Sokoban: Enhancing general single-agent search methods using domain knowledge

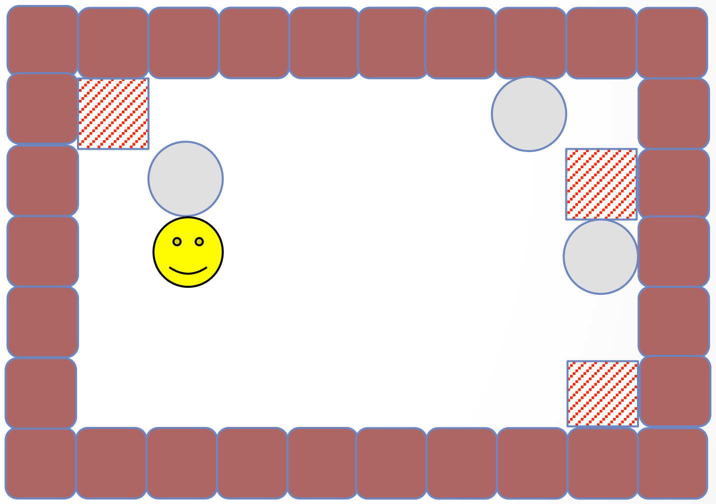
Andreas Junghanns – Jonathan Schaeffer

presented by Pascal Düblin

What is Sokoban?

- Computer game
- Goal:
  - Push stones with the man (smiley) on goal squares (red shaded squares)
- Rules:
  - No pull, only push moves
  - If a stone cannot be pushed and isn't on a goal square, the game is lost

- Example:



Sokobans search-space

Property	Specifics	24-Puzzle	Rubik's Cube	Sokoban
Branching factor	Average Range	2.37 1-3	13.35 12-15	12 0-136
Solution length	Average Range	100+ 1-unkown	18 1-20	260 97-674
Search-space size	Upper bound	10^{35}	10^{19}	10^{98}
Calculation of Lower bound	Full Incremental	$O(n)$ $O(1)$	$O(n)$ $O(1)$	$O(n^3)$ $O(n^2)$
Underlying graph		Undirected	Undirected	Directed

Application-dependent techniques

- Sokoban-solver: Rolling Stone
- Node limit: 20 million nodes
- Basis for Rolling-Stone: IDA*
- 3 years of work
- 90 Sokoban test problems

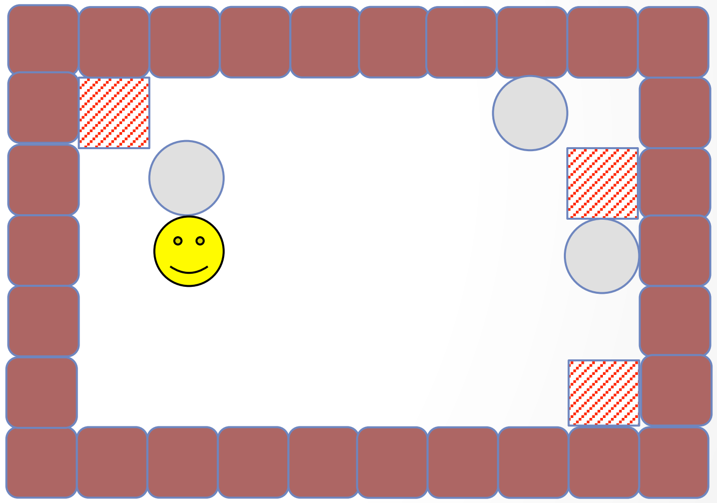
Basic Implementation: IDA*

- IDA* = Iterative deepening A*
- Similar approach like iterative deepening depth first search
- F-value of A* is limited
- This limit surges with each iteration of the depth first search

Simple lower Bound

- Heuristic for IDA*
- Manhattan Distance to nearest goal square
- Sum of all distances
- In example: 5
- Problems solved: 0

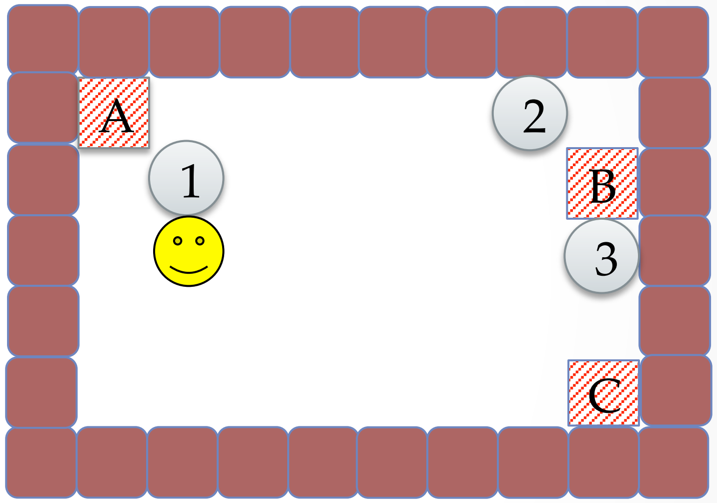
- Example:



Minimum matching lower bound (R0) (1)

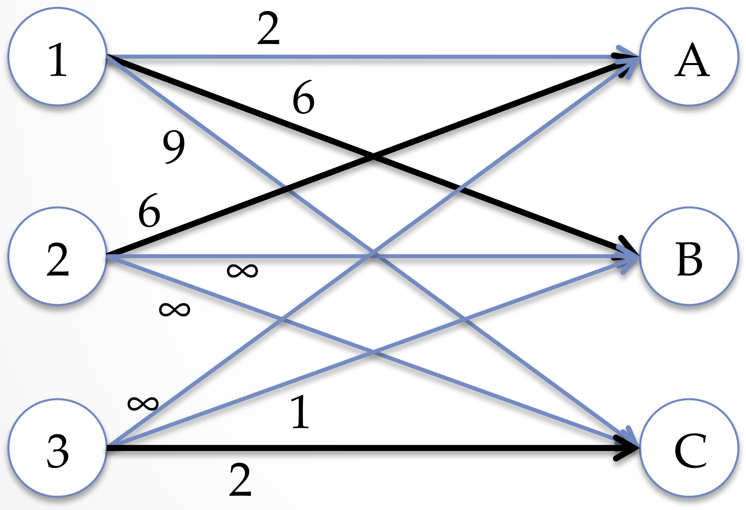
- Improved heuristic for IDA*
- Each goal square can only be taken from one stone
- Manhattan Distance to nearest and reachable goal squares

- Example:

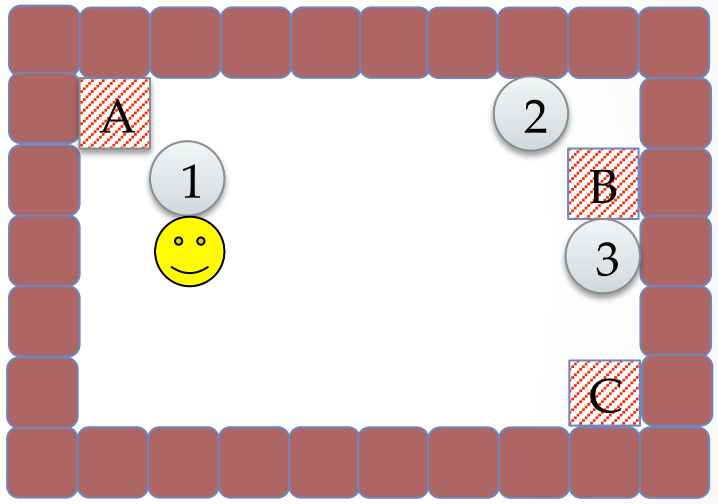


Minimum matching lower bound (R0) (2)

- Algorithm realize that the goal square of a stone is not always the nearest

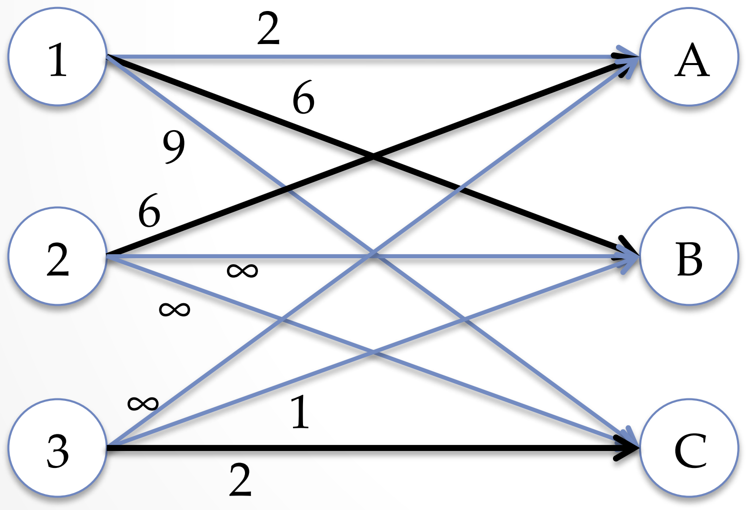


- Example:

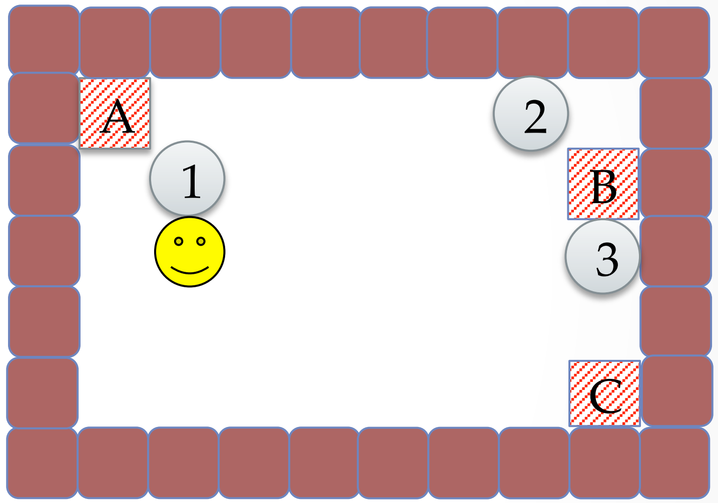


Minimum matching lower bound (R0) (3)

- Algorithm realize that the goal square of a stone is not always the nearest



- Example:

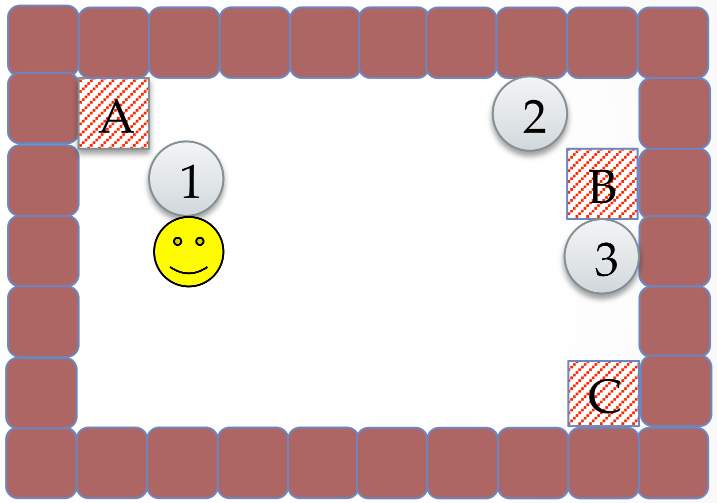


Goals:	A	B	C
1	2	**6**	9
2	**6**	-	-
3	-	1	**2**

Minimum matching lower bound (R0) (4)

- Heuristic value: 14
- Better heuristic value, h closer at h*
- The result is an enormous reduction of the search space
- Problems solved: 0

- Example:



Goals:	A	B	C
1	2	**6**	9
2	**6**	-	-
3	-	1	**2**

Transposition table (R1)

- All visited states stored in the transposition table
- Avoid visiting duplicated states/nodes
- Duplicate elimination before expanding next node
- Similar to close list
- Problems solved: 5

Move ordering (R2)

- Order how nodes will be expanded

- Actions (Moves) are sorted with the most promising actions first.

- Sorting criteria:

1. Move the same stone
2. Move that mimimize the lower bound (optimal move)
3. Stone first that is nearest of its goal square
4. Non optimal moves sorted by the same criteria as above

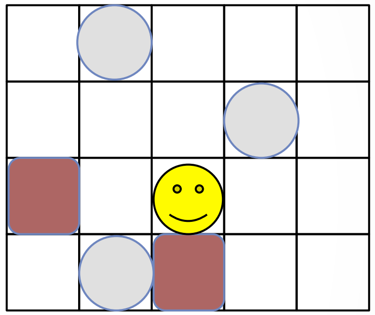
- Problems solved: 4

Deadlock table (R3)

- 4 x 5 region
- Find all arangements of stones, wall squares and the man
- Store all deadlocks
- During IDA* search: check state against the deadlock table

- Problems solved: 5

- Example:



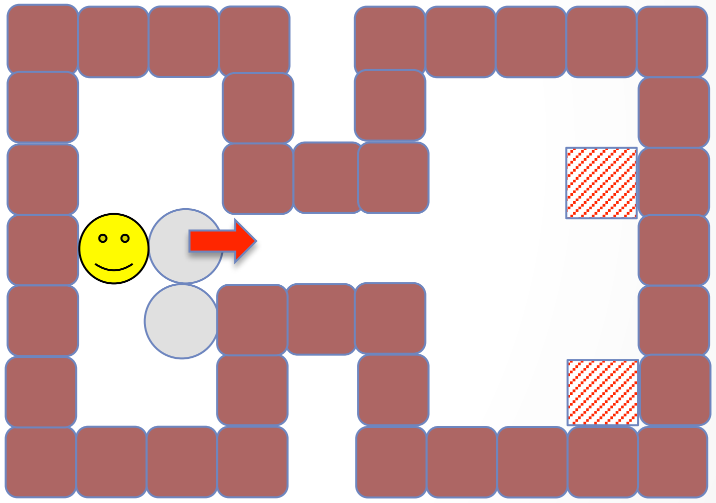
Tunnel macros (R4)

