Dynamic Programming

Dynamic Programming

We have seen **MPC** (Model Predictive Control), for Control . //not in 2022

Nonlinear MPC can be used for Planning , however, the larger the prediction horizon, the more expensive its optimization process.

In planning, we do not usually have a high number of states, but we may require a "plan" (a sequence of actions) covering a high number of steps.

We see an example, in simulation now, to feel the problem.

Dynamic Programming

We have seen **MPC** (Model Predictive Control), for Control . It is one of the most advanced approaches in Control.

Nonlinear MPC can even be used for Planning , however, the larger the prediction horizon, the more expensive its optimization process.

In planning, we do not usually have a high number of states, but we may require an extended "plan" (a sequence of actions, covering a high number of steps.)

We discuss now, about a very efficient approach for nonlinear Control and for Planning.

Principle of Optimality

Consider a system whose transition equation is :

$$x' = f\left(x, u\right)$$

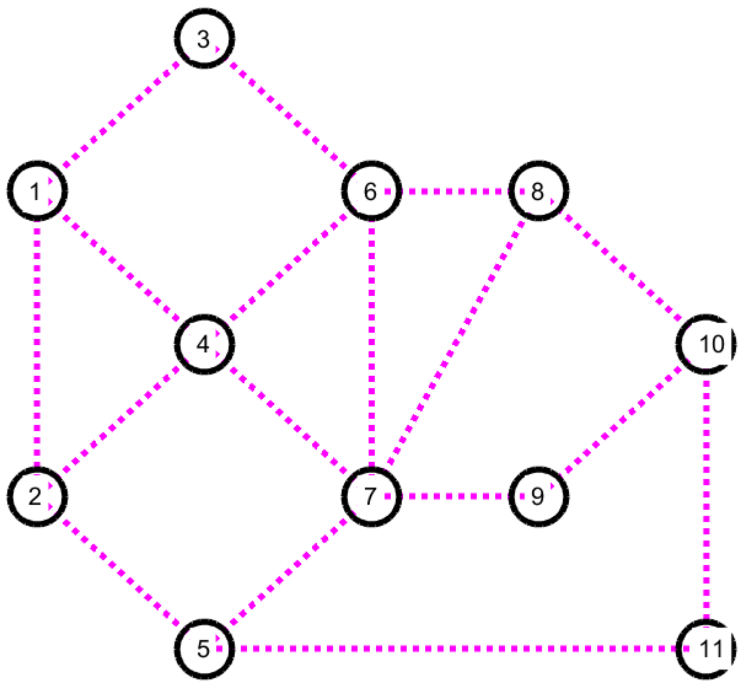
$$x \in \Omega$$

$$u \in U$$

Ω : set of discrete states

U : set of feasible control actions

$$x' = f(x,u)$$
$$x \in \Omega$$
$$u \in U$$



Example 1:
States : in a graph having 11 nodes (each of them is a "state")
Action: transition from a node to another connected node.
Connections can be unidirectional or bidirectional
(in this case, assumed to be bidirectional)

$$x' = f(x, u)$$

$$x \in \Omega$$

$$u \in U$$

Ω : set of discrete states

U : set of feasible control actions

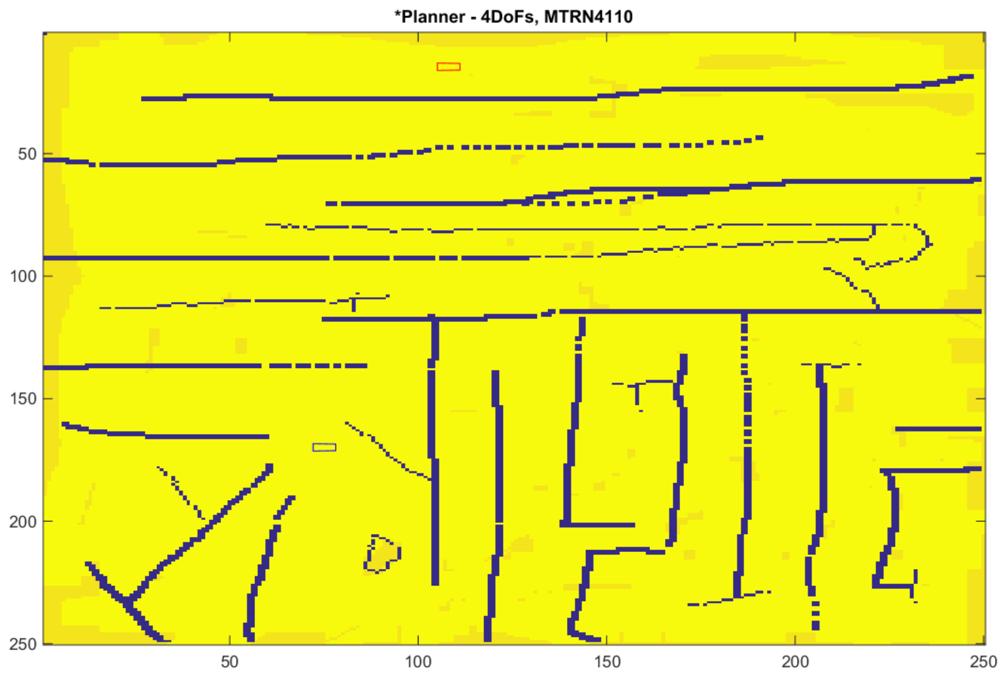
Example 2:

