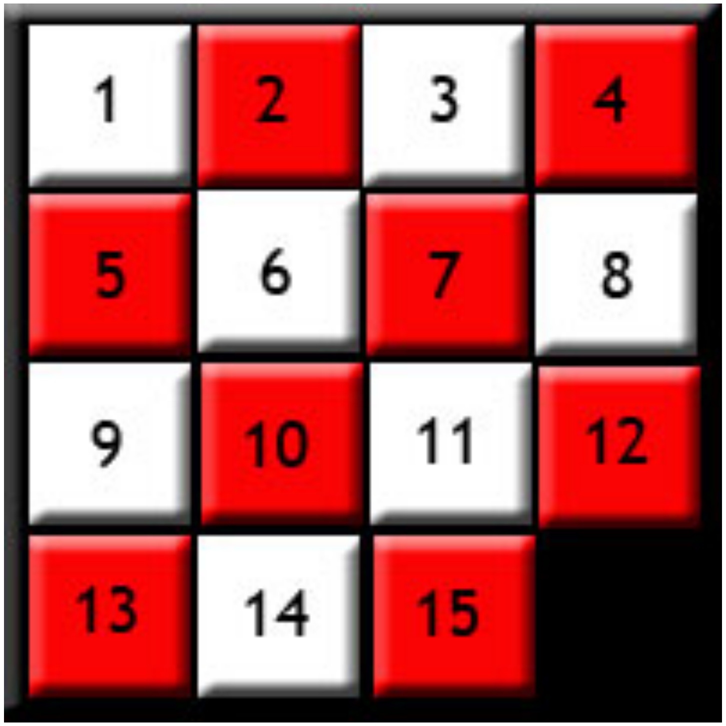
Puzzle Solving (single-agent search)

Robert Holte
Computing Science Department
University of Alberta

Puzzle = 1-player game

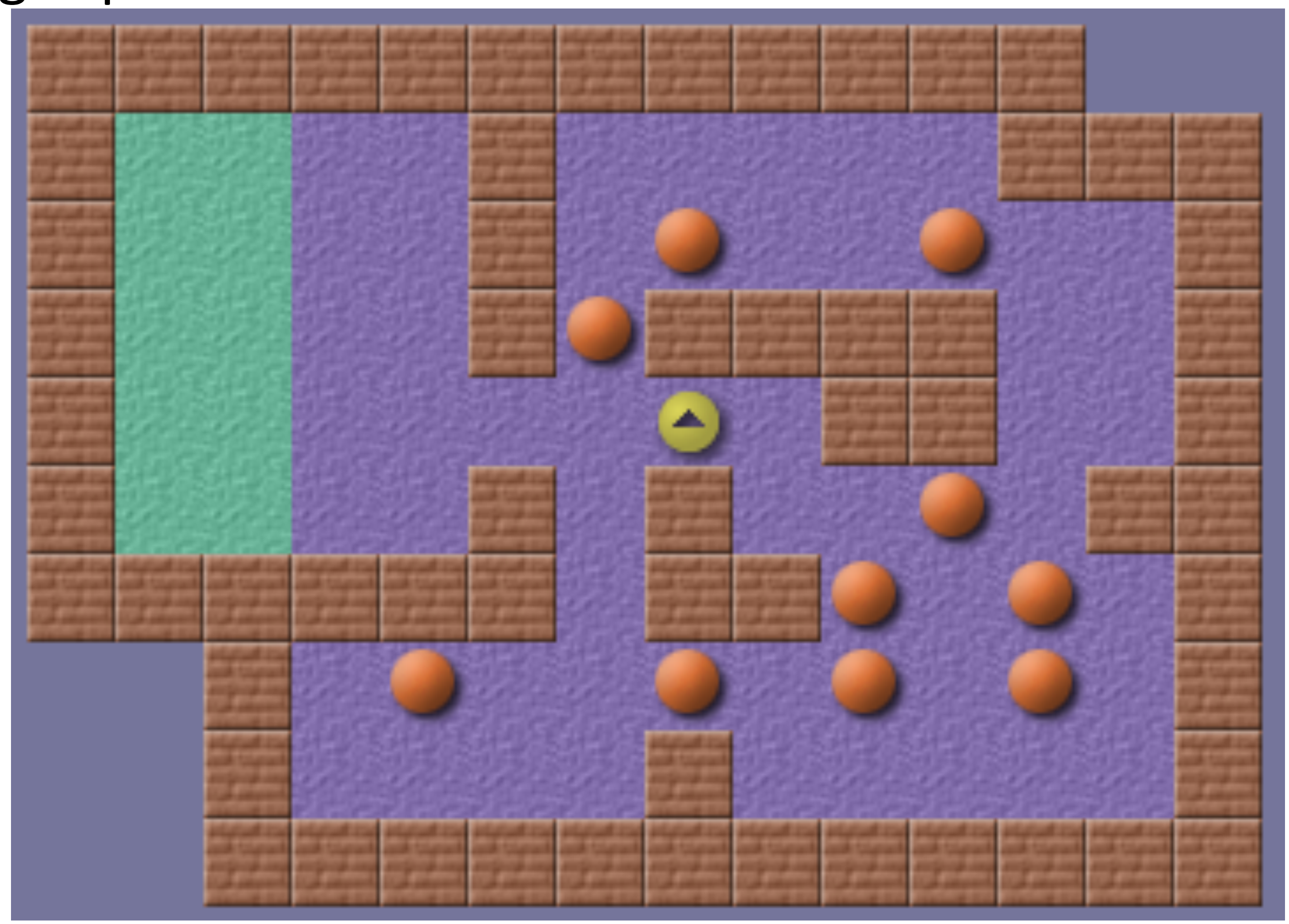
- Examples:
  - Rubik's Cube
  - Sliding-tile puzzle

- In this talk puzzles have
  - deterministic actions
  - perfect information
  - no chance events



Puzzles can be PSPACE-complete

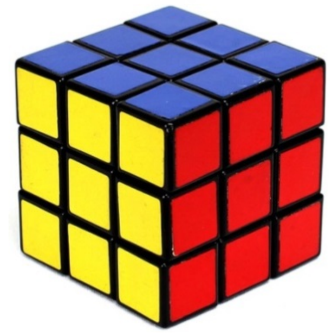
- Sokoban solvability first proven PSPACE-complete by Joe Culberson
- Visiting expert: André Grahl Pereira



Heuristic Search

What is State Space Search?

GIVEN



- Start state
- Goal state
- Successor function (maps a state to a set of states)
- Cost function (if x is a successor of s, cost(s,x) is the non-negative cost of reaching x from s ("edge cost"))

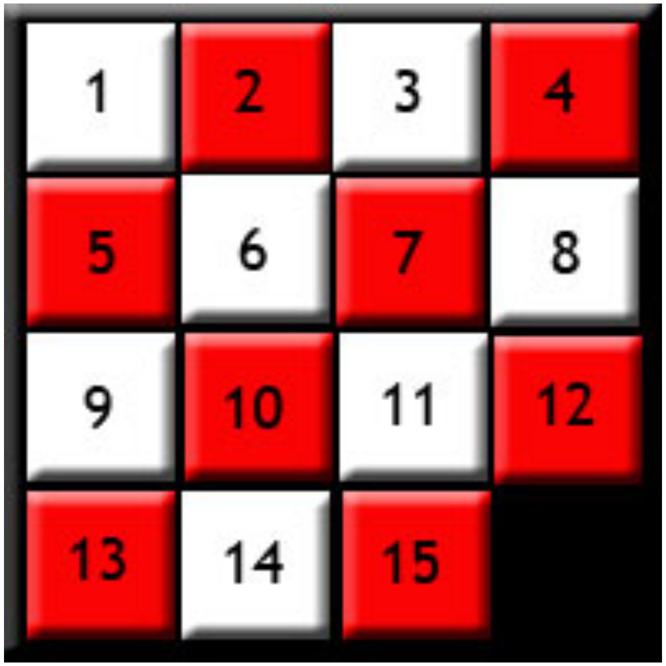
FIND a path from start to goal.

