Analysis and resolution of combinatorial games

Universidad de Málaga

Carlos Linares López

Planning and Learning Group (PLG)
Computer Science Department
Universidad Carlos III de Madrid

November, 28, 2013

Motivation

Combinatorial games offer a good opportunity for research in AI:

- They follow very simple rules
- They engender exponentially large state spaces
- They exemplify the usage of programming techniques and optimization algorithms
- For which no optimal strategy is known
- They can be easily extended to more complex cases
- Progress can be easily tracked

. . . and they are fun!

Talking about state spaces . . . I

"*Everything is a computation. But a Turing Machine is a graph. So everything is a graph*"

Scope I

We focus on domain-dependent solvers as opposed to domain-independent solvers (i. e., *Automated Planning*)

(Get ready to all sorts of *tricks*!)

Scope II

We focus on deterministic environments as opposed to domains with uncertainty

Talking about state spaces . . . II

State spaces in the context of combinatorial games

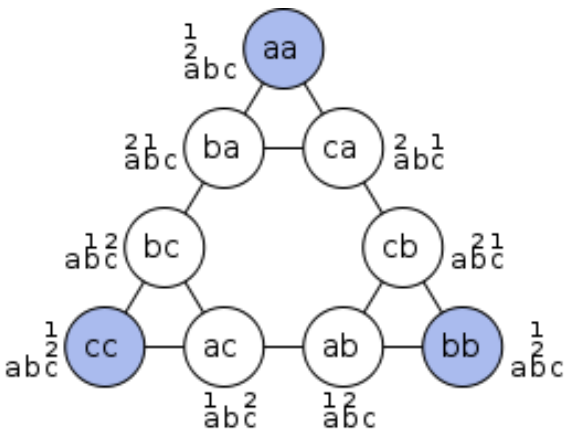
A *state space* is a weighted directed graph $\mathcal{S} = \langle T, \Pi, C \rangle$, where T is a finite set of states, $\Pi \subseteq T \times T$ is a set of directed edges (ordered pairs of states) representing state transitions, and $C : \Pi \longrightarrow \mathbb{N}$ is the edge cost function

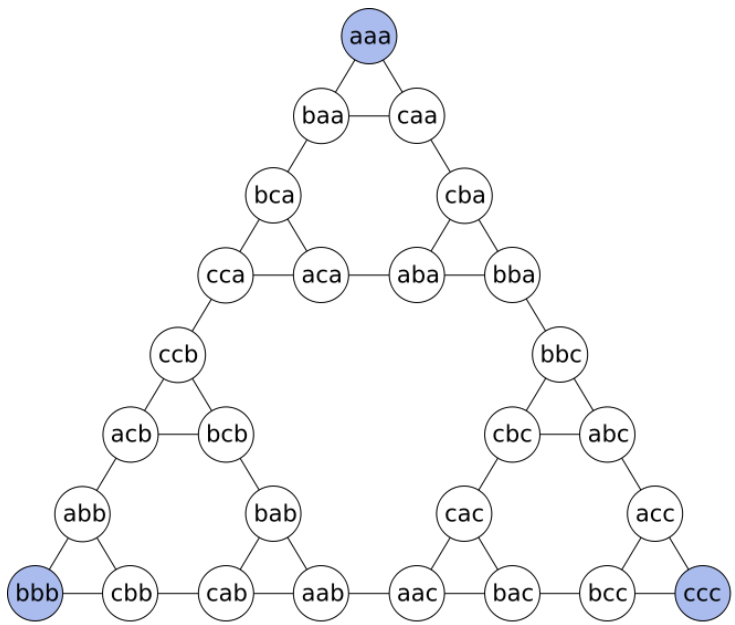
Talking about state spaces . . . III

- State spaces might consist of different components (see *N*-Puzzle)
- Some combinatorial games consist of *permutations* (e. g., Rubik's cube) and their state space is distinguished with the name "*permutation state spaces*"
  Some permutations might not be legal (e. g., terms like "*the legal cube group*" are used)

