Ideas of Dynamic Programming in Optimal Graph-search and Sampling-based Planners

Hongkai YE

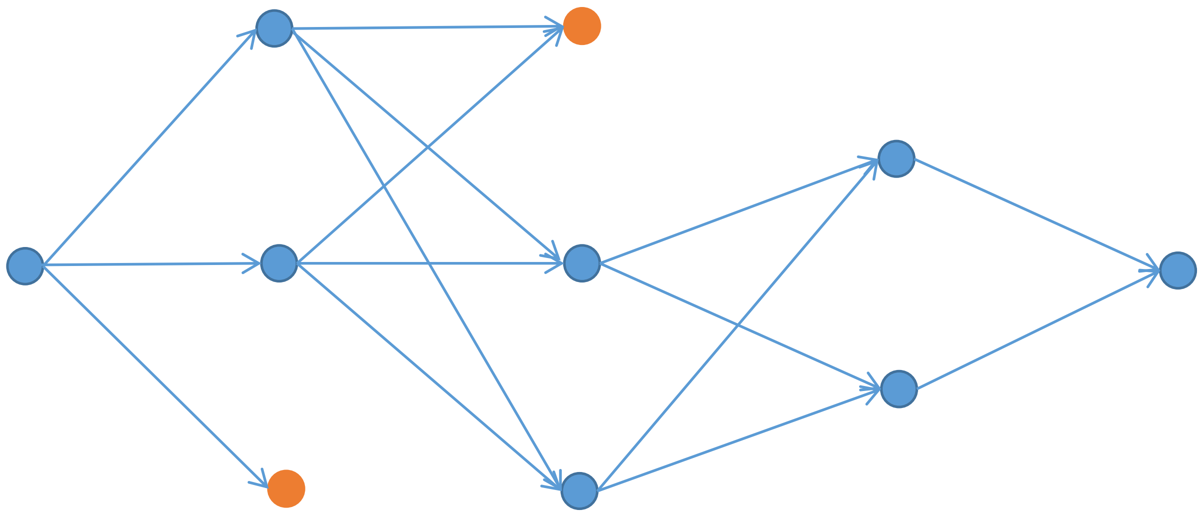
2021.10.05

- DP functional equation
- Direct Methods
- Successive Approximation Methods
  - Pull
  - Push

DP functional equation (discrete form)

$$f(j) = \min_{i \in B(j)} \{f(i) + D(i,j)\}\ ,\ j \in C \backslash \{1\}$$

B(j) represents for predecessors of j,
and D(i, j) the cost of transition from state i to state j.



Compute backwards?

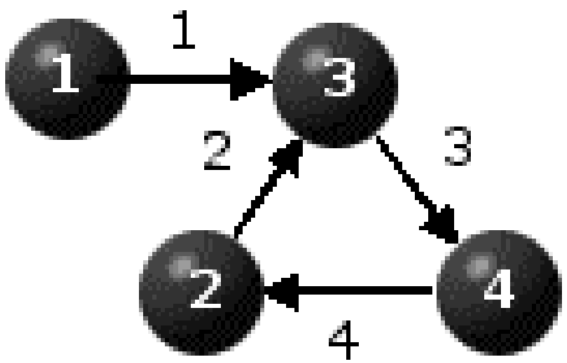
For shortest path problems, the difference is merely to store values of cost-to-go (Backwards) or cost-to-come (Forwards).

DP functional equation (discrete form)

$$f(j) = \min_{i \in B(j)} \{f(i) + D(i,j)\} \ , \ j \in C \backslash \{1\}$$

B(j) represents for predecessors of j,
and D(i, j) the cost of transition from state i to state j.

