ST. FRANCIS XAVIER
UNIVERSITY

CSCI-564
CONSTRAINT PROCESSING AND HEURISTIC SEARCH

LECTURE 12 – STATE SPACE PRUNING

Dr. Jean-Alexis Delamer

Recap

- Most of problem can be represented as state space problem.
- We can use a graph representation to search and find the optimal solution.
  - Most algorithms have a high time complexity.
- We saw that the graph can be too large to be stored in memory.
  - It is possible to prune the search tree to limit the memory consumption.

State Space Pruning

- An effective way to tackle large problem is to prune the search tree.
  - Works in a memory constraint environment.
  - Intuitively works in time constraint environment.

- Why are we pruning?
  - Some branch might not lead to a goal state.
  - Some lead to inferior solutions
  - Some are redundant.

State Space Pruning

- State space pruning reduce the node branching factor.
  - Less successor nodes have to be analyzed.
- Basically, a smaller part of the state space is generated (explored).
  - Saves both time and space.
- However, there might be a trade-off between time and space complexity.
  - Complex data structures implies time effort to maintain it.

State Space Pruning

- Most pruning techniques relies on observed regularities in the search space.
  - You can exploit patterns to reduce the search efforts.

- Patterns can be provided by a domain expert.
- Or it can be detected/learned automatically.

State Space Pruning

- Static pruning techniques detect pruning before executing the main algorithm.
- In other cases, the pruning rules need to be inferred during execution.
- They can be combined.
  - Layered search algorithms.
    - Top-level search for problem solutions
    - While lower-level refines the pruning knowledge.

State Space Pruning

- When pruning the state space, it's important to make sure to not prune important branches.

- Similar to heuristics, we need to show that a pruning rule is admissible.

- What is an admissible pruning rule?
  - A pruning rule is admissible if at least one optimal solution will be reachable from the initial state.

State Space Pruning

- A pruning rule is solution preserving if there exists at least one path from the initial state to the goal in the reduced state space.

- Admissible pruning strategies and admissible heuristics (estimates) are fundamentally different concepts.
  - But together they help to overcome time and space complexity.

Admissible State Space Pruning

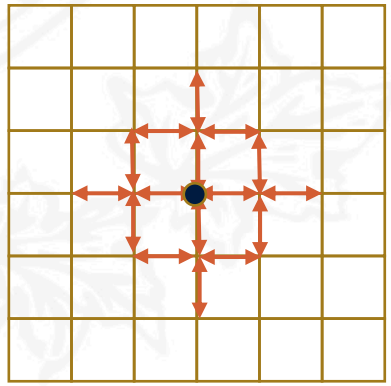
- Definition (Admissible Pruning):
  - Technique that reduces the branching factor a state space problem while preserving the existence of optimal solutions.

- Algorithms like A* or IDA* should be able to find the optimal solutions even if you're using an admissible pruning.

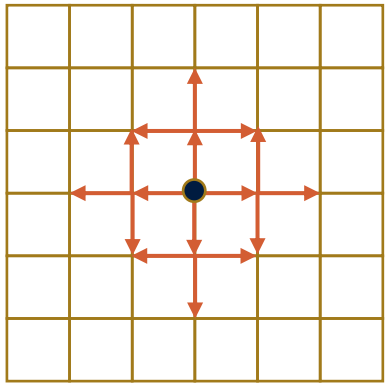
- Most algorithms used for memory constrained environment implements a basic form of pruning.
  - Remember, the pruning of the search tree with upper bounds

Admissible State Space Pruning

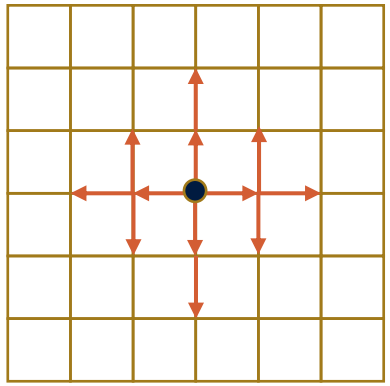
- Example



An agent can move in four directions without restriction. (No pruning)