States : elements of a grid
Actions: transition from a cell to an adjacent cell
(Again, we see the example of the Mining Truck)

$$x' = f(x,u)$$
$$x \in \Omega$$
$$u \in U$$

Ω : set of discrete states

U : set of feasible control actions



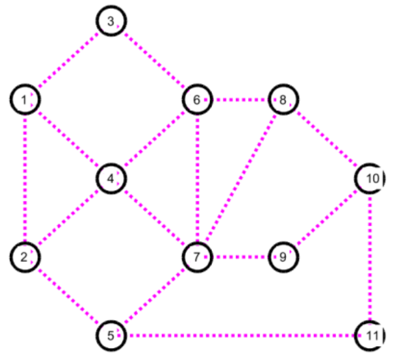
Example 2:

States : elements of a grid
Actions: transition from a cell to an adjacent cell
(Again, we see the example of the Mining Truck)

$$x' = f(x, u)$$

$$x \in \Omega$$

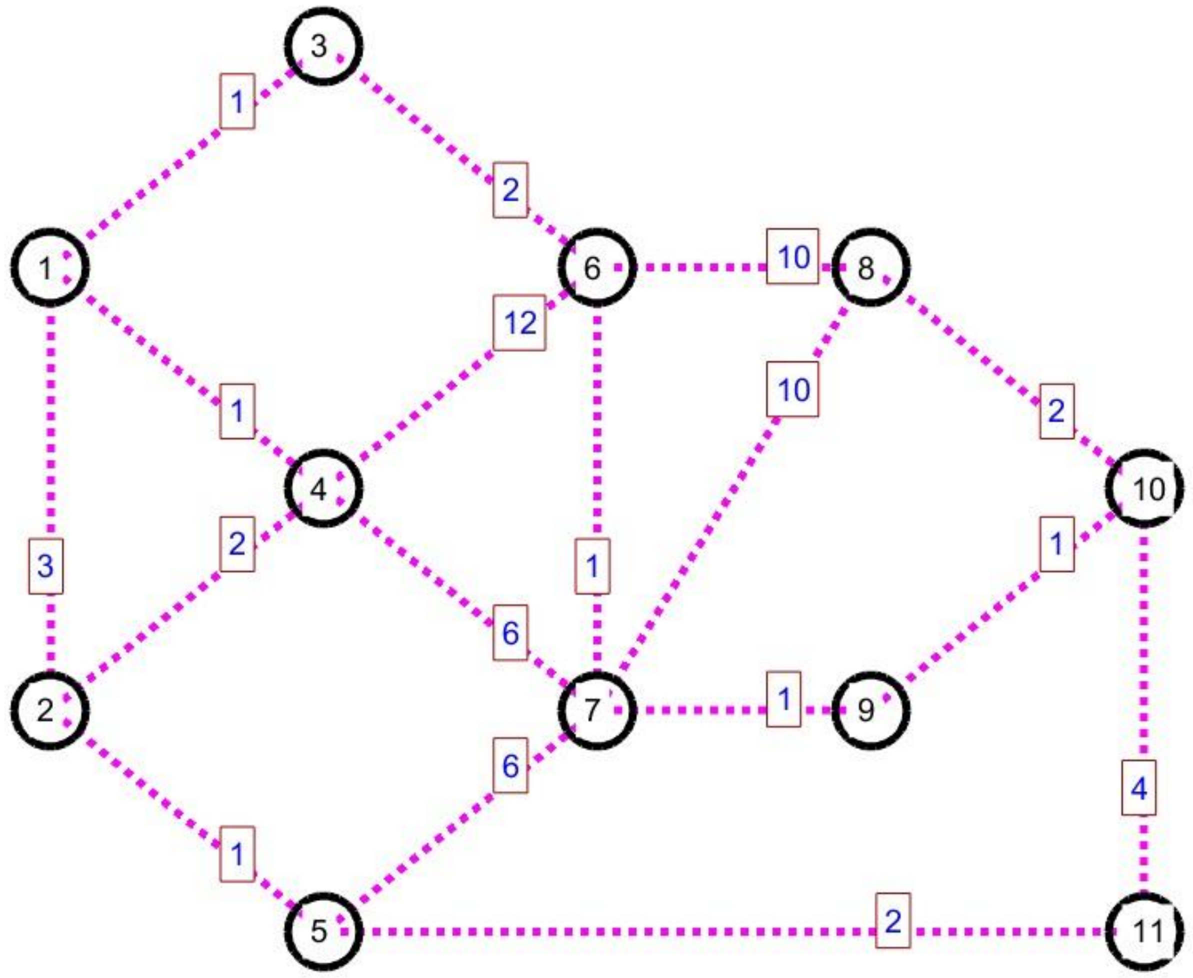
$$u \in U$$



