Artificial Intelligence 4 Games

Pathfinding Overview

2020

Jakub Kowalski, University of Wrocław

Pathfinding

Basic Formulation

- ➔ Given a graph, a start node, a goal node
- ➔ Our task is to find the shortest (cheapest) path from the start to the goal
- ➔ In more general case, edges have weights
- ➔ Many more sophisticated variants

Applications

All types of movement:

- ★ Units in strategy games, NPCs in role playing games, all entities in simulation/tycoon games, enemies in FPS, ...

Basic Algorithms

- ★ Breadth First Search
- ★ Dijkstra
- ★ Best-First Search
- ★ A*

Uninformed Search

Breadth First Search (BFS)

- ➔ We expand equally in all directions
- ➔ The edges need to have uniform weights
- ➔ Worst case performance $O(|V|+|E|)$
- ➔ Complete (always finds the path) and optimal (found path is the best one)

Dijkstra's algorithm

- ➔ We expand lower cost paths first
- ➔ Works with weighted edges
- ➔ Worst case performance $\Theta(|V|\log|V|+|E|)$
- ➔ Complete and optimal

Informed Search

Best-First (Greedy) Search

- ➔ We use heuristic - estimation of how far is from a given node to the goal. E.g. on a plane - the straight line distance
- ➔ Expand nodes that “should be” closest to the goal
- ➔ Mostly get to the goal fast using some nonoptimal path
- ➔ In the worst case it can explore the whole graph like a badly guided DFS
- ➔ Requires cycle checking to be complete

A*

- ➔ We combine uniform-cost approach (“backward cost” ***g(x)***) with the best-first approach given by the heuristic ***h(x)***
- ➔ Thus, the A* algorithm use the ordering based on ***f(x) = g(x) + h(x)***.
- ➔ Works similar as Dijkstra
(if h(x)=0 then it simply is Dijkstra)
- ➔ Complete
- ➔ Optimal with a proper heuristic

A*

Pseudocode

- *closedSet* := {}
- *openSet* := {START}
- g(x) = 0 **if** x==START, **otherwise** ∞
- f(x) = h(x) **if** x==START, **otherwise** ∞
- **while** *openSet* **is not** empty:
  - *current* := *openSet*.dequeue_lowest_f()
  - **if** *current*==GOAL: **return** Success
  - *closedSet*.add(*current*)
  - **for each** *neighbor* **of** *current*:
    - **if** *neighbor* **in** *closedSet*: **continue**
    - *varf* := g(*current*)+ cost(*current*, *neighbor*) + h(*neighbor*)
    - **if** *neighbor* **not in** *openSet* **or** f(*neighbor*) > *varf*:
      - *openSet*.add(*neighbor*)
      - f(*neighbor*) = *varf*
  - **return** Failure

Heuristic

The heuristic function has to be admissible (optimistic), i.e. ***h(x) <= true_cost(x, goal)***.

To ensure that A* finds optimal solution, the heuristic function has to be consistent (monotone), i.e. ***f*** is not decreasing along any path: ***h(x) <= true_cost(x, x') + h(x')***

All consistent heuristics are admissible.