Talking about state spaces . . . IV

- It is a misconception to believe that state spaces of combinatorial games are randomly generated. Indeed, they usually follow some structure that can be discovered and exploited
  Indeed, the state space of the Towers of Hanoi is the Sierpinski's triangle





Problem space

Definition

A *problem space* consists of a state space with one (or more) nodes distinguished as the *start state*, s and one (or more) nodes distinguished as the *target state*, t

Goal I

There are different ways to deal systematically with these problems. Remarkably:

- Operations research: The problem of finding the shortest path in a graph can be solved with a Mixed Integer Programming (MIP) task
- Algebra: If the state spaces show some structure, they might be studied with *group theory* (generators, subgroups, cosets, etc.)
- Logic: Some seemingly simple problems can be solved in the light of computational logic (e. g., rewritting logic, narrowing, ...)
- Computer Science: But graphs can be also efficiently traversed as *search trees*

Goal II

Goal

Our goal is to build an algorithmic theory that solves exponentially hard problems

Programming Language

The programming language matters!

- Interpreted programming languages (LISP, JAVA, ...): good for fast prototyping but rather inefficient (e.g., garbage collector, memory management, etc.)
- Compiled programming languages (C, C++, etc.): good for showing progress but sometimes awkward or too difficult

Data structures

As said before, be ready to all sorts of tricks!

- Data structures matter! (B-TREEs, AVT-TREEs, BDDs, ZDDs, etc.)
- But also how operations on data structures are implemented (e.g, `image`, `pre-image` in BDDs)

Suggestion

Do not be too impatient! Do not going straight to the point and giving some consideration to the programming language/data structure usually pays off, if not always!

Programming Techniques

Some combinatorial games can be efficiently solved using different Programming Techniques:

- Divide-and-Conquer: Towers of Hanoi (3 pegs)
- Dynamic Programming: those that can be addressed with the Sprague-Grundy theorem (e. g., Nim, Dots-and-boxes, etc.)
- Greedy search: 8-Puzzle, Nine Men's Morris

Blind search

Algorithm	Complete?	Admissible	Memory	Time
Breadth-First	✔	✔	$O(b^d)$	$O(b^d)$
Depth-first	✖	✖	$O(bd)$	$O(b^d)$
Bidirectional	✔	✔	$O(b^{\frac{d}{2}})$	$O(b^{\frac{d}{2}})$

Table: Characterization of various uninformed search algorithms

Results

Very modest success can be achieved with these algorithms:

- 8-Puzzle
- $2 \times 2 \times 2$ Rubik's cube
- 10-Pancake
- (11,2)-TopSpin

. . . since they do scale up very poorly

Heuristics

Informal definition

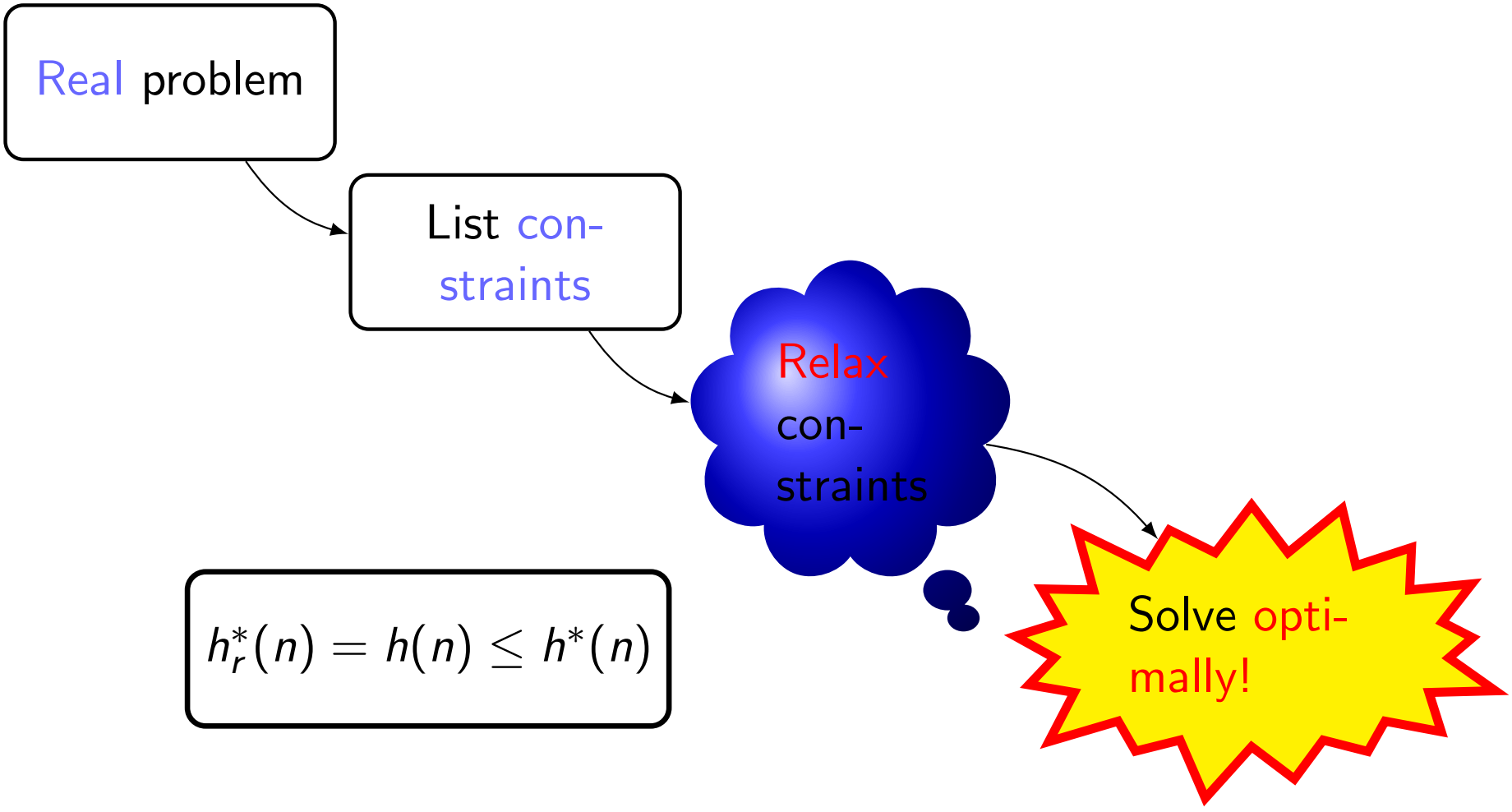
Heuristics are just "rules of thumb"

They were typically generated using "1% inspiration and 99% perspiration"