An agent can't use the inverse action. (Prunning based on predecessor)

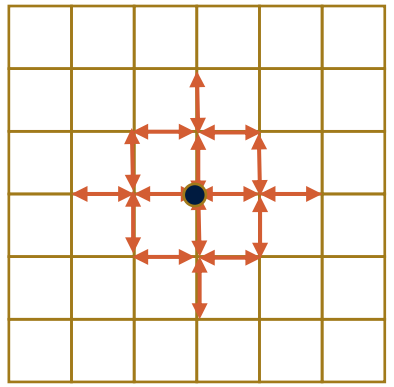


We add more pruning rules. (Pruning based on a set of rules)

Admissible State Space Pruning

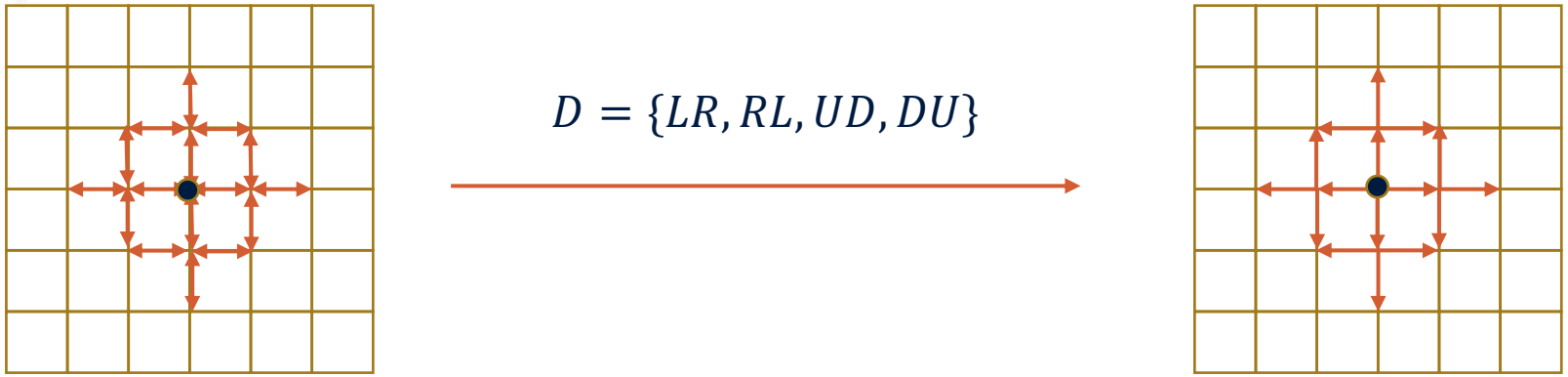
- Most combinatorial problems are described implicitly.
  - The state space problem is considered in a labeled representation.
  - Let Σ the set of different action labels.

- Example
  - Gridworld problem
  - $\Sigma = \{U, D, L, R\}$



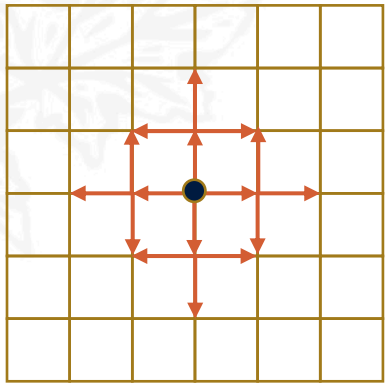
Admissible State Space Pruning

- We can use substring pruning.
  - Prunes paths from that contain one forbidden words $D \in \Sigma^*$.
  - These words are called duplicates.
  - It is known a priori that the same state can be reach trough a shortcut.
    - Shorter action sequence.