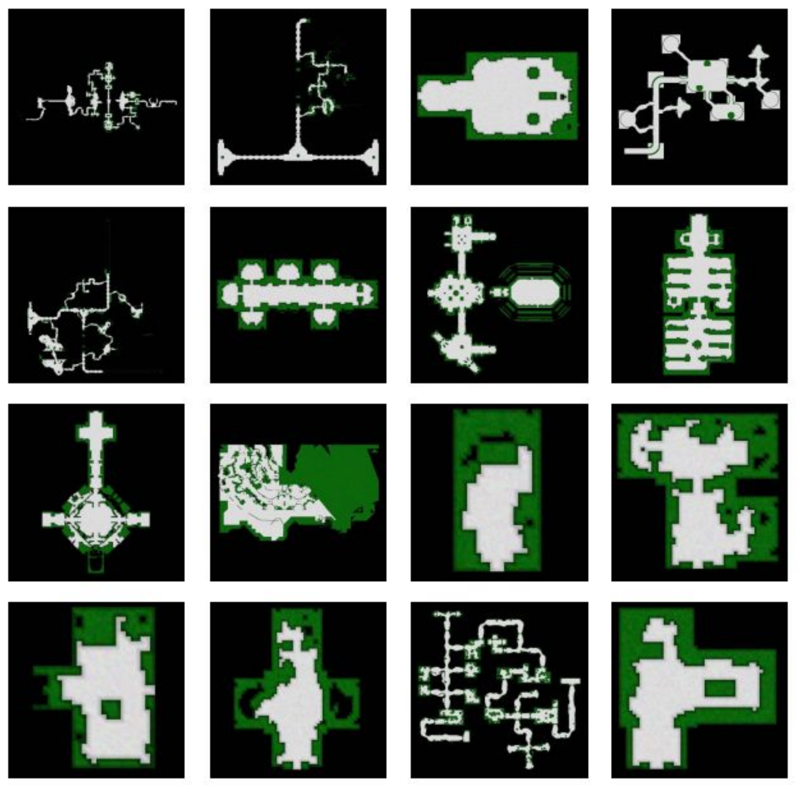
Pathfinding in Games

Literature

★ Botea, A., Bouzy, B., Buro, M., Bauckhage, C., Nau, D. Pathfinding in games. Dagstuhl Follow-Ups, vol 5, pp. 21-31, 2013.
★ Abd Algfoor, Z., Sunar, M. S., Kolivand, H. A comprehensive study on pathfinding techniques for robotics and video games. International Journal of Computer Games Technology, 2015.
★ Sturtevant, N., GPPC: Grid-Based Path Planning Competition: http://movingai.com/GPPC/

Pathfinding in Games

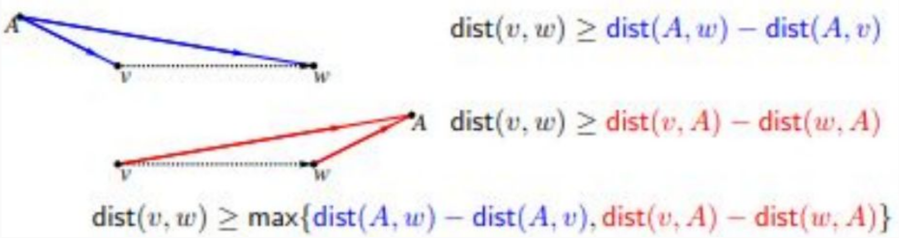
- hierarchical planning,
- non-trivial heuristic functions,
- dynamic changes in the environment,
- multiple targets,
- multi-agent pathfinding,
- adversarial pathfinding,
- various types of terrains and mobilities of the units,
- incomplete information,
- real-time constraints, memory constraints,
- inventory-driven pathfinding,
- various graph types (grid, hex, navmesh),
- interpretation: doorways are nodes / doorways are edges,
- …



Landmark Heuristics

ALT

- ➔ We choose a small set of landmark points
- ➔ And precompute distances between each node and landmark
- ➔ We can use those distances as a heuristic, as the maximum distance difference over a subset of landmarks is admissible



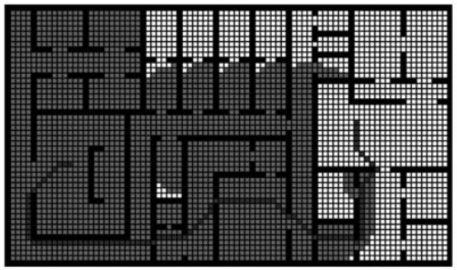
$\mathrm{ALTBest}_P$

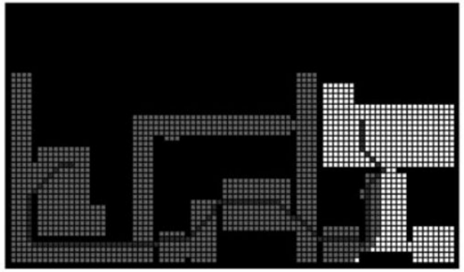
- ➔ We have a predefined set of *P* landmarks,
- ➔ During a search we choose a single landmark that gives the highest ***h*** value for the root node (initially, the start point)
- ➔ The heuristic is maximum of the ALT value for the selected point and the Manhattan heuristic
- ➔ $\mathrm{ALTBest}_P$ is worse in terms of quality than ALT, but it is cheaper per node.

Room-based Heuristics

Dead-end heuristic

- ➔ In preprocessing phase, we decompose map into disjoint areas
- ➔ During a search query, we start with identifying and removing from consideration irrelevant areas (by setting the heuristic values to ∞)

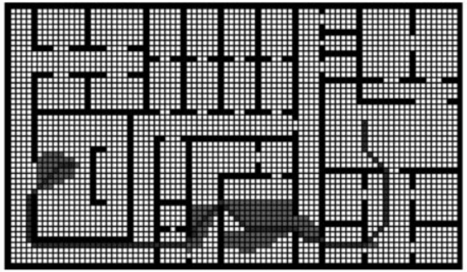




Gateway heuristic

- ➔ We decompose map into areas, borders between areas form gates
- ➔ Then we precompute distances between all gates
- ➔ And use gate distance within a heuristic