Typically want to minimize path length (or cost).

Successors Defined by Operators

- Operators (rules, moves) define how one state can be transformed into another.
- An operator has two parts:
  - **precondition**: defines the set of states to which the operator can legally be applied.
  - **effect**: defines how a state is changed when the operator is applied to it.
- Also, each operator has a non-negative cost (in this talk all operators cost 1).

15-puzzle Operators (Example)



- **DOWN(X)**: move the tile in location X down.
- **Preconditions:**
  - X ≤ 12
  - location X+4 is empty
- **Effects:**
  - the tile that was in X is now in X+4
  - X is now empty

Generic "Breadth-first" Search

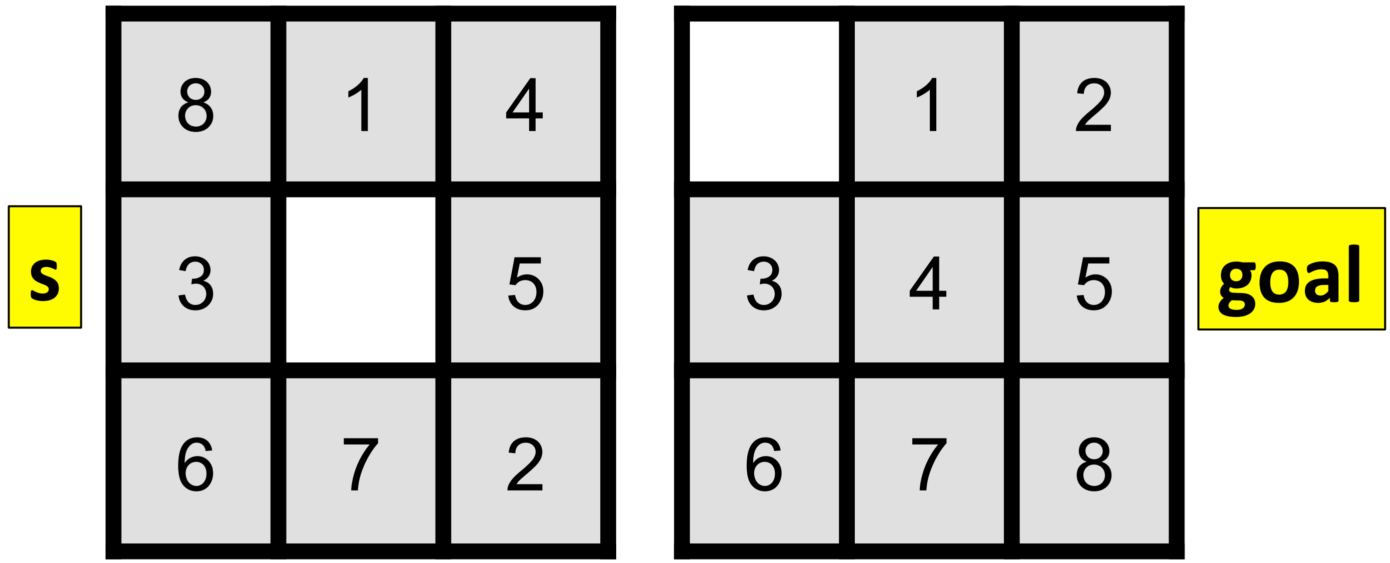
Which one?

1. Put the start state on pri
2. Repeat:
   a) If OPEN is empty, exit with fail e. O erwise…
   b) Remove a state, n, from OPEN.
   c) If n is a goal state, exit with success. Otherwise…
   d) Compute n's successors ("expand" state n)
   e) Add a successor to OPEN if it has never been seen before, or if the new path to it (via n) is cheaper than any previously generated path to it.

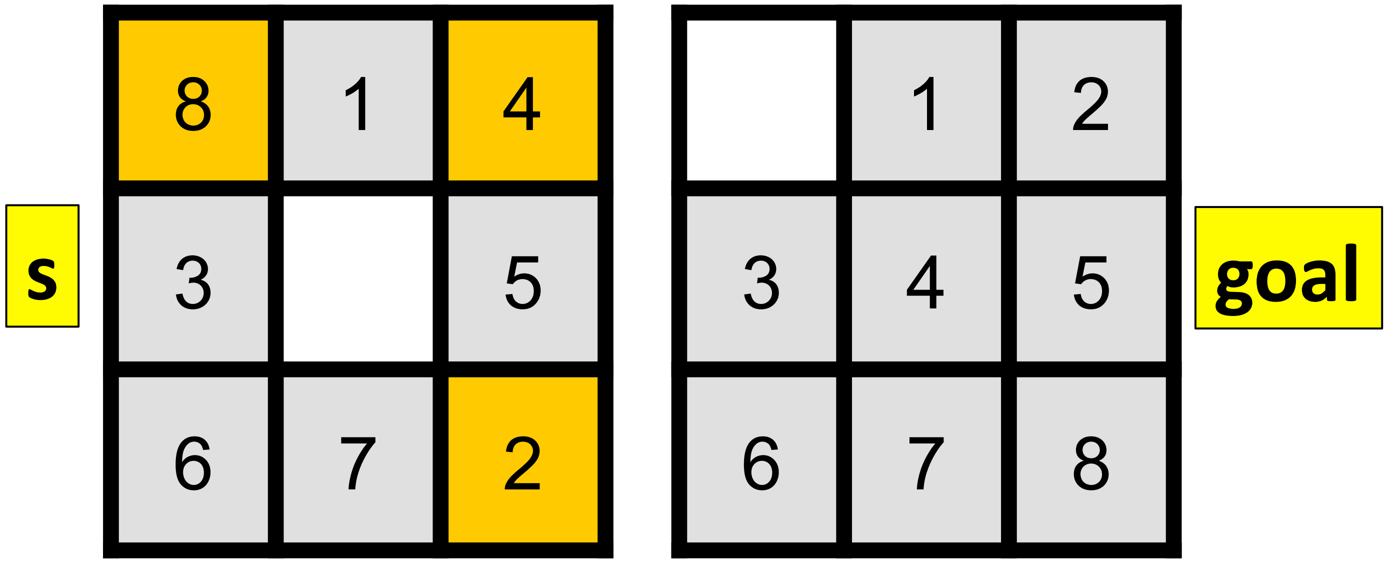
Heuristic Functions

- Like an evaluation function in a game, a heuristic function maps a state to a number indicating how promising the state is.
- In order to be sure of returning the optimal solution (least-cost path to goal), the heuristic cannot be an arbitrary evaluation function.
- An <u>admissible</u> heuristic never overestimates distance to goal ($h(n) \leq d(n,goal)$ for all n).