Thomas Edison

Far before the appearance of search algorithms

Constraint relaxation



Improving over constraint relaxation

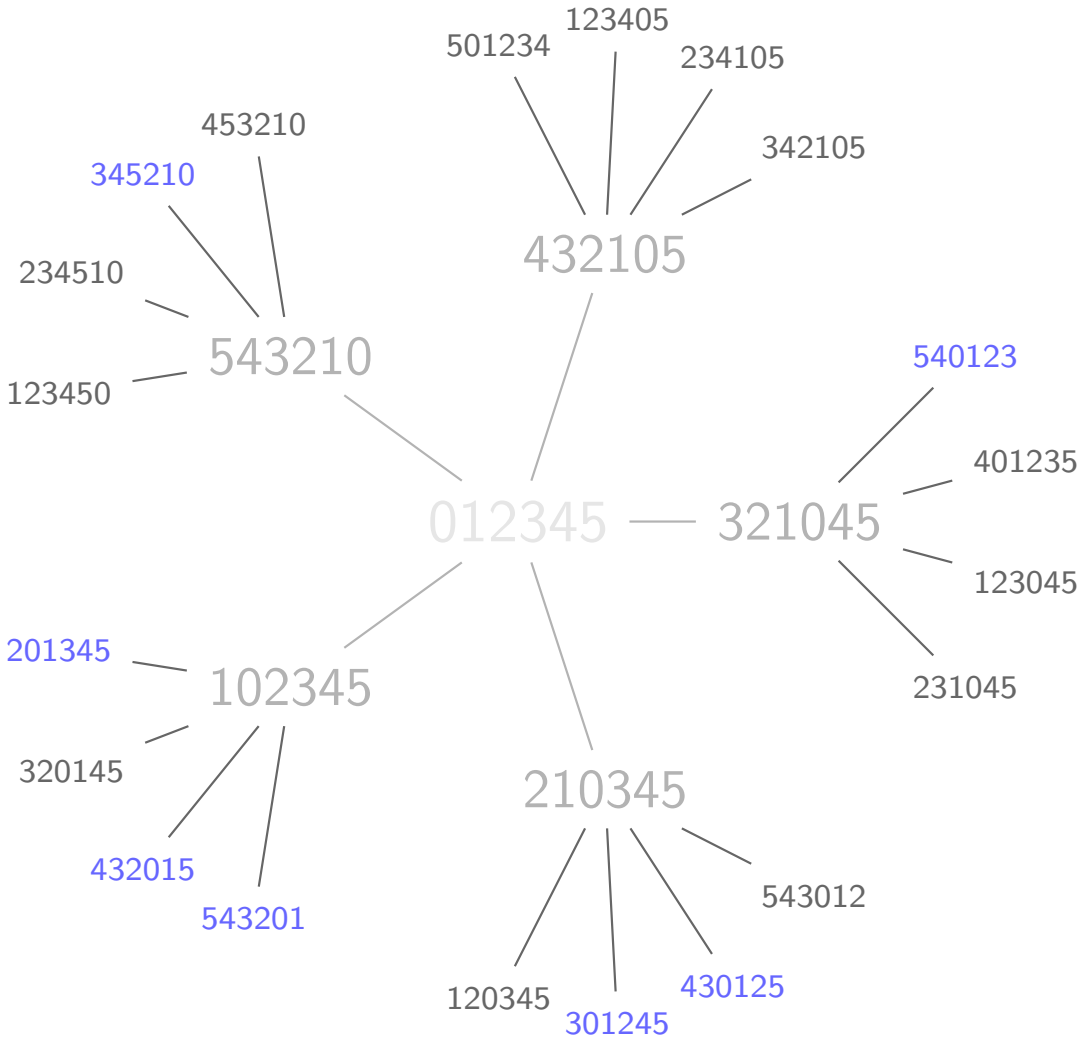
Domain-dependent

- 8-Puzzle: The linear conflict was obtained with additional observations
- Pancake: The gap heuristic is obtained theoretically from the landmarks heuristic

Domain-independent

- Perimeter search was originally devised as a domain-independent idea for improving heuristic estimations
- Duality suggests taking the maximum of two admissible estimates of two nodes which are known to have the same optimal cost

Perimeter search

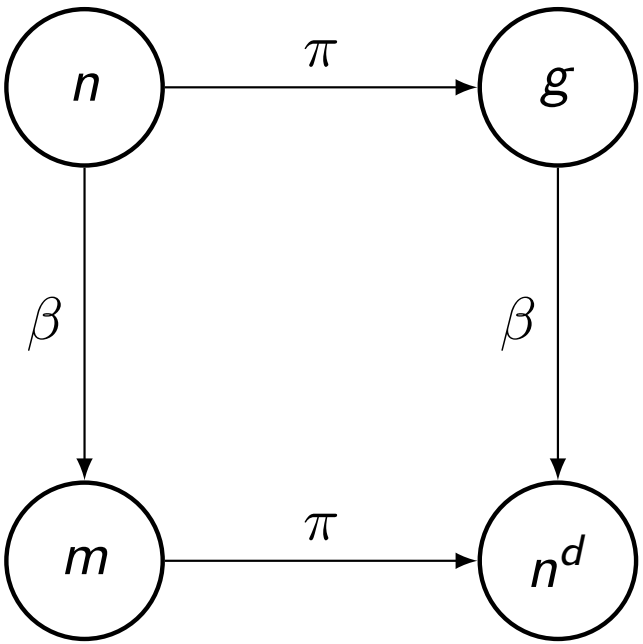


Improves the heuristics by the perimeter depth:

$$h_d = \max_{i=1,N} \{h(n, m_i)\}$$

It inspired a new search algorithm where some perimeter nodes can be ignored

Duality



$$n^d = \pi(\beta(n)) = \beta(\pi(n))$$

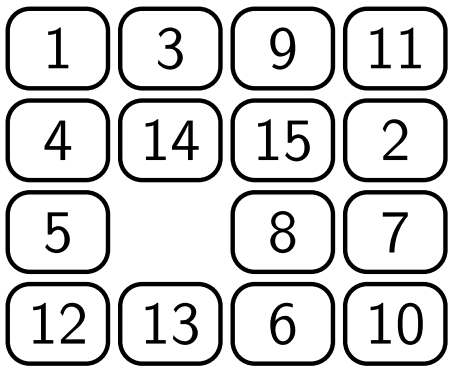
- Only applicable to permutation state spaces
- It also inspires a new bidirectional search algorithm

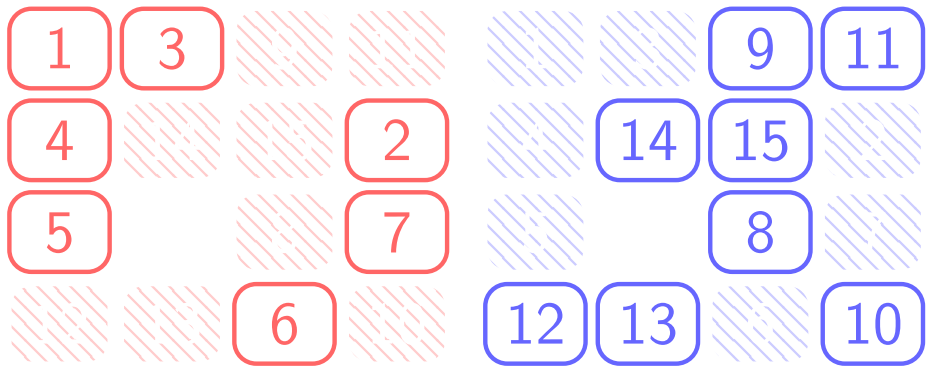
Pattern Databases I

Still, automated systems can improve over the previous approaches

Abstractions

An *abstraction mapping* $\psi_i : \mathcal{S} \longrightarrow \mathcal{A}_i$ between state space $\mathcal{S}$ and abstract state space $\mathcal{A}_i$, is defined by a mapping between the states of $\mathcal{S}$ and the states of $\mathcal{A}_i$, $\psi_i : T \longrightarrow T_i$





