Pattern Databases

Robert Clausecker <clausecker@zib.de>

State Abstractions

- search problems are hard to solve
- if we remove constraints, the problems is easier
- for example
  - TSP: allow a tree instead of a tour
  - Sokoban: ignore everything but pushing boxes
  - Rubiks Cube: ignore edge tiles
  - 15 Puzzle: ignore identity of some tiles
  - 15 Puzzle: ignore some tiles completely

    [Edelkamp, Schrödl 2012]

State Abstractions

- a solution to the original problem still works in the abstraction
  - thus, a solution for the abstracted problem is an admissible heuristic for the original problem
- how do we solve the abstracted problem?
  - have a fast P-time algorithm to solve it?

State Abstractions

- a solution to the original problem still works in the abstraction
  - thus, a solution for the abstracted problem is an admissible heuristic for the original problem
- how do we solve the abstracted problem?
  - ~~have a fast P-time algorithm to solve it?~~
  - just do it every time we evalute the heuristic?

Valtorta's Theorem

Let u be any state necessarily expanded, when finding a path from v_0 to z with $v_0, z \in S$ and let $\varphi : S \rightarrow S'$ be any abstraction mapping; and the heuristic estimate $h(u)$ be computed by blindly searching from $\varphi(u)$ to $\varphi(z)$. If the problem is solved by the A* algorithm using h, then either u itself will be expanded, or $\varphi(u)$ will be expanded.

[Edelkamp, Schrödl 2012]

State Abstractions

- a solution to the original problem still works in the abstraction
  - thus, a solution for the abstracted problem is an admissible heuristic for the original problem
- how do we solve the abstracted problem?
  - ~~have a fast P-time algorithm to solve it?~~
  - ~~just do it every time we evalute the heuristic?~~
  - solve all abstract states once, tabulate the results (but beware of space usage)

Space Considerations

k	APDB size	ZPDB size	avg	max
2	600	608	1.01	2
3	13 800	14 472	1.04	2
4	303 600	339 048	1.12	3
5	6 375 600	7 871 280	1.23	4
6	127 512 000	181 008 000	1.42	5
7	2 422 728 000	4 066 655 040	1.68	6
8	43 609 104 000	87 358 400 640	2.00	7
9	741 354 768 000	1 759 513 674 240	2.37	8
10	11 861 676 288 000	32 787 717 580 800	2.76	10
11	177 925 144 320 000	560 680 553 664 000	3.15	11
12	2 490 952 020 480 000	8 749 801 518 796 800	3.51	13

in a Nutshell

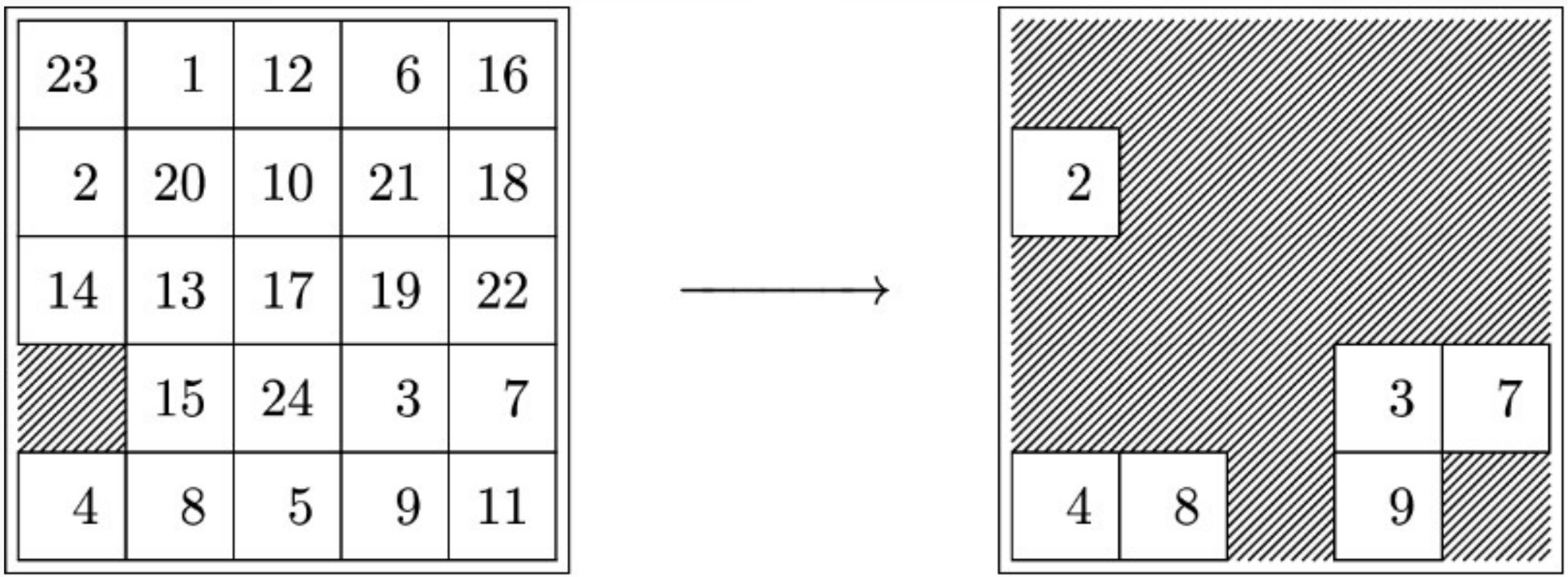
- simplify the problem space (abstraction)
  - to make the problem solvable
  - to make the tables fit the available storage
- the simplified problem is never harder than the original
  - thus admissible as a heuristic
- tabulate results for the simplified space
  - cannot do this just-in-time (cf. Valtorta's theorem)
- and use that table as heuristic during the search

