behind the true depth-first algorithms on the near-miss instances. This discrepancy can be explained by the depth limit, which strongly violates a basic assumption of **IPNS**: that the expected amount of work to disprove a physical state is constant throughout the tree. Thus, we expect that the discrepancy would disappear after adapting **IPNS** to the depth limit, or when searching without one.

6.4 Analysis

In this section, we conduct a brief analysis of the time and space complexity of our new algorithms. No directional algorithm is best in general; for specific classes of belief-state trees, however, clear differences do arise between the algorithms. In the following analysis, we focus on disproofs (since the algorithms generate the same trees for proofs), and ignore illegal moves and transpositions.

Recall that in any tree with all terminal leaves at the depth limit, **DFS** dominates **BFS** in the sense that for every fixed branch ordering, the set of nodes expanded by **DFS** will be a subset of the set of nodes expanded by **BFS**. We can make an analogous claim comparing the operation of **DUB** and **UDB**:

Theorem 3 *In a tree with all $false$ leaves at the depth limit, for any fixed branch ordering, the set of physical states expanded by **DUB** will be a subset of the set of physical states expanded by **UDB**.*

In this class of trees, **UDB** and **DUB** visit the same set of belief-state nodes with the same order of first visit. However, **DUB** does depth-first rather than uncertainty-first searches of each belief-state-tree branch, allowing it to find the $false$ leaves faster. Theorem 3 nearly holds for our problem database, because shallow $false$ leaves arise only from stalemates and Black checkmates, which are relatively rare in the positions we create.

[10]Incidentally, unlike any of our other algorithms, when a move in its current plan is disproved, **GL-DBU** can salvage the remainder of the plan.

Using a simple tree model, we can also approximate the best-case speedup and worst-case memory requirements for our new algorithms. Consider a belief-state tree rooted at an OR-node of size u_0, with depth d and fixed branching factors m_W, m_B, p_W, and p_B for the White and Black moves and percepts. In this tree, examine an arbitrary EXPAND-node 2-ply from the depth limit with size u', and define u=$u' * m_B$. If the belief-state tree has no terminal nodes, then the following table shows how many physical states each directional algorithm must construct to disprove the EXPAND-node (not including elements of the EXPAND-node itself):

DFS	$u + (u/p_B) * m_W$
UDB	$u + m_W$
DUB & **DBU**	$1 + m_W$

Since a majority of the tree's physical states will be located within 2-ply of the depth limit, we can approximate the overall performance of our search algorithms by the number of physical states they construct within its deepest 2-ply. Furthermore, because all four algorithms visit the same set of belief-state nodes in trees without terminal nodes, by setting u' to the *average* belief-state size of visited EXPAND-nodes 2-ply from the depth limit, we can interpret the values in the above table as approximately proportional to *run times*. Thus, in the best case, **DUB** and **DBU** are faster by roughly a factor of the average belief-state size in the tree. This is consistent with our observed speedup: in our 5-ply database, the average value of u (as defined above) is approximately 60.

Under the above tree model, with the additional stipulation that physical states be evenly distributed among percepts, the worst-case asymptotic memory requirements for efficient implementations of the algorithms are as follows:

DFS, **UDB**, & **DUB**	$O(u_0 * p_B \sum_{i=0}^{\lfloor d/2 \rfloor} (\frac{m_B}{p_B * p_W})^i)$
DBU	$O(u_0 * \sum_{i=0}^{\lfloor d/2 \rfloor} (m_B)^i)$
PNS & **IPNS**	$O(u_0 * \sum_{i=0}^{\lfloor d/2 \rfloor} (m_B m_W)^i)$

UDB and **DUB** store only a proving branch plus physical states for other possible percepts, whereas **DBU** must store a proof tree and **IPNS** must store the entire belief-state tree. Because of **IPNS**'s large memory requirements, one might attempt to construct a depth-first variant of the algorithm, analogous to recent work on ordinary **PNS** [Sakuta, 2001].

7 Interleaved state estimation and search

The depth-first method for state estimation described in Section 2 can be interleaved with the **DBU** checkmate-finding algorithm described in Section 6. As each new state is found by the state estimation algorithm, it is integrated into the current proof tree. This process continues until a disproof is found (early termination) or the entire belief state has been proven.