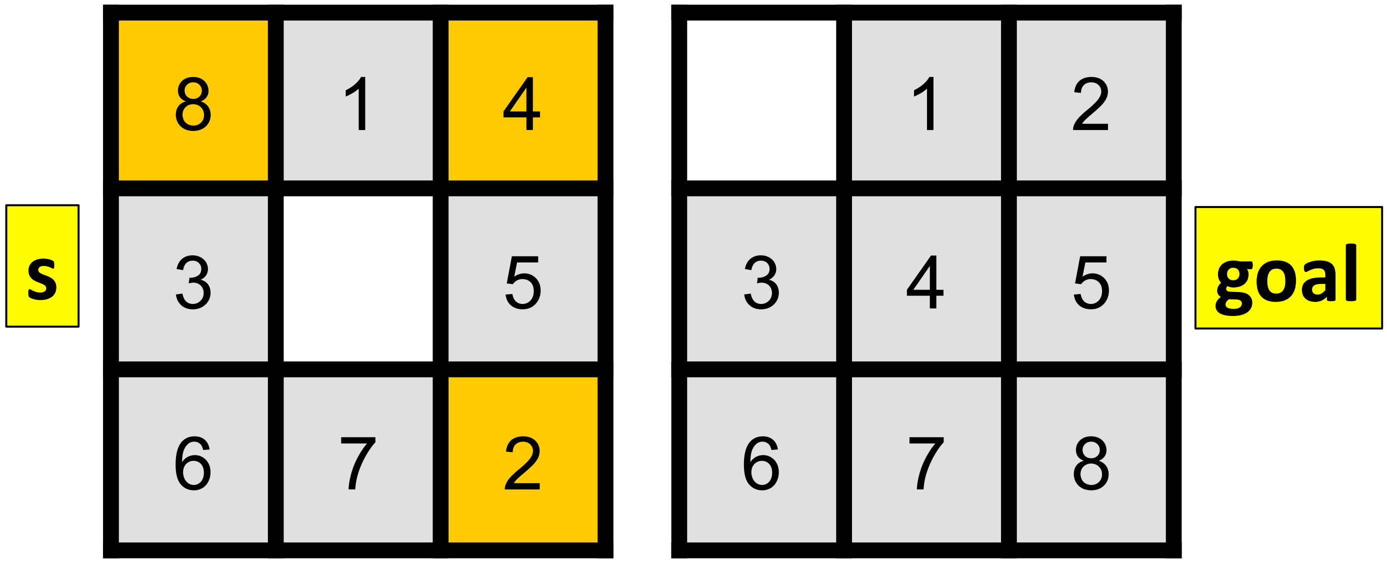
Estimate the distance from s to goal.



Misplaced Tiles = 3



Manhattan Distance = 8



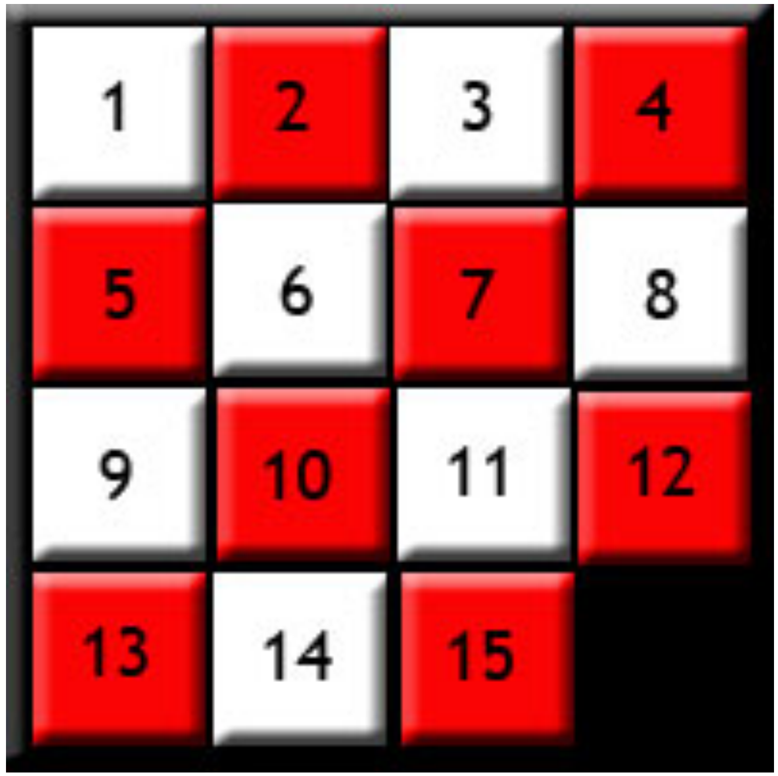
MD(8) = 4

MD(4) = 2

MD(2) = 2

MD(other tiles) = 0

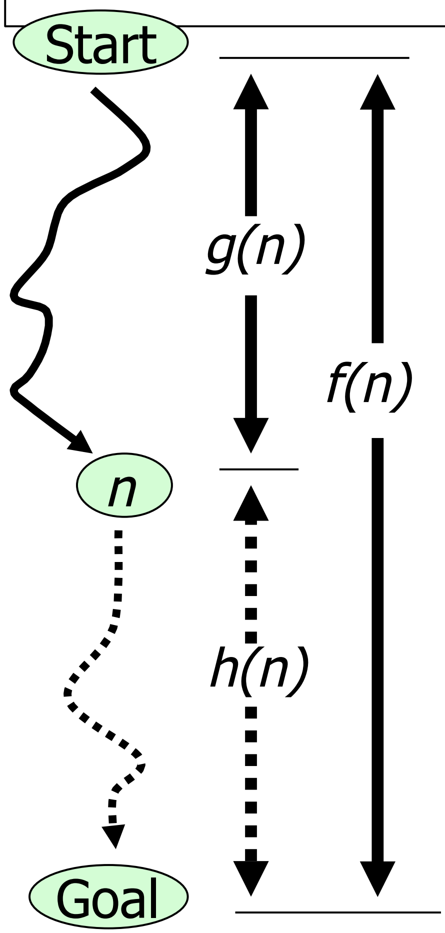
Heuristics Speed up Search



10,461,394,944,000 states

heuristic search examines 36,000

Notation



- g(n) = distance from start to node n along our current path (not necessarily optimal)
- h(n) = estimated distance from n to goal
- f(n) = g(n)+h(n) = estimated distance from start to goal via n (using our current path to n)

Generic "Breadth-first" Search

Which one?

1. Put the start state on pri
2. Repeat:
   a) If OPEN is empty, exit with failure. Otherwise…
   b) Remove a state, n, from OPEN.
   c) If n is a goal state, exit with success. Otherwise…
   d) Compute n's successors ("expand" state n)
   e) Add a successor to OPEN if it has never been seen before, or if the new path to it (via n) is cheaper than any previously generated path to it.

Which State to Remove From OPEN?

- Dijkstra's algorithm: minimum g(n)
- A*: minimum f(n)
  - A* with an admissible heuristic is guaranteed to return an optimal solution.
  - The same is true of IDA* (Iterative Deepening A*), a depth-first version of the basic algorithm that needs memory linear in the solution depth (A* can require exponential memory).

Using Abstraction to Create Heuristics

The Big Idea

Create a simplified version of your problem.

Use the exact distances in the simplified version as heuristic estimates in the original.

