Navigating Large Graphs: Introduction to Shortest Path Algorithms

Khoruzhii Kirill

04.07.2024

Problem statement

The graph is given via

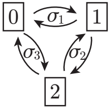
- target vertex V_0
- available moves $\{\sigma_j\}$

$V_0 = \boxed{0}\boxed{1}\boxed{2}$

$\sigma_1 = (1,\ 0,\ 2)$

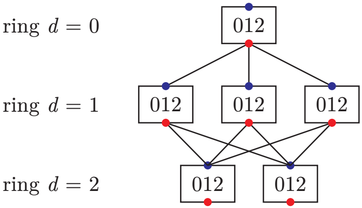
$\sigma_2 = (0,\ 2,\ 1)$

$\sigma_3 = (2,\ 1,\ 0)$



Graph structure:

- distance $d(V) \stackrel{\text{def}}{=} \min\limits_{\text{path}} \operatorname{len} \operatorname{path}(V, V_0)$
- ring $d' \stackrel{\text{def}}{=} \{V \mid d(V) = d'\}$
- V_0 eccentricity $\stackrel{\text{def}}{=} \max\limits_{V} d(V)$ (may be the same as diameter D)



Example: Rubik's Cube

- state as vector of numbers $(0, 0, 0, 0, 1, 1, 1, 1, \ldots)$
- permutations behave as

$$f_0 = \begin{pmatrix} 0 & 1 & 2 & 3 & \ldots \\ 0 & 1 & 19 & 17 & \ldots \end{pmatrix}$$

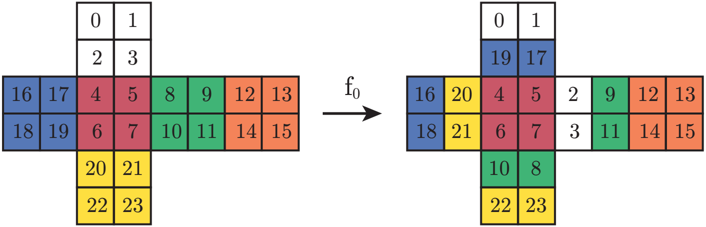


Figure 1: Example of permutation, code here

Scale of the disaster

- for Cayley graphs the vertex degree is $n = \mathrm{card}\{\sigma_j\}$
- as a consequence exponential growth $\mathrm{card\,ring}\, d' \propto n^{d'}$

	2x2	3x3	4x4	5x5
Rubik's Cube	3.7×10^6	4.3×10^{19}	7.4×10^{45}	2.8×10^{74}
Sliding Puzzle	1.2×10^1	1.8×10^5	1.0×10^{13}	7.8×10^{24}

Table 1: Graph sizes for different puzzles

	2x2	3x3	4x4	5x5
Rubik's Cube	14	26	≈ 48	≈ 80
Sliding Puzzle	4	31	80	≈ 150

Table 2: Graph diameters for different puzzles

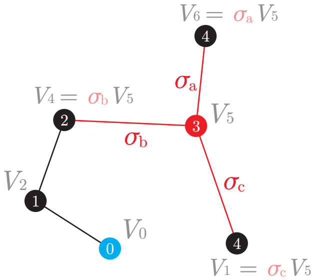
Deep Approximate Value Iteration (DAVI)

Bellman equation:

$$\boxed{J'(V) = 1 + \min_{\sigma} J(\sigma V)}$$

Data annotation by Bellman Equation:

1. $d(V_2) = d(V_0) + 1 = 1$
2. $d(V_4) = d(V_2) + 1 = 2$
3. $d(V_5) = d(V_4) + 1 = 3$
4. $d(V_1) = d(V_5) + 1 = 4$
   $d(V_6) = d(V_5) + 1 = 4$



Deep Approximate Value Iteration (DAVI)

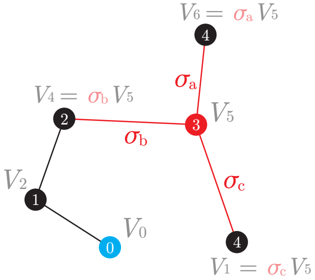
Bellman equation:

$$\boxed{J'(V) = 1 + \min_{\sigma} J(\sigma V)}$$

Model $J(V)$ is trained to predict d.

Until convergence repeat:

- for $V \in$ dataset calculate $J'(V)$
- train model to predict $J'(V)$



Instead of storing a dictionary with all the vertices, we train the model to «remember»/«understand» the values found according to Bellman.

[1] S. McAleer et al., Solving the rubik's cube with approximate policy iteration

Model $J(V)$ is trained to predict d.

