Exploiting Belief State Structure in Graph Search

Jason Wolfe & Stuart Russell
University of California - Berkeley
Introduction

• Recognizing already-solved subproblems can be essential for efficient search, e.g.,
  • A* graph search, α-β with transposition table
  • Subsumption in theorem proving
  • “Nogoods” in CSP & SAT solving, planning

• We present a novel, effective application to partially observable games & planning

• Generalize notion of graph search to incorporate subsumption relationships
Motivation

- Working on finding Kriegspiel checkmates
  - chess -- but opponent pieces and moves secret
  - symmetric percepts: illegal/check/capture/...
  - players attempt moves until one found legal ("try-until-feasible" property)
- Branching factor is huge (worst case ~30!), but we want to scale to 7-ply and beyond
- Developed methods here to help tame this branching factor
(4x4) **Kriegspiel**
(4x4) Kriegspiel

White to move and checkmate within 3 turns
A (4x4) Kriegspiel checkmate

White to move and checkmate within 3 turns
Outline

• Background:
  • Partially observable planning & games
  • Belief-state AND/OR trees
  • Search algorithms DFS and DBU

• Exploiting related belief states:
  • Graph versions of DFS and DBU
  • Data structures for subset lookups

• Experimental results & conclusions
Partially Observable Planning

- World state is **partially observable** (PO)
- Actions may be **nondeterministic**
- Plans may be **contingent** on observations
- Goal: **strong**, fixed-depth (acyclic) plans
  - Guaranteed to reach goal in fixed # actions
  - Find by searching AND-OR tree where nodes correspond to agent’s **belief states**

[cf. Bertoli et. al. (2001), Sakuta and Iida (2001)]
Guaranteed Wins in PO Games

• Strategies that guarantee the optimal payoff within $k$ moves.

• Isomorphic to strong acyclic plans
  • Treat other agents like nondeterminism
  • Works because win must be guaranteed (i.e., work for all possible opponent strategies)
**Belief-state and/or Trees**

![Diagram of belief-state and/or trees]
Belief-state and/or Trees

- OR-nodes
  - agent choice points
  - one child per move
  - child has possible action outcomes
  - proven iff all goal states or some child proven
Belief-state and/or Trees

• **OR-nodes**
  • agent choice points
  • one child per move
  • child has possible action outcomes
  • proven iff all goal states or some child proven

• **AND-nodes**
  • observation points
  • one child per obs.
  • children partition states
  • proven iff all children proven
**Search algorithm** DFS

- Simple approach: execute ordinary AND–OR search algorithm (e.g. **DFS**) on belief-state tree
  [Sakuta and Iida 2000; Bertoli et. al. 2001; Bolognesi and Ciancarini 2004]

- Essentially just codifies definitions on previous slide (pseudocode provided in paper)
DFS
DFS
DFS
DFS

*a*1

1

*a*2

o1

o2

*a*3

*a*4

o1

o2

Exploiting Belief State Structure in Graph Search

Jason Wolfe and Stuart Russell, UC Berkeley
DFS
DFS
DFS
DFS

Exploiting Belief State Structure in Graph Search

Jason Wolfe and Stuart Russell, UC Berkeley
Exploiting Belief State Structure in Graph Search
DFS

Exploiting Belief State Structure in Graph Search
DFS

Exploiting Belief State Structure in Graph Search

Jason Wolfe and Stuart Russell, UC Berkeley
Exploiting Belief State Structure in Graph Search
Exploiting Belief State Structure in Graph Search

Jason Wolfe and Stuart Russell, UC Berkeley
DFS
DFS
DFS
DFS
Search Algorithm DBU

- **DBU** = depth, breadth, then uncertainty

- Builds proofs **one physical state** at a time
  - fail fast if subset found unsolvable; enables **minimal disproofs**
  - can be significantly faster than **DFS**
    [Russell + Wolfe (2005)]
DBU

Exploiting Belief State Structure in Graph Search

Jason Wolfe and Stuart Russell, UC Berkeley
DBU

Exploiting Belief State Structure in Graph Search

Jason Wolfe and Stuart Russell, UC Berkeley
Exploiting Belief State Structure in Graph Search

Jason Wolfe and Stuart Russell, UC Berkeley

DBU

![Diagram of DBU structure with nodes and edges representing belief states and actions.](image)
Exploiting Belief State Structure in Graph Search