- Macros:

Combine a group of moves

- All tunnel moves are made all at once

- Problems solved: 6



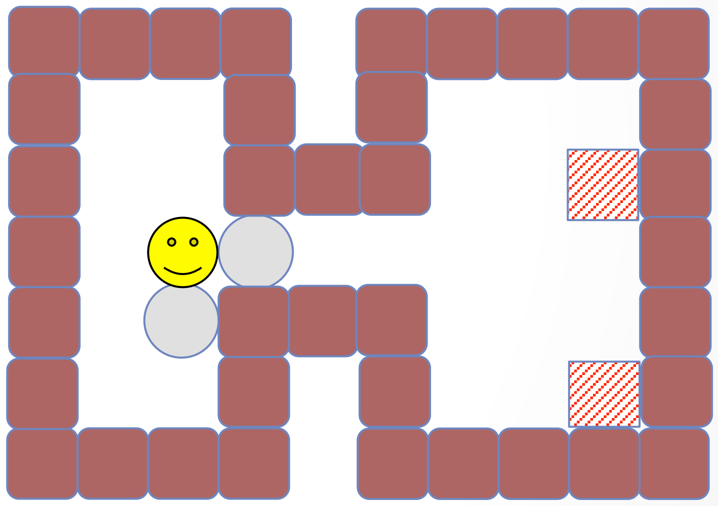
Tunnel macros (R4)

- Macros:

Combine a group of moves

- All tunnel moves are made all at once

- Problems solved: 6



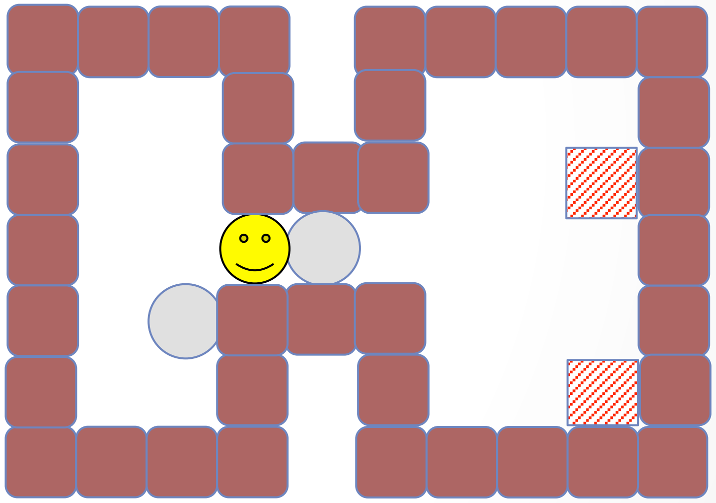
Tunnel macros (R4)

- Macros:

Combine a group of moves

- All tunnel moves are made all at once

- Problems solved: 6



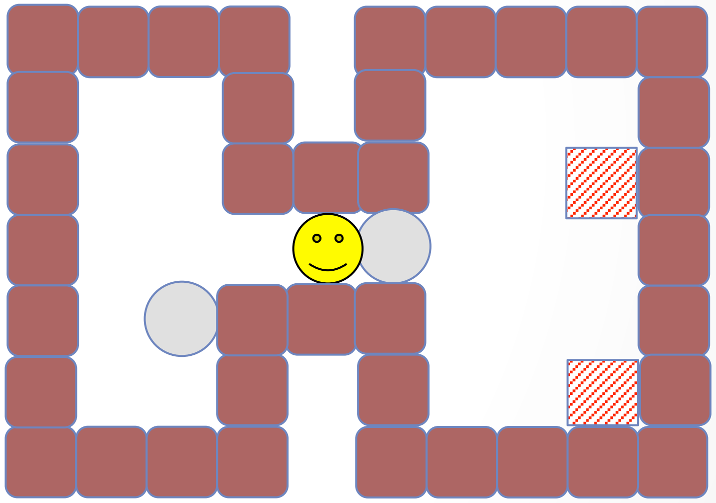
Tunnel macros (R4)

- Macros:

Combine a group of moves

- All tunnel moves are made all at once

- Problems solved: 6



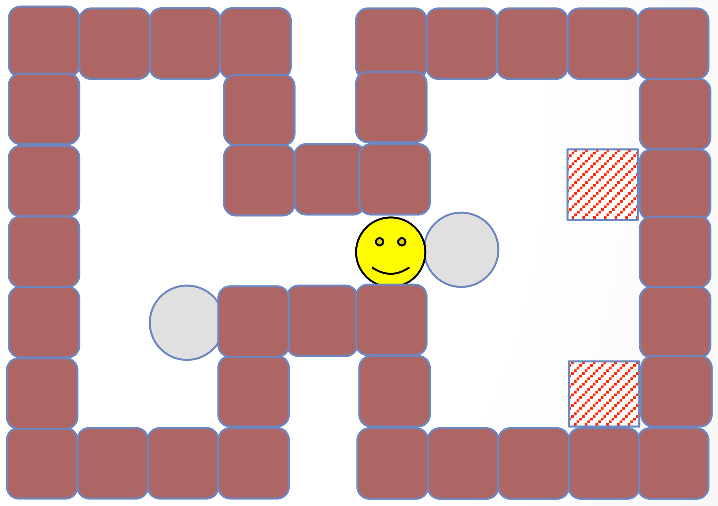
Tunnel macros (R4)

- Macros:

Combine a group of moves

- All tunnel moves are made all at once

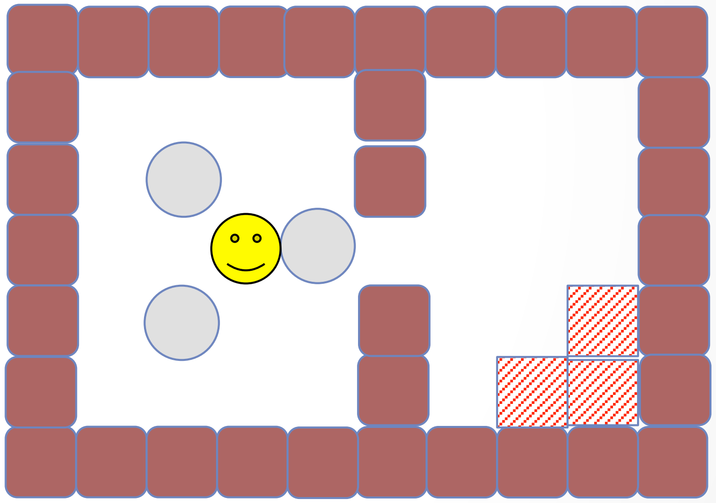
- Problems solved: 6



Goal macros (R5) (1)