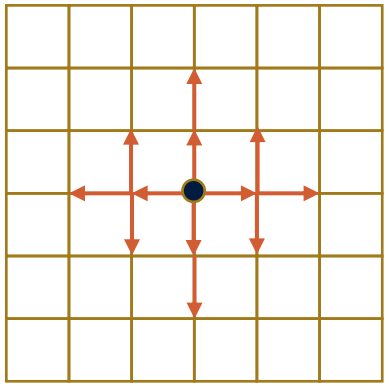


Admissible State Space Pruning

- We can distinguish shortcuts and duplicates.
- We create a lexicographical ordering $\leq_l$.



Reject the following duplicates
$D = \{DR, DL, UR, UL\}$



Admissible State Space Pruning

- How can we find pairs (d, c) that don't reject any optimal solution?
  - where d is a duplicate and c a shortcut

- Run BFS
  - Records all states in a hash table
  - When a duplicate is encountered (conflict in the hash table), the largest sequence of action is recorded as a duplicate.
- In undirected search spaces
  - Looking for cycles
  - Compare a newly generated node with all nodes in the generating path
  - The cycles is split in cycles and duplicates

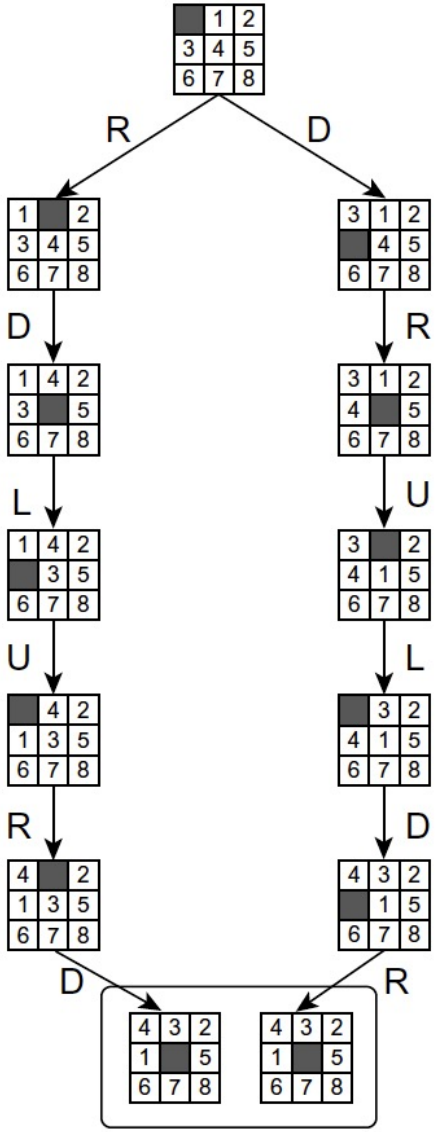
Admissible State Space Pruning

- Examples $(n^2 - 1)$-puzzle:
  - When splitting a full-length cycle, one part must be inverted (all its actions have to be reversed).
    - $RDLURDLURDLU$ with inverse $DRULDRULDRUL$
  - If they have the same size, we need more criteria.

- Create a lexicographic order on the set of actions $\Sigma = \{U, D, L, R\}$
  - Let $U \leq_l R \leq_l D \leq_l L$
  - We impose L to be inverse to R, noted $L^{-1} = R$
  - U inverse to D

Admissible State Space Pruning

First Part (Duplicate)	Second Part	Inverse (Shortcut)
RDLURDLURDL	ϵ	
RDLURDLURDL	U	D
RDLURDLURD	LU	DR
RDLURDLUR	DLU	DRU
RDLURDLU	RDLU	DRUL
RDLURDL	RDLUR	DRULD
RDLURD	RDLURD	DRULDR



Admissible State Space Pruning

- It can be extended to weighted search spaces.
- Definition (Pruning pair):
  - Let G be a weighted problem graph with action label set Σ and weight function $w: \Sigma^* \to \mathbb{R}$. A pair $(d, c) \in \Sigma^* \times \Sigma^*$ is a pruning pair if:
    1. $w(d) > w(c)$, or, if $w(d) = w(c)$, then $c \leq_l d$;
    2. $\forall u \in S$: d is applicable in u if and only if c is applicable in u; and
    3. $\forall u \in S$ we have that applying d on S yields the same result as applying c on S, if they are both applicable; that is, $d(u) = c(u)$
- Example:
  - In the gridworld problem, if in c the agent moves a larger distance in x- or y-direction than in d, it cannot be a shortcut.