Jason Wolfe and Stuart Russell, UC Berkeley
DBU
Exploiting Belief State Structure in Graph Search

Jason Wolfe and Stuart Russell, UC Berkeley

DBU

\[ \begin{align*}
\text{o}_1 & \quad \text{o}_2 \\
a_1 & \quad a_2 \\
a_3 & \quad a_4 \\
a_3 & \quad a_4
\end{align*} \]
DBU
Exploiting Belief State Structure in Graph Search

DBU

Jason Wolfe and Stuart Russell, UC Berkeley
DBU
DBU

Exploiting Belief State Structure in Graph Search

Jason Wolfe and Stuart Russell, UC Berkeley

32
Exploiting Belief State Structure in Graph Search

Jason Wolfe and Stuart Russell, UC Berkeley
DBU

Exploiting Belief State Structure in Graph Search
DBU
DBU
DBU

Exploiting Belief State Structure in Graph Search

Jason Wolfe and Stuart Russell, UC Berkeley
Exploiting Belief State Structure in Graph Search

Jason Wolfe and Stuart Russell, UC Berkeley
Exploiting Belief State Structure in Graph Search

Jason Wolfe and Stuart Russell, UC Berkeley
Belief State AND/OR Graph Search

- Identifying repeated nodes can prevent:
  - re-doing previous work
  - wasting effort considering cyclic plans
- Previous work: exactly repeated nodes
  - [Bertoli et al. (2001)] rely on canonicality of BDD-based belief state representations
  - [Sakuta and Iida (2001)] use hashing on sets of explicitly represented physical states
Key Observations

• Since a plan for belief state $b$ must work at all of its physical states:
  • a plan for $b$ works on all subsets of $b$
  • no plans for $b$ $\Rightarrow$ all supersets of $b$ unsolvable

• We will fully exploit these facts by:
  • re-using proofs from supersets
  • re-using disproofs from subsets
  • avoiding generalized cycles (ancestor is subset)
Exploiting Belief State Structure in Graph Search

 DFS ⊆ Graph Search

Proven

Disproven

Stack
DFS ⊆ Graph Search

Exploiting Belief State Structure in Graph Search

Jason Wolfe and Stuart Russell, UC Berkeley
Exploiting Belief State Structure in Graph Search

DFS ⊆ Graph Search

(axioms)

\[ \text{proven} \]

\[ \text{disproven} \]

\[ \text{stack} \]

\{a,b\}
Exploiting Belief State Structure in Graph Search

DFS ⊆ Graph Search

proven

disproven

stack

{a,b} {b,c}
DFSC ⊆ Graph Search

proven

{t}

disproven

stack

{a,b} {b,c}
Exploiting Belief State Structure in Graph Search

DFS is a subset of Graph Search.

proven
{t} {b,c}
disproven
stack
{a,b}
Exploiting Belief State Structure in Graph Search

**DFSC** \( \subseteq \) **Graph Search**

Proven elements:
- \( \{t, b, c\} \)

Disproven elements:
- Stack:
  - \( \{a, b\} \)
  - \( \{d\} \)
Exploiting Belief State Structure in Graph Search

DFSC ⊆ Graph Search

proven
{t} \{b,c\}

disproven

stack
{a,b} \{d\}

...
DFS ⊆ Graph Search

Proven: \{t\} \{b,c\}

Disproven:

Stack:
\{a,b\} \{d\}
Exploiting Belief State Structure in Graph Search

DFS ⊆ Graph Search

proven

\{t\} \{b,c\}

disproven

stack

\{a,b\} \{d\}
Exploiting Belief State Structure in Graph Search

DFS ⊆ Graph Search

proven
\{t\} \{b,c\}
disproven
\{d\}
stack
\{a,b\}
Exploiting Belief State Structure in Graph Search

DFS ⊆ Graph Search

Proven: \{t\} \{b, c\}

Disproven: \{d\}

Stack: \{a, b\} \{c, a\}
DFS ⊆ Graph Search

Proven:
- \{t\} \{b,c\}

Disproven:
- \{d\}

Stack:
- \{a,b\} \{c,a\}
Exploiting Belief State Structure in Graph Search

DFS ⊆ Graph Search

Proven
\{t\} \{b,c\}

Disproven
\{d\}

Stack
\{a,b\} \{c,a\}
DFSC ⊆ Graph Search

proven
{t} {b,c}

disproven
{d}

stack
{a,b} {c,a}
Exploiting Belief State Structure in Graph Search
Exploiting Belief State Structure in Graph Search