Additive Pattern Databases

Operator Partitionings

- transitions in the problem space can be described by *operators*
  - Rubiks Cube: F, B, U, D, L, R
  - 24 puzzle: move tile *x* up/down/left/right
  - Towers of Hanoi: transfer from *x* to *y*
- one kind of abstraction: ignoring operators
  - gained by ignoring some operators
  - i.e. executing them is free

for the 24 Puzzle



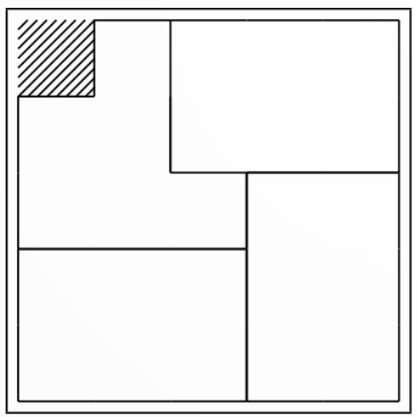
a 24 puzzle abstracted to the tiles { 2, 3, 4, 7, 8, 9 }
[Clausecker 2017]

Additive Heuristics

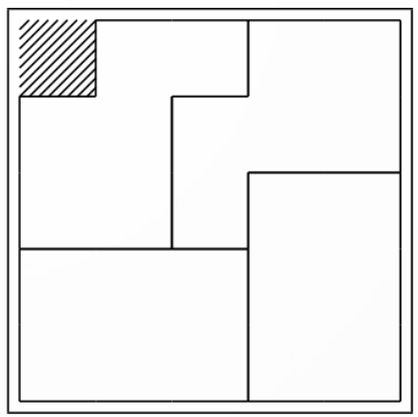
- if two abstractions care for a disjoint set of operators, the sum of their *h* values is still admissible (and consistent)
  - worse than using a single abstraction
  - but a single abstraction won’t fit our memory

    [Seipp et al. 2017]
- if we can put each operator into some abstraction, we get a fairly good heuristic

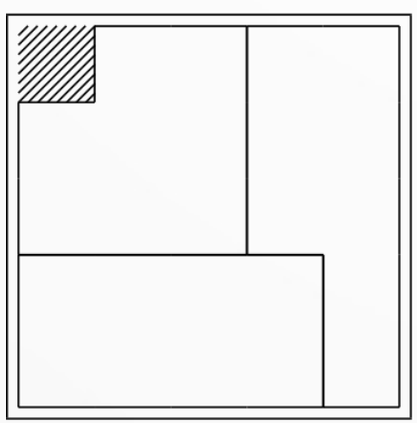
Examples



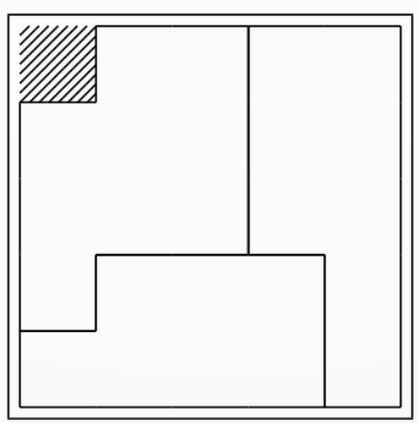
6-6-6-6 orig.