- In Sokoban goal squares are often grouped in a gaol area
- In this case Sokoban can be split in two subproblems:

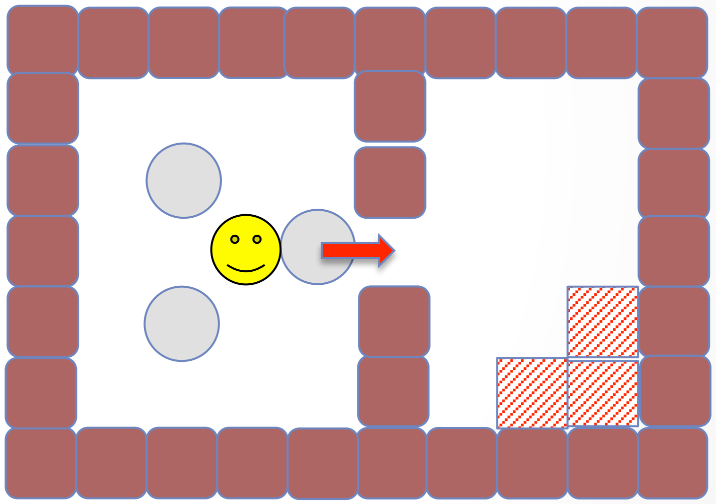
- Example:



Goal macros (R5) (1)

- In Sokoban goal squares are often grouped in a gaol area
- In this case Sokoban can be split in two subproblems:
  - Push a stone on an entrance square

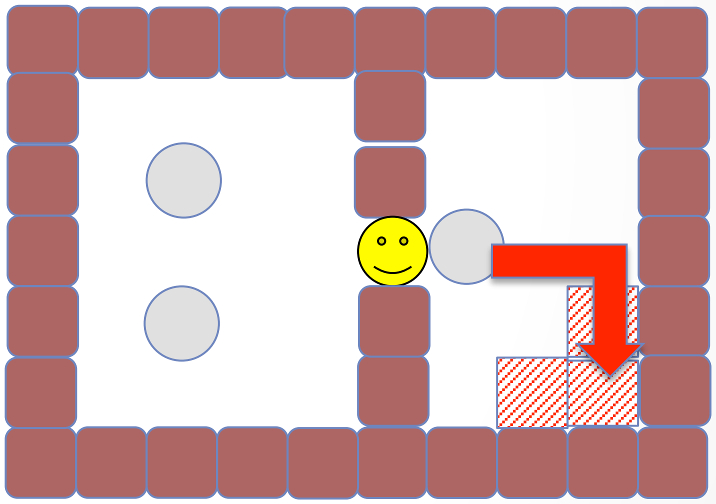
- Example:



Goal macros (R5) (1)

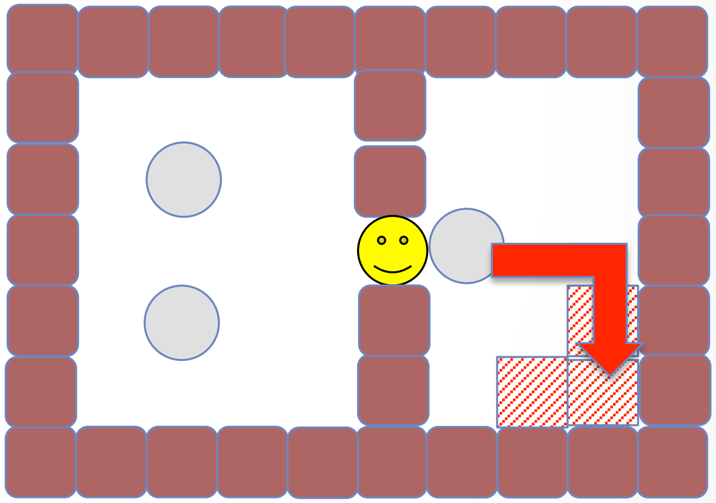
- In Sokoban goal squares are often grouped in a gaol area
- In this case Sokoban can be split in two subproblems:
    - Push a stone on an entrance square
    - Push a the stone on a goal square

- Example:



Goal macros (R5) (2)

- Precomputed for each problem and room, in which specified order the man has to push the stones on their goal squares.
- Example:

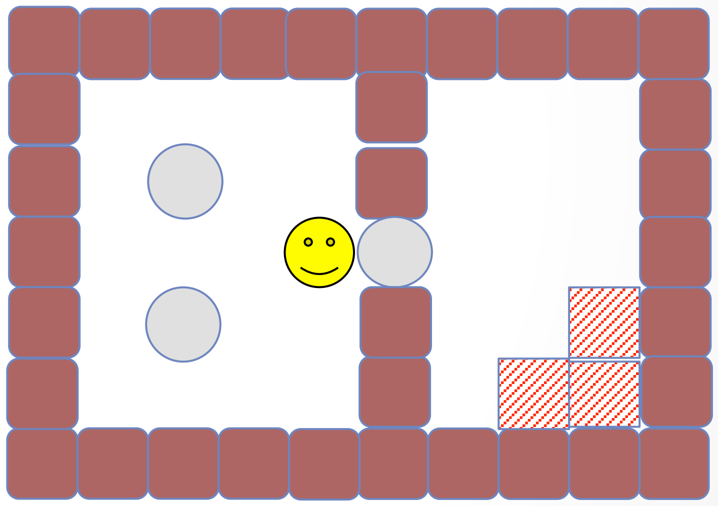


Goal macros (R5) (3)

- If a stone is on an entrance square, the goal macro will be executed

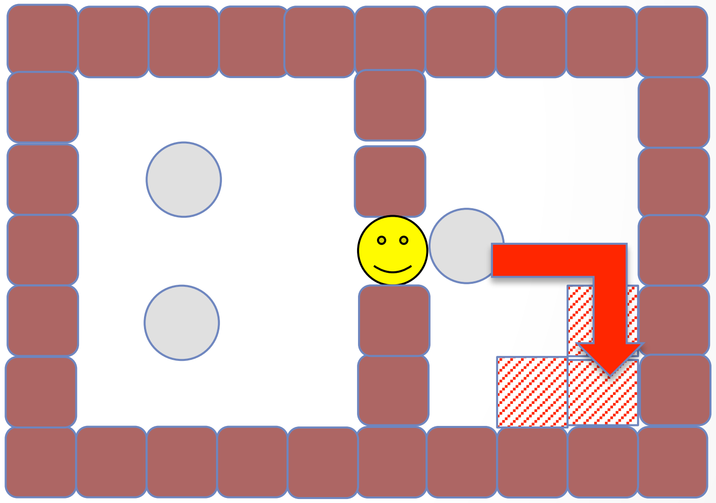
- Problems solved: 17

- Example:



Goal macros (R5) (4)