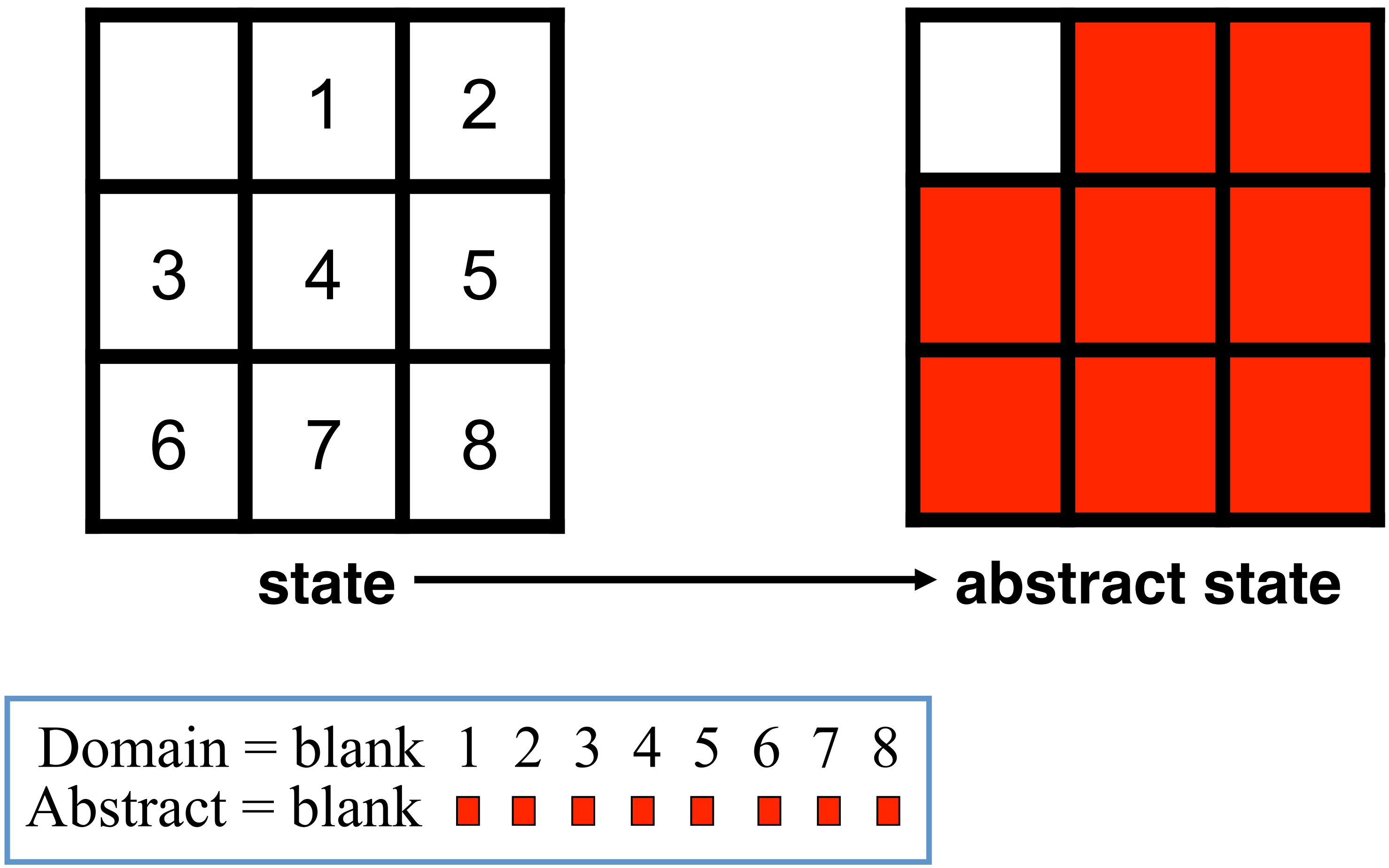
Example: 8-puzzle

	1	2
3	4	5
6	7	8

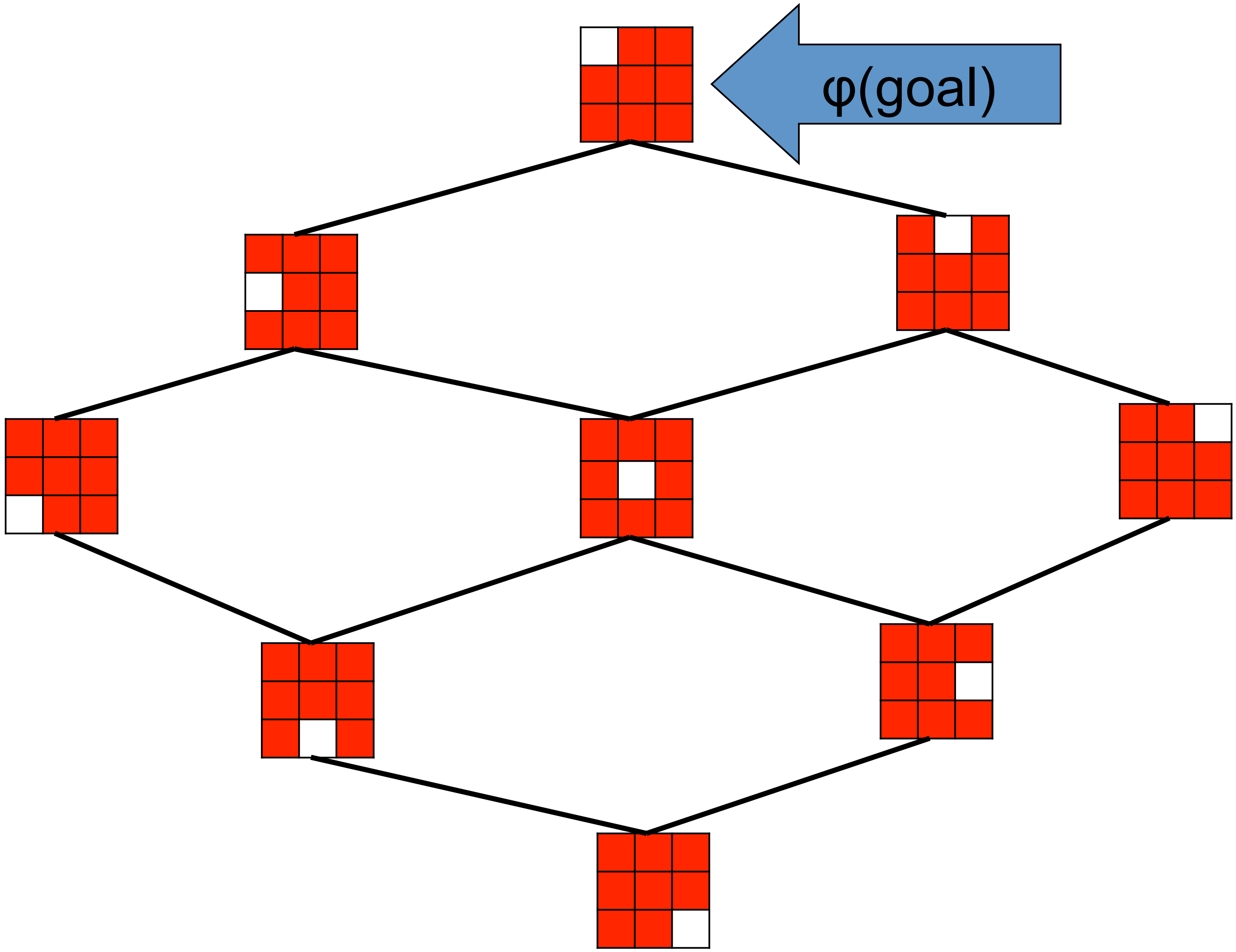
181,440 states

Domain = blank 1 2 3 4 5 6 7 8

Domain abstraction



Abstract State Space

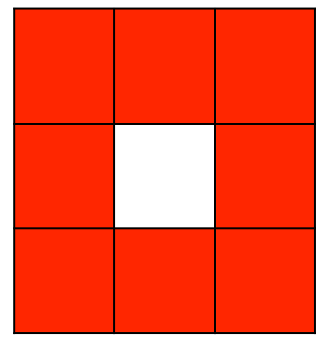


Calculating h(s)

Given a state, s

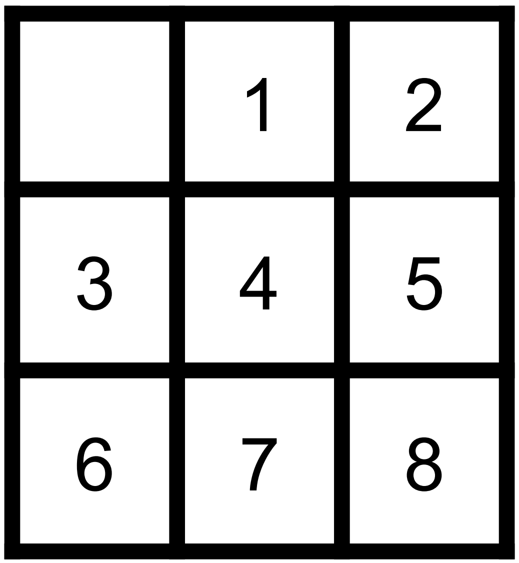
8	1	4
3		5
6	7	2

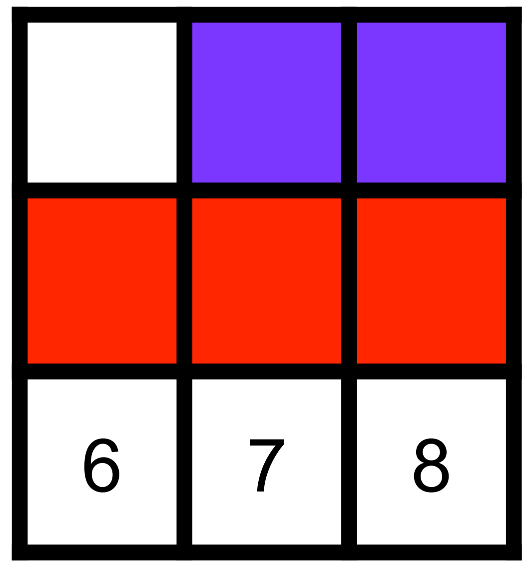
Compute the corresponding abstract state, φ(s)