DFSC ⊆ Graph Search

proven
\{t\} \{b,c\}

disproven
\{d\}

stack
\{a,b\} \{c,a\}
Exploiting Belief State Structure in Graph Search

DFS ⊆ Graph Search

proven

\{t\} \{b,c\}

disproven

\{d\}

stack

\{a,b\} \{c,a\}
DFS ⊆ Graph Search

DFS search is a special case of graph search. In DFS, the search order is determined by the order in which nodes are added to the stack. The stack is a last-in-first-out (LIFO) data structure, which means that the last node added to the stack is the first node to be removed. This ensures that the most recently visited node is always processed next, which can help to avoid getting stuck in infinite loops.

The diagram shows a graph with nodes labeled a, b, c, d, and t. The edges indicate the relationships between the nodes. The stack at each step is shown in the table on the right side of the diagram. The table has three columns: "proven", "disproven", and "stack".

The "proven" column contains a set of nodes that have been proven to be true. The "disproven" column contains a set of nodes that have been proven to be false. The "stack" column contains the nodes that are currently being processed.

In the example shown in the diagram, the search starts at node t and proceeds to nodes a and b in the first step. The stack contains the set {a, b}. In the second step, node c is added to the stack, and the stack becomes {a, b, c}. In the third step, node d is added to the stack, and the stack becomes {a, b, c, d}. The search then proceeds to node t in the fourth step, and the stack becomes {a, b, c, d, t}.

This process continues until all nodes in the graph have been visited. The final stack contains all nodes that have been visited, and the search is complete.
Exploiting Belief State Structure in Graph Search

DFS ⊆ Graph Search

proven
{t} {b,c} {c,a} {a,b}

disproven
{d}

stack
{a,b} {c,a}
DFSC Graph Search

- Previous versions exist for equality case
- Idea: memoization + cycle avoidance
  - mark belief states as proven, disproven, stack
  - at new belief state, find relevant related ones
  - must avoid the Graph History Interaction (GHI) problem
DBUC Graph Search

- No graph version previously existing
- Modifications are similar to DFS, except subsumption relationships can change
DBUC Graph Search

- No graph version previously existing
- Modifications are similar to DFS, except subsumption relationships can change
DBUC Graph Search

• No graph version previously existing
• Modifications are similar to DFS, except subsumption relationships can change
DBUC Graph Search

- No graph version previously existing
- Modifications are similar to DFS, except subsumption relationships can change
DBUC Graph Search

- No graph version previously existing
- Modifications are similar to DFS, except subsumption relationships can change
Finding Related Belief States

• Need data structure that:
  • finds relevant belief states related to query set
  • can efficiently add new sets & change status
  • constant-time lookups, updates not possible?

• We use variant of inverted files, found to work well in many different settings
  [Helmer and Moerkotte (1999)]
Inverted Files

• Hash ea. state -> list of belief states it’s in
Inverted Files

- Hash each state -> list of belief states it’s in
- 1:{a,c,d}
Inverted Files

- Hash ea. state -> list of belief states it’s in
- 1: \{a, c, d\}, 2: \{b, d\}
Inverted Files

- Hash each state -> list of belief states it’s in
- \(1: \{a, c, d\}, 2: \{b, d\}, 3: \{a, b, c\}\)
Inverted Files

• Hash ea. state -> list of belief states it’s in
• 1:{a,c,d}, 2:{b,d}, 3:{a,b,c}

Example: query {b,c,d}
Inverted Files

• Hash ea. state -> list of belief states it’s in

• 1:{a,c,d}, 2:{b,d}, 3:{a,b,c}

• Example: query {b,c,d}
Inverted Files

- Hash each state -> list of belief states it's in
- $1: \{a, c, d\}$, $2: \{b, d\}$, $3: \{a, b, c\}$

- Example: query $\{b, c, d\}$
Inverted Files

• Hash ea. state -> list of belief states it’s in
• 1:{a,c,d}, 2:{b,d}, 3:{a,b,c}