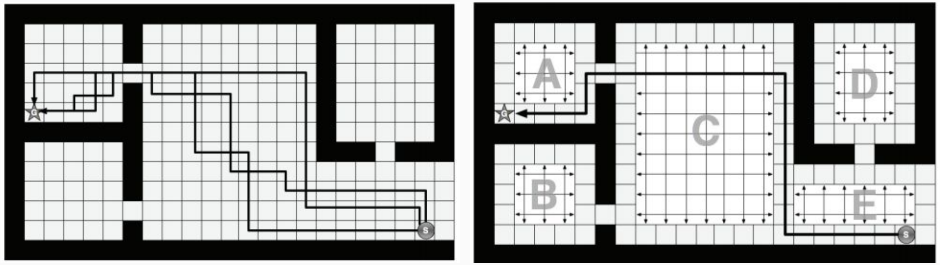
$$h^G(n,g) = min \sum_i \sum_j h^l(n,G_i) + H(G_i,G_j) + h^l(G_j,g)$$



Symmetry Elimination

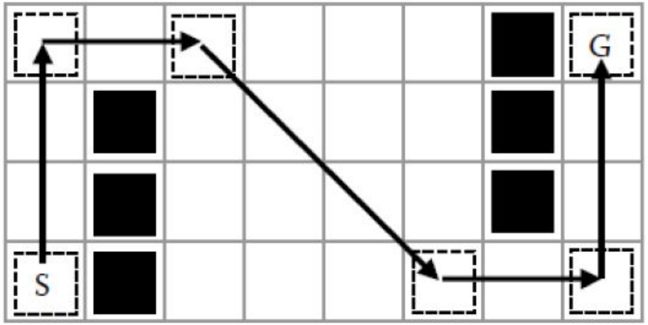
One of the main problems with A* pathfinding on grid maps is related to open spaces, where exist exponential number of optimal paths.

This can be solved by e.g. decomposing maps into obstacle-free rooms and pruning all nodes except the ones on the perimeter.



JPS

- Ultrafast A* improvement over the uniform cost octile grids
- Uses successor-pruning technique to remove redundant paths from consideration
- Replace neighbors with further away nodes and jumps directly to them omitting opening intermediate nodes



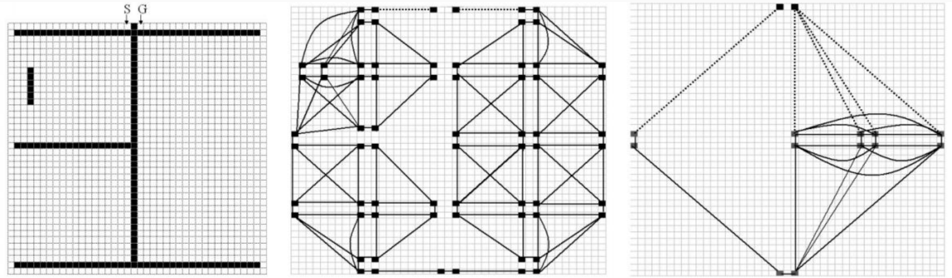
Hierarchical Pathfinding

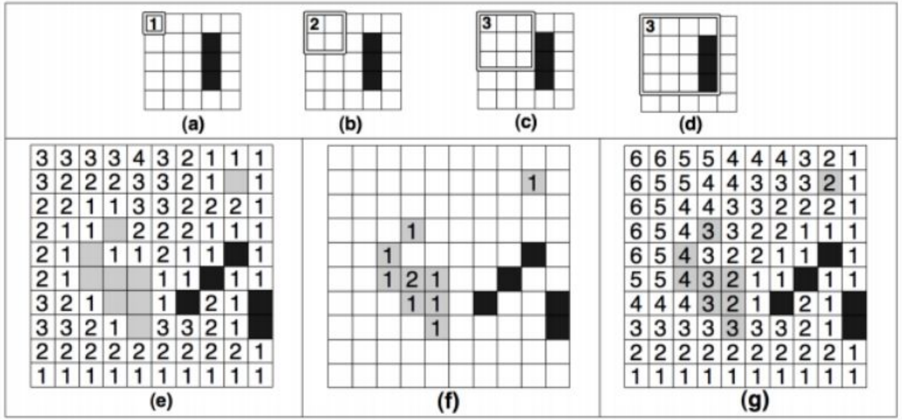
HPA*

- ➔ In preprocessing step we decompose map into disjoint square sectors
- ➔ We put some entrances on the edges, and compute true distances between entrances within one sector
- ➔ Pathfinding uses A* on the abstract graph,
- ➔ Refinement step to smooth the path