The DP functional equation
does not constitute an algorithm!



$$\begin{aligned} f(1) &= 0 \\ f(2) &= 4 + f(4) \\ f(3) &= \min\{2 + f(2), 1 + f(1)\} \\ f(4) &= 3 + f(3) \end{aligned}$$

Direct DP Methods

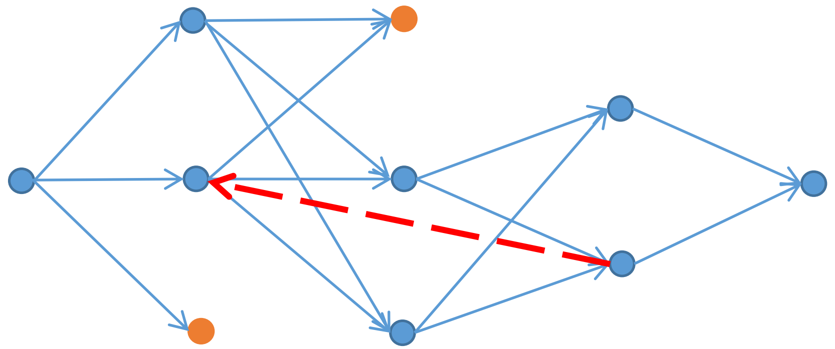
Generic Direct Method

$$D(i,j) = \infty, \forall i,j \in C, i \geq j$$

Initialization: $F(1) = 0$

Iteration: For $j = 2, \ldots, n$ Do:

$$F(j) = \min_{i \in B(j)} \{F(i) + D(i,j)\}$$



The cost-to-come value of each state can be determined by processing it **only once.**

- While computing the value of f(j), all the relevant values of f(i) must have **already been computed** (somehow).
- The graph must be **acyclic**.

Direct DP Methods

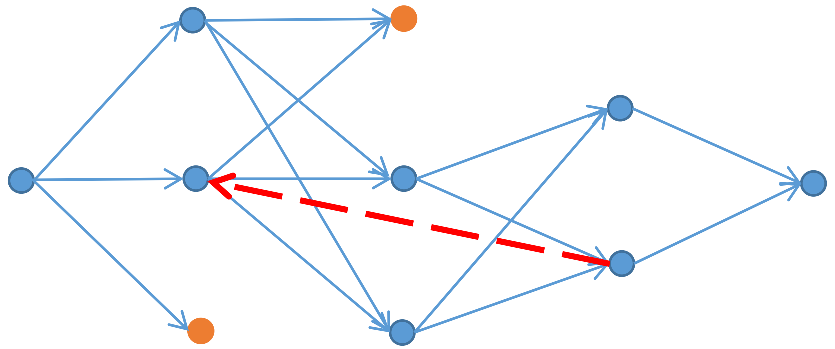
Generic Direct Method

$$D(i,j) = \infty, \forall i,j \in C, i \geq j$$

Initialization: $F(1) = 0$

Iteration: For $j = 2, \ldots, n$ Do:

$$F(j) = \min_{i \in B(j)} \{F(i) + D(i,j)\}$$



In motion planning, state graphs (or grids) are almost surely cyclic.
Each state can be in any level (step), and one can only access states by range queries or k-NN queries.

Sampling-based methods introduce some kinds of randomness in generating the **implicit** random geometric graph **(RGG) incrementally** or **in batch**. It's still graph-search.

Successive Approximation Methods

An initial approximation for all f(j) is constructed, and then it is successively updated (hopefully improved).

Different SA methods differ in two aspects:

1. The way the updating is carried out.

2. The order in which the updating mechanism is conducted. (Heuristics)

Two Updating “Philosophies”

$$\textbf{Pull at j :}\ \ F(j) = \min_{i \in B(j)} \{F(i) + D(i,j)\}\ ,\ \ j \in C, B(j) \neq \{\}.$$

Update value of current state being processed.
Same as DP functional equation, but states may be re-visited.

$$\textbf{Push at j :}\ F(i) = \min\{F(i), F(j) + D(j,i)\},\ \ i \in A(j).$$

Update values of current state's immediate successors.
This is exactly how most graph-search planners (Dijkstra, A*, ...) work.