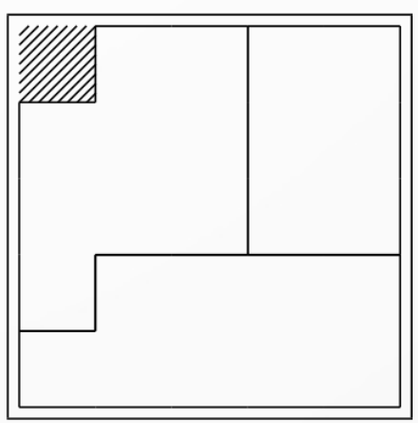
6-6-6-6 new



8-8-8



9-8-7



9-9-6

some partitionings of the 24-puzzle

Building Pattern Databases

PDB Construction

- we construct a PDB by breadth-first search of the abstracted search space
- no need for open and close lists: just store them in the PDB
- when all nodes are closed, we are done

Building Blocks

- a **goal state** $z \in S$
- an **abstraction** A
- an **index set** $I_A = \{ 0, 1, \ldots, s_A - 1 \}$
  - with s_A being the **PDB size**
- an **index function** $idx_A(v) : S \rightarrow I_A$
  - performs state abstraction and indexing
- an **inverse index function** $idx_A^{-1}(n) : I_A \rightarrow S$
  - such that $n = idx_A(idx_A^{-1}(n))$ for all n
- a **move function** $move_A(v) : S \rightarrow 2^S$
  - computes all configurations adjacent to v in A

PDB Construction

1. $pdb[0, \ldots, s_A - 1] \leftarrow \infty$
2. $pdb[idx_A(z)] \leftarrow 0$
3. $r \leftarrow 0$
4. **while** $\exists e \in I_A.\ pdb[e] = r$ **do**
    1. $V_r \leftarrow \{\ idx_A^{-1}(e) \mid pdb[e] = r\ \}$
    2. $V_{r+1} \leftarrow \{\ u \mid v \in V_r,\ u \in move_A(v),\ pdb[idx_A(u)] = \infty\ \}$
    3. **for** $v \in V_{r+1}$ **do** $pdb[idx_A(v)] \leftarrow r + 1$
    4. $r \leftarrow r + 1$
5. **return** *pdb*

[Clausecker 2017]

PDB Verification

- to verify that the PDB is correct, we verify that it is correct for every entry
  - global correctness follow from local correctness [Clausecker 2017]
- we check that
  - the entry for z is 0
  - adjacent entries have a distance of no more than 1
  - progress is possible from every configuration

PDB Verification

1. **if** $pdb[idx_A(z)] \neq 0$ **then** return *invalid*
2. **for** $e \in I_A$ **do**
   1. $v \leftarrow idx^{-1}(e)$
   2. **if** $\exists u \in moves_A(v).\ pdb[e] - pdb[idx_A(u)] > 1$ **then return** *invalid*
   3. if $v \neq z \wedge \neg\exists u \in moves_A(v).\ pdb[idx_A(u)] < pdb[e]$ **then return** *invalid*
3. **return** *valid*

[Clausecker 2017]

Index Functions

- sparse mapping
  - just use a k-dimensional array for k pieces of state and ignore all invalid combinations
  - pretty fast and easy to implement
  - but wastes a lot of space
- compact mapping
  - map states tightly to I_A
  - slower and harder to implement
  - can work with a lot less space

Index Functions

- for the 24 puzzle: need to store a (partial) permutation
  - inversion numbers
    - complicated bit fiddling
    - slightly faster
    - many variants to experiment with
  - Fisher Yates shuffle
    - just a bunch of data moves
    - slightly slower
    - pretty simple to implement
  - http://github.com/FUZxxl/permcode

Literature

- [Edelkamp, Schrödl 2012] S. Edelkamp and S. Schrödel: *Heuristic Search: Theory and Applications*, Morgan Kaufmann, 2012
- [Clausecker 2017] R. Clausecker: *Notes on the Construction of Pattern Databases*, ZIB Technical Report 17-59, 2017
- [Seipp et al. 2017] J, Seipp, Th. Keller and M. Helmert: *A Comparison of Cost Partitioning Algorithms for Optimal Classical Planning*, ICAPS-2017 proceedings, 2017