Until convergence repeat:

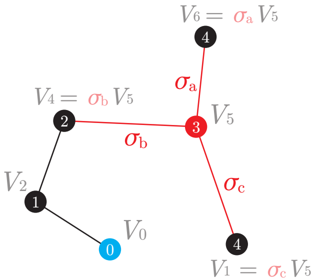
- for $V \in$ dataset calculate $J'(V)$
- train model to predict $J'(V)$

Important details:

- dataset generation
- model architecture
- vertex representation
- model-based pathfinding

Bellman equation:

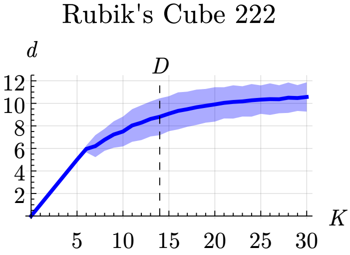
$$\boxed{J'(V) = 1 + \min_{\sigma} J(\sigma V)}$$

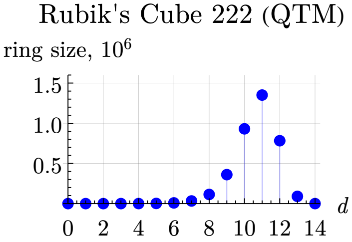


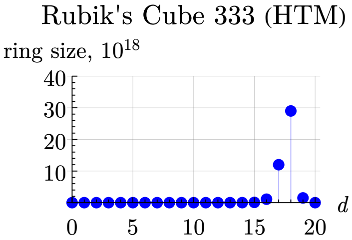
Dataset generation

For balanced dataset, we do $K \in [1, D_{\text{est}}]$ random steps from V_0.

It is important to carefully choose $K_{\max} \sim D_{\text{est}}$.







Heuristic guided path search: A*

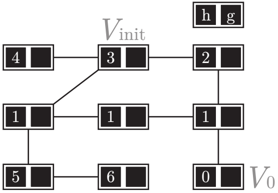
The cost of each node V in the search tree:

$$f(V) = g(v) + h(V), \quad \text{with} \begin{cases} g(v) & \text{path cost} \\ h(v) & \text{heuristic function} \end{cases}$$

A* algorithm

while $V_\mathrm{n} \neq V_0$:

1. $V_\mathrm{n} = \underset{V \in \mathrm{queue}}{\operatorname{argmin}} f(V)$
2. for $\sigma \in \{\sigma_j\}$:
   if $g(V_n) + 1 < g(\sigma V_n)$:
   extend queue with σV_n
   upd $g(\sigma V_\mathrm{n}) = g(V_\mathrm{n}) + 1$



The path is guaranteed to be the shortest if $h(V) \leqslant d(V)$!

Heuristic guided path search: A*

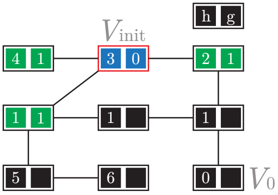
The cost of each node V in the search tree:

$$f(V) = g(v) + h(V), \quad \text{with} \begin{bmatrix} g(v) & \text{path cost} \\ h(v) & \text{heuristic function} \end{bmatrix}$$

A* algorithm

while $V_{\mathrm{n}} \neq V_0$:

1. $V_{\mathrm{n}} = \underset{V \in \text{queue}}{\operatorname{argmin}} f(V)$
2. for $\sigma \in \{\sigma_j\}$:
   if $g(V_n) + 1 < g(\sigma V_n)$:
   extend queue with σV_{n}
   upd $g(\sigma V_{\mathrm{n}}) = g(V_{\mathrm{n}}) + 1$



The path is guaranteed to be the shortest if $h(V) \leqslant d(V)$!

Heuristic guided path search: A*

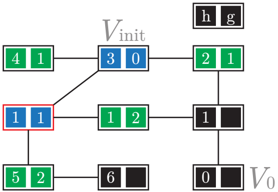
The cost of each node V in the search tree:

$$f(V) = g(v) + h(V), \quad \text{with} \begin{bmatrix} g(v) & \text{path cost} \\ h(v) & \text{heuristic function} \end{bmatrix}$$

A* algorithm

while $V_\mathrm{n} \neq V_0$:

1. $V_\mathrm{n} = \underset{V \in \text{queue}}{\operatorname{argmin}} f(V)$
2. for $\sigma \in \{\sigma_j\}$:
   if $g(V_n) + 1 < g(\sigma V_n)$:
   extend queue with σV_n
   upd $g(\sigma V_\mathrm{n}) = g(V_\mathrm{n}) + 1$



The path is guaranteed to be the shortest if $h(V) \leqslant d(V)$!