Two Updating “Philosophies”

Successive Approximation for $f(j) = \min_{i \in B(j)} \{f(i) + D(i,j)\}, j \in C \backslash \{1\}; f(1) = 0$	
Pull	Push
Initialize: $Q = A(1)$ $F(1) = 0; F(i) = \infty, i \in C \backslash \{1\}$ **Iterate:** While $(\|Q\| > 0)$ Do: $j =$ Select_From(Q) $Q = Q \backslash \{j\}$ $F(j) = \min_{i \in B(j)} \{F(i) + D(i,j)\}$ $Q = Q \bigcup A(j)$ End Do	**Initialize:** $Q = \{1\}$ $F(1) = 0; F(i) = \infty, i \in C \backslash \{1\}$ **Iterate:** While $(\|Q\| > 0)$ Do: $j =$ Select_From(Q) $Q = Q \backslash \{j\}$ for $i \in A(j)$ Do: $G = F(j) + D(j,i)$ if$(G < F(i)) Do:$ $F(i) = G$ $Q = Q \bigcup \{i\}$ End Do End Do End Do

Dijkstra and Push

Dijkstra	$F(i) = \min\{F(i), F(j) + D(j,i)\}, i \in A(j) \cap U$
Push	$F(i) = \min\{F(i), F(j) + D(j,i)\}, i \in A(j)$

While $(|Q| > 0)$ Do:
    $\boxed{j = \text{Select\_From(Q)}}$
    $Q = Q \backslash \{j\}$
    $\boxed{\text{for } i \in A(j) \text{ Do:}}$
        $G = F(j) + D(j,i)$
        $\text{if}(G < F(i)) Do:$
            $F(i) = G$
            $Q = Q \bigcup \{i\}$
        End Do
    End Do
End Do

While $(|Q| > 1)$ Do:
    $\boxed{j = \arg\min\{F(i) : i \in Q\}}$
    $Q = Q \backslash \{j\}$
    $\boxed{\text{for } i \in A(j) \bigcap Q \text{ Do:}}$
        $G = \min\{F(i), F(j) + D(j,i)\}$
        $\text{If}(G < F(i))$ Do:
            $F(i) = G$
            $Q = Q \bigcup \{i\}$
        End Do
    End Do
End Do

Dijkstra and Push

Dijkstra	$F(i) = \min\{F(i), F(j) + D(j,i)\}, i \in A(j) \cap U$
Push	$F(i) = \min\{F(i), F(j) + D(j,i)\}, i \in A(j)$

Differences:

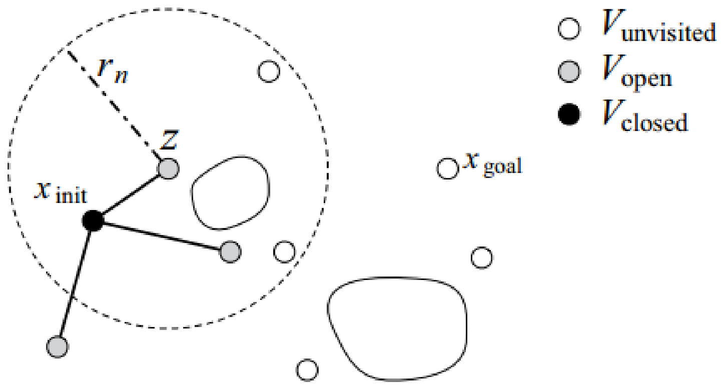
in Dijkstra or A*, only the F(·) values of the immediate successors of j **that have not been processed yet**, are updated. (by labeling states as open and closed):

How come?