h(s) = distance(φ(s),φ(goal)) = **2**

Finer-grained Domain Abstraction





Domain = blank 1 2 3 4 5 6 7 8
Abstract = blank 6 7 8

30,240 abstract states

Other Ways to Create Heuristics

- Domain Abstraction is by no means the only way to create heuristics.
- Devising new ways to estimate distances in a state space is an active research area. Recent methods include:
  - Merge-and-Shrink Abstraction
  - Cartesian Abstraction
  - Delete Relaxation (and red/black versions)
  - h^m
  - Operator-counting methods

Towards a High-Performance Compiler for State-Space Search

joint work with Neil Burch

How to Represent a State Space?

1. **Domain-specific**: write specialized code for each state space.
   – High performance (memory and time)
   – Little code re-use from one space to another
   – "procedural" representation of the successor function
2. **Domain-independent**: write the state space definition in a declarative language.

Efficiency relative to domain-specific??

PSVN

- State = vector of length N.
  - Each entry of the vector is called a state variable.
  - Each state variable has a finite domain of possible values.
- Each operator is of the form LHS => RHS.
  - LHS is the operator's precondition
  - RHS is the operator's effect
  - Both are vectors of length N. Each entry is either:
    - Constant (from the appropriate domain)
    - Variable (same variable can occur more than once)

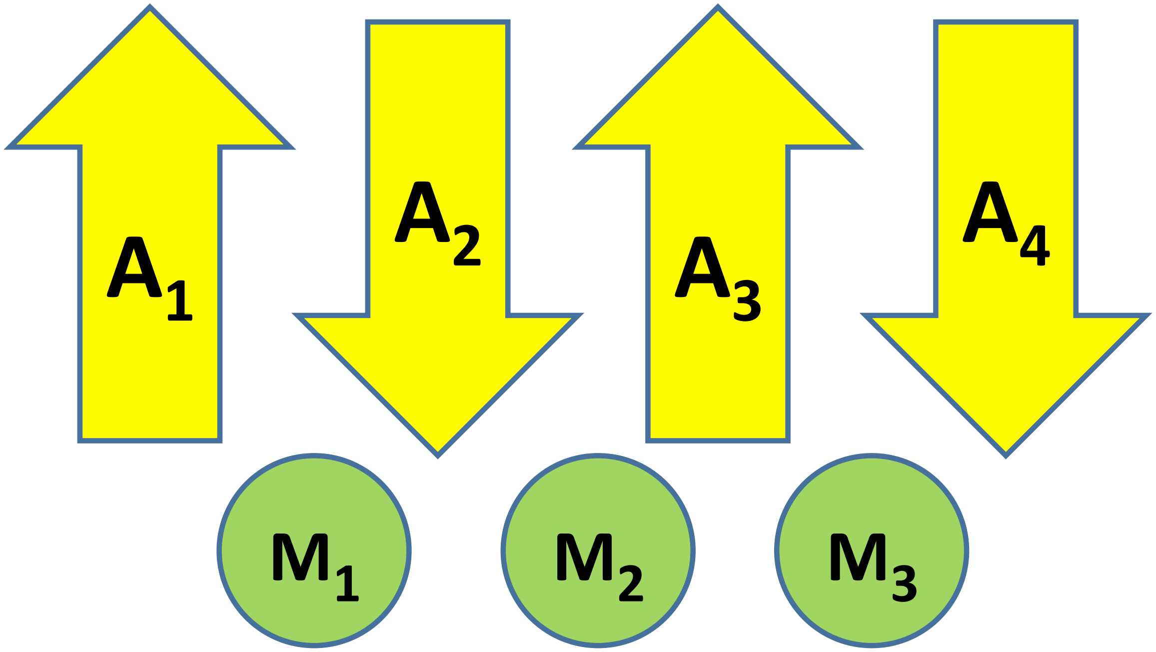
Examples (N=4)

0 **A** **B** X => 0 **B** **A** X

X **A** **A** B => **B** A A B   COST 5

In these examples, numbers are constants and letters are variables.

The 4-Arrow Puzzle



operator M_i: flip A_i and A_{i+1}

4-Arrow Puzzle, M_1 PSVN Rules

0 0 A B => **1 1** A B

0 1 A B => **1 0** A B

1 0 A B => **0 1** A B

1 1 A B => **0 0** A B

The PSVN rules for M_2 and M_3 are similar.

How to Represent a State Space?

1. **<u>Domain-specific</u>**: write specialized code for each state space.
   - High performance (memory and time)
   - Little code re-use from one space to another
   - "procedural" representation of the successor function
2. **<u>Domain-independent</u>**: write the state space definition in a declarative language.

Efficiency relative to domain-specific??