HAA*

- ➔ Extension of HPA*
- ➔ Allow several types of terrain
- ➔ Handles units of variable sizes and different terrain traversal capabilities





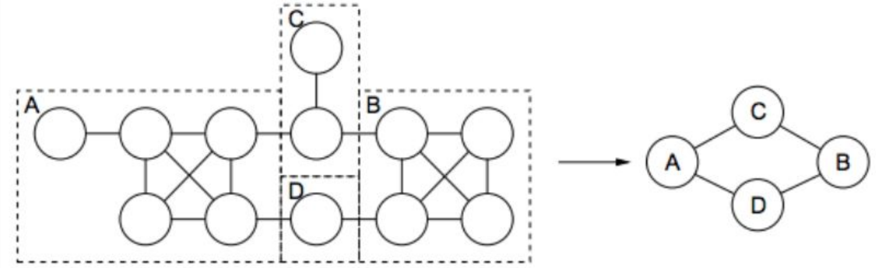
Hierarchical Pathfinding

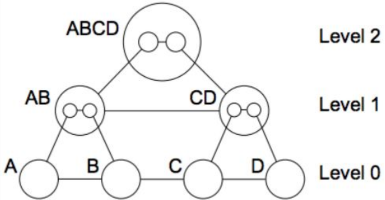
PRA*

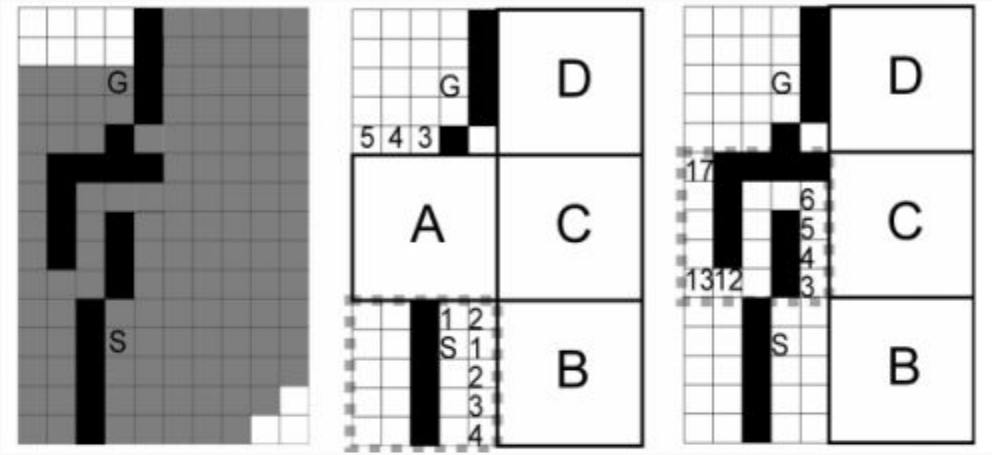
- ➔ Builds bottom-up hierarchical representation by merging small fully connected regions
- ➔ We search top-down, common parent mean path between regions

Block A*

- ➔ We precompute a database of all m×n block topologies
- ➔ And for each block compute distances between all boundary nodes







