University of Basel

Single-Player Chess as a Planning Problem

Ken Rotaris <ken.rotaris@stud.unibas.ch>
Institute, University of Basel

Date: 04.07.2022

Planning? Single-Player Chess?

> “**Planning** is the art and practice of **thinking before acting**”
>
> — Patrik Haslum

Motivation

- Check the **validity** of a given chess problem (Puzzles)

Motivation

- Check the **validity** of a given chess problem (Puzzles)
- Study **ways** to solve Single-Player Chess

Motivation

- Check the **validity** of a given chess problem (Puzzles)
- Study **ways** to solve Single-Player Chess
- Comparison: **general purpose planner** vs. **domain specific planner**

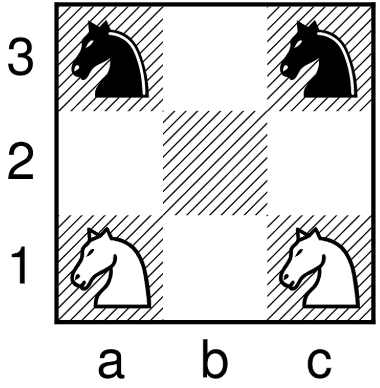
Motivation

- Check the **validity** of a given chess problem (Puzzles)
- Study **ways** to solve Single-Player Chess
- Comparison: **general purpose planner** vs. **domain specific planner**
- **Test limits** of a general-purpose planner (PDDL)
  - Bottleneck: Computing valid moves involving the King

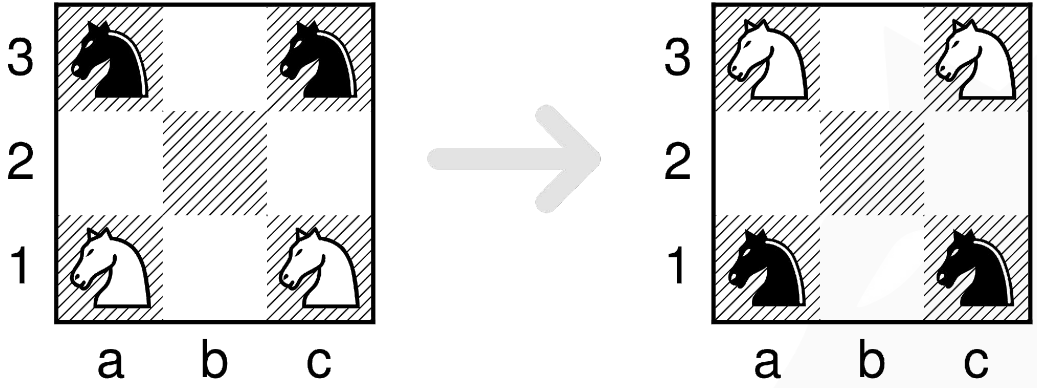
Motivation

- Check the **validity** of a given chess problem (Puzzles)
- Study **ways** to solve Single-Player Chess
- Comparison: **general purpose planner** vs. **domain specific planner**
- **Test limits** of a general-purpose planner (PDDL)
  - Bottleneck: Computing valid moves involving the King
- Chess engines are **awesome**!

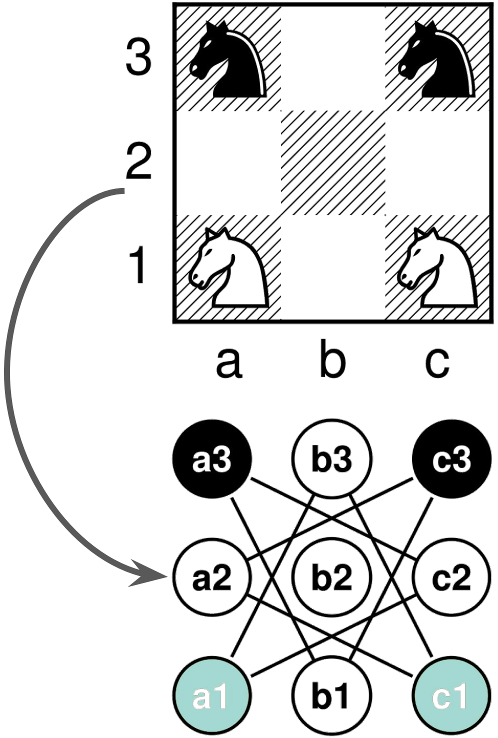
Example: Guarini Problem (1512 AD)



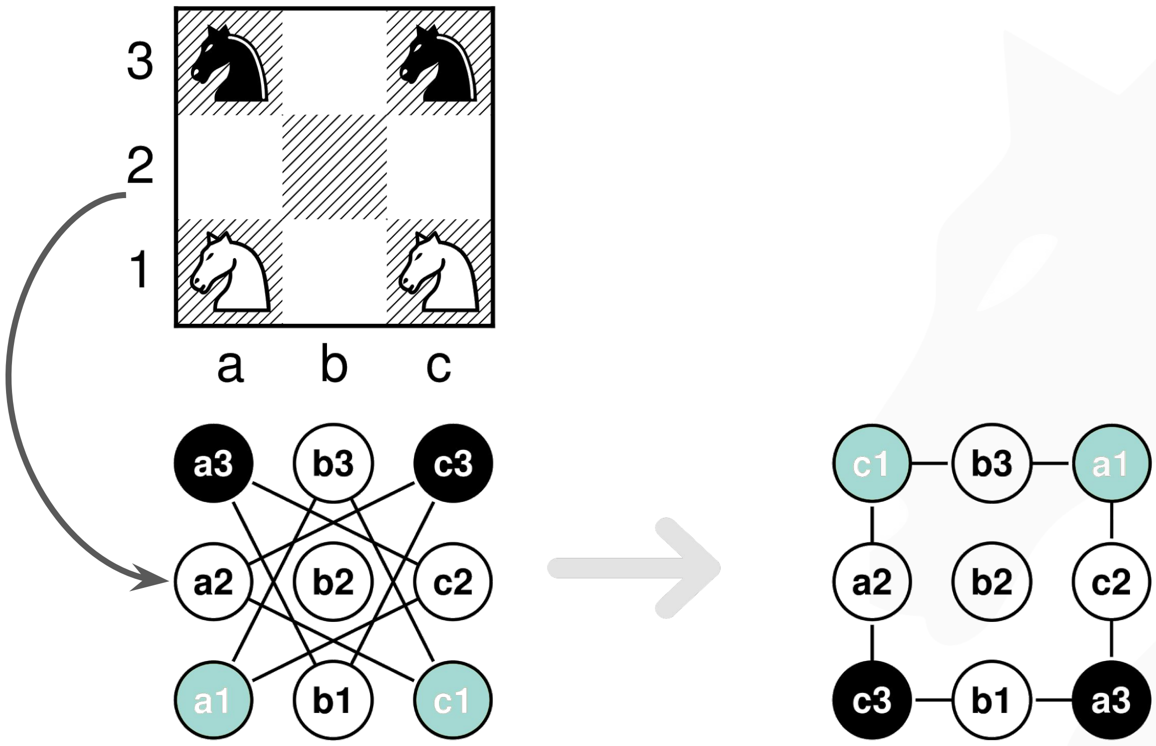
Example: Guarini Problem (1512 AD)



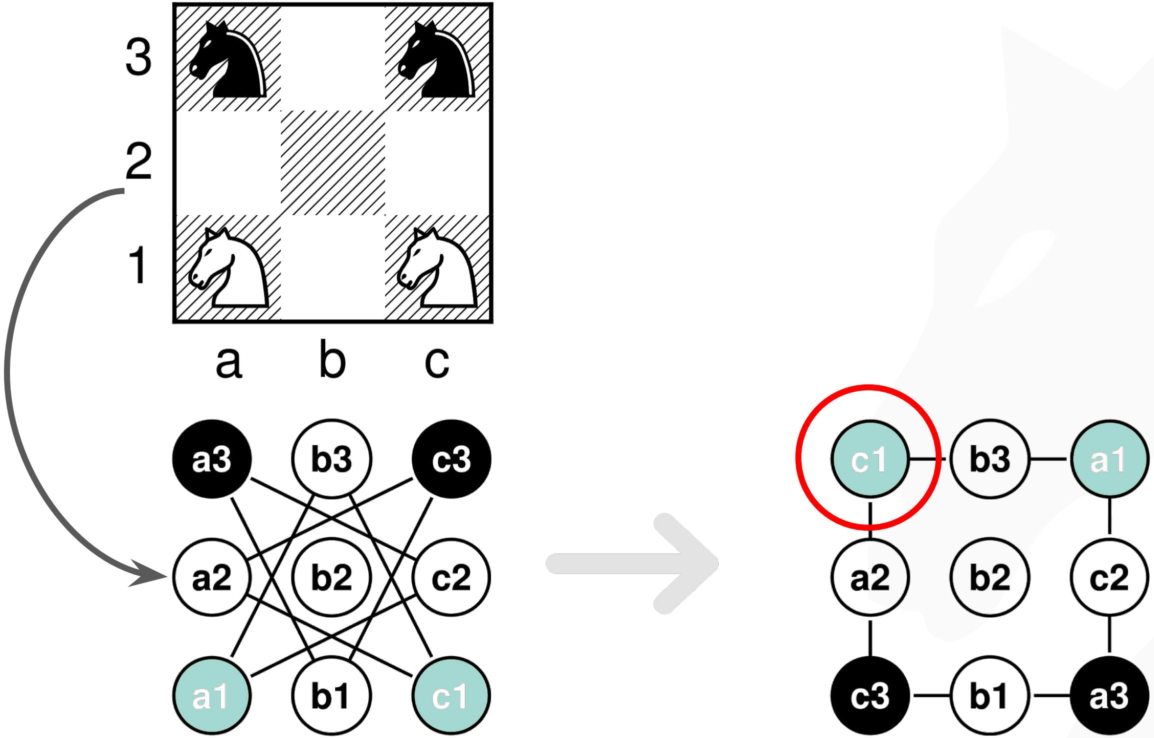
Example: Guarini Problem (1512 AD)



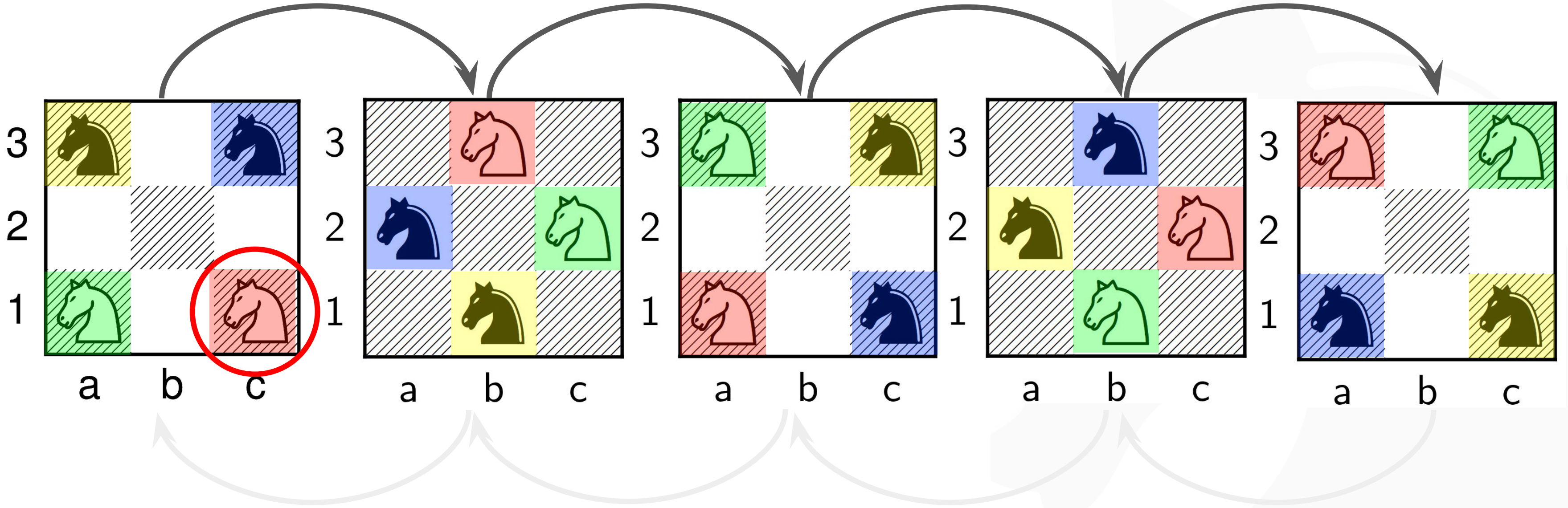
Example: Guarini Problem (1512 AD)