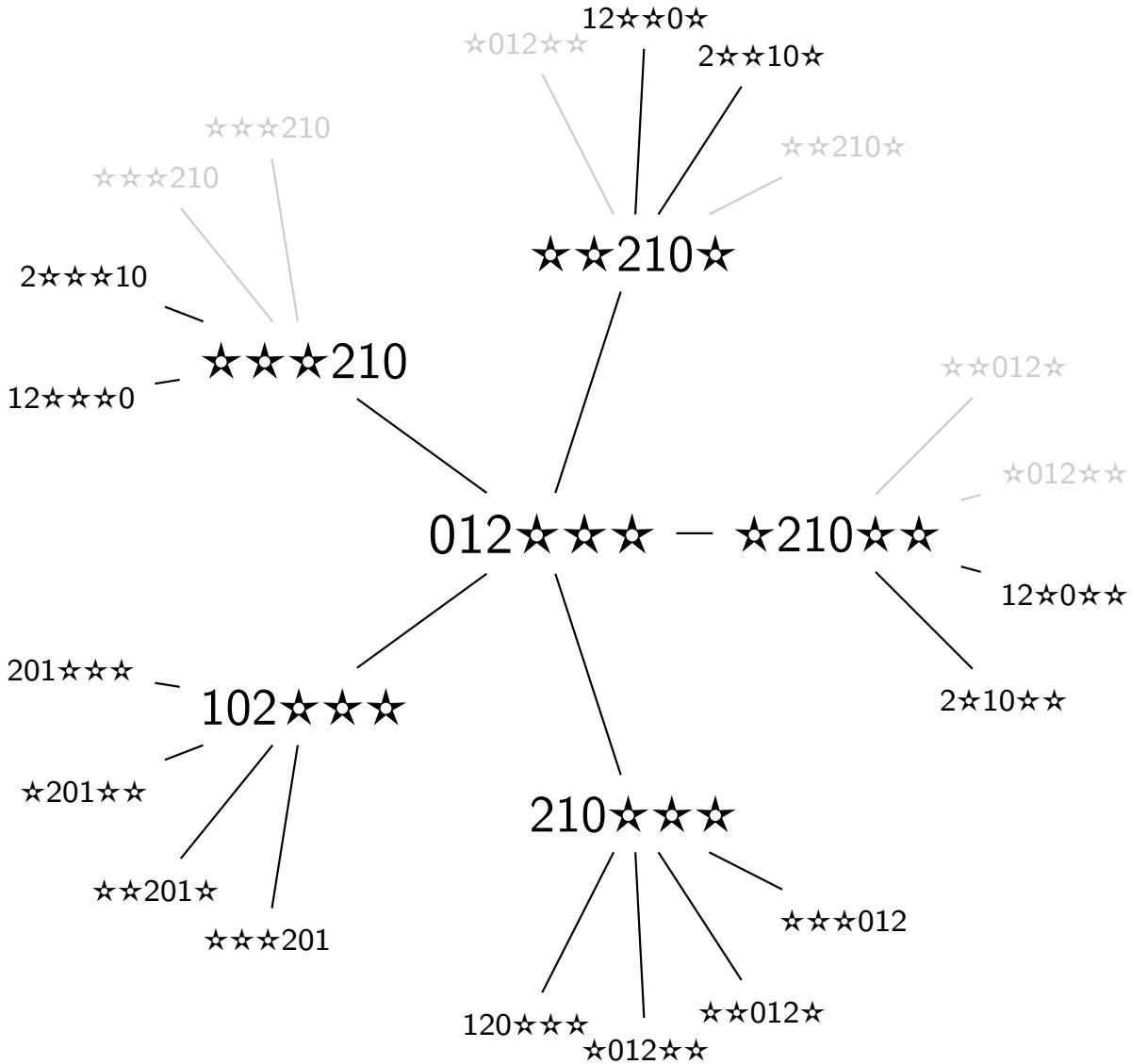
Pattern Databases II

MAX PDBs: From the goal pattern a breadth-first backwards search can be conducted to derive the optimal path cost of all patterns.
In the presence of N PDBs, one can simply:

$$h = \max_{i=1,N} \{h_i(\psi_i(n))\}$$

which led to the first reporting of optimal solution in the $3 \times 3 \times 3$-Rubik's cube

Pattern Databases III



Pattern Databases IV

ADD PDBs: Or even better! One can compute only the cost of those operators affecting each pattern and then to add them:

$$h = \sum_{i=1,N} \{h_i(\psi_i(n))\}$$

And for the first time we were able to solve random instances of the 15-Puzzle in a matter of 10^{-2} seconds on average