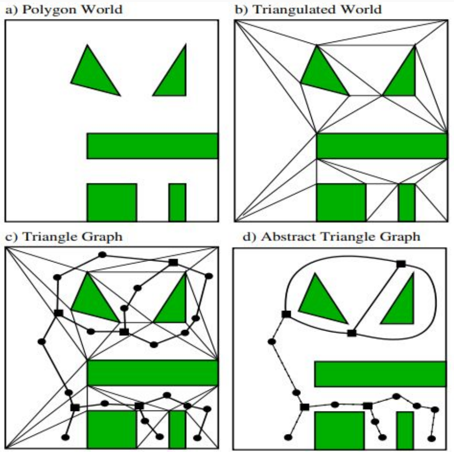
Triangulation-based Environments

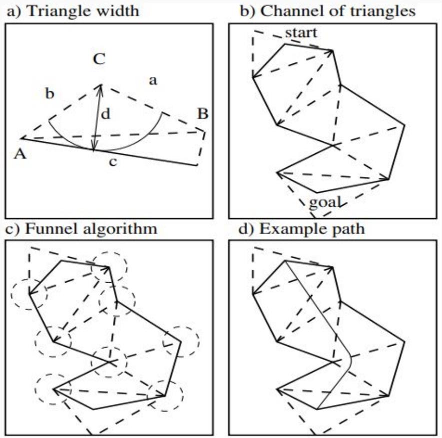
TA*

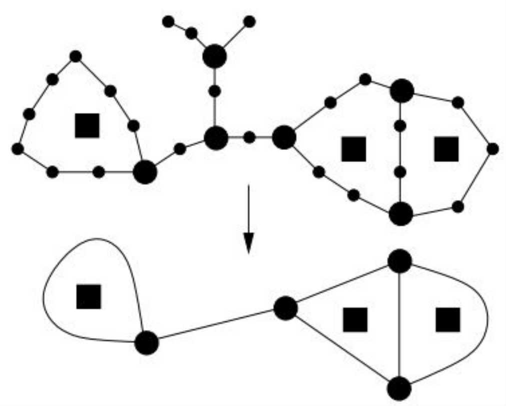
- ➔ Maps are represented with obstacles defined by polygons
- ➔ Uses A* like algorithm on graphs induced by the triangulation
- ➔ Finds optimal any-angle paths for circular objects

TRA*

- ➔ TA* improvement reducing the triangulation graph to contain only nodes of degree 3
- ➔ Runs much faster



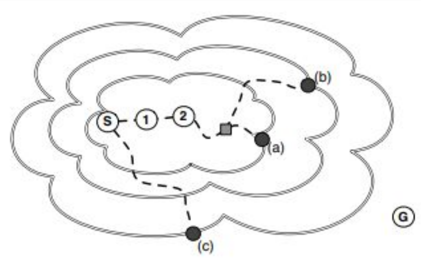




Real-time Search

TBA*

- ➔ Standard A* that can be interrupted
- ➔ The most promising node on the open list is traced back to the start
- ➔ If it passes through the agent's position he simply follows the path
- ➔ Otherwise, the agent backtracks his steps towards the start
- ➔ Or use shortcut-search enhancement

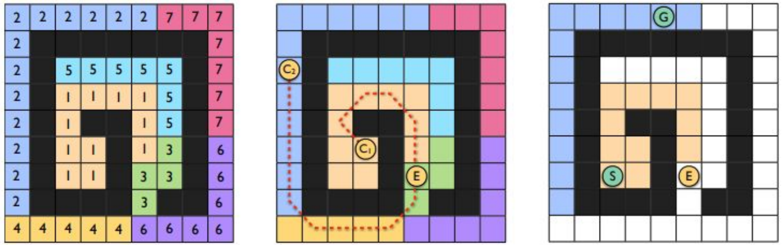


AA*, ARA*

- ➔ A* is called multiple times, with ***h*** value multiplied by gradually decreasing ε≥1
- ➔ Repair variant reuses and updates the path

LRTA*, D LRTA*

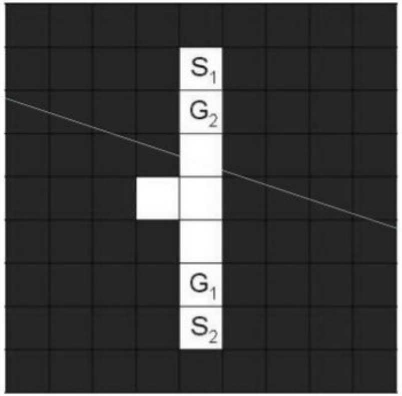
- ➔ Planning, learning, acting steps
- ➔ Partitioning on regions, computed paths between
- ➔ Entries to regions set as subgoals



Cooperative Pathfinding

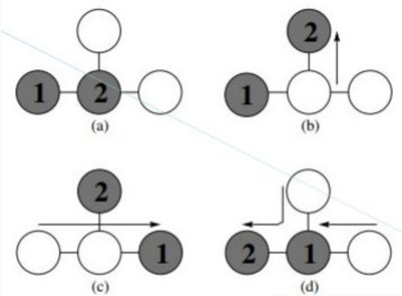
Cooperative A*

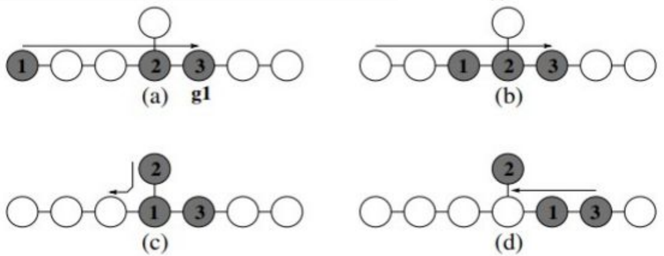
- ➔ Agents reserves their paths
- ➔ They can wait until the path is free.
- ➔ Reservation table is hashable
- ➔ The initial ordering of the agents has huge impact on the outcome.