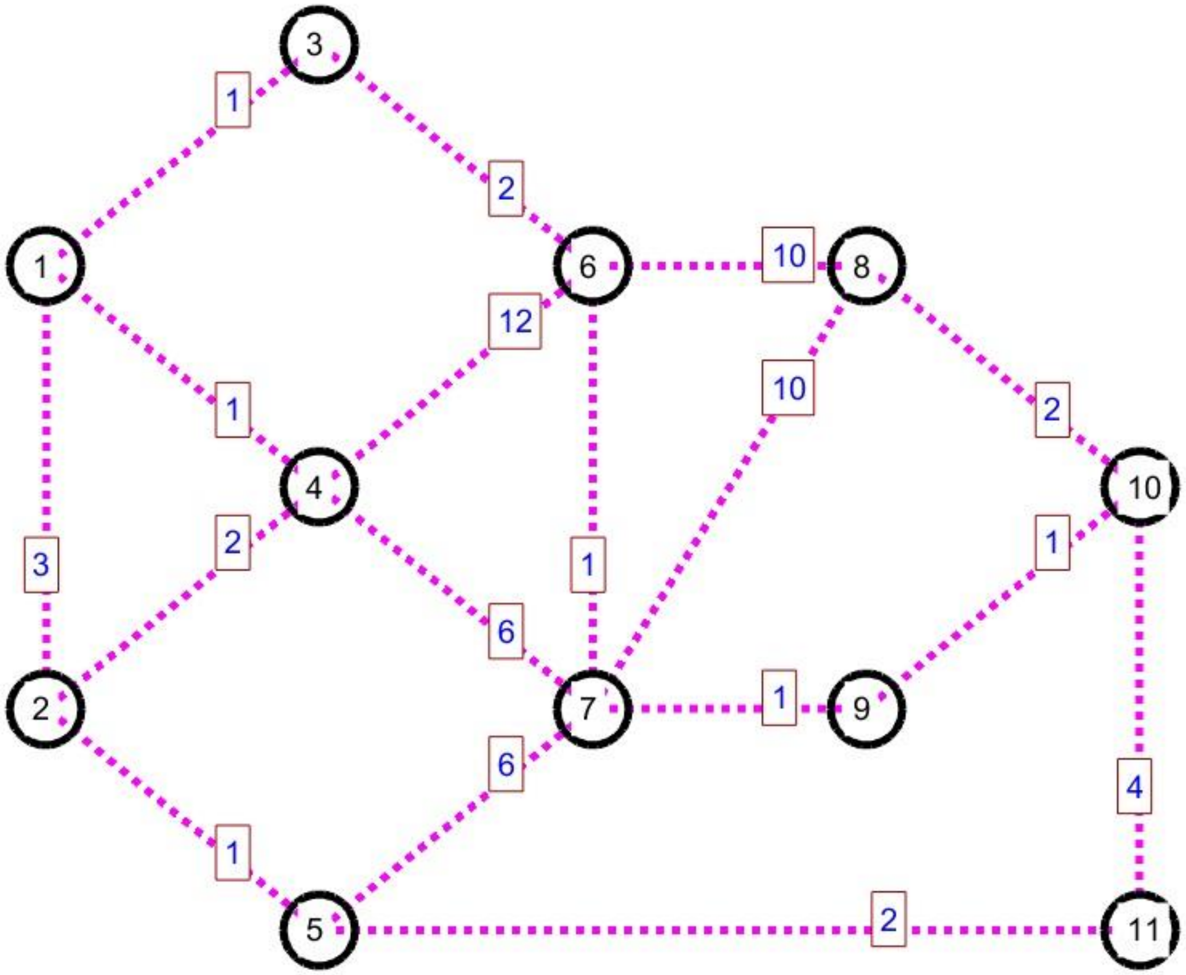
Ω : set of discrete states

U : set of feasible control actions

Note that , from the current state, just a small subset of the state space is reachable by performing one transition. However, by performing multiple transitions, the set of reachable states (from current state x) is larger, it can eventually be the full domain , except certain sub-regions which may be unreachable.
We ask now about how to reach a required GOAL (goal state) provided that the system's current state is x.