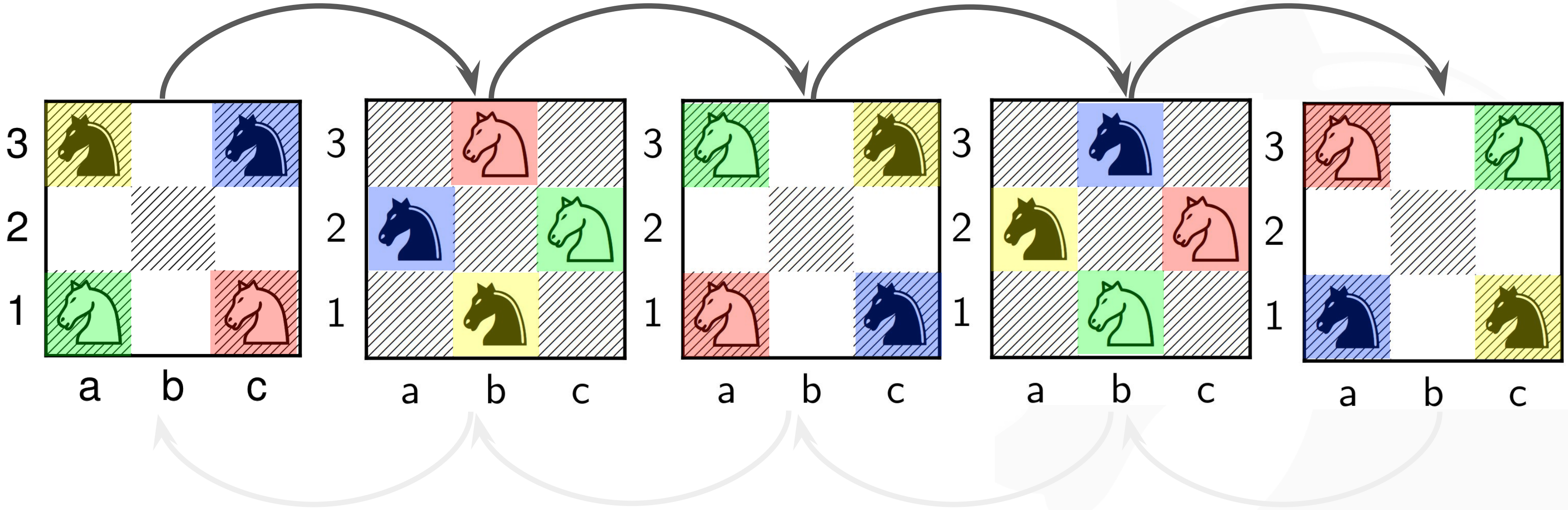
Example: Guarini Problem (1512 AD)



Example: Guarini Problem (1512 AD)



Example: Guarini Problem (1512 AD)



This is intuitive for humans but **how can we communicate the problem to an engine**?

Encoding

- We need to encode the problem in such a way that there is **no doubt on how the system can behave** in any given state

Encoding

- We need to encode the problem in such a way that there is **no doubt on how the system can behave** in any given state
- Encoding in a **formal language** for the domain-independent planner

Encoding

- We need to encode the problem in such a way that there is **no doubt on how the system can behave** in any given state.
- Encoding in a **formal language** for the domain-independent planner

Definition

A **domain-independent planner** is a generic tool which is independent of the problem at hand.

Encoding

- We need to encode the problem in such a way that there is **no doubt on how the system can behave** in any given state.
- Encoding in a **formal language** for the domain-independent planner

Definition

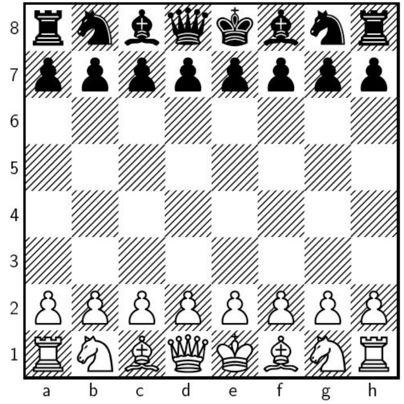
A **domain-independent planner** is a generic tool which is independent of the problem at hand.

⇒ A **domain-dependent planner** on the other hand **does not have the restrictions of a formal language** that needs **to be used to encode the problem** (we can encode it in whichever way we want).

Domain Specific Approach

1. **General** Purpose Planner Approach
   (domain-<u>in</u>dependent planner)

General Purpose Planner: Pipeline



→ **PDDL** → FAST DOWNWARD → **Plan**

Encoding

- absolute pins
- checks
- blocked movement
- inhibited moves
- normal moves
- castling
- double pawn moves
- en passant moves
- pawn promotions
- taking turns
- game termination
- etc.

The PDDL Language: Objects and Predicates

Example: Static Predicate

```
(:predicates
        (same_diagonal ?square_1 ?square_2 - square) → static predicate      } ∈ Domain file
)
(:objects
        (a1 a2 a3 b1 b2 b3 c1 c2 c3 - square) → objects                       } ∈ Problem file
)
(:init
        (same_diagonal a1 b2)
        (same_diagonal b2 a1)
        (same_diagonal b2 c3)
        …
)
```

→ Example: (same_diagonal a1 b2) = true *(any not defined combination of squares is false)*

The PDDL Language: Types of Predicates

Three types of predicates and their evaluation time:

1. **Static predicates***: The value does not change after it is defined (given as part of the model).*

The PDDL Language: Types of Predicates

Three types of predicates and their evaluation time:

1. **Static predicates***: The value does not change after it is defined (given as part of the model).*
2. **Fluent predicates**: *The value* ***changes via the ':effect'-section*** *if an action is applicable.*

The PDDL Language: Types of Predicates

Three types of predicates and their evaluation time:

1. **Static predicates***: The value does not change after it is defined (given as part of the model).*
2. **Fluent predicates**: *The value* ***changes via the ':effect'-section*** *if an action is applicable.*
3. **Derived predicates**: *The value is computed anew* ***whenever a state is generated****.*

The PDDL Language: Derived Predicates

Example: Capturable by white Piece

```
(:predicates
	(is_white ?figure - figure)  → static predicate
	(at ?figure - figure ?square - square)  → fluent predicate
	(occupied_by_black ?square - square)  → fluent predicate
	(capturable ?figure - figure ?from_square ?to_square - square)  → derived predicate
)
(:derived (capturable ?figure - figure ?from_square ?to_square - square)
	(and (at ?figure ?from_square)
		(is_white ?figure)
		(occupied_by_black ?to_square)
	)
)
```

→ Example: (capturable white_bishop_1 a1 a3) = true if there is a black figure on a3

The PDDL Language: Actions

Putting it all together:

Example: Capturing a black Piece with a white Bishop

```
(:action bishop_capture
        :parameters (?bishop - bishop ?from_square ?to_square - square)
        :precondition (and (capturable ?bishop ?from_square ?to_square)
                           (same_diagonal ?from_square ?to_square)
                      )
        :effect (and (not(at ?bishop ?from_square))
                     (at ?bishop ?to_square)
                     (not(occupied_by_black ?square))
                )
)
```

PDDL Specifics of the Model (The Rook)

Implementation: Rook Move (I. Objects)

```
(:objects
        ;locations:
        n1 n2 n3 n4 n5 n6 n7 n8  - location

        ;object pieces:
        king_w1 king_b1 - king
        rook_w1 - rook
)
```

PDDL Specifics of the Model (The Rook)

Implementation: Rook Move (II. Parameters & Preconditions)

```
(:action rook_move
      :parameters (?rook - rook ?from_file ?from_rank ?to_file ?to_rank - location)
      :precondition (and (valid_position)
                    (at ?rook ?from_file ?from_rank)
                    (myturn ?rook)
                    (or
                        (and
                            (= ?from_file ?to_file)
                            (vert_reachable ?rook ?from_file ?from_rank ?to_file ?to_rank)
                        )
                        (and
                            (= ?from_rank ?to_rank)
                            (horiz_reachable ?rook ?from_file ?from_rank ?to_file ?to_rank)
                        )
                    ))
      :effect (next slide…)
```

Either vertically or horizontally reachable

PDDL Specifics of the Model (The Rook)

Implementation: Rook Move (III. Effect Section)

```
(and  (forall (?figure - figure)
              (when
                    (and  (at ?figure ?to_file ?to_rank)
                          (not_same_color ?rook ?figure))
                    (and (not (at ?figure ?to_file ?to_rank))
                          (removed ?figure))
              )
      )
      (when (white_s_turn)
            (not(white_s_turn))
      )
      (when (not(white_s_turn))
            (white_s_turn)
      )
      (not(valid_position))
      …
```

static predicate (not_same_color)

If there is some figure of opposite color at the destination square…

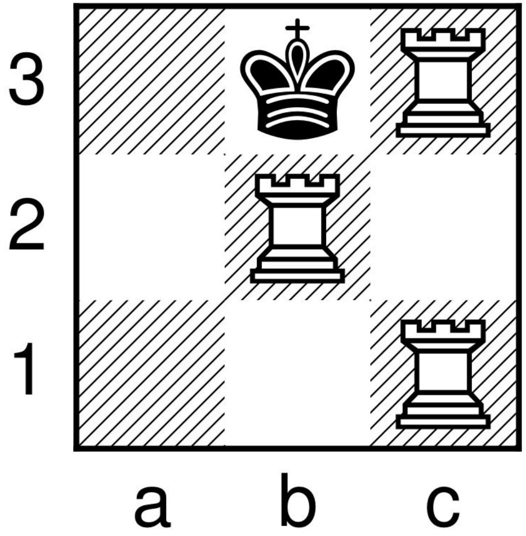
…remove that figure from the board

To prevent there being two figures at the same square

Turn taking: switch turn after every action/ move

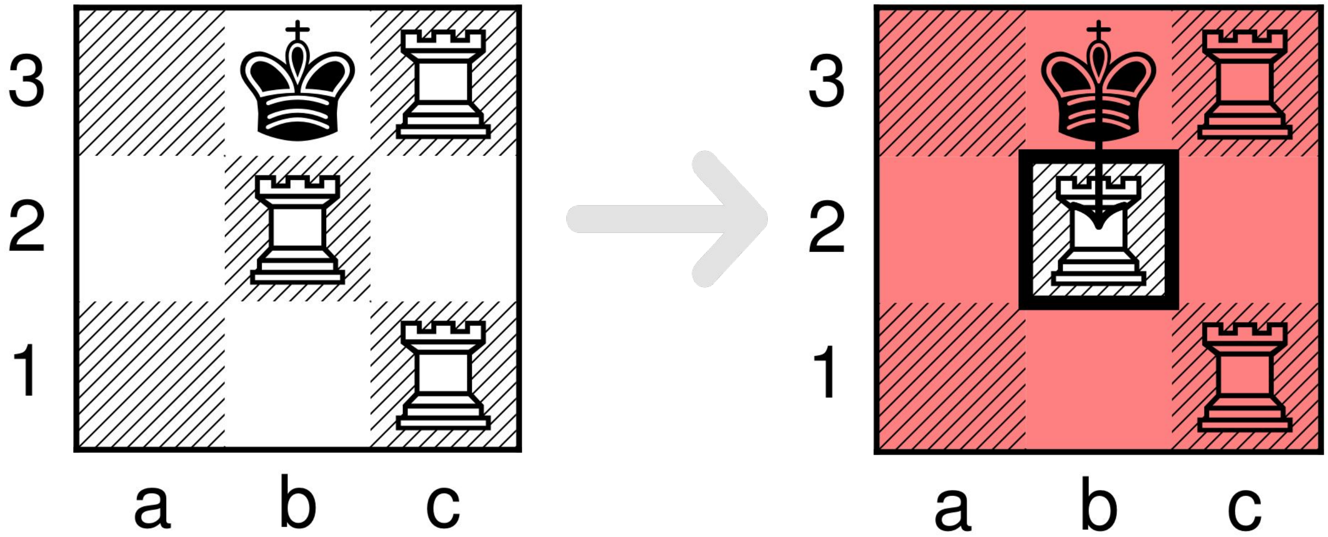
PDDL Specifics of the Model (Challenges)

Red Zone: Consider the following state:



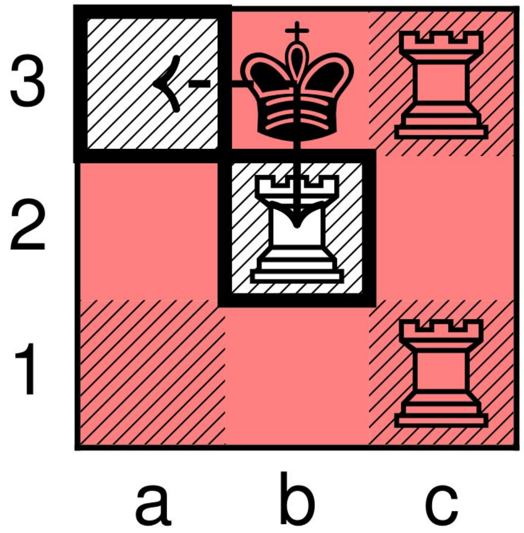
PDDL Specifics of the Model (Challenges)

Red Zone: Consider the following state:



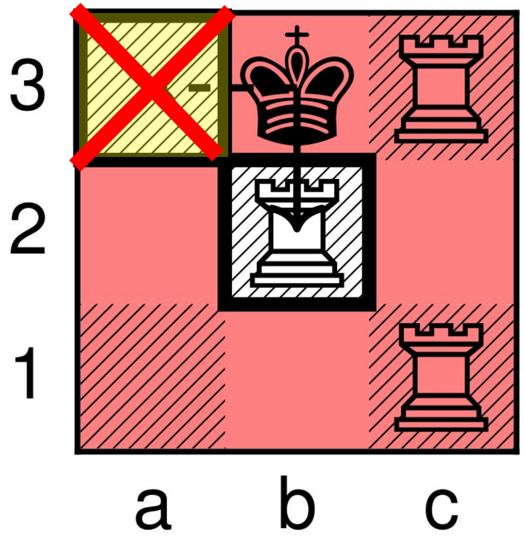
PDDL Specifics of the Model (Challenges)

Problem:



PDDL Specifics of the Model (Challenges)

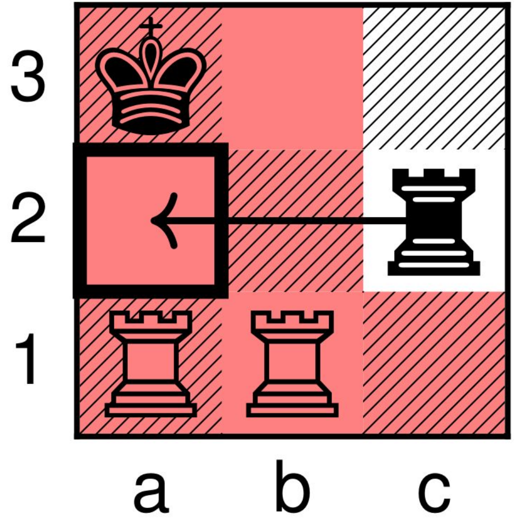
Problem: King can protect himself and move behind his own ‘shadow’:



PDDL Specifics of the Model (Challenges)

Other interesting problems include the detection of:

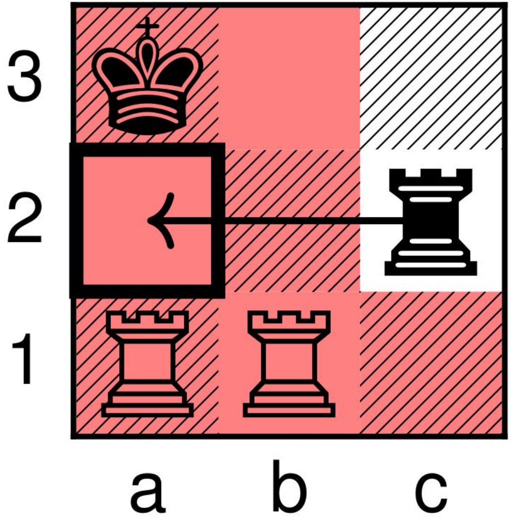
Forced Moves:



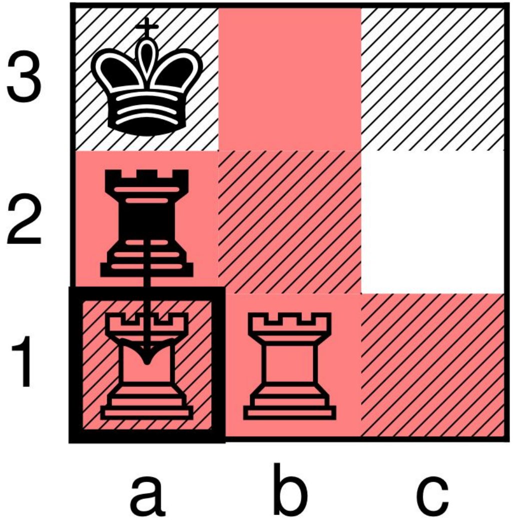
PDDL Specifics of the Model (Challenges)

Other interesting problems include the detection of:

Forced Moves:



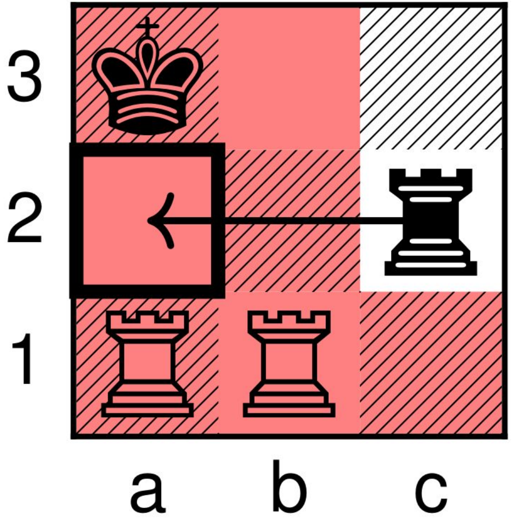
Absolute Pins:



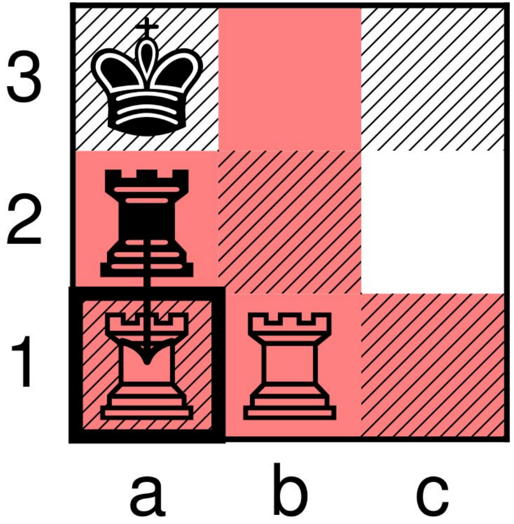
PDDL Specifics of the Model (Challenges)

Other interesting problems include the detection of:

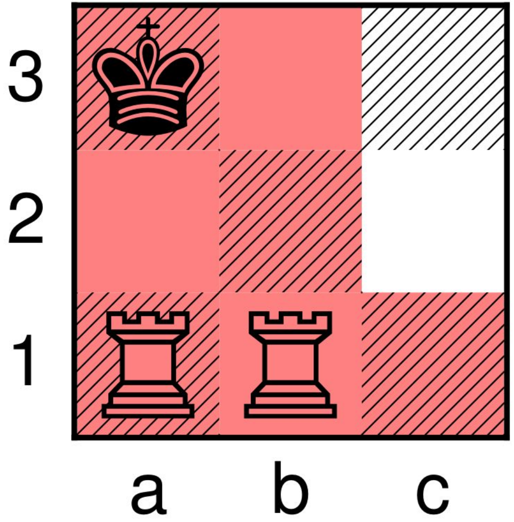
Forced Moves:



Absolute Pins:



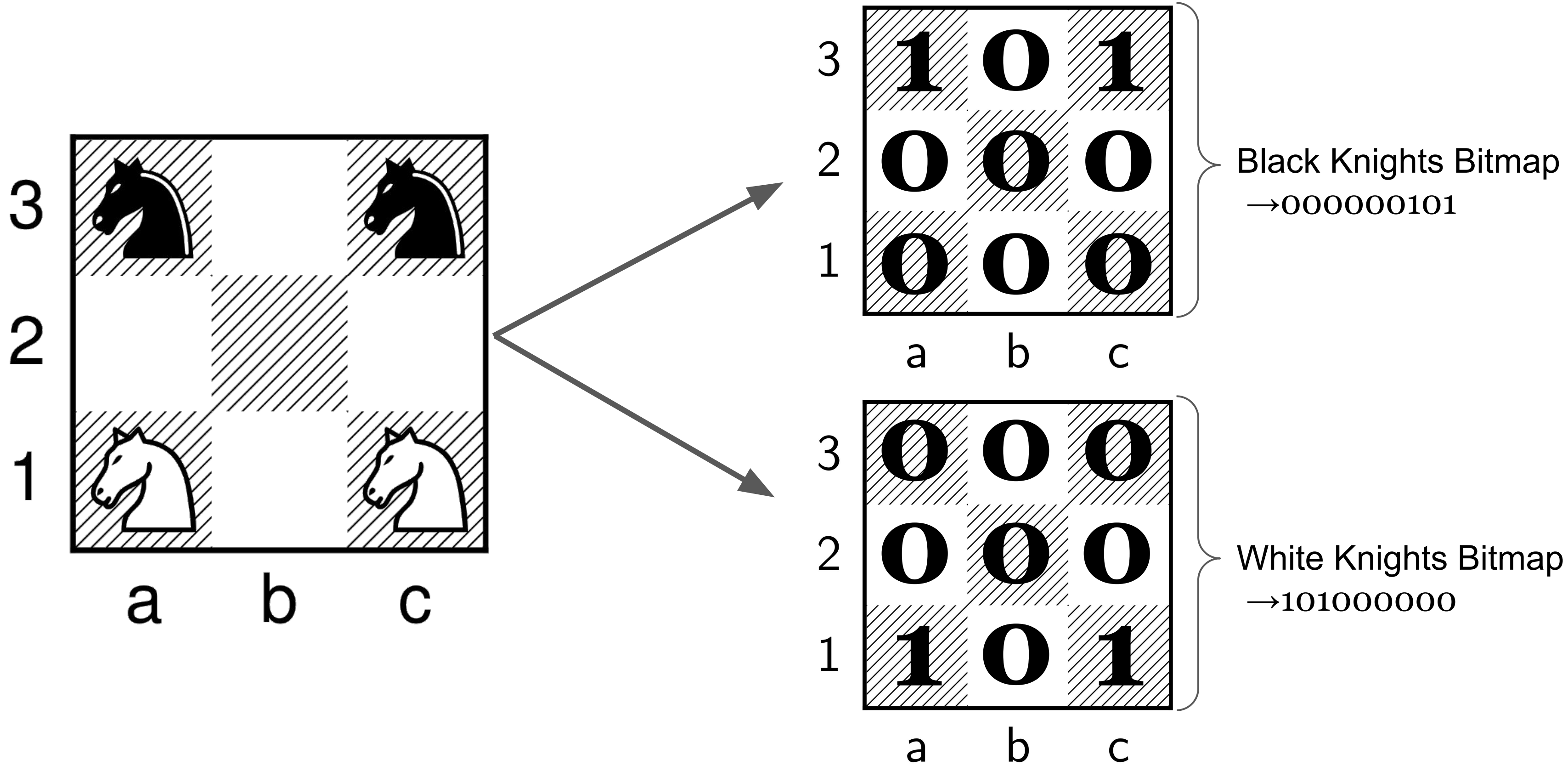
Mating Positions:



Domain Specific Approach: State Representation

General Purpose Planner Approach ⟶ Domain **specific** Approach

Domain Specific Approach: State Representation



Domain Specific Approach: Search

Implemented search types:

- Breadth First Search (**BFS**)

Domain Specific Approach: Search

Implemented search types:

- Breadth First Search (**BFS**)
- Best First Search instantiations:

Domain Specific Approach: Search

Implemented search types:

- Breadth First Search (**BFS**)
- Best First Search instantiations:
    - **Greedy Best First Search**

Domain Specific Approach: Search

Implemented search types:

- Breadth First Search (**BFS**)
- Best First Search instantiations:
  - **Greedy Best First Search**
  - **A*** Search

Domain Specific Approach: Search

Implemented search types:

- Breadth First Search (**BFS**)
- Best First Search instantiations:
  - **Greedy Best First Search**
  - **A*** Search
  - **weighted A*** Search

Domain Specific Approach: Search

Implemented search types:

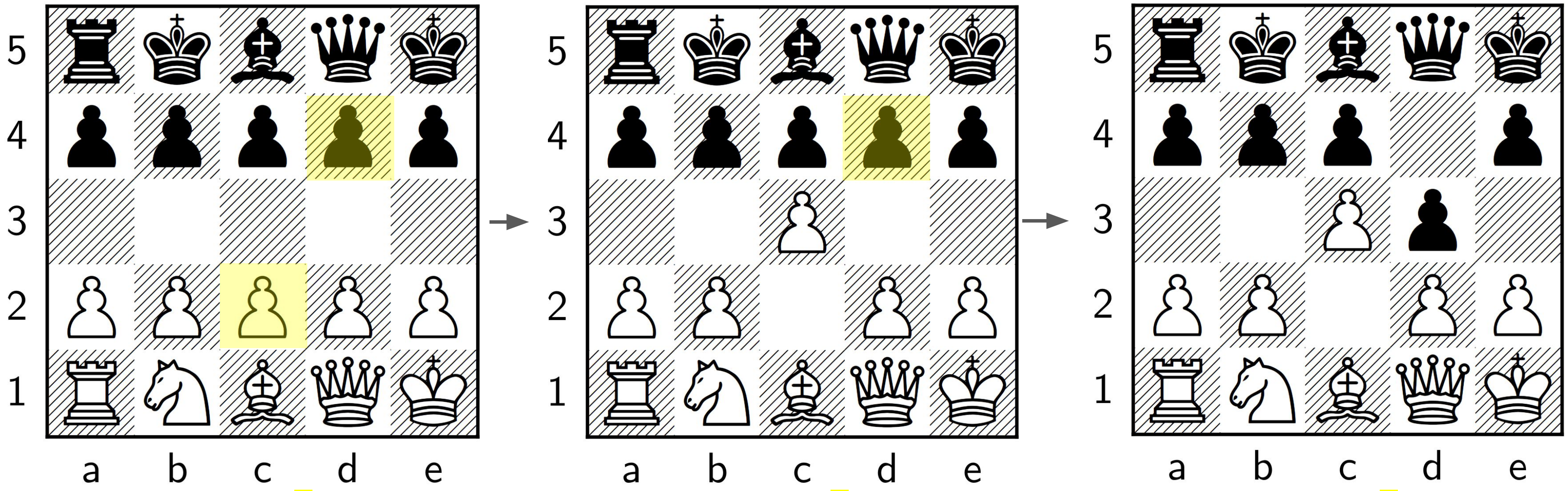
- Breadth First Search (**BFS**)
- Best First Search instantiations:
  - **Greedy Best First Search**
  - **A*** Search
  - **weighted A*** Search

→ **Heuristic**: We count the number of chess pieces that are out of order when compared to the goal state.