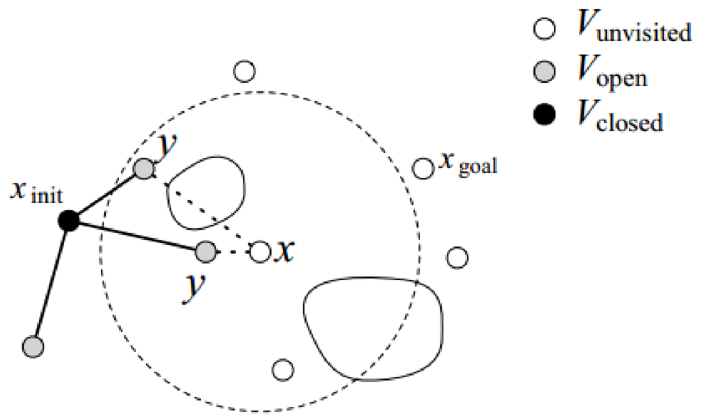
(1) It is updated in ascending order of cost-to-come.
(2) State transition cost is non-negative.

FMT*

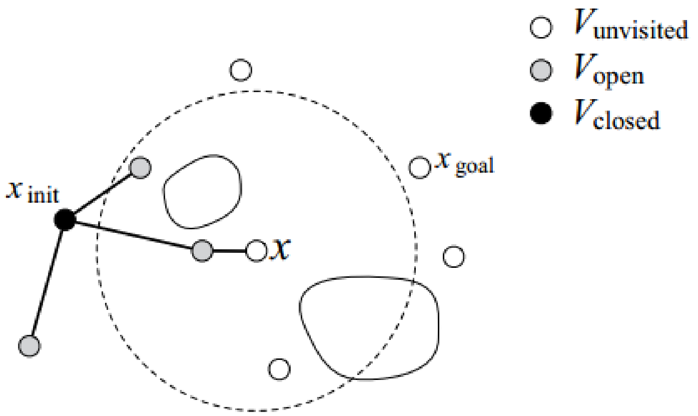
- Application of DP direct method.
- The lazy mechanism does count.
- A lazy check on Dijkstra works the same (Lazy PRM*).



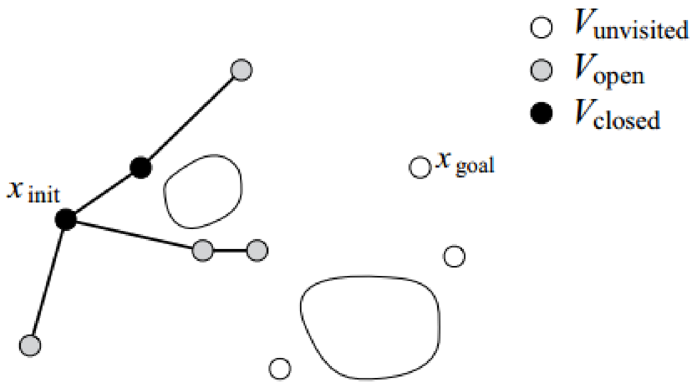
(a) Lines 2–3: FMT* selects the lowest-cost node z from set V_{open} and finds its neighbors within $V_{unvisited}$.



(b) Lines 4–5: given a neighboring node x, FMT* finds the neighbors of x within V_{open} and searches for a locally optimal one-step connection. Note that paths intersecting obstacles are also lazily considered.



(c) Line6:FMT* selects the locally optimal one-step connection to x ignoring obstacles, and adds that connection to the tree if it is collision-free.



(d) Lines 7–8: After all neighbors of z in $V_{unvisited}$ have been explored, FMT* adds successfully connected nodes to V_{open}, places z in V_{closed}, and moves to the next iteration.

Practical Performance

- Sampling Strategy

  Deterministic/Random; Batch/Incremental

- Heuristic Search

  Informed search; Improved heuristics

- Lazy Check

  Delaying edge evaluations until necessary (when they belong to the best candidate solution, or in order of potential solution quality...).

- Bidirectional Search

- Hybirid

  Introduce local optimization.