Pruning Automata

- Assuming we don't need to check condition 2 and 3.
- Searching for a duplicate in the current search tree path can slow down the exploration.

- Finding any of a set of strings in a text is the bibliographic search problem.
  - It's solved by constructing a substring acceptor.
  - It's a finite state automaton.

Pruning Automata

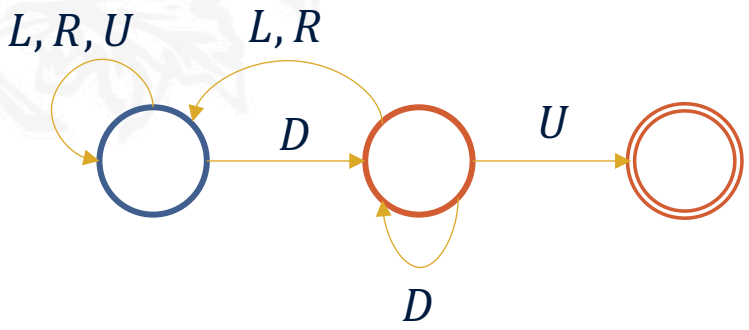
- The automatons runs synchronous to the exploration and prunes the tree if it reaches an accepting state.
- Each state transition induces an automaton transition.

- It's important for algorithm such as IDA*.
  - The time to generate one successor for these algorithms is already reduced to a constant.

Pruning Automata

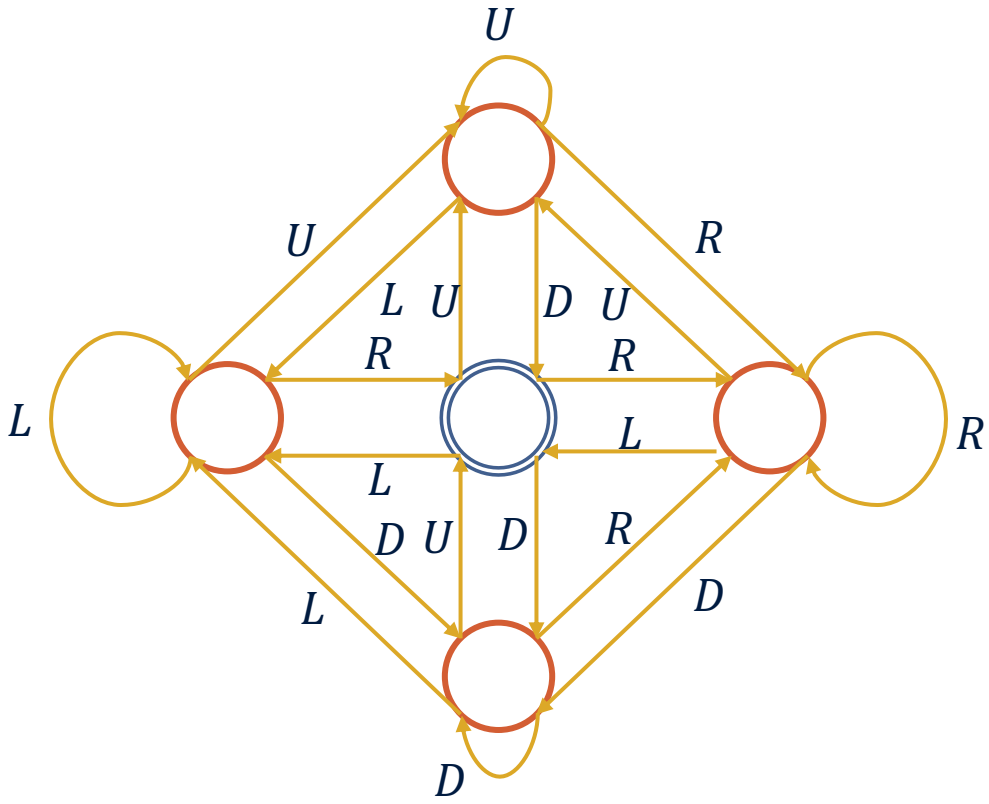
- In the gridworld we want to prune the actions sequences : DU, UD, LR, RL.