Domain Specific Approach: Heuristic

Compute the number of out of place figures:

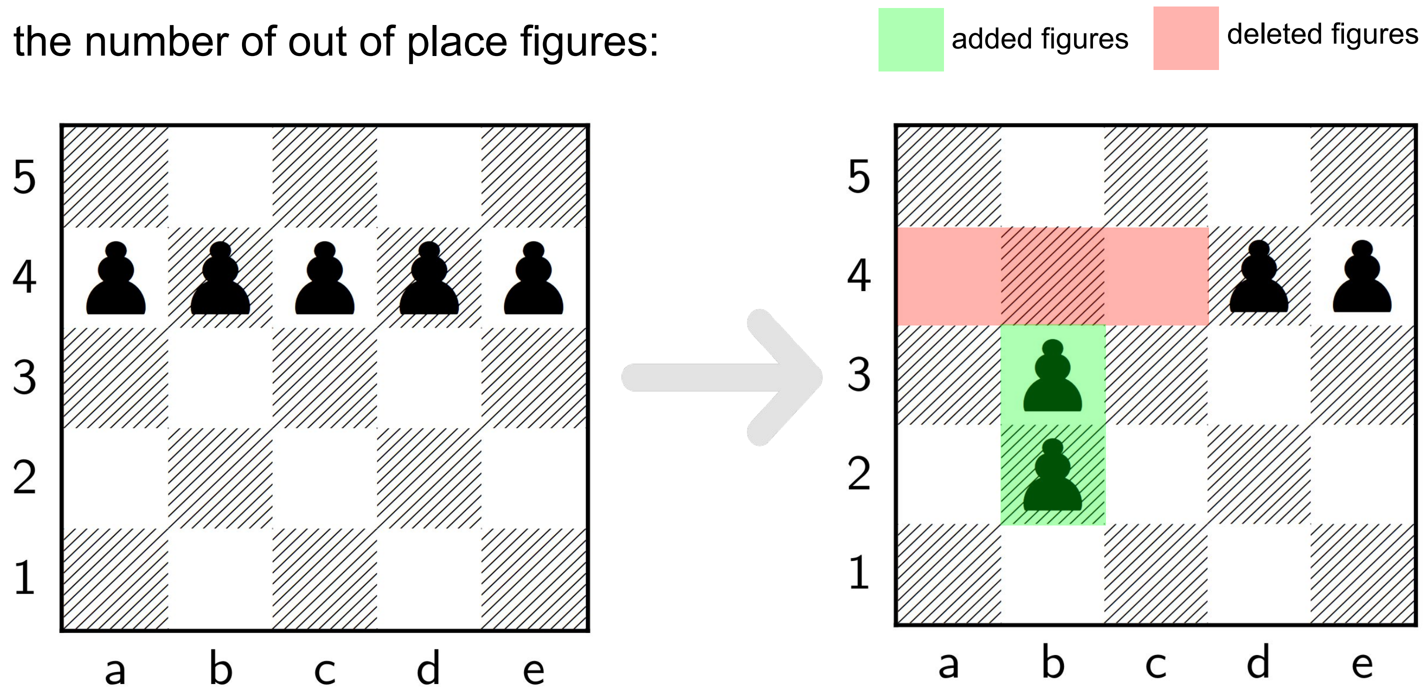
Initial State: $\mathbf{s_0}$ | Intermediate State: $\mathbf{s'}$ | Goal state: $\mathbf{s_*}$



$h(\mathbf{s_0})=2$ | $h(\mathbf{s'})=1$ | $h(\mathbf{s_*})=0$

Domain Specific Approach: Heuristic

Compute the number of out of place figures:



We cannot just compute the amount of flipped bits…

Domain Specific Approach: Heuristic

```
h = Long.bitCount(curr[i] & ~(curr[i] & goal[i]))
```

Current State:

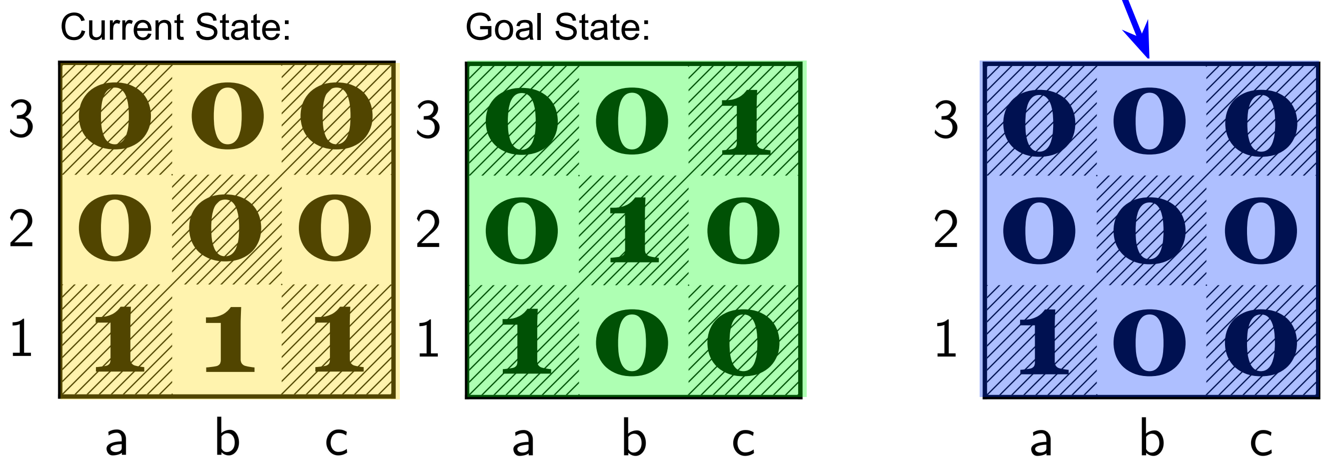
	a	b	c
3	0	0	0
2	0	0	0
1	1	1	1

Goal State:

	a	b	c
3	0	0	1
2	0	1	0
1	1	0	0

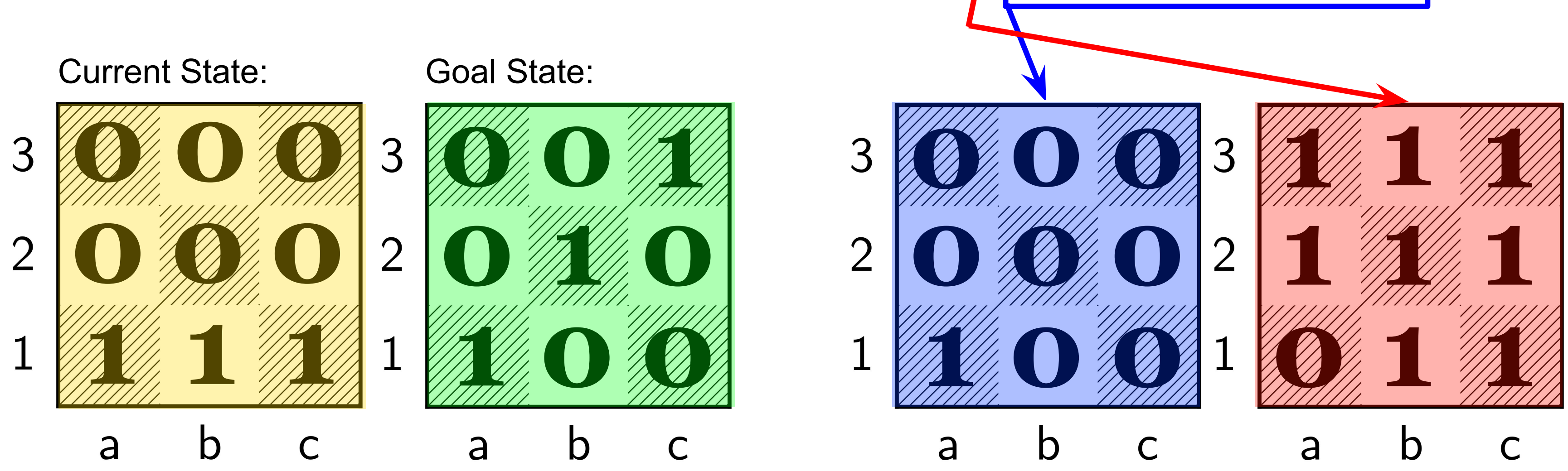
Domain Specific Approach: Heuristic

```
h = Long.bitCount(curr[i] & ~(curr[i] & goal[i]))
```



Domain Specific Approach: Heuristic

```
h = Long.bitCount(curr[i] & ~(curr[i] & goal[i]))
```



Domain Specific Approach: Heuristic

```
h = Long.bitCount(curr[i] & ~(curr[i] & goal[i]))
```

Current State:

	a	b	c
3	0	0	0
2	0	0	0
1	1	1	1

Goal State:

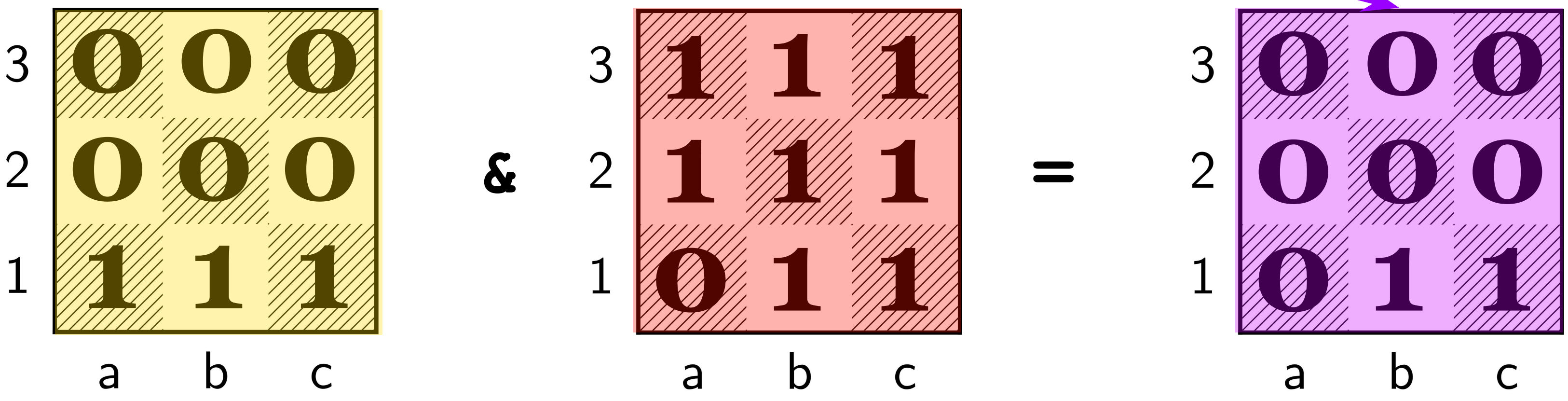
	a	b	c
3	0	0	1
2	0	1	0
1	1	0	0

	a	b	c
3	1	1	1
2	1	1	1
1	0	1	1

Domain Specific Approach: Heuristic

```
h = Long.bitCount(curr[i] & ~(curr[i] & goal[i]))
```

Current State:



h = 2

Domain Specific Approach: Properties of h()

- **Goal Aware?**

Domain Specific Approach: Properties of h()

✓ **Goal Aware**

Domain Specific Approach: Properties of h()

- ✓ **Goal Aware**
- ❏ **Safe?**

Domain Specific Approach: Properties of h()

- ✓ **Goal Aware**
- ❏ **Safe?**
  Two cases where the heuristic value is assigned to Infinity:

Domain Specific Approach: Properties of h()