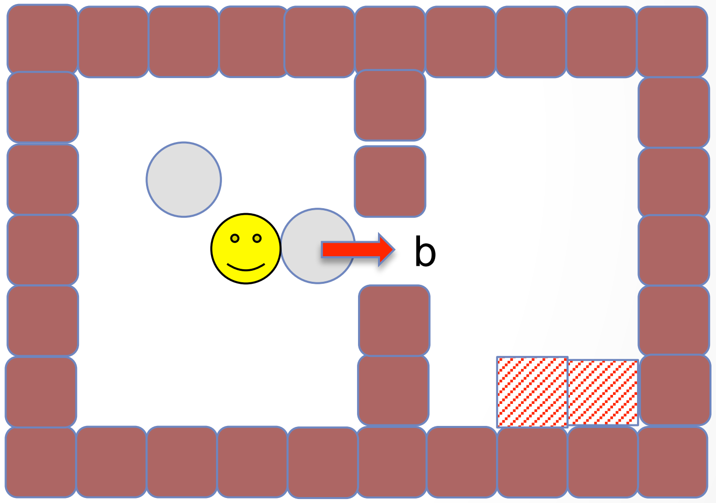
- Goal macro moves are grouped to one
- All other possible moves are ignored
- Example:



- Problems solved: 17

Goal cuts (R6) (1)

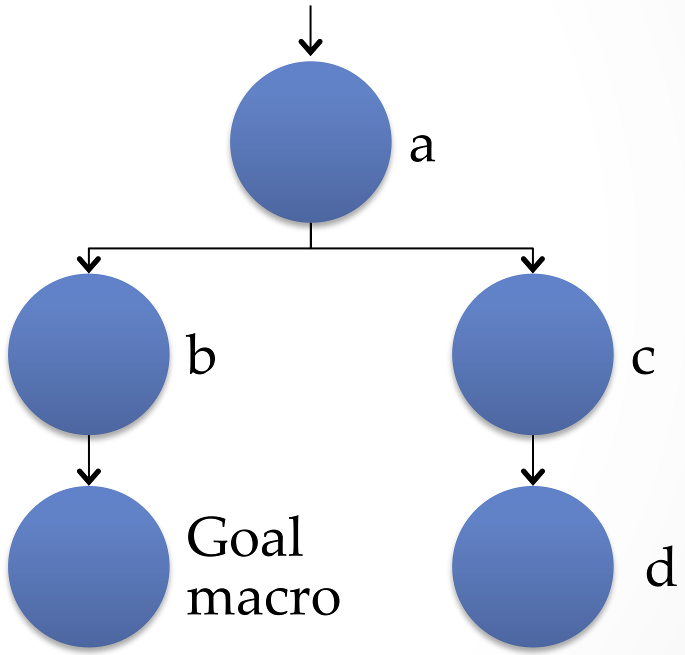
- If push „b“ starts a goal macro (stone reaches a entrance square), all childs of the parent of „b“ will be pruned
- Example:



- Problems solved: 24

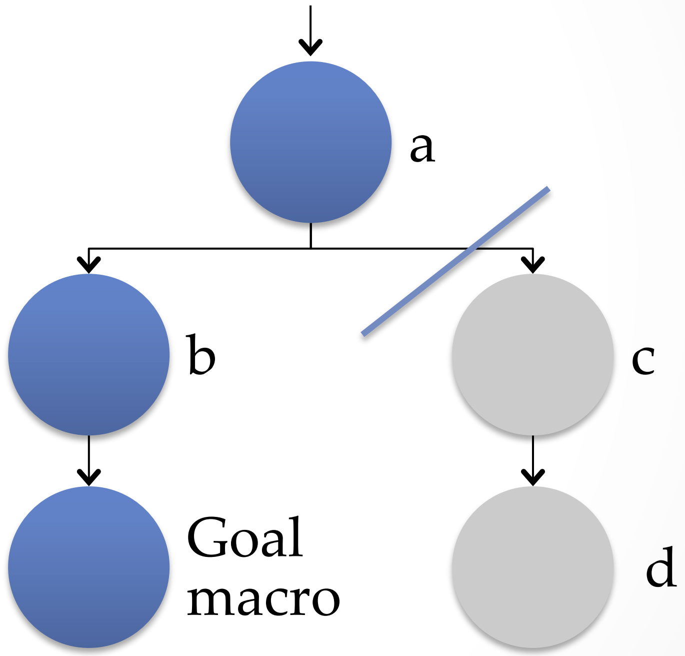
Goal cuts (R6) (2)

- If push „b“ starts a goal macro (stone reaches a entrance square), all childs of the parent of „b“ will be pruned
- In example the branch with move „c“ and „d“ is pruned after the goal macro is reached
- Problems solved: 24



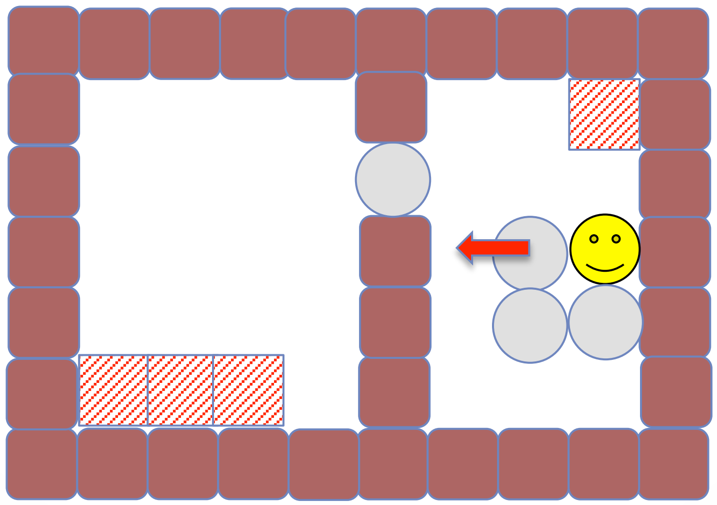
Goal cuts (R6) (3)

- If push „b“ starts a goal macro (stone reaches a entrance square), all childs of the parent of „b“ will be pruned
- In example the branch with move „c“ and „d“ is pruned after the goal macro is reached
- Problems solved: 24



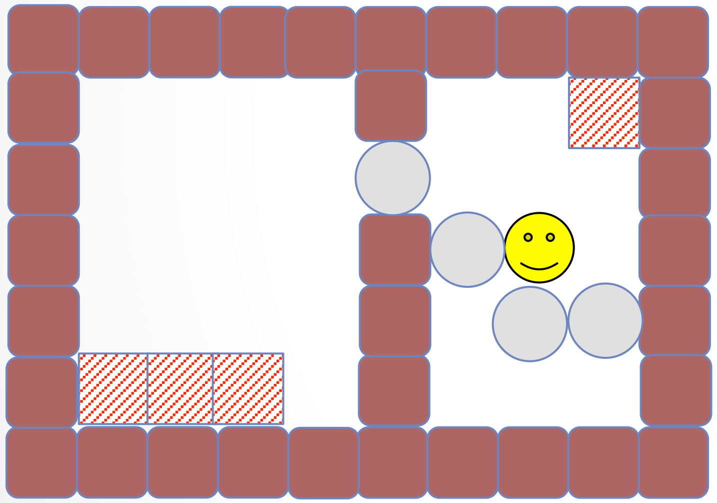
Pattern search (R7) (1)