• Example: query \{b,c,d\}
**Inverted Files**

- Hash each state -> list of belief states it's in
- $1:\{a,c,d\}, \ 2:\{b,d\}, \ 3:\{a,b,c\}$

**Example:** query $\{a,c\}$
Inverted Files

• Hash ea. state -> list of belief states it’s in
• 1:{a,c,d}, 2:{b,d}, 3:{a,b,c}

• Example: query {a,c}
Inverted Files

• Hash ea. state -> list of belief states it’s in
• 1:{a,c,d}, 2:{b,d}, 3:{a,b,c}

• Example: query {a,c}
Inverted Files

• Hash ea. state -> list of belief states it’s in
• 1:{a, c, d}, 2:{b, d}, 3:{a, b, c}

Example: query {a, c}
Experiments

• Two very different domains

• Six total algorithms:
  • $\text{DFS}$ and $\text{DBU}$ (tree search)
  • $\text{DFS}=\text{}$ and $\text{DBU}=\text{}$ (hash tables)
  • $\text{DFSC} \subset \text{}$ and $\text{DBUC} \subset \text{}$ (inverted files)
Experiments

• Two very different domains

• Six total algorithms:
  • DFS and DBU (tree search)
  • DFS= and DBU= (hash tables)
  • DFS⊆ and DBU⊆ (inverted files)

*Best previous algorithm
Experiments

• Two very different domains

• Six total algorithms:
  • DFS and DBU (tree search)
  • DFS= and DBU= (hash tables)
  • DFS⊆ and DBU⊆ (inverted files)
  *Best previous algorithm
  ◆New algorithm
Experiments: Vacuum World

- Domain description:
  - grid world, some squares dirty
  - current square observable
  - actions: \textit{left}, \textit{right}, \textit{up}, \textit{down}, \textit{suck}
  - \textit{right} \& \textit{down} \textbf{may} dirty source square

- Problem instances:
  - world is $2 \times N$
  - initial belief state $=$
  - depth limit $= 3N+1$
Experiments: Vacuum World

Solution Time vs. Problem Size

- DFS
- DBU
- DFS =
- DBU =
- DFS ⊆
- DBU ⊆

Median solution time (s)

Problem size
Experiments: Vacuum World

Solution Time vs. Problem Size

Median solution time (s)

Problem size

DFS \subseteq DBU

DFS = DBU

DFS \subseteq DBU

DFS \subseteq DBU

Graph showing solution time vs. problem size for different algorithms (DFS and DBU) with different markers for equality and subset relationships.
## Experiments: Vacuum World

<table>
<thead>
<tr>
<th>Size</th>
<th>$2 \times 4$</th>
<th>$2 \times 5$</th>
<th>$2 \times 6$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Seconds</td>
<td>States</td>
<td>Seconds</td>
</tr>
<tr>
<td>DFS</td>
<td>502.3</td>
<td>49036K</td>
<td>*</td>
</tr>
<tr>
<td>DBU</td>
<td>174.6</td>
<td>5892K</td>
<td>*</td>
</tr>
<tr>
<td>*DFS=</td>
<td>3.4</td>
<td>257K</td>
<td>46.1</td>
</tr>
<tr>
<td>◆DBU=</td>
<td>0.5</td>
<td>11K</td>
<td>6.2</td>
</tr>
<tr>
<td>◆DFS≤</td>
<td>0.8</td>
<td>36K</td>
<td>5.1</td>
</tr>
<tr>
<td>◆DBU≤</td>
<td>0.4</td>
<td>10K</td>
<td>2.4</td>
</tr>
</tbody>
</table>

* Exceeded 10,000 seconds
** Exceeded 400 MB RAM
Experiments: Kriegspiel

(7-ply problems; large branching factor)

Proportion Problems Solved vs. Time Limit

- DBU ⊆
- DBU =
- DFS ⊇
- *DFS =

Proportion problems solved vs. Solution time limit (s)
Discussion

• >1 order of magnitude speedup over previous algorithms, in 2 domains
  • Subset testing gives big gains for low overhead

• Future work:
  • apply to other algorithms (e.g., PNS)
  • extend to symbolic representations (e.g., BDDs)
  • memory bounded search, garbage collection
  • ...

Exploiting Belief State Structure in Graph Search
QUESTIONS?