- ✓ **Goal Aware**
- ❏ **Safe?**
  Two cases where the heuristic value is assigned to Infinity:
  1. **Number of pawns < number of missing figures** (of the color who's turn it is)

Domain Specific Approach: Properties of h()

- ✓ **Goal Aware**
- ❏ **Safe?**
  Two cases where the heuristic value is assigned to Infinity:
  1. **Number of pawns < number of missing figures** (of the color who’s turn it is)
  2. If **last pawn of a color is further ahead than in the goal state** (Pawns cannot move backwards)

Domain Specific Approach: Properties of h()

- ✓ **Goal Aware**
- ✓ **Safe**
- ❑ **Consistent?**

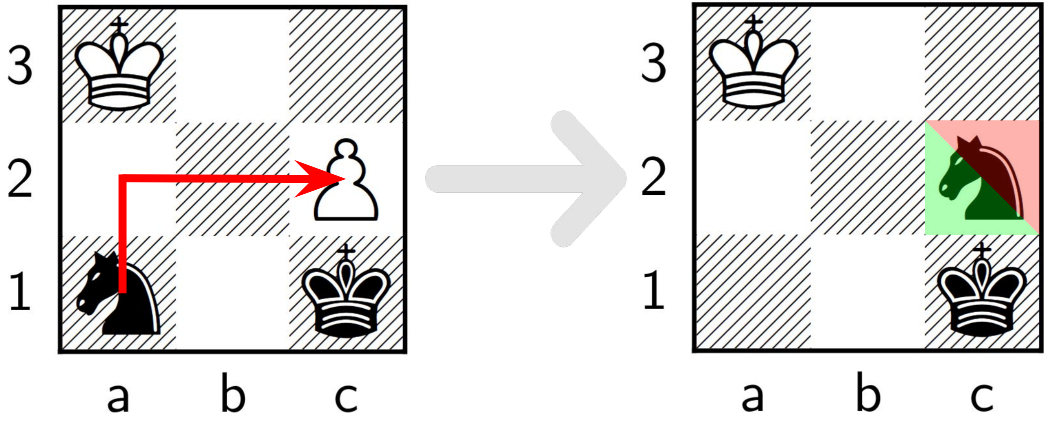
Domain Specific Approach: Properties of h()

- ✓ **Goal Aware**
- ✓ **Safe**
- ❏ **Consistent?**
  ⇒ h(s) value is not allowed to drop by more than one

Domain Specific Approach: Properties of h()

- ✓ **Goal Aware**
- ✓ **Safe**
- ❑ **Consistent?**
  ⇒ h(s) value is not allowed to drop by more than one

Consider this position:



Domain Specific Approach: Properties of h()

- ✓ **Goal Aware**
- ✓ **Safe**
- ❑ ~~**Consistent**~~

Domain Specific Approach: Properties of h()

- ✓ **Goal Aware**
- ✓ **Safe**
- ~~❑ **Consistent**~~
- ~~❑ **Admissible**~~

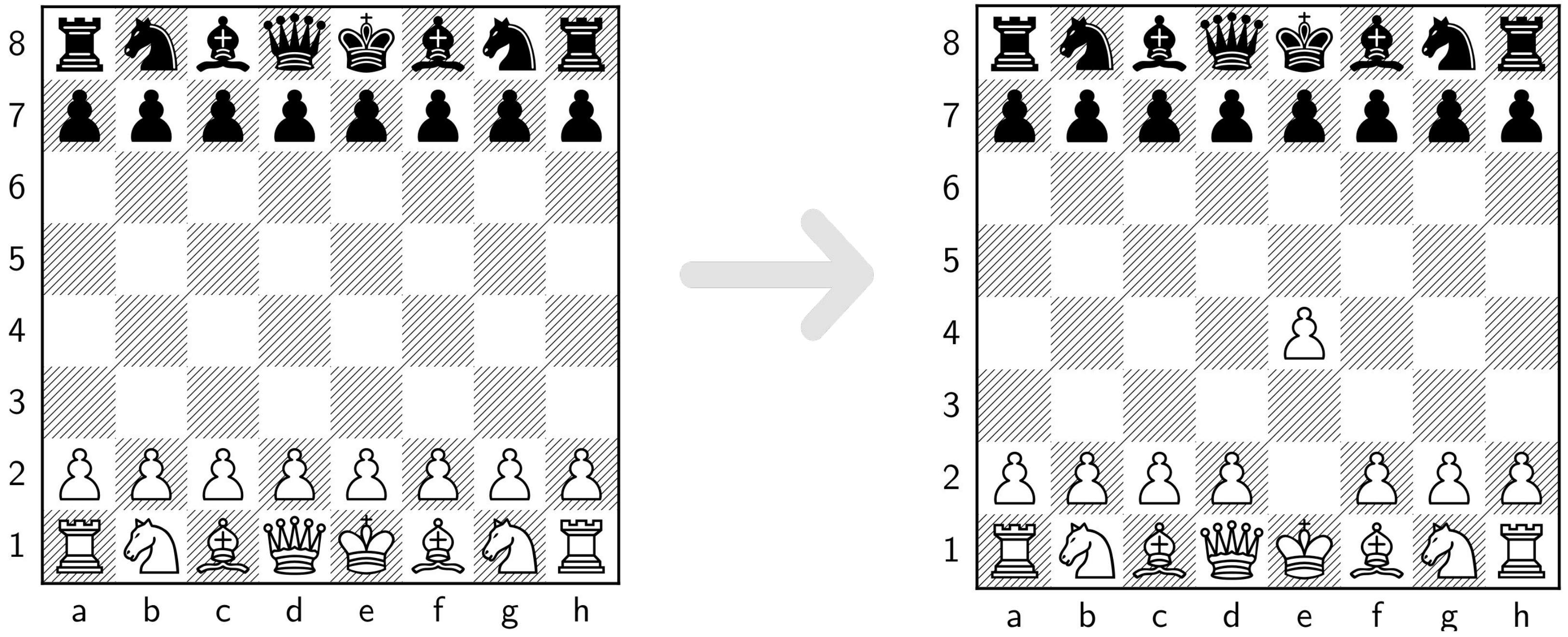
Experiments: Setup

Limits: 10 min., 8GB (memory)
Environment: Ubuntu, 16GB (memory), Intel Core i7 (with 4 x 2.7GHz)

Fast Downward with **GBFS** and the **FF-heuristic** (*--search "eager_greedy([ff])"*)

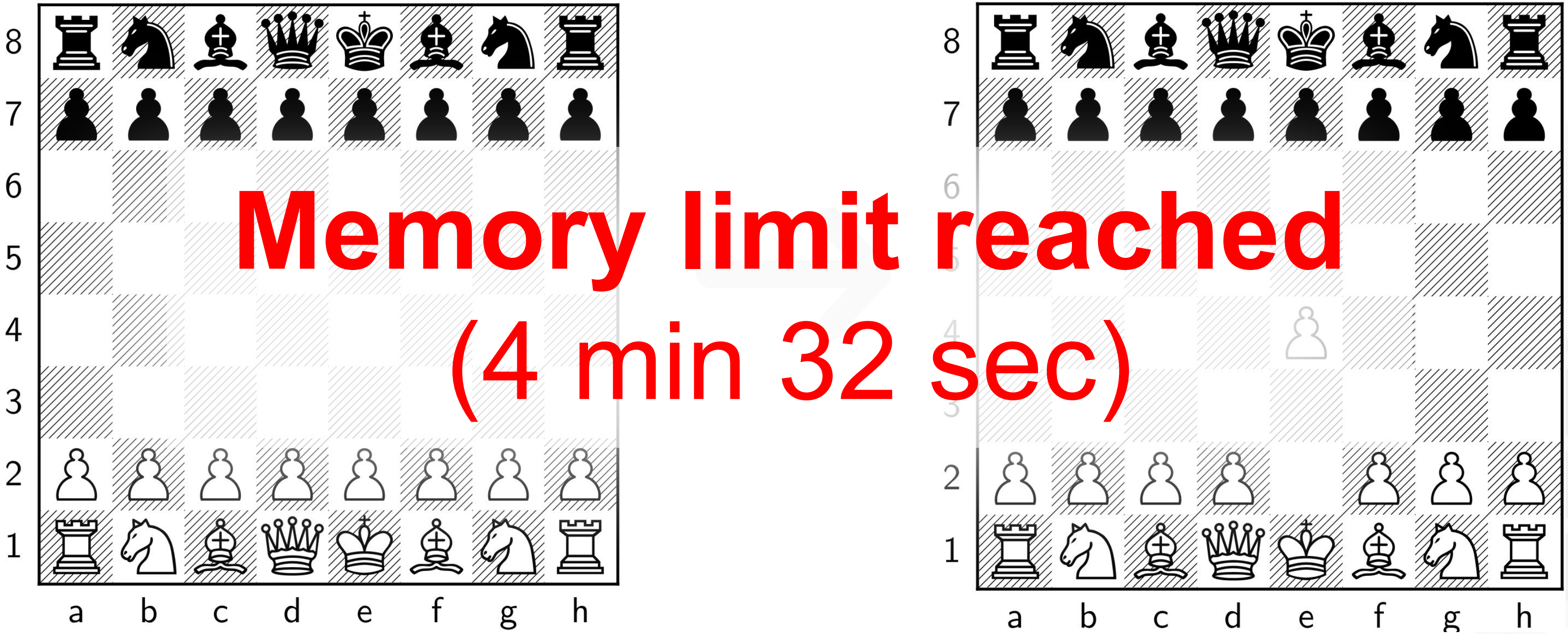
Experimental Results

1. **PDDL** Approach: First Experiment (8x8 board, 1 Pawn move)



Experimental Results

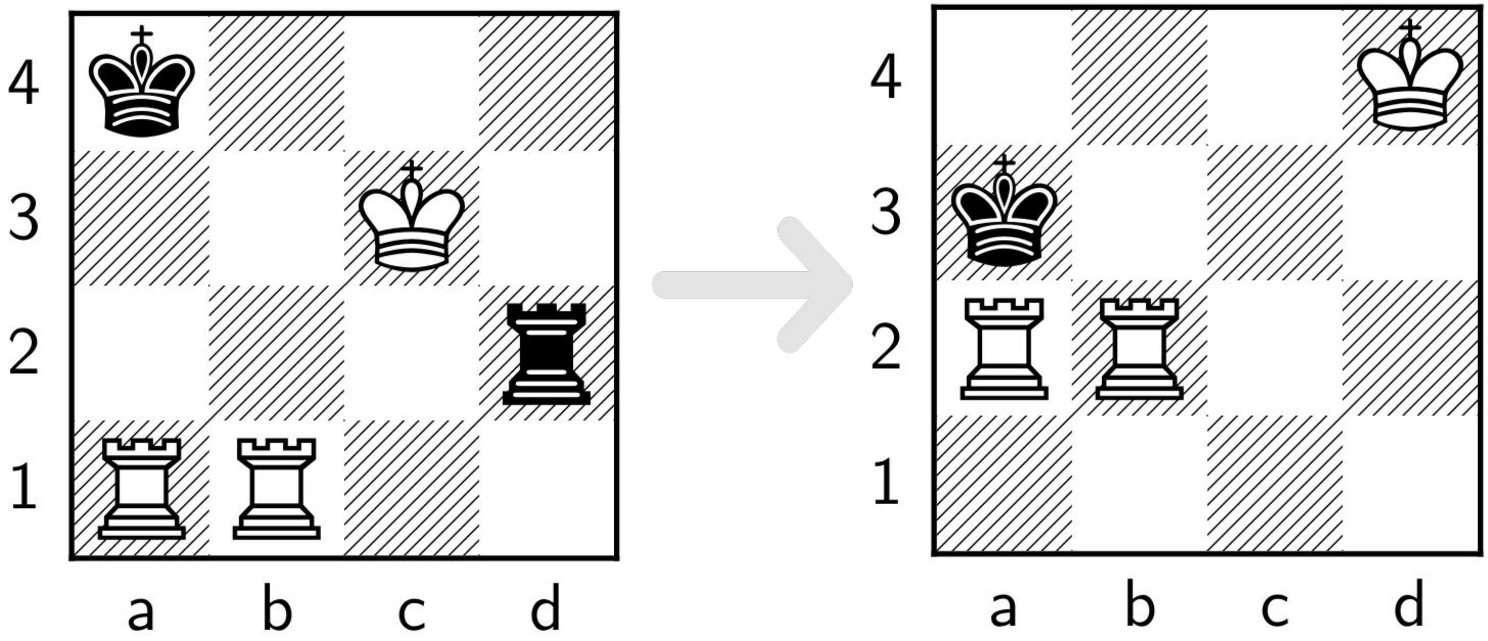
1. **PDDL** Approach: First Experiment (8x8 board, 1 Pawn move)



Memory limit reached
(4 min 32 sec)