Heuristic guided path search: A*

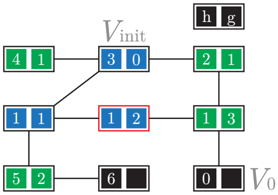
The cost of each node V in the search tree:

$$f(V) = g(v) + h(V), \quad \text{with} \begin{bmatrix} g(v) & \text{path cost} \\ h(v) & \text{heuristic function} \end{bmatrix}$$

A* algorithm

while $V_\mathrm{n} \neq V_0$:

1. $V_\mathrm{n} = \underset{V \in \text{queue}}{\operatorname{argmin}} f(V)$
2. for $\sigma \in \{\sigma_j\}$:
   if $g(V_n) + 1 < g(\sigma V_n)$:
   extend queue with σV_n
   upd $g(\sigma V_\mathrm{n}) = g(V_\mathrm{n}) + 1$



The path is guaranteed to be the shortest if $h(V) \leqslant d(V)$!

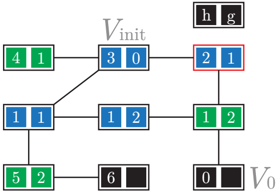
The cost of each node V in the search tree:

$$f(V) = g(v) + h(V), \quad \text{with} \quad \begin{bmatrix} g(v) & \text{path cost} \\ h(v) & \text{heuristic function} \end{bmatrix}$$

A* algorithm

while $V_\mathrm{n} \neq V_0$:

1. $V_\mathrm{n} = \underset{V \in \mathrm{queue}}{\operatorname{argmin}} f(V)$
2. for $\sigma \in \{\sigma_j\}$:
   if $g(V_n) + 1 < g(\sigma V_n)$:
   extend queue with σV_n
   upd $g(\sigma V_\mathrm{n}) = g(V_\mathrm{n}) + 1$



The path is guaranteed to be the shortest if $h(V) \leqslant d(V)$!

Heuristic guided path search: A*

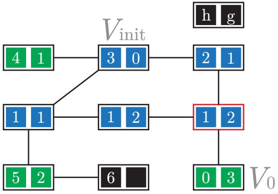
The cost of each node V in the search tree:

$$f(V) = g(v) + h(V), \quad \text{with } \begin{bmatrix} g(v) & \text{path cost} \\ h(v) & \text{heuristic function} \end{bmatrix}$$

A* algorithm

while $V_\mathrm{n} \neq V_0$:

1. $V_\mathrm{n} = \operatorname*{argmin}_{V \in \mathrm{queue}} f(V)$
2. for $\sigma \in \{\sigma_j\}$:
   if $g(V_n) + 1 < g(\sigma V_n)$:
   extend queue with σV_n
   upd $g(\sigma V_\mathrm{n}) = g(V_\mathrm{n}) + 1$



The path is guaranteed to be the shortest if $h(V) \leqslant d(V)$!

Heuristic guided path search: A*

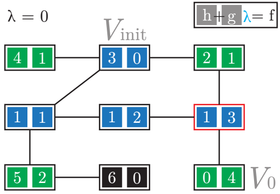
The cost of each node V in the search tree:

$$f(V) = \lambda g(v) + h(V), \quad \text{with} \begin{bmatrix} g(v) & \text{path cost} \\ h(v) & \text{heuristic function} \end{bmatrix}$$

A* algorithm

while $V_\mathrm{n} \neq V_0$:

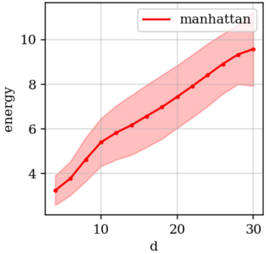
1. $V_\mathrm{n} = \underset{V \in \mathrm{queue}}{\mathrm{argmin}}\, f(V)$
2. for $\sigma \in \{\sigma_j\}$:
   - if $g(V_n) + 1 < g(\sigma V_n)$:
     - extend queue with σV_n
     - upd $g(\sigma V_\mathrm{n}) = g(V_\mathrm{n}) + 1$



The path is guaranteed to be the shortest if $h(V) \leqslant d(V)$!

Heuristics

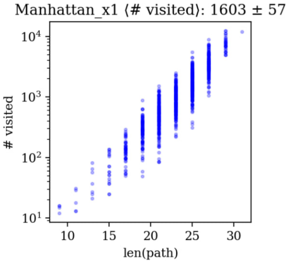
- Hamming distance (number of correct tiles)
- Manhattan distance. For Sliding Puzzle 3x3 (180k states)

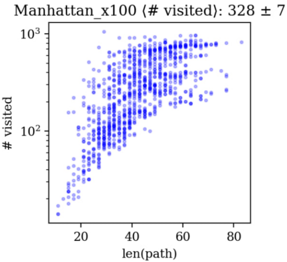


- K-prediction
- KL-div*

Heuristics

- Hamming distance (number of correct tiles)
- Manhattan distance. For Sliding Puzzle 3x3 (180k states)