Push and Swap

- ➔ Suboptimal but complete
- ➔ Introduces two high-level operations.
- ➔ Push: forcing all other agents to get out the way
- ➔ Swap: swaps positions of two agents





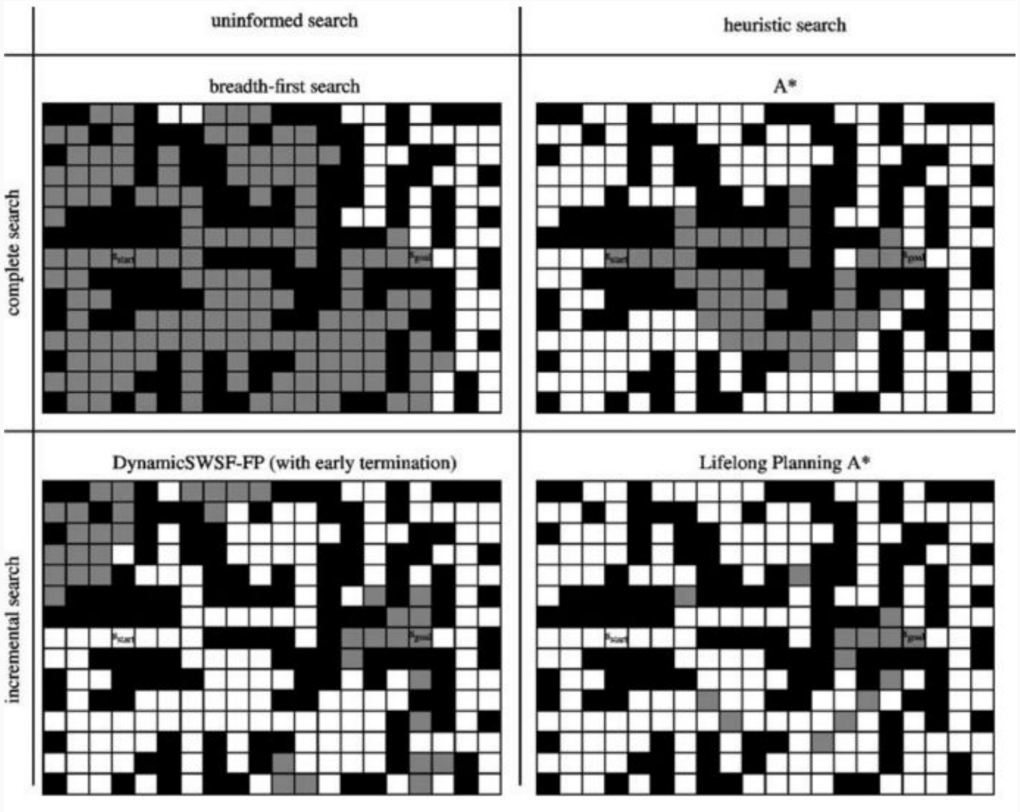
Dynamic Environment

LPA*

- Incremental version of A*.
- Tracks changes between distance from the start, and distances from the start of the node's predecessors
- Forces recalculations when required

D*, Focused D*, D*Lite

- Search backwards (from goal to start)
- Node can be: new/open/closed/raise/lower



Playing Games with A*

Literature

★ Tristan Penman: N-puzzle https://tristanpenman.com/demos/n-puzzle/

★ André G. Pereira, Marcus Ritt, Luciana S. Buriol: Optimal Sokoban solving using pattern databases with specific domain knowledge, Artificial Intelligence, vol. 227, pp. 52-70, 2015.

★ Julian Togelius, Sergey Karakovskiy, Robin Baumgarten: The 2009 Mario AI Competition. IEEE CEC, 2010.

A* as a Game Playing Algorithm

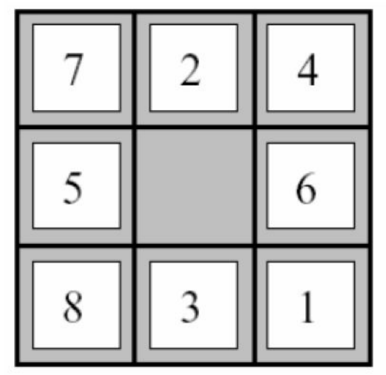
A* can be also used for playing (i.e. solving a game). Then our “map” is actually a graph of game states, and “moves” are legal actions.

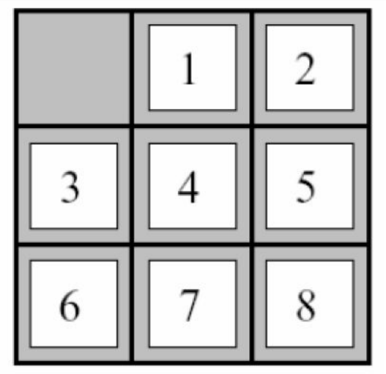
This is particularly popular approach when dealing with single-player games, which turns “playing” into a planning problem.