Pattern Databases V

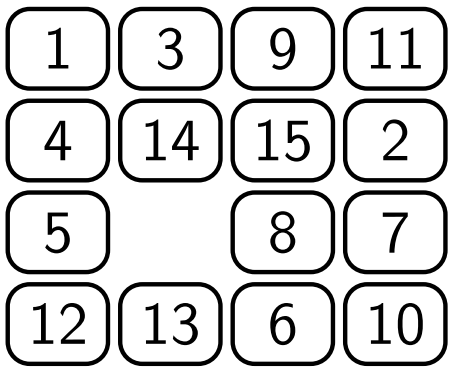
Cost-splitting: But wait! If they are not disjoint and the same operator affects $k \leq N$ abstractions then one can just simply divide every cost path by the number of operators, κ, affecting each pattern:

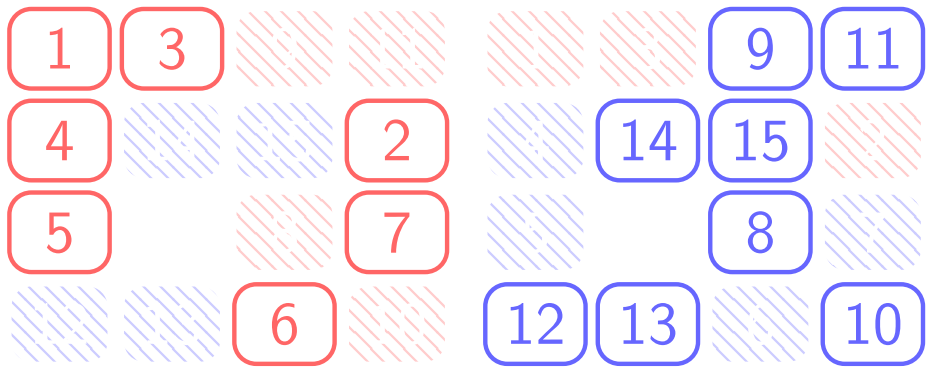
$$h = \max_{i=1,N} \left\{ \frac{h_i(\psi_i(n))}{\kappa} \right\}$$

which turned out to work very well in the (18,4)-TopSpin but not in the Rubik's cube

Pattern Databases VI

A different variant distinguishes the "*don't care*" symbols





And other variants also store different occurrences of the same pattern

Pattern Databases VII

To the outer world!

In the end, Pattern Databases were generalized to the case of domain-independent solvers and a further generalization of them,merge-and-shrink has been proven to be very successful in the same setting —getting various awards at the last International Planning Competition.

N-Puzzle

- 15-Puzzle is not the state-of-the-art anymore
- But some instances of the 24-Puzzle can take days to be solved
- The best approach consists of applying IDA* + ADD PDBs + duality

K-Pancake

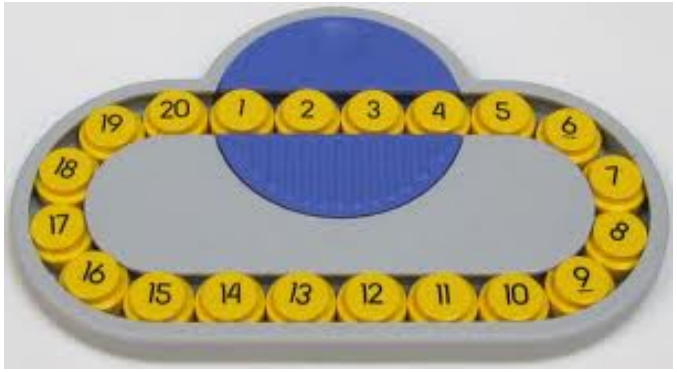
- The 17-Pancake was the state-of-the-art for sometime.
- Usually solved with IDA* + (cost-splitting) PDBs
- But a simple idea based on constraint relaxation yielded the gap heuristic
- Current state of the art is 60-Pancake solved with IDA* + gap

It has been also shown that the Pancake problem shows symmetries that can be exploited with relative-order PDBs

(N, K)-TopSpin

- Current state of the art is the $(18, 4)$-TopSpin
- The best approach consists of applying IDA* + (COST-SPLITTING) PDBs considering circular permutations