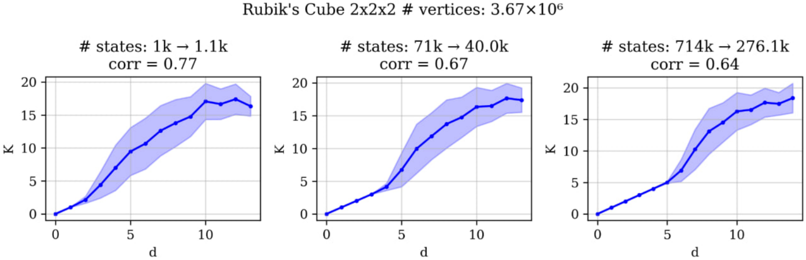




- K-prediction
- KL-div*

Heuristics

- Hamming distance (number of correct tiles)
- Manhattan distance
- K-prediction



- KL-div*

Unscrambling

- model is trained to predict probability distribution of the source vertex
- the shorter a path, the more likely it is to occur randomly
- the cumulative probability $p_1 p_2 \ldots$ of a random training scramble increases as the number of moves decreases.
- beam search (greedy algorithm) based on cumulative p

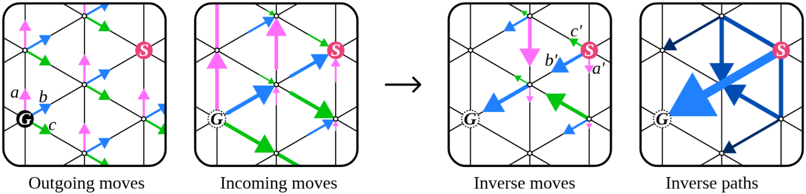


Figure 2: A miniature instance of combinatorial search with a predefined goal from [3].

[3] K. Takano, Self-Supervision is All You Need for Solving Rubik's Cube, 2023

Movement on a graph with heuristics h as a process of searching for a state with the lowest energy $E(V) = h(V)$.

Metropolis-Hastings algorithm

init $V = V_0$
while $V \neq V_0$:

1. Choose a random $\sigma \in \sigma_j$
2. if $E(\sigma V) < E(V)$:
   $V = \sigma V$
3. else with $p = e^{-\beta(E(\sigma V) - E(V))}$
   $V = \sigma V$

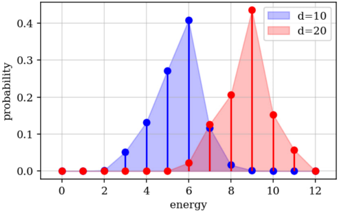
The behavior is highly dependent on the inverse temperature β.

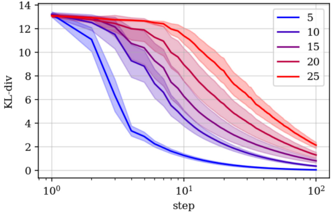
Movement on a graph with heuristics h as a process of searching for a state with the lowest energy $E(V) = h(V)$.

Metropolis-Hastings algorithm

init $V = V_0$
while $V \neq V_0$:

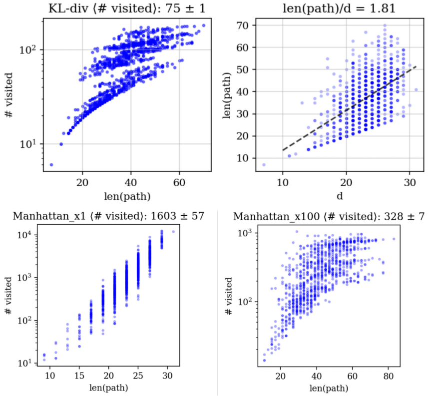
1. Choose a random $\sigma \in \sigma_j$
2. if $E(\sigma V) < E(V)$:
   $V = \sigma V$
3. else with $p = e^{-\beta(E(\sigma V) - E(V))}$
   $V = \sigma V$

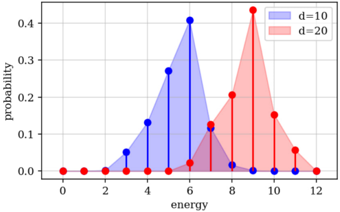


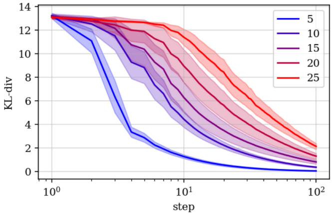


The behavior is highly dependent on the inverse temperature β.

Movement on a graph with heuristics h as a process of searching for a state with the lowest energy $E(V) = h(V)$.







The behavior is highly dependent on the inverse temperature β.