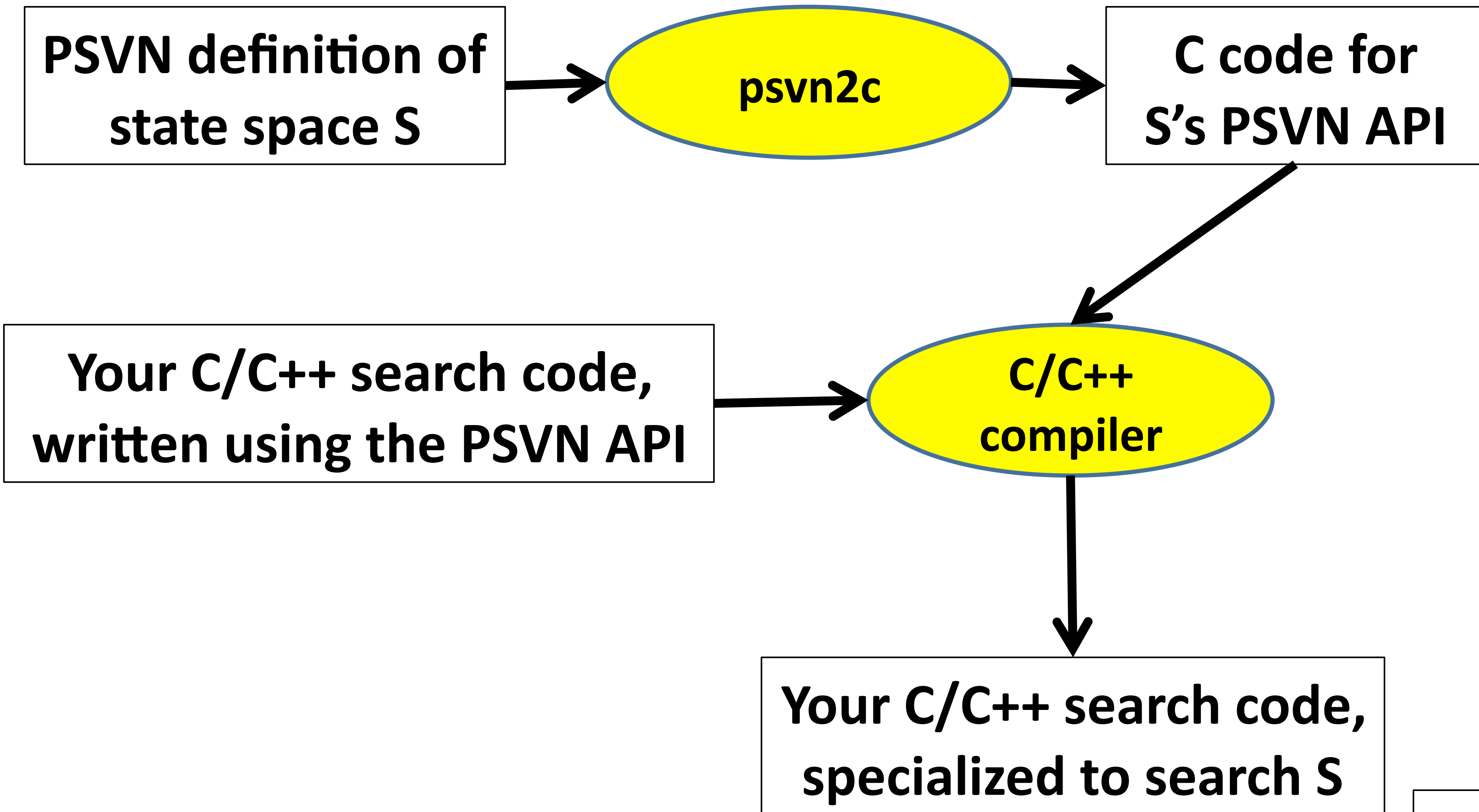
Research Goal

- Build a domain-independent state-space search system whose performance on any given domain is equal (or superior) to good domain-specific code.

  (performance = memory usage as well as run time)

- Approach: Compile (translate) PSVN to C code.

psvn2c, a Compiler for PSVN



Iterating Through a State's Children

```
init_forward_iter( iter );
while((rule=next_fwd_iter(
                iter,&state)) >= 0){
apply_fwd_rule(rule,&state,&child);
if(is_goal( &child ))
    ...
}
```

Why Am I Optimistic?

1. Compilers are capable of deeper, more complex, and more thorough analysis than (most) humans (for the part of the code the compiler is responsible for).

2. Many of the domain properties exploited by humans in writing domain-specific code can be automatically detected (and then exploited in the same way).

Knowledge of Redundant Sequences

- Rubik's Cube branching factor reduced from 18 to 13.35:

  "Since twisting the same face twice in a row is redundant, ruling out such moves reduces the branching factor to 15 after the first move. Furthermore, twists of opposite faces of the cube are independent and commutative... Thus, for each pair of opposite faces we arbitrarily chose an order, and forbid moves that twist the two faces consecutively in the opposite order."

  (Rich Korf, AAAI, 1997)

- (16,4)-TopSpin branching factor reduced from 16 to 8.9

psvn2c's analysis is more extensive, reduces it to 7.8

Goal: Automatically Eliminate Redundant Operator Sequences

- Operator sequence R is redundant with operator sequence S iff:
  1. Cost(R) ≥ Cost(S)
  2. Matches(x,R) ⇒ Matches(x,S)
  3. Matches(x,R) ⇒ R(x)=S(x)

Notation: R ≥ S means R is redundant with S.

- If we can automatically determine that R ≥ S, we can avoid duplicate effort by refusing to fully execute R – we execute all of R except its last operator ("move"), hence the name "move pruning".

Checking Single Operators

Operator R is redundant with operator S iff:

1. Cost(R) ≥ Cost(S) …… **trivial to check**
2. Matches(x,R) ⇒ Matches(x,S)

   …… **is R's LHS more specific than S's ?**

1. Matches(x,R) ⇒ R(x)=S(x)

   …… **after unifying LHS's are the RHS's identical?**

Example

(R) 0 0 1 A => 1 0 A 1

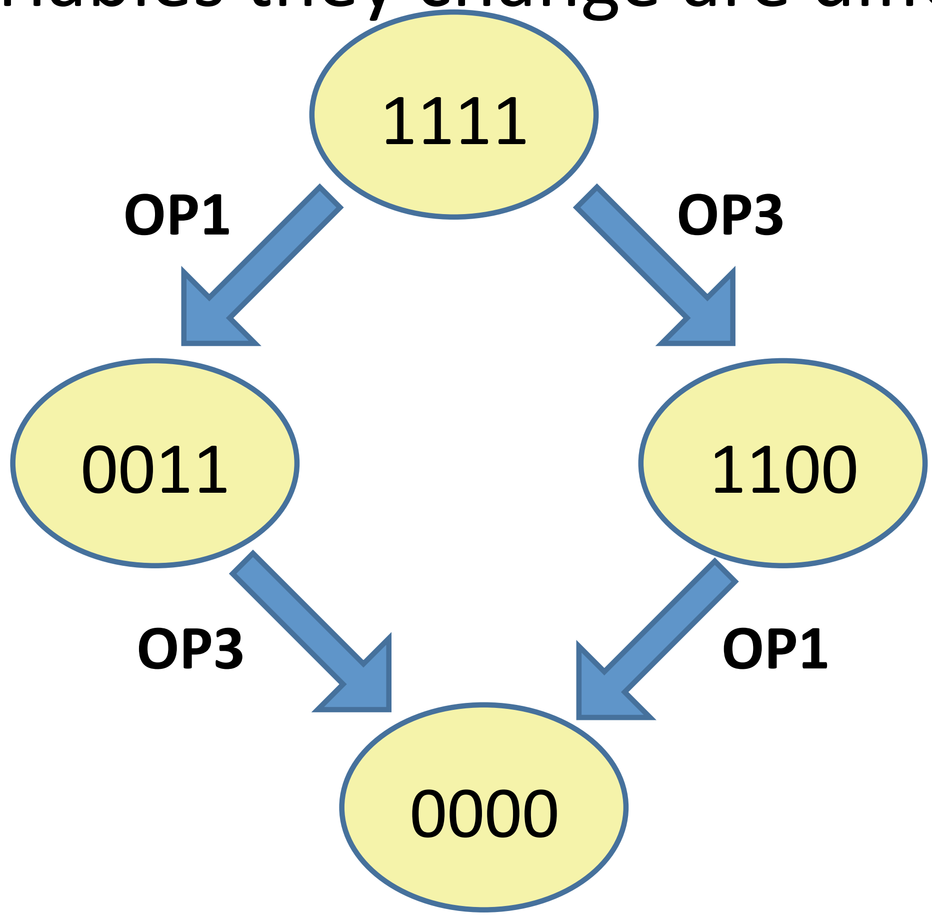
(S) W W X Y => 1 W Y X

1. Cost(R) ≥ Cost(S) …… **yes**
2. Matches(x,R) ⇒ Matches(x,S) …… **W=0, X=1, Y=A**
3. Matches(x,R) ⇒ R(x)=S(x) …… **S's RHS = 1 0 A 1**

What about operator sequences?

Example: 4-Arrow Puzzle

- OP1 and OP3 obviously are commutative since the variables they change are different.



Macro-Operators

- Any sequence of PSVN operators can be represented by a single PSVN operator ("macro-operator").
- The macro-operator's LHS represents the conditions that must be true for the entire sequence to be executed.
- Its RHS represents the net effect of applying the entire sequence of operators.
- Simple iterative "move composition" algorithm for constructing the macro-op for a sequence.