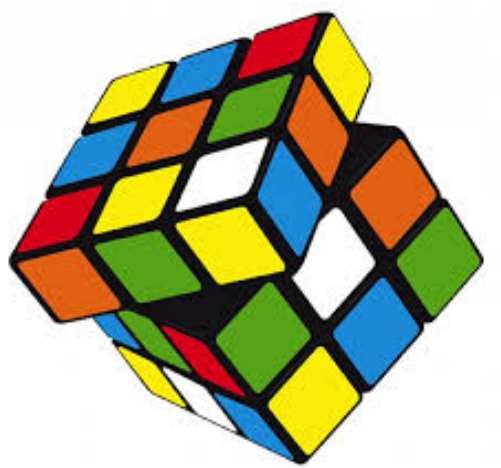
If (cost-splitting) PDBs are used, it was empirically shown that duality has disastrous consequences



Rubik's cube

- Current state of the art is $3 \times 3 \times 3$-Rubik's cube
- The best approach consists of applying IDA* + MAX PDBs exploiting symmetries

Additionally, some experiments have been conducted storing large PDBs in secondary memory devices but still more research has to be invested

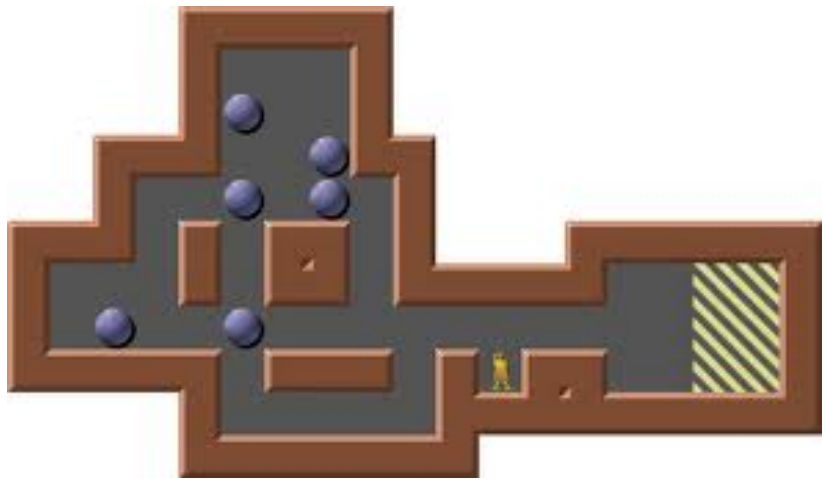


(N, K)-Towers of Hanoi

- Current research focuses on 4 pegs
- Current state of the art is 30 discs
- The best approach consists of applying IDA* + ADD PDBs exploiting symmetries

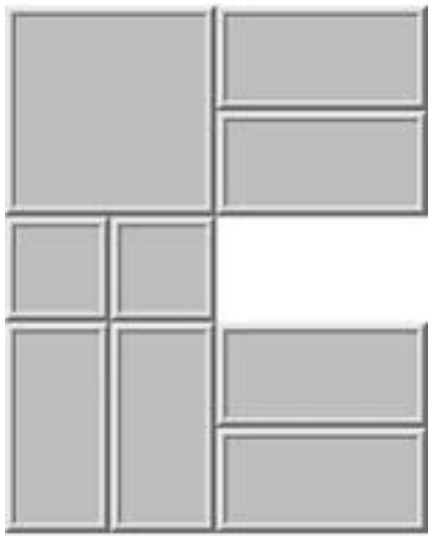
Sokoban

- Sokoban is PSPACE-complete
- Not much progress since 1999
- Current state of the art includes IDA* + lower bounds + transposition tables + move ordering + deadlock tables + macro moves
- Still, there are unsolved instances in the typical testbed



Klotski

- Klotski is another sliding-tile puzzle (NP-completeness)
- Almost no work on it, if any



Pacman

- Pacman is NP-hard
- Check out `www.pacman-vs-ghosts.net`

Classic Nintendo

- Mario Bros is NP-hard
- Check out `www.marioai.org`

Hope you enjoyed...

Be curious!