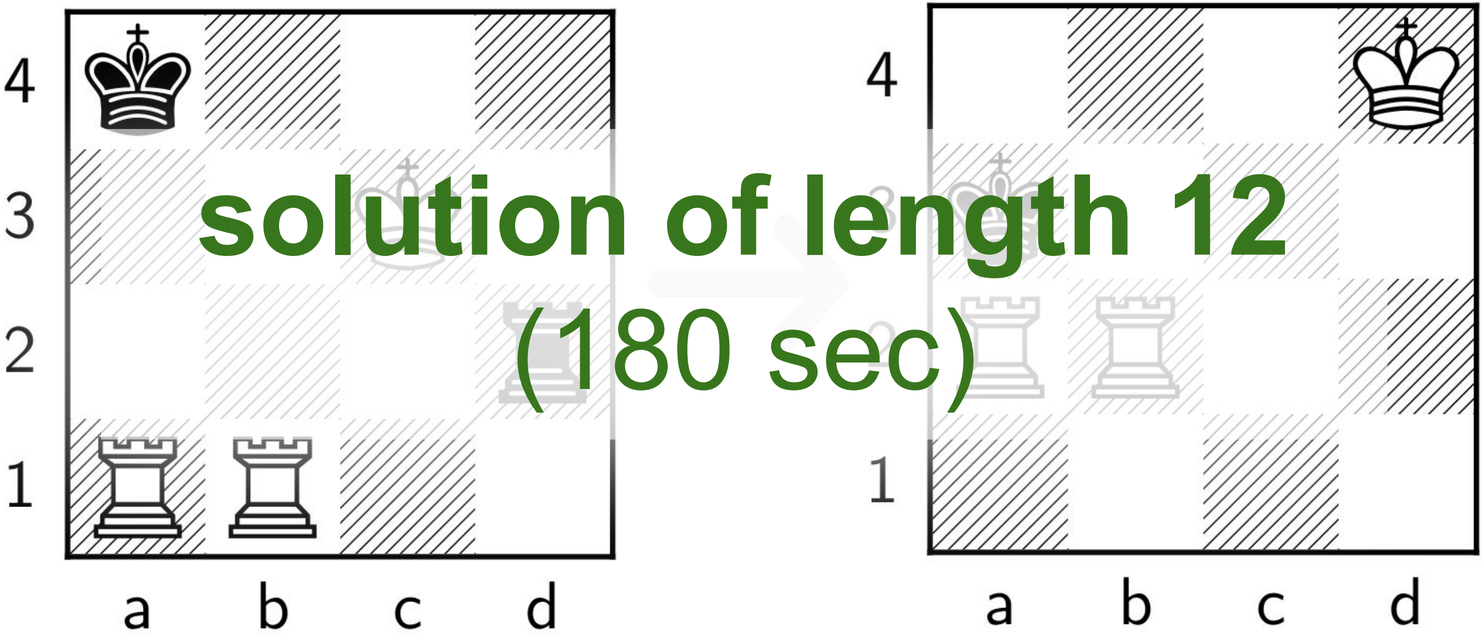
Experimental Results

2. **PDDL** Approach: Second Experiment (4x4 board)



Experimental Results

2. **PDDL** Approach: Second Experiment (4x4 board)



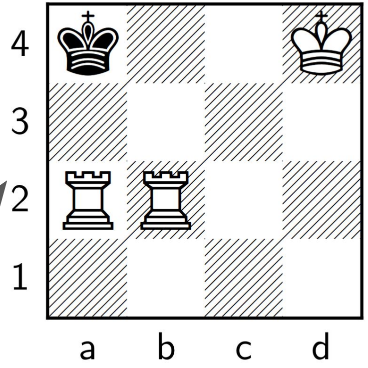
solution of length 12

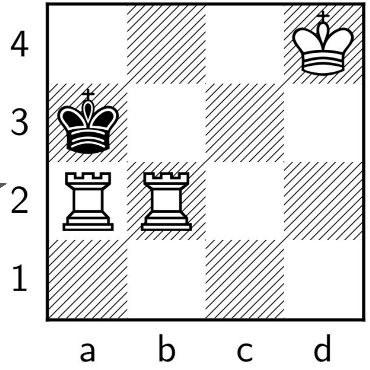
(180 sec)

Experimental Results

2. **PDDL** Approach: Second Experiment (4x4 board)

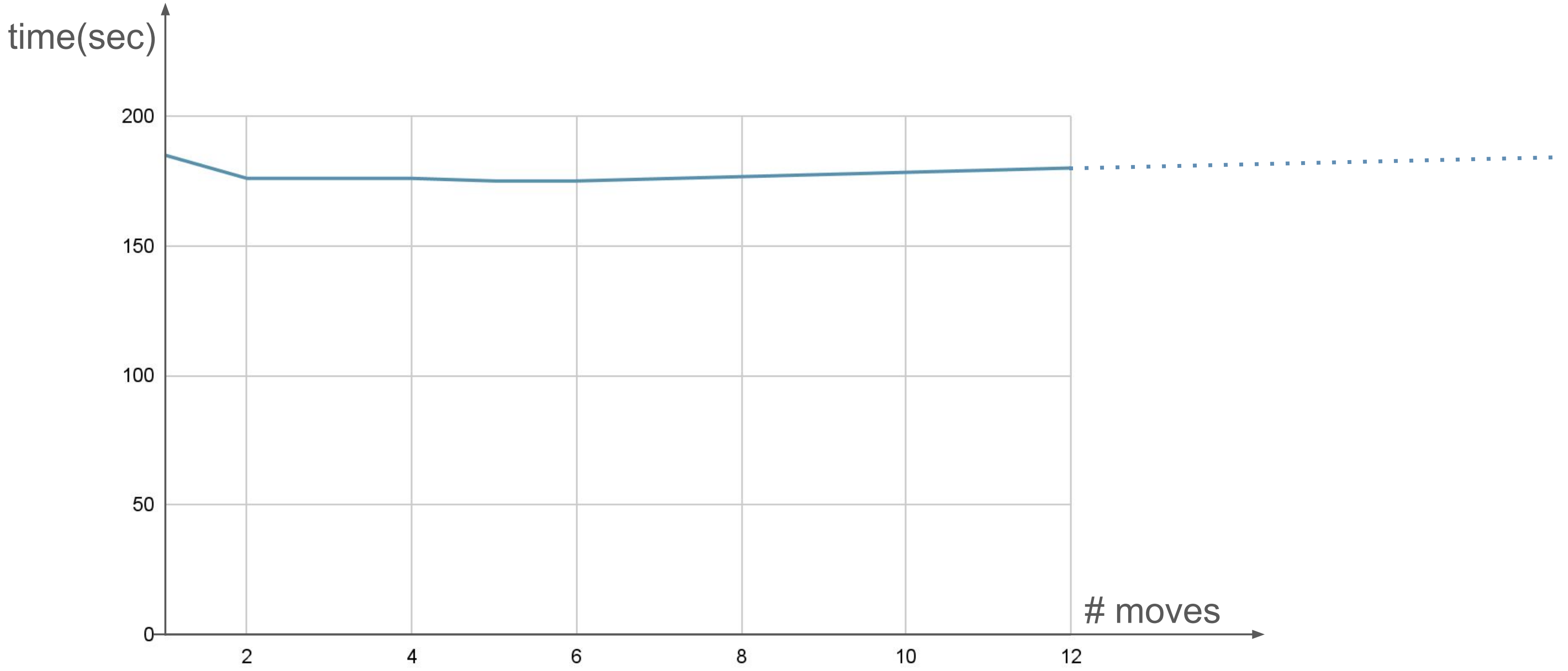
Row	Time	Moves	Goal State
1	185s	1	k1K1/4/r3/RR2
2	176s	2	k2K/4/r3/RR2
3	176s	3	3K/k3/r3/RR2
4	176s	4	3K/k3/rR1/R3
5	175s	5	k2K/4/rR1/R3
6	175s	6	k2K/4/RR1/4
7	180s	12	3K/k3/RR1/4





Experimental Results

2. **PDDL** Approach: Second Experiment (4x4 board)



Experimental Results

3-4. **PDDL** Approach: 5x5 and 6x6 board

Experimental Results

3-4. **PDDL** Approach: 5x5 and 6x6 board

Time limit reached

(10 min)

Experimental Results

5-7. **PDDL** Approach: 7x7 and 8x8 board

Experimental Results

5-7. **PDDL** Approach: 7x7 and 8x8 board

Memory limit reached

(81 sec & 108 sec)

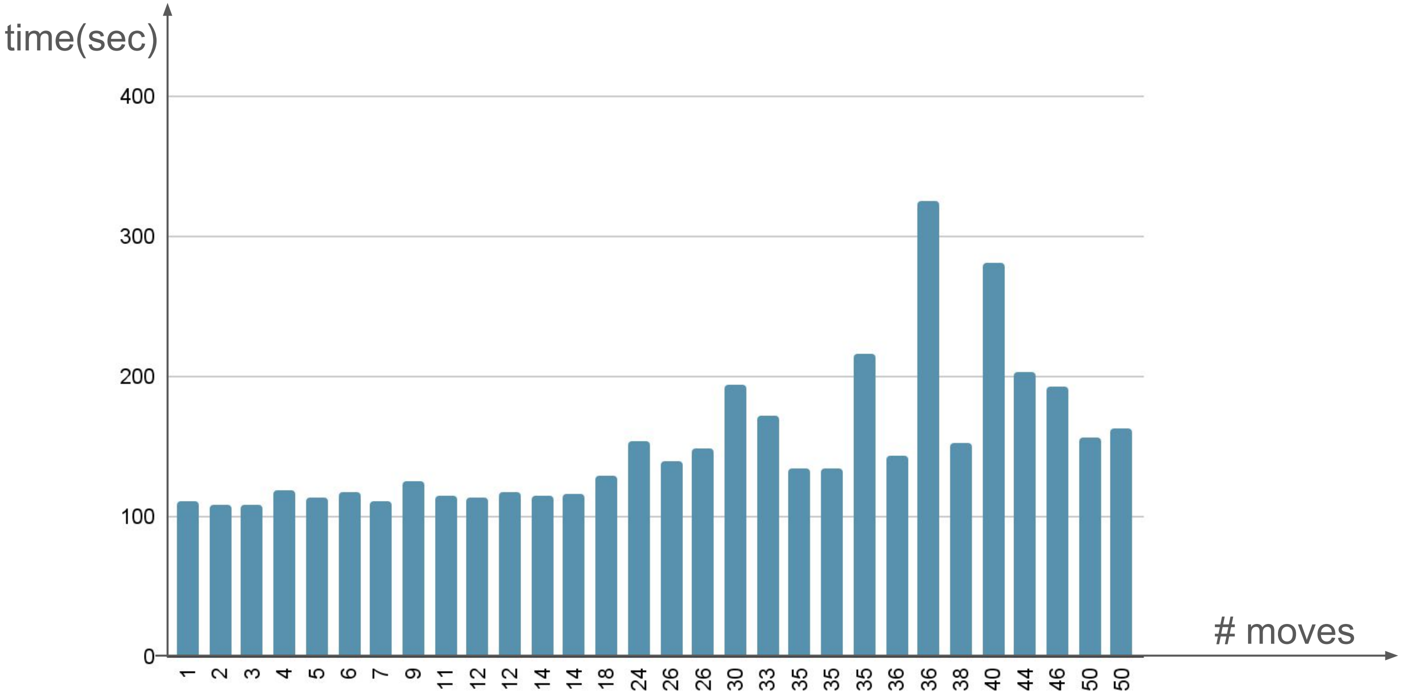
Experimental Results

8. **PDDL** Approach: **Without the Red-Zone/ without Kings** (8x8 board)