Example (4-Arrow Puzzle)

0 **0** A B => **1** **1** A B

X **1** **0** Y => X **0** **1** Y

Example (4-Arrow Puzzle)

0 0 A B => **1 1** A B

X **1 0** Y => X **0 1** Y

Macro-operator:

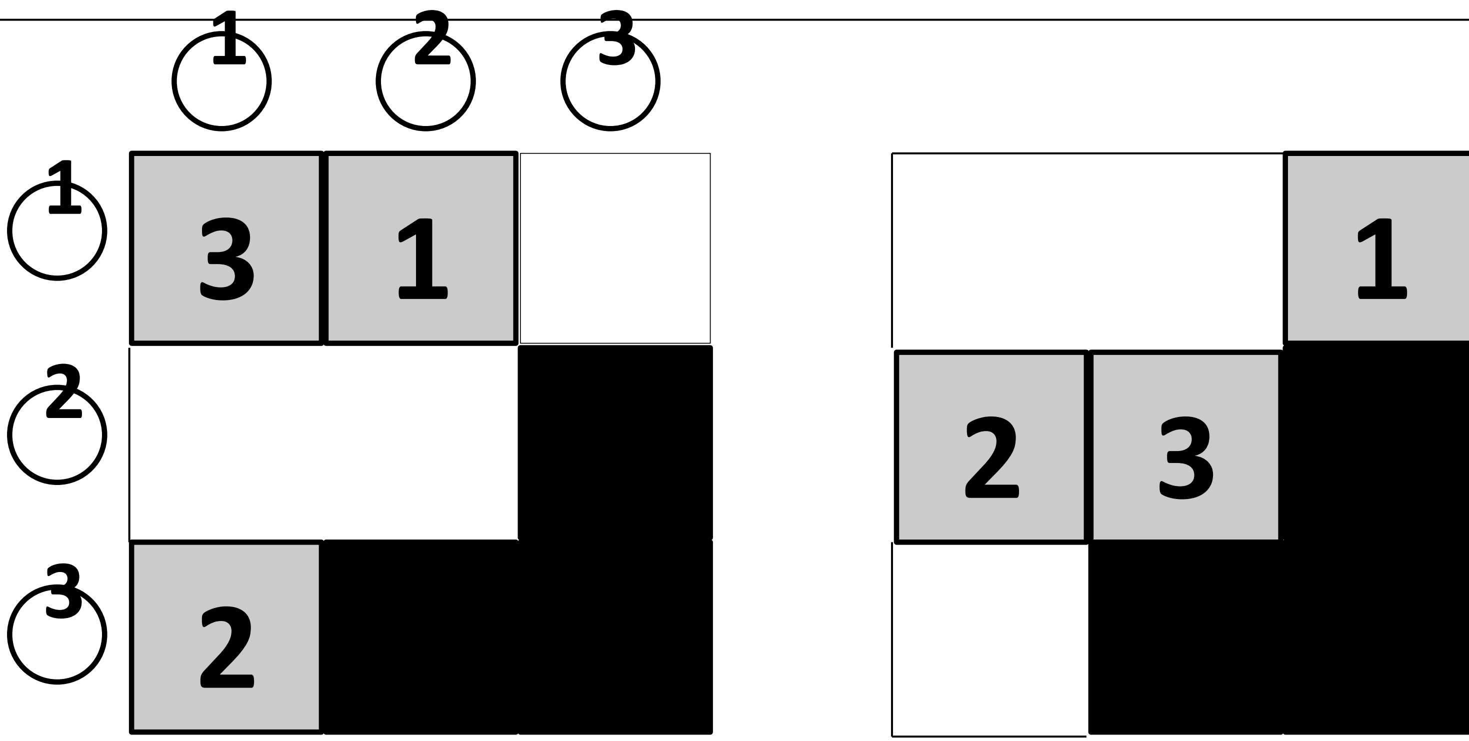
0 0 0 B => **1 0 1** B

PSVN's Move Pruning (version 1)

1. Create a macro-operator for every PSVN rule sequence length L or less.
2. Compare every macro-op (R) to every other macro-op (S):
   a) If R > S, mark R for move pruning.
   b) If R ≡ S, mark one of them for move pruning.

is this correct?

Move Pruning Gone Wrong



- There are 8 optimal solutions, and quite a few redundant operator sequences.
- If all the redundant operator sequences are eliminated, no optimal solutions remain!

What Goes Wrong?

Three of the redundancies discovered:

1. 12R-11D ≡ 11D-12R
2. 11D-12R-21R > 12R-11R-12D
3. 11R-12D-31U > 11D-21R-31

Three of the optimal solutions:

12R-11D-21R-31U

12R-11R-12D-31U

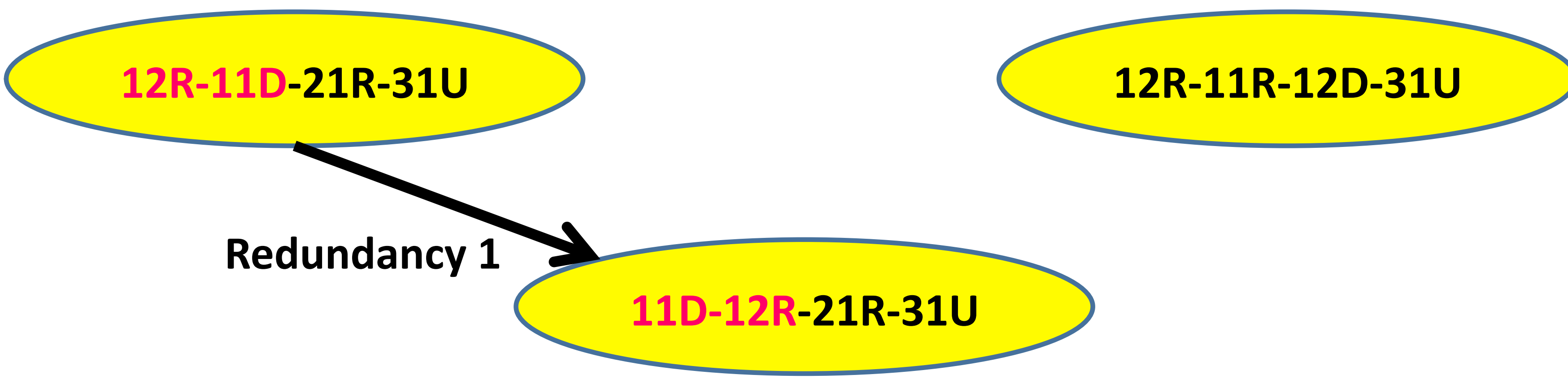
11D-12R-21R-31U

What Goes Wrong?

Three of the redundancies discovered:

1. 12R-11D ≡ 11D-12R
2. 11D-12R-21R > 12R-11R-12D
3. 11R-12D-31U > 11D-21R-31

Three of the optimal solutions:



What Goes Wrong?

Three of the redundancies discovered:

1. 12R-11D ≡ 11D-12R
2. 11D-12R-21R > 12R-11R-12D
3. 11R-12D-31U > 11D-21R-31U

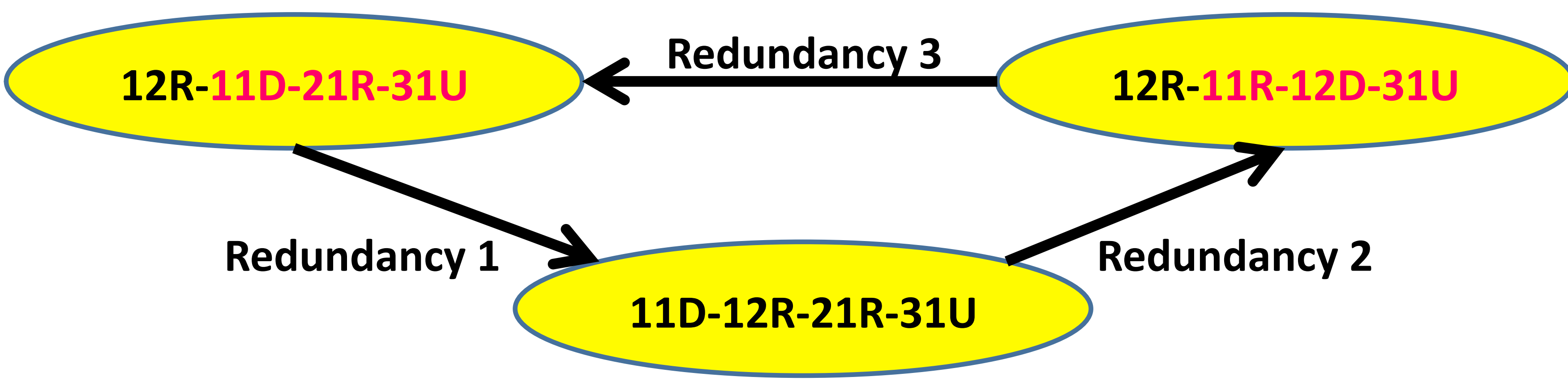
Three of the optimal solutions:

What Goes Wrong?