- Goal: Find Deadlock or increase lower bound estimation (Slide: Overestimation)
- PIDA*: IDA* version for pattern search
- Example:

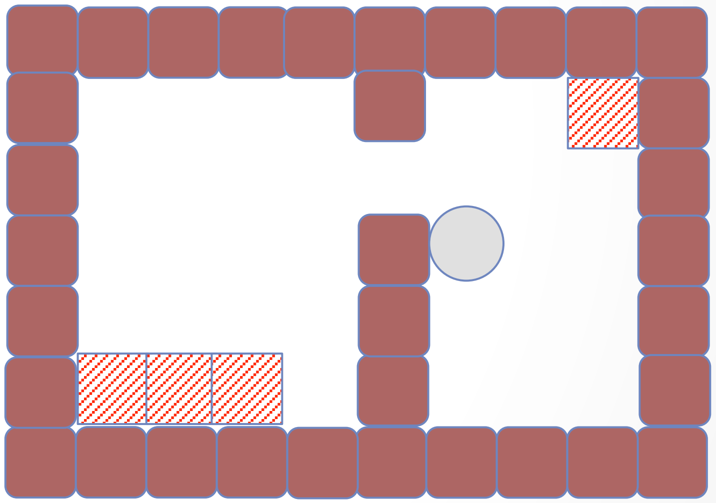


Pattern search (R7) (2)

- Original maze:

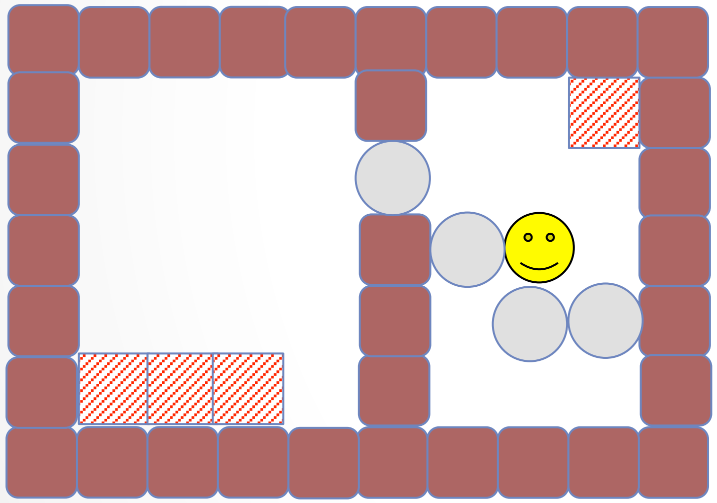


- Test maze:

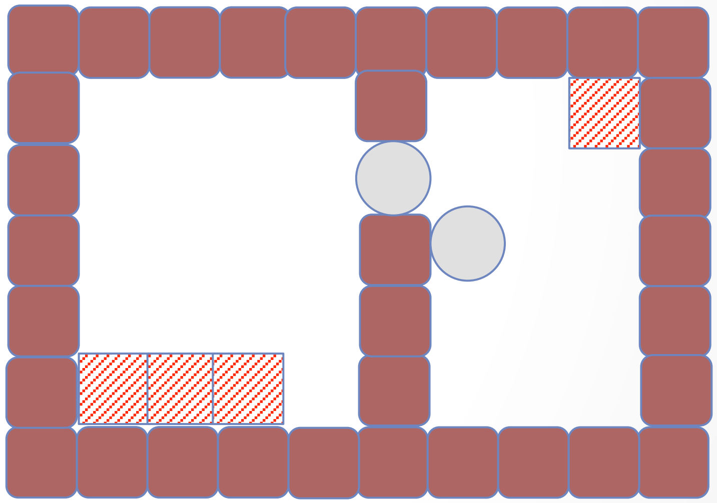


Pattern search (R7) (2)

- Original maze:

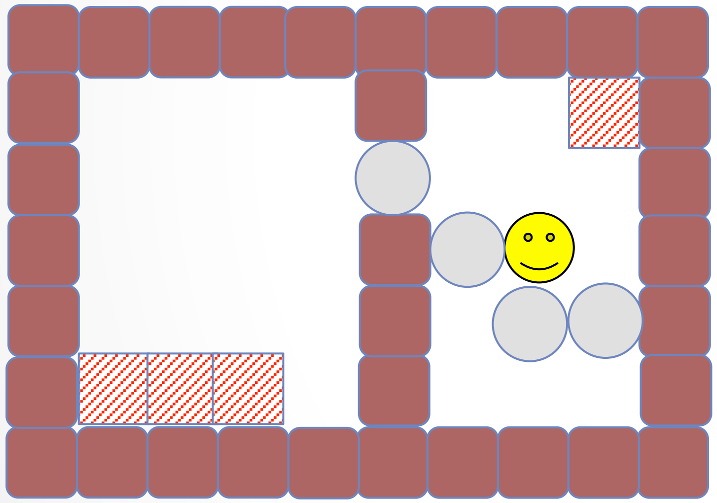
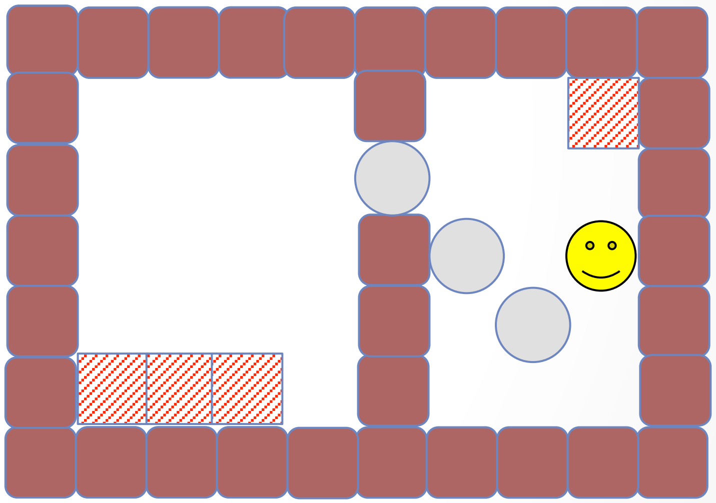


- Test maze:



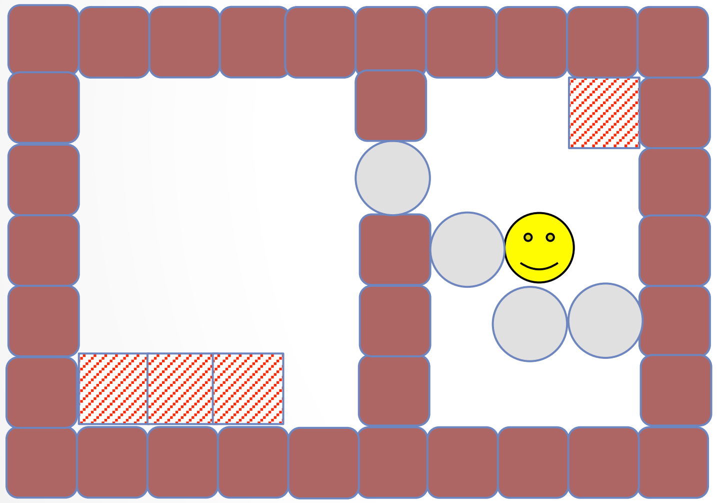
Pattern search (R7) (2)

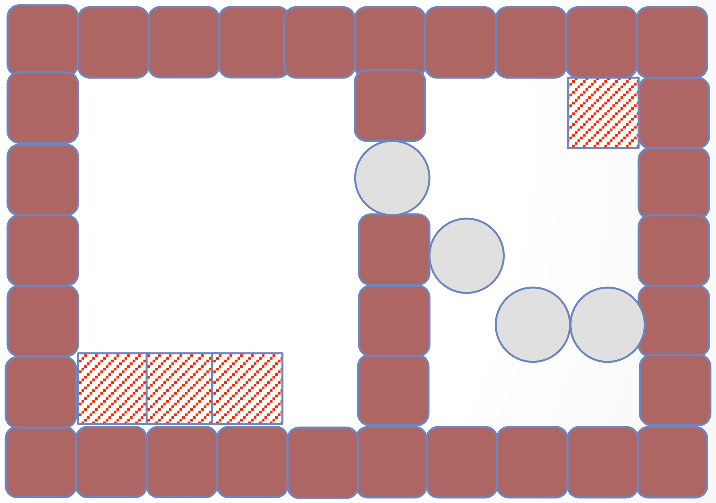
- Original maze:
- Test maze:

Pattern search (R7) (2)

- Original maze:
- Test maze:



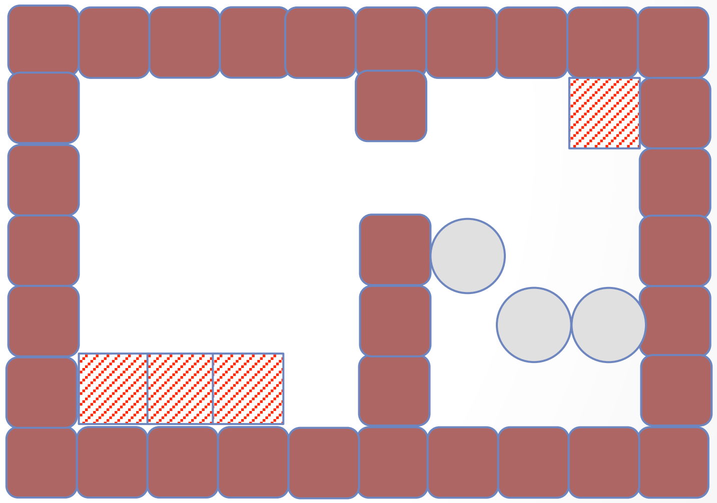


- => Deadlock occurs

Pattern search (R7) (3)

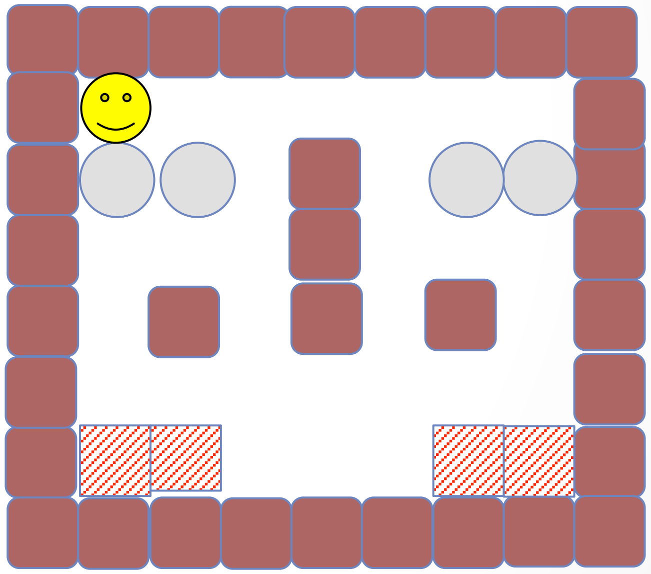
- Final steps:
  - Calculate minimum deadlock pattern
  - Add pattern to pattern table
- During IDA* search:
  - Check state against pattern search table
- Problems solved: 48

- Minimum pattern:



Relevance cuts (R8)

- Goal: to find independent subproblems
- Move only if the last # number of moves influence it
- Properties of influence/ relevance:
  1. Alternatives
  2. Goal-Skew
  3. Connection
  4. Tunnel
- Problems solved: 50



Overestimation (R9)

- A* is optimal, if h() is admissible
  - Admissible: $\forall n,\ h(n) \leq C(n)\ \mid\ C(n)$: actual cost to reach goal from *n*
- Non-admissible h => search often more accurate, but no longer optimal
- Sum of max penalty per stone added to h (penalties calculated from pattern search)
- => optimality no longer garanteed
- Problems solved: 54

Rapid random restarts (R10)

- Restarts with more randomization in move ordering (less strictness)
- A certain number of restarts with the same f-limit
- If f-limit increases, the randomization of move ordering will fall back to zero

- Problems solved: 57

Comparison table

	# solved	In prop. to 90	In prop. to 57	# less solved without this approach	In prop. to 90	In prop. To 57
Transposition table	5	5.6%	8.8%	19	21.1%	33.3%
Move ordering	4	4.4%	7.0%	0-1	0-1.1%	0-1.8%
Deadlock table	5	5.6%	8.8%	0-1	0-1.1%	0-1.8%
Tunnel macros	6	6.7%	10.5%	0-1	0-1.1%	0-1.8%
Goal macros	17	18.9%	29.8%	33	36.7%	57.9%
Goal cuts	24	26.7%	42.1%	0-1	0-1.1%	0-1.8%
Pattern search	48	53.3%	84.2%	22	24.4%	38.6%
Relevance cuts	50	55.6%	87.7%	0-1	0-1.1%	0-1.8%
Overestimation	54	60%	94.7%	0-1	0-1.1%	0-1.8%
RR Restart	57	63.3%	100%	0-1	0-1.1%	0-1.8%