Row	Time	Moves	Goal State
1	111s	1	rnbq1bnr/pppppppp/8/8/4P3/8/PPPP1PPP/RNBQ1BNR
2	109s	2	rnbq1bnr/ppppp1pp/8/5p2/4P3/8/PPPP1PPP/RNBQ1BNR
3	109s	3	rnbq1bnr/ppppp1pp/8/5P2/8/8/PPPP1PPP/RNBQ1BNR
4	119s	4	rnbq1b1r/ppppp1pp/5n2/5P2/8/8/PPPP1PPP/RNBQ1BNR
5	114s	5	rnbq1b1r/ppppp1pp/5n2/5P2/8/3P4/PPP2PPP/RNBQ1BNR
6	117s	6	r1bq1b1r/ppppp1pp/2n2n2/5P2/8/3P4/PPP2PPP/RNBQ1BNR
7	111s	7	r1bq1b1r/ppppp1pp/2n2n2/5P2/8/3P4/PPPB1PPP/RN1Q1BNR
8	125s	9	r1bq1b1r/ppp1p1pp/2n2n2/3p1P2/8/3P4/PPPB1PPP/RN1Q1BNR
9	114s	12	r1bq1b1r/ppp1p1pp/2n2n2/3p1P2/8/3P1Q2/PPPB1PPP/RN3BNR
10	115s	11	r1bq1b1r/ppp1p1pp/5n2/3p1P2/3n4/3P1Q2/PPPB1PPP/RN3BNR
11	116s	14	r1bq1b1r/ppp1p1pp/5n2/3Q1P2/3n4/3P4/PPPB1PPP/RN3BNR
12	117s	12	r1b2b1r/ppp1p1pp/5n2/3q1P2/3n4/3P4/PPPB1PPP/RN3BNR
13	115s	14	r1b2b1r/ppp1p1pp/5n2/3q1P2/3n4/1P1P4/P1PB1PPP/RN3BNR
14	129s	18	r1b2b1r/ppp1p1pp/5n2/4qP2/3n4/1P1P4/P1PB1PPP/RN3BNR
15	139s	26	r1b2b1r/ppp1p1pp/5n2/4qP2/3n4/1PPP4/P2B1PPP/RN3BNR
16	154s	24	r1b2b1r/ppp1p1pp/5n2/1n2qP2/8/1PPP4/P2B1PPP/RN3BNR
17	149s	26	r1b2b1r/ppp1p1pp/5n2/1n2qP2/8/1PPP1N2/P2B1PPP/RN3B1R
18	135s	35	r1b2b1r/ppp1p1pp/5n2/4qP2/8/1PnP1N2/P2B1PPP/RN3B1R
19	134s	35	r1b2b1r/ppp1p1pp/5n2/4qP2/8/1PNP1N2/P2B1PPP/R4B1R
20	144s	36	r1b2b1r/ppp1p1pp/5n2/5P2/8/1PqP1N2/P2B1PPP/R4B1R
21	194s	30	r1b2b1r/ppp1p1pp/5n2/5P2/8/1PBP1N2/P4PPP/R4B1R
22	216s	35	r4b1r/ppp1p1pp/5n2/5b2/8/1PBP1N2/P4PPP/R4B1R
23	325s	36	3r1b1r/ppp1p1pp/5n2/5b2/8/1PBP1N2/P4PPP/R4B1R
24	172s	33	3r1b1r/ppp1p1pp/5B2/5b2/8/1P1P1N2/P4PPP/R4B1R
25	153s	38	3r1b1r/ppp1p2p/5p2/5b2/8/1P1P1N2/P4PPP/R4B1R
26	281s	40	3r1b1r/ppp1p2p/5p2/5b2/7N/1P1P4/P4PPP/R4B1R
27	193s	46	5b1r/ppp1p2p/5p2/5b2/7N/1P1r4/P4PPP/R4B1R
28	163s	50	5b1r/ppp1p2p/5p2/5b2/7N/1P1B4/P4PPP/R6R
29	203s	44	5b1r/ppp1p2p/5p2/8/7N/1P1b4/P4PPP/R1R5
30	156s	50	7r/ppp1p2p/5p1b/8/7N/1P1b4/P4PPP/R1R5
31	timeout	-	7r/ppp1p2p/5p1b/8/7N/1P1b4/P4PPP/R2R4

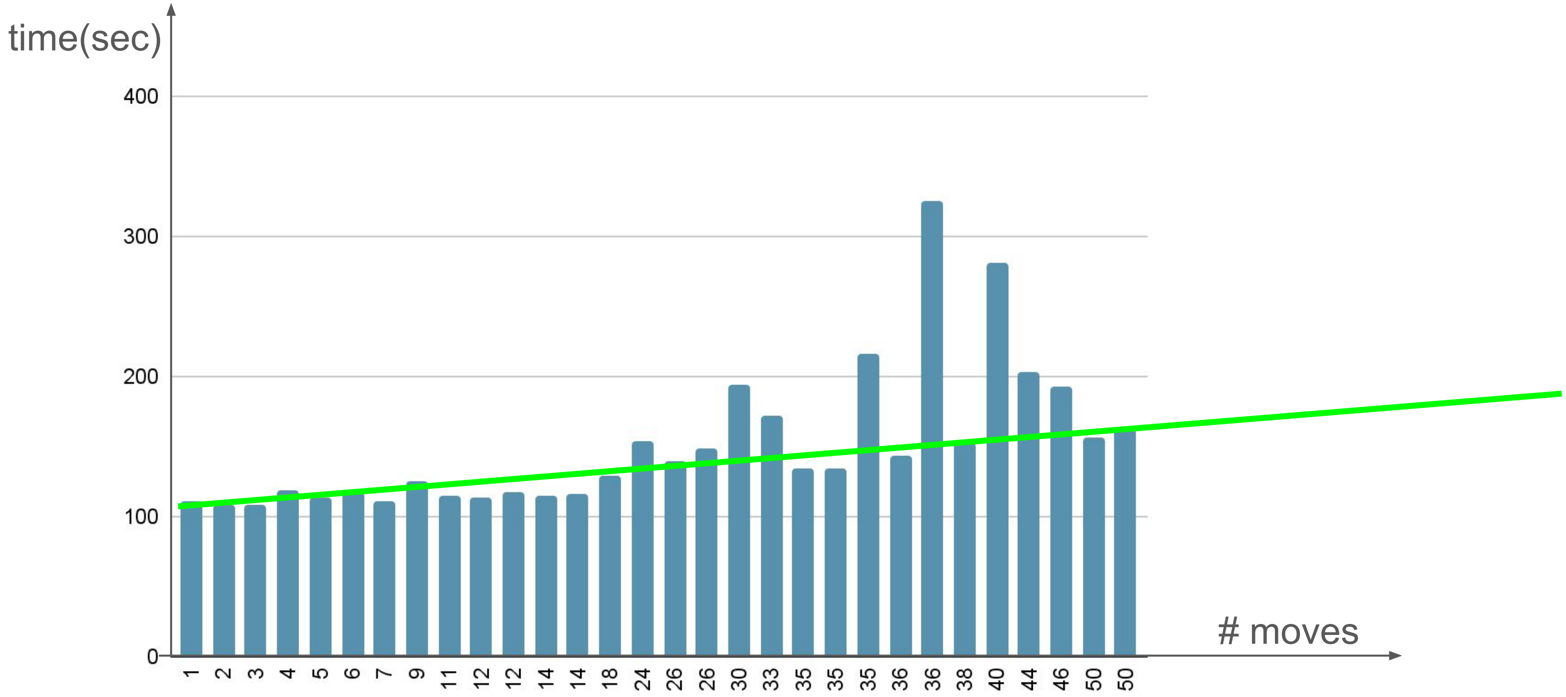
Experimental Results

8. **PDDL** Approach: **Without the Red-Zone/ without Kings** (8x8 board)



Experimental Results

8. **PDDL** Approach: **Without the Red-Zone/ without Kings** (8x8 board)



Experimental Results

General Purpose Planner Approach ⟶ Domain **specific** Approach

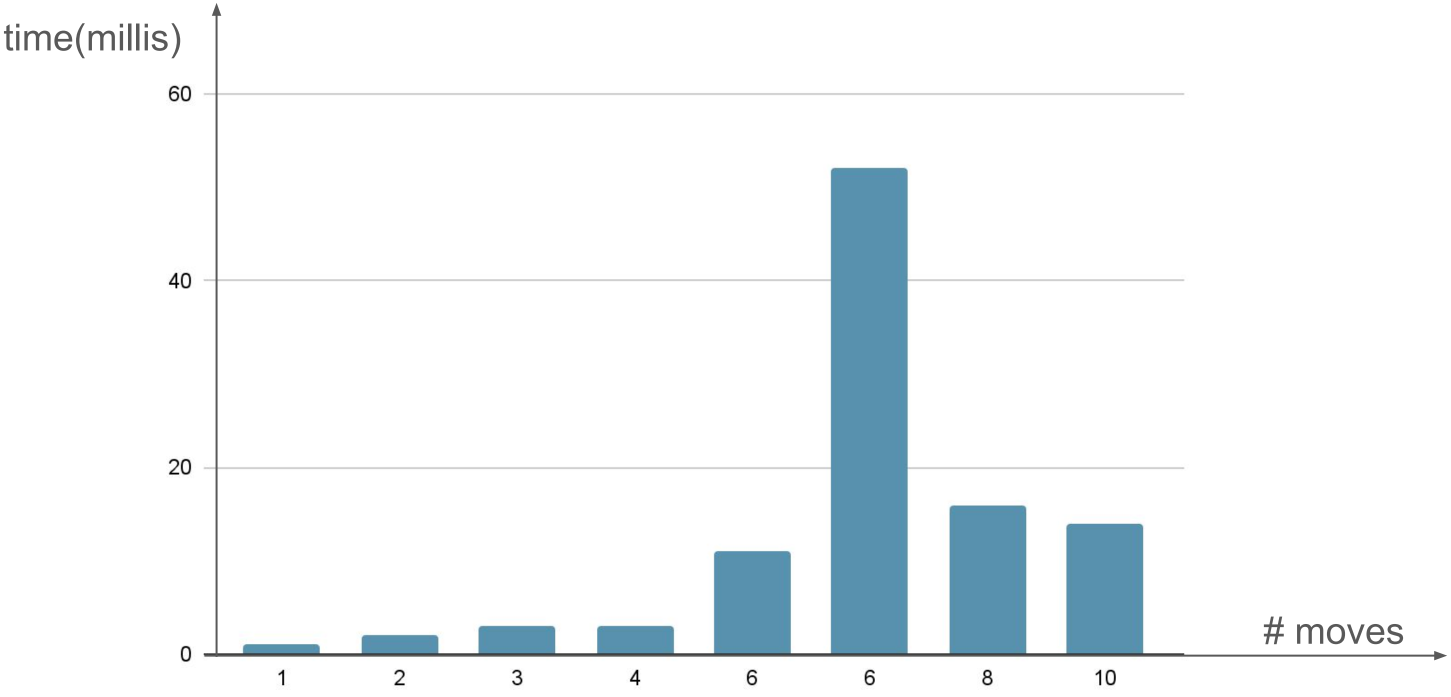
Experimental Results

9. **Domain Specific** Approach: using **GBFS** (8x8 board)

Row	Time	Moves	Nodes Expanded
1	1ms	1	20
2	2ms	2	40
3	3ms	3	121
4	3ms	4	171
5	11ms	6	1365
6	52ms	6	9746
7	14ms	10	1866
8	16ms	8	2267
9	-	-	-

Experimental Results

10. **Domain Specific** Approach: using **GBFS** (8x8 board)



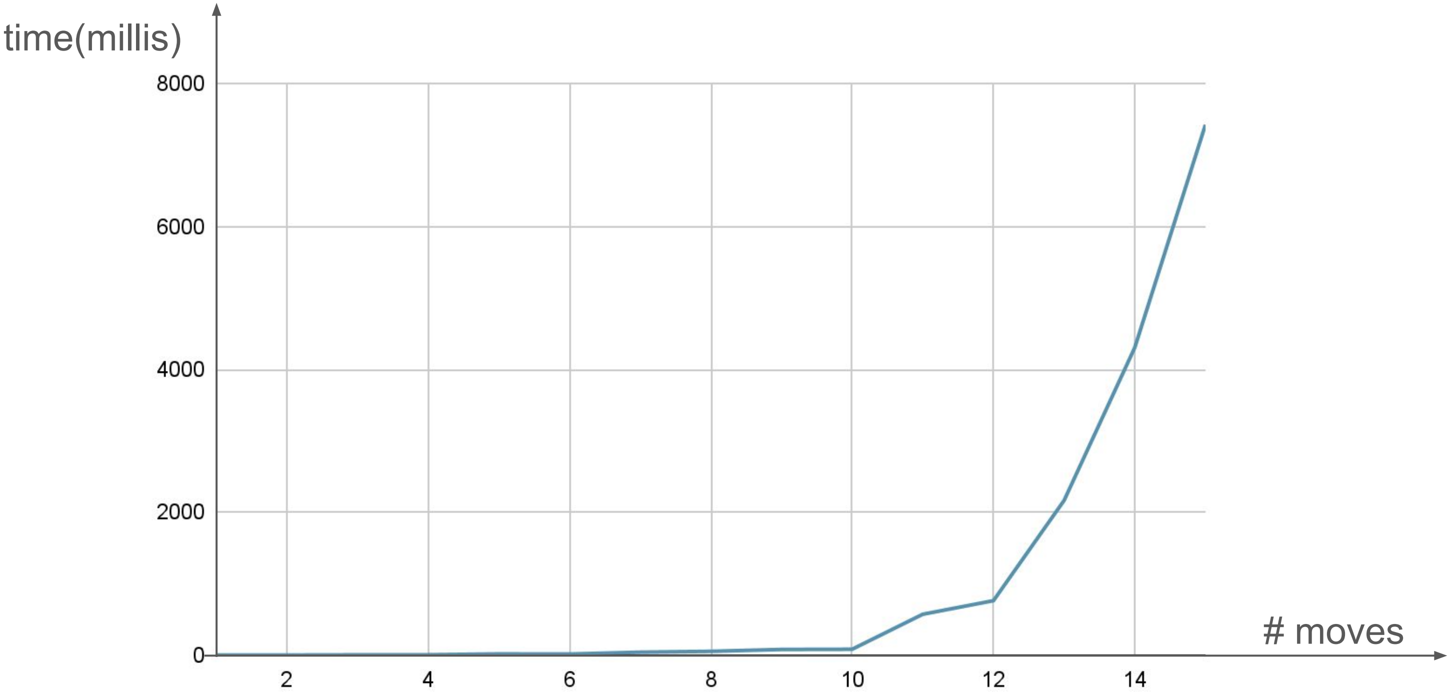
Experimental Results

10. **Domain Specific** Approach: using **A*** (8x8 board)