- An automaton for DU:



Pruning Automata

- In the gridworld we want to prune the actions sequences : DU, UD, LR, RL.
- The full automata:



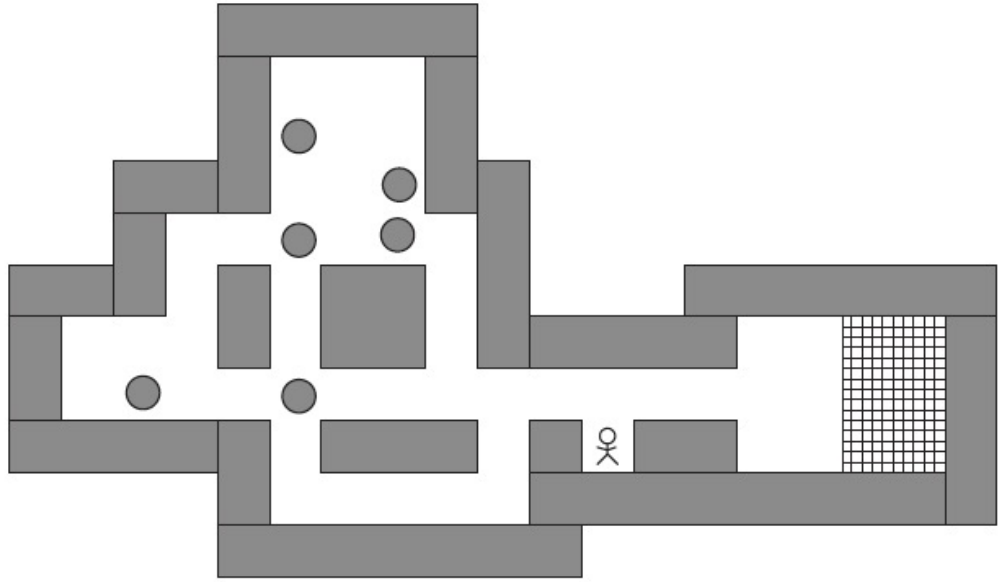
Pruning Automata

- A pruning strategy cut off branches in the search tree.
- It reduces the average node branching factor.

Puzzle	Construction	Duplicate	States	Without	With
Eight-Puzzle	BFS	35,858	137,533	1.732	1.39
Fifteen-Puzzle	BFS	16,442	55,441	2.130	1.98
Fifteen-Puzzle	CYC	58,897	246,768	2.130	1.96
Twenty-Four-Puzzle	BFS+CDBF	127,258	598,768	2.368	2.235
2^3 Rubik's Cube	BFS	31,999	24,954	6.0	4.73
3^3 Rubik's Cube	BFS	28,210	22,974	13.34	13.26