Three of the redundancies discovered:

1. 12R-11D ≡ 11D-12R
2. 11D-12R-21R > 12R-11R-12D
3. 11R-12D-31U > 11D-21R-31U

Three of the optimal solutions:

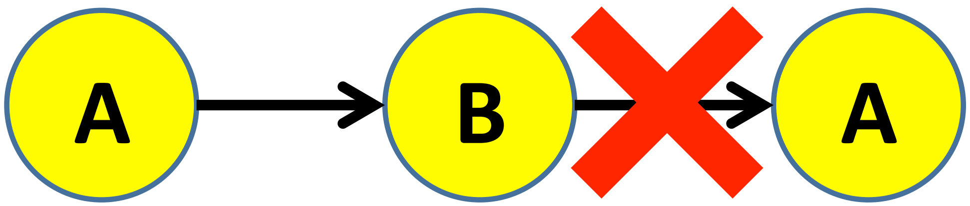


Provably Correct Solution

- Impose an order on the operators. This induces an order on the sequences (e.g. lexicographic order).
- Only allow sequence R to be pruned if it is redundant with a sequence S that is earlier in the order than R.
- This ensures no move pruning cycles exist, so at least one optimal solution will remain unpruned.

Experimental Evaluation

- Depth-first search to depth d.
- Basic version uses parent-pruning (PP).



- Compare that to a version with no PP but with move pruning (MP)
  - for sequences of length L=2 or less
  - for sequences of length L=3 or less

Results (totals over 100 states)

Domain (depth)	PP	MP, L=2	MP, L=3
16-Arrow Puzzle (15)	> 3600s	0.39s	0.39s
(14,3)-TopSpin (9)	> 3600s	53.60s	9.96s
Work or Golf (13)	> 3600s	19.76s	5.60s

Results (time required for MP)

Domain (depth)	PP	MP, L=2	MP, L=3
16-Arrow Puzzle (15)	> 3600s	0.07s 0.39s	18.58s 0.39s
(14,3)-TopSpin (9)	> 3600s	< 0.01s 53.60s	1.11s 9.96s
Work or Golf (13)	> 3600s	2.98s 19.76s	15m 4s 5.60s

Move Pruning Summary

Automatic move pruning methods…

- equal or exceed human analysis (e.g. TopSpin, Rubik's Cube, 15-puzzle, Towers of Hanoi)
- Apply to domains which are tricky to do correctly (Work or Golf)
- Even if they just accomplish parent-pruning, they are faster
- Are tricky… needed a formal proof of correctness to be sure our method was sound

Conclusion

- A compiler for state spaces can generate high-performance code.
- Automatic move pruning analysis can be of enormous benefit to algorithms based on depth-first search, and can exceed what people would do by hand.
- Rigorous formal analysis has been necessary to guarantee correctness.

Interested in trying PSVN?
Contact me – rholte@ualberta.ca

UNUSED

“Serious” Puzzles

- Pickup & Delivery (logistics) problems
- Pathfinding problems
  - GPS navigation
  - computer games
- Planning problems (find a sequence of actions that achieves a goal given the current situation)
- Edit distance
  - biological sequence alignment

Heuristics Defined by Abstraction

- An **abstraction** of state space S is any state space $\phi(S)$ such that:
  - for every state $s \in S$ there is a corresponding state $\phi(s) \in \phi(S)$.
  - $distance(\phi(s_1), \phi(s_2)) \leq distance(s_1, s_2)$.

- Exact distances in $\phi(S)$ are admissible and consistent heuristics for searching in S.

Two Research Communities

- Heuristic Search
  - technology focused (state-space search guided by a heuristic function, a lot like game-tree algorithms)
  - usually interested in optimal (least-cost) solutions
- Planning
  - task focused ("find a sequence of actions …")
  - uses a variety of technologies (e.g. SAT)
  - "satisficing", not (much) concerned with solution cost
- Historically separate, but today the best planners use heuristic search.

Example of Problem-Specific Code

- "Implementing Fast Heuristic Search Code", Ethan Burns et al. SoCS 2012
- High-performance implementation of IDA* for the 15-puzzle, based on three main ideas.

Burns et al., IDEA #1

"replace virtual method calls with C++ templates … the template instantiates our search algorithm at compile-time … all virtual method calls are replaced with normal function calls…"

Our approach largely achieves this, your search code is generic and is specialized for a domain at compile time.

Burns et al., IDEA #2

Exploit: every operator in the 15-puzzle is 1-to-1.

(operator op is 1-to-1 iff op(x)=op(y) $\Longrightarrow$ x=y)

Example (4-Arrow puzzle operator):

0 0 A B => **1 1** A B

A PSVN rule is 1-to-1 iff all the variables in its LHS occur in its RHS.

X Y A B => **1 1** A B

Burns et al., IDEA #3

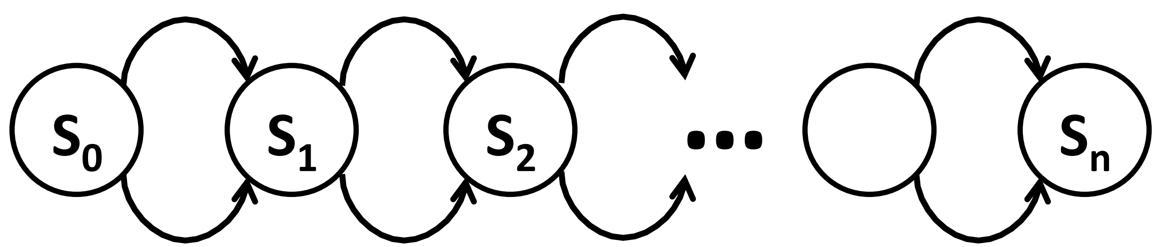
Exploit: every operator in the 15-puzzle has an inverse, and we know which operator it is.

Can we automatically determine if a PSVN rule has an inverse among the rules?

YES

Transpositions

- Transposition = two operator sequences that lead to the same state.
- For example, in the space below there are two operators that lead from S_0 to S_1.
- How many sequences are there from S_0 to S_n?



Goal: Automatically Eliminate Redundant Operator Sequences

- Operator sequence R is redundant with operator sequence S iff:
  1. Cost(R) ≥ Cost(S)
  2. Pre(R) ⊆ Pre(S)
  3. R(x)=S(x) for all x ϵ Pre(R)

Notation: R ≥ S means R is redundant with S.

- If we can automatically determine that R ≥ S, we can avoid duplicate effort by refusing to fully execute R – we execute all of R except its last operator ("move"), hence the name "move pruning".

Checking Single Operators

Operator R is redundant with operator S iff:

1. Cost(R) ≥ Cost(S) …… **trivial to check**
2. Pre(R) ⊆ Pre(S) …… **is R's LHS more specific than S's ?**
3. R(x)=S(x) for all x ϵ Pre(R)

   …… **after unifying LHS's are the RHS's identical?**

Example

(R) `0 0 1 A => 1 0 A 1`

(S) `W W X Y => 1 W Y X`

1. Cost(R) ≥ Cost(S) …… **yes**
2. Pre(R) ⊆ Pre(S) …… **W=0, X=1, Y=A**
3. R(x)=S(x) for all x ϵ Pre(R) …… **S's LHS = 1 0 A 1**

What about operator sequences?

What is State-Space Search?

GIVEN:

- Start state
- Goal state (sometimes: goal test)
- Successor function (maps a state to a set of states)

FIND a path from start to goal.

Path = sequence of states: $S_1, S_2, \ldots, S_N$ such that $S_{i+1} \in \text{successors}(S_i)$

Typically want to minimize path length (or cost).