A few examples of games that can be dealt with A*:

- N-puzzle
- Sokoban
- Mario
- Codingame Fall Challenge 2020 :-)

8-Puzzle





Heuristics

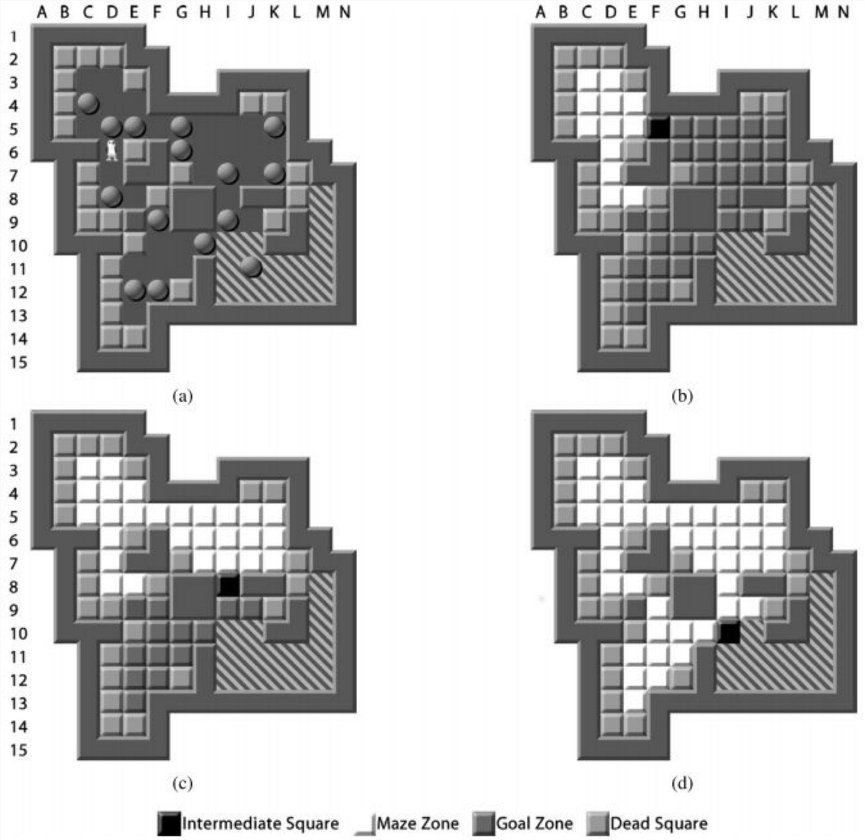
Popular method of generating heuristics is relaxation of the problem.

We used that with pathfinding: straight-line distance is simply no-obstacle assumption.

- What If we could just swap any two tiles? *Number of misplaced tiles.*
- What if we could slide any direction at any time? *Total Manhattan distance.*

We usually got trade-off between the quality of the heuristic (less nodes expanded) and its computational cost.

Sokoban



- IDA*
- Quite complicated heuristic
- Instance decompositions
- Pattern databases
- Dead squares detection
- Domain-dependent tie-breaking rules
- ...

Mario

[Video](#)

- Goal is to reach the right side of the screen
- As fast as possible
- Interruptible A*: best node so far is used
- Slight heuristic overestimation
- Recalculating plan every two game ticks

Bonus material: Video games

StarCraft

- ★ http://www.codeofhonor.com/blog/the-starcraft-path-finding-hack
- ★ https://youtu.be/I9mCau4a130
- ★ https://youtu.be/paX8nHGPpXA