Comparison table

	# solved	In prop. to 90	In prop. to 57	# less solved without this approach	In prop. to 90	In prop. To 57
Minimum matching	0	0%	0%	0-1	0-1.1%	0-1.8%
Transposition table	5	5.6%	8.8%	19	21.1%	33.3%
Move ordering	4	4.4%	7.0%	0-1	0-1.1%	0-1.8%
Deadlock table	5	5.6%	8.8%	0-1	0-1.1%	0-1.8%
Tunnel macros	6	6.7%	10.5%	0-1	0-1.1%	0-1.8%
Goal macros	17	18.9%	29.8%	33	36.7%	57.9%
Goal cuts	24	26.7%	42.1%	0-1	0-1.1%	0-1.8%
Pattern search	48	53.3%	84.2%	22	24.4%	38.6%
Relevance cuts	50	55.6%	87.7%	0-1	0-1.1%	0-1.8%
Overestimation	54	60%	94.7%	0-1	0-1.1%	0-1.8%
RR Restart	57	63.3%	100%	0-1	0-1.1%	0-1.8%

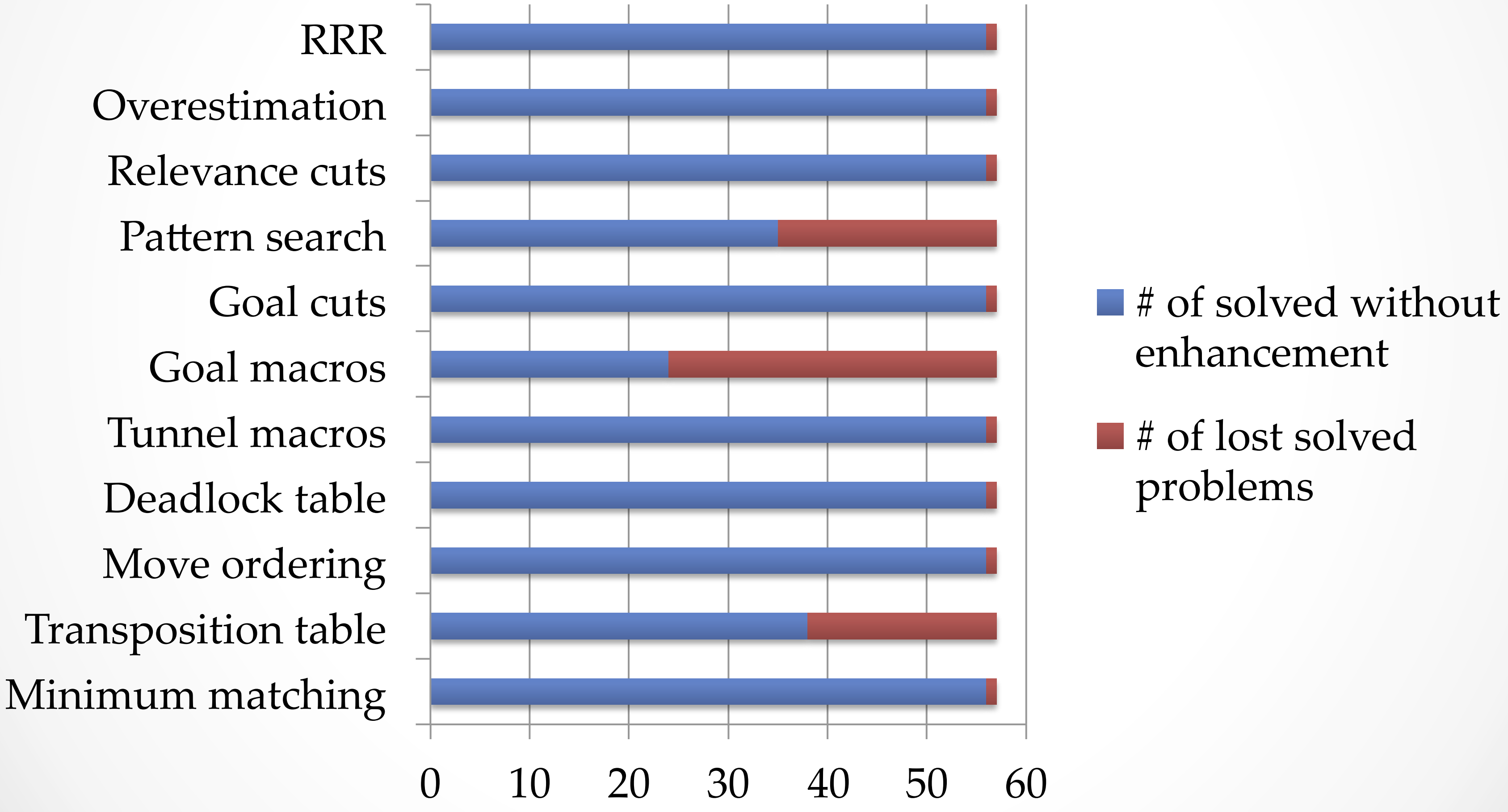
Comparison table

	# solved	In prop. to 90	In prop. to 57	# less solved without this approach	In prop. to 90	In prop. To 57
Minimum matching	0	0%	0%	0-1	0-1.1%	0-1.8%
Transposition table	5	5.6%	8.8%	19	21.1%	33.3%
Move ordering	4	4.4%	7.0%	0-1	0-1.1%	0-1.8%
Deadlock table	5	5.6%	8.8%	0-1	0-1.1%	0-1.8%
Tunnel macros	6	6.7%	10.5%	0-1	0-1.1%	0-1.8%
Goal macros	17	18.9%	29.8%	33	36.7%	57.9%
Goal cuts	24	26.7%	42.1%	0-1	0-1.1%	0-1.8%
Pattern search	48	53.3%	84.2%	22	24.4%	38.6%
Relevance cuts	50	55.6%	87.7%	0-1	0-1.1%	0-1.8%
Overestimation	54	60%	94.7%	0-1	0-1.1%	0-1.8%
RR Restart	57	63.3%	100%	0-1	0-1.1%	0-1.8%

Comparison table

	# solved	In prop. to 90	In prop. to 57	# less solved without this approach	In prop. to 90	In prop. To 57
Minimum matching	0	0%	0%	0-1	0-1.1%	0-1.8%
Transposition table	5	5.6%	8.8%	19	21.1%	33.3%
Move ordering	4	4.4%	7.0%	0-1	0-1.1%	0-1.8%
Deadlock table	5	5.6%	8.8%	0-1	0-1.1%	0-1.8%
Tunnel macros	6	6.7%	10.5%	0-1	0-1.1%	0-1.8%
Goal macros	17	18.9%	29.8%	33	36.7%	57.9%
Goal cuts	24	26.7%	42.1%	0-1	0-1.1%	0-1.8%
Pattern search	48	53.3%	84.2%	22	24.4%	38.6%
Relevance cuts	50	55.6%	87.7%	0-1	0-1.1%	0-1.8%
Overestimation	54	60%	94.7%	0-1	0-1.1%	0-1.8%
RR Restart	57	63.3%	100%	0-1	0-1.1%	0-1.8%

Independent enhancement test



Conclusions

- IDA* is simple to implement and solve domain-independent problems
- Combination with domain-dependent knowledge can result in a greatly improved search performance
- In such a vaste search space like sokoban problems, domain-dependent approaches can help solving them
- In some case, the way to find domain-dependent knowledge is to study the problem solutions

Discussion

- Questions?