Computation times for "interleaved" vs. "sequential" methods (using **GL-DBU**) applied to each 3-ply database instance are shown in Figure 9. As expected, interleaving can provide substantial time savings on near-miss instances by eliminating the need for full state estimation, but has no effect on the solving of mate instances.

8 Conclusions and further work

We have proposed a new family of statewise-incremental solvers for belief-state AND-OR trees, and have shown them to yield large performance improvements on a database of Kriegspiel checkmate problems. Future work will enhance our complete Kriegspiel player with belief-state transposition tables (as explored by Sakuta [2001]) and improved methods for approximate state estimation and nonterminal evaluation, as well as evaluate further applications of incremental belief-state search. In particular, we plan to investigate dynamic move orderings and iterative deepening, further analyze the combination of incremental search and approximate state estimation, and apply incremental search to existing methods for general play. One might also consider incrementalization of partially observable planners and of QBF solvers more generally.

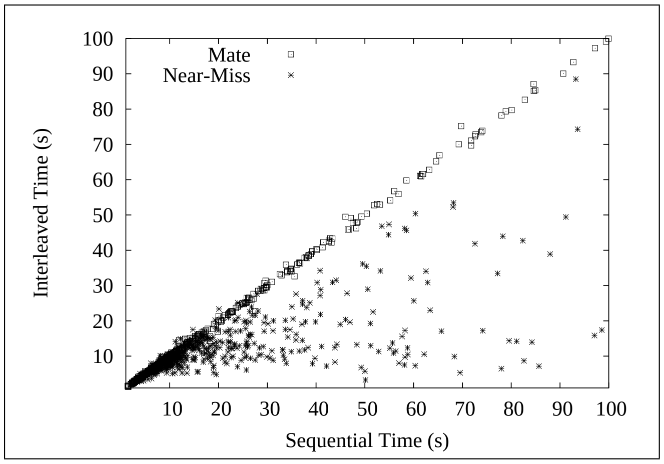

Figure 9: Results for interleaved state estimation and search.

References

[Allis, 1994] L. V. Allis. *Searching for Solutions in Games and Artificial Intelligence*. PhD thesis, University of Limburg, 1994.

[Bertoli *et al.*, 2001] P. Bertoli, A. Cimatti, M. Roveri, and P. Traverso. Planning in nondeterministic domains under partial observability via symbolic model checking. In *IJCAI*, 2001.

[Bolognesi and Ciancarini, 2003] A. Bolognesi and P. Ciancarini. Computer Programming of Kriegspiel Endings: The Case of KR vs. K. In *Advances in Computer Games 10*, 2003.

[Bolognesi and Ciancarini, 2004] A. Bolognesi and P. Ciancarini. Searching over Metapositions in Kriegspiel. In *Computers and Games 2004*. Springer-Verlag, 2004.

[Ciancarini *et al.*, 1997] P. Ciancarini, F. DallaLibera, and F. Maran. Decision Making under Uncertainty: A Rational Approach to Kriegspiel. In *Advances in Computer Chess 8*, 1997.

[Ferguson, 1992] T. Ferguson. Mate with Bishop and Knight in Kriegspiel. *Theoretical Computer Science*, 96:389–403, 1992.

[Ferguson, 1995] T. Ferguson. Mate with the Two Bishops in Kriegspiel. Technical report, UCLA, 1995.

[Ginsberg, 1999] M. L. Ginsberg. GIB: Steps toward an expert-level bridge-playing program. In *IJCAI*, 1999.

[Koller and Pfeffer, 1997] D. Koller and A. Pfeffer. Representations and solutions for game-theoretic problems. *Artificial Intelligence*, 94:167–215, 1997.

[Parker *et al.*, 2005] A. Parker, D. Nau, and V. S. Subrahmanian. Game-tree search with combinatorially large belief states. In *IJCAI*, 2005. (In press).

[Russell and Norvig, 2003] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, 2003.

[Sakuta, 2001] M. Sakuta. *Deterministic Solving of Problems with Uncertainty*. PhD thesis, Shizuoka University, 2001.