Row	Time	Moves	Nodes Expanded
1	2ms	1	20
2	2ms	2	40
3	5ms	3	221
4	6ms	4	260
5	18ms	5	2043
6	16ms	6	1458
7	42ms	7	6377
8	54ms	8	9767
9	80ms	9	17148
10	83ms	10	18018
11	573ms	11	234’446
12	763ms	12	304’938
13	2s 166ms	13	1’180’648
14	4s 305ms	14	2’253’356
15	7s 424ms	15	4672735
16	-	-	-

Experimental Results

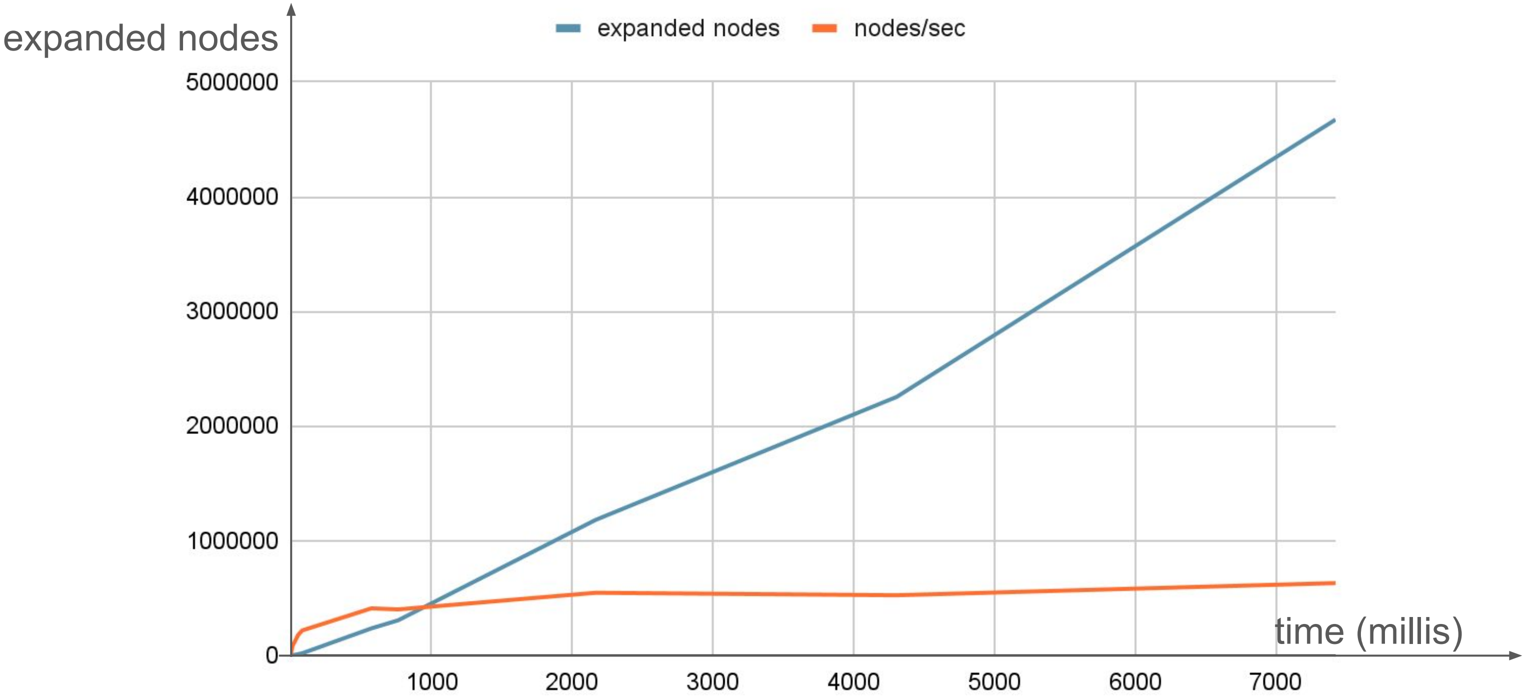
10. **Domain Specific** Approach: using **A*** (8x8 board)



Experimental Results

10. **Domain Specific** Approach: using **A*** (8x8 board)

~500’000 nodes/sec



Experimental Results

Conclusion

Experimental Results

Conclusion

- Red Zone = Bottleneck of the PDDL implementation
  ⇒ Valid moves involving the King are expensive
  ⇒ Domain independent approach not practical for boards bigger than 4x4

Experimental Results

Conclusion

- **Red Zone = Bottleneck** of the PDDL implementation
  - ⇒ Valid moves involving the King are expensive
  - ⇒ **Domain-<u>in</u>dependent approach not practical** for boards bigger than 4x4
- The results of the PDDL implementation reflect our expectations
  - ⇒ PDDL is used for domain-independent problems
    - ⇒ **Domain-<u>in</u>dependent approach can not be optimized as much as a domain-dependent approach**.

Experimental Results

Conclusion

- **Red Zone = Bottleneck** of the PDDL implementation
  ⇒ Valid moves involving the King are expensive
  ⇒ **Domain-<u>in</u>dependent approach not practical** for boards bigger than 4x4
- The results of the PDDL implementation reflect our expectations
  ⇒ PDDL is used for domain-independent problems
  ⇒ **Domain-<u>in</u>dependent approach can not be optimized as much as a domain-dependent approach**.
- **Memory bottleneck** in the domain-dependent approach

Experimental Results

Conclusion

- **Red Zone = Bottleneck** of the PDDL implementation
  ⇒ Valid moves involving the King are expensive
  ⇒ **Domain-<u>in</u>dependent approach not practical** for boards bigger than 4x4
- The results of the PDDL implementation reflect our expectations
  ⇒ PDDL is used for domain-independent problems
  ⇒ **Domain-<u>in</u>dependent approach can not be optimized as much as a domain-dependent approach**.
- **Memory bottleneck** in the domain-dependent approach
- **A*** is not as efficient as GBFS but it **can find solutions where GBFS fails**.

University of Basel

Natural Science Faculty

Department of Mathematics and Computer Science

Artificial Intelligence
Research Group
ai.dmi.unibas.ch

Questions?

ken.rotaris@stud.unibas.ch

Backup: “**Locking**” Mechanism

- Fluid predicate **‘valid_position’** *(true at the start)*
- **After every action**: (not(**valid_position**)) → **set to false**
- Only the unlocking action is applicable now (all other actions are “locked”).

```
(:action check_if_last_move_was_valid ;check same colored king
    :parameters (?figure - figure ?from_file ?from_rank - location)
    :precondition (and
                        (not(valid_position))
                        (at ?figure ?from_file ?from_rank)
                        (myturn ?figure)
                        (opposite_king_not_in_check ?figure)
                  )
    :effect (and
                  (valid_position)
            )
)
```

Backup: PDDL Domain File **Statistics**

Name	**Amount**
#types	8
#actions	18
total #predicates	61
#static predicates	19
#fluent predicates	9
total #derived predicates	33
#recursive derived predicates	6
#non-recursive derived predicates	27

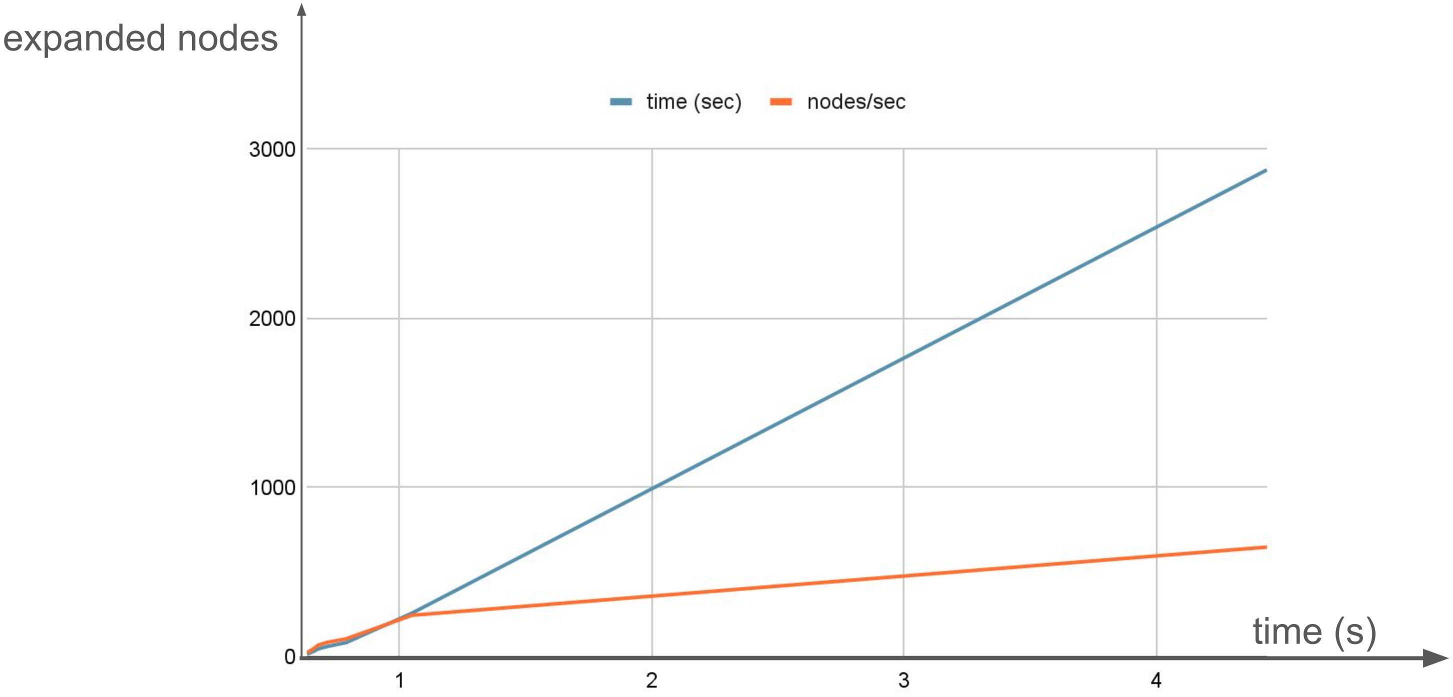
Backup: Experimental **PDDL** Results

2. **PDDL** Approach: Second Experiment (4x4 board)

Row	Time	Moves	Goal State	t	States Generated
1	185s	1	k1K1/4/r3/RR2	0.634782s	15
2	176s	2	k2K/4/r3/RR2	0.663288s	33
3	176s	3	3K/k3/r3/RR2	0.679977s	47
4	176s	4	3K/k3/rR1/R3	0.715359s	61
5	175s	5	k2K/4/rR1/R3	0.790793s	84
6	175s	6	k2K/4/RR1/4	1.05123s	258
7	180s	12	3K/k3/RR1/4	4.4395s	2879

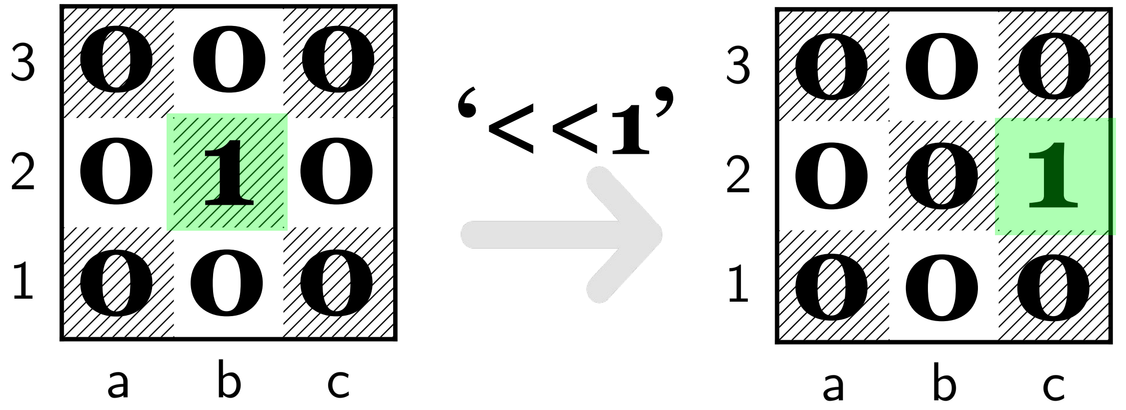
Backup: Experimental **PDDL** Results

2. **PDDL** Approach: Second Experiment (4x4 board)



Backup: **Bitmap Movements**

Movement to the right (shift by 1 to the left):



Backup: **Bitmap Movements**

Diagonal Movement (shift by 4 to the left):