Command & Conquer

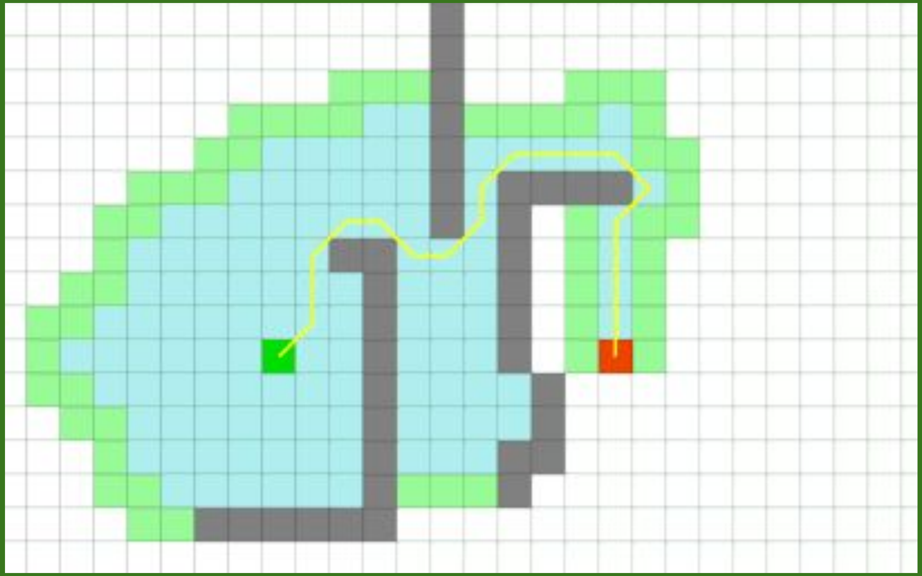
- ★ https://youtu.be/Wb84Vi7XFRg?t=525

Cities:Skylines / Sim City

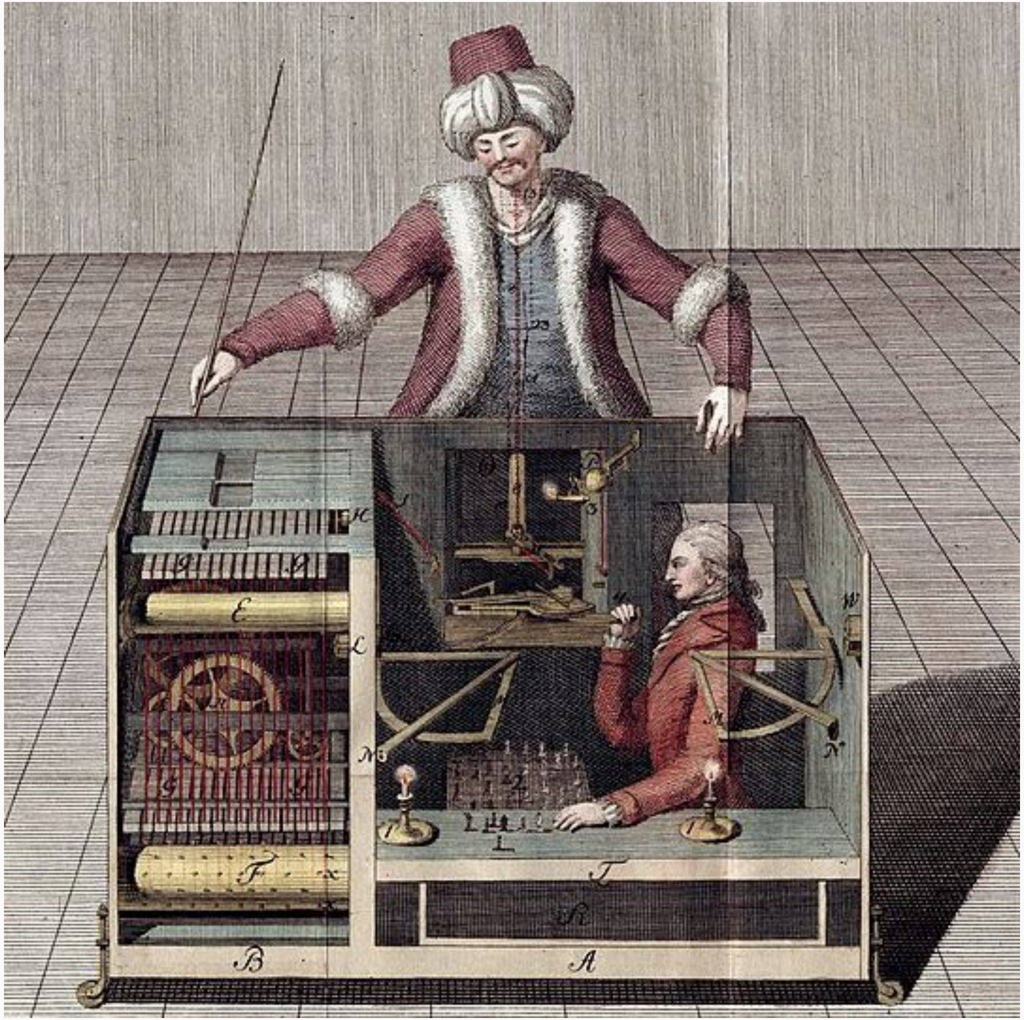
- ★ https://youtu.be/MeNxJVOL9eM
- ★ https://youtu.be/zHdyzx_ecbQ

Summary

★ Pathfinding is actually a complex task
★ (We just scratched the surface of the problem)
★ It is also one of the most important tasks
★ Most algorithms are based on A*



Thanks!



Bonus reference quiz