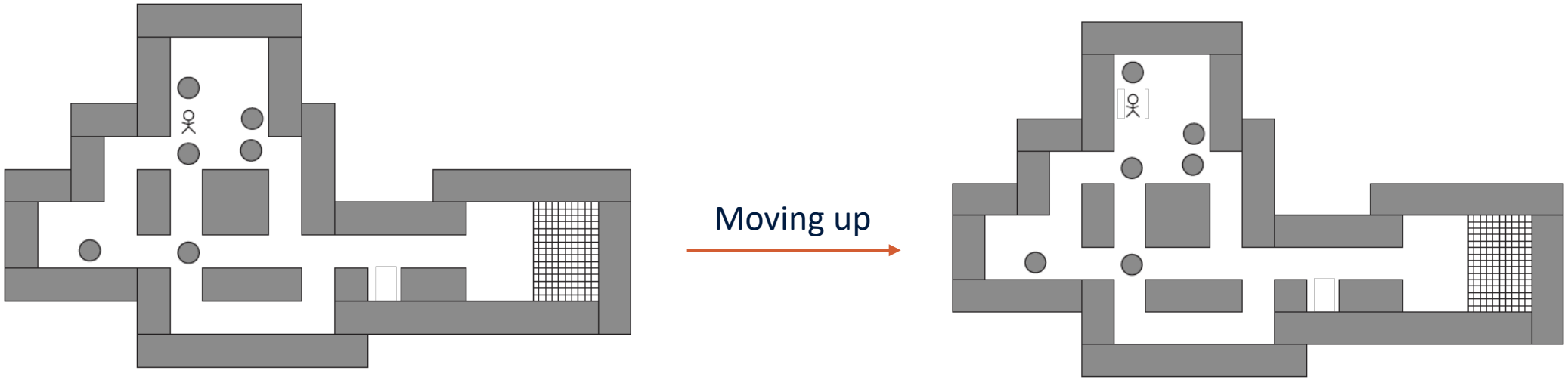
Pruning Dead Ends

- In a domain like the $(n^2 - 1)$-puzzle every state that you can reach by a sequence of actions remains solvable.
- In several domains, it's not the case.
  - Can you think of a problem?



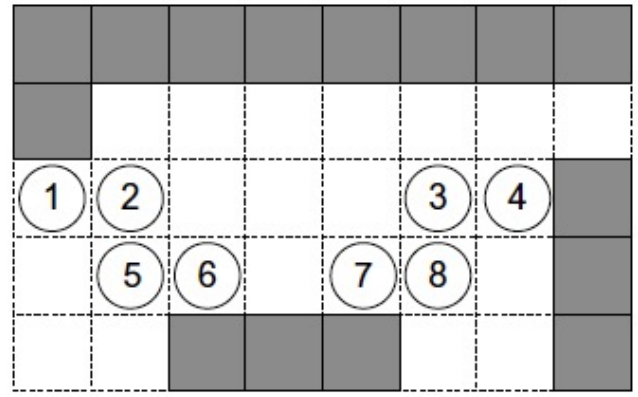
Pruning Dead Ends

- In those problems some actions cannot be reversed.
  - We call it a dead-end.



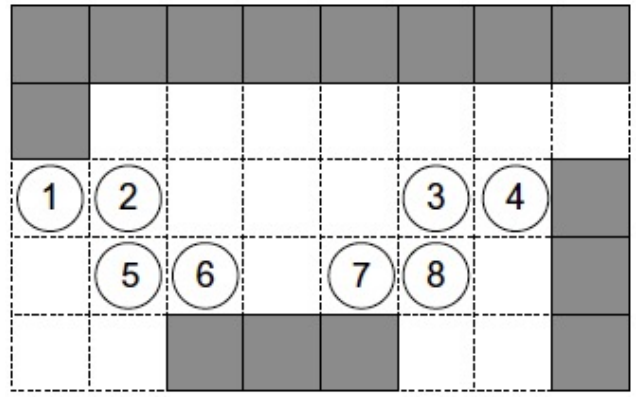
Pruning dead ends

- The goal of dead-end pruning is to detect these branches.
- A ball is considered free if it has either no horizontal neighbor or no vertical neighbor.
- Is it sufficient?
  - No. Some are not free at first but can be freed.



Pruning dead ends

- Initially, place all nonfree balls into a queue.
- Iteratively try to remove the nonfree balls from the queue until it becomes empty, or nothing changes anymore.
- If in one iteration every ball stays inside the queue, some of which are not located on goal fields, the position is a dead-end.
- Complexity:
  - In the worst case $O(n^2)$.
  - In most cases, linear.



Pruning dead ends

- The objective is to learn and generalize dead-end positions.
  - It's also called bootstrapping.
- Each dead-end found is stored and used to prune the tree.

- Depending on the given resources and heuristics.
  - The dead-end can invoked in every expansion step.
  - Occasionally.
  - In critical situation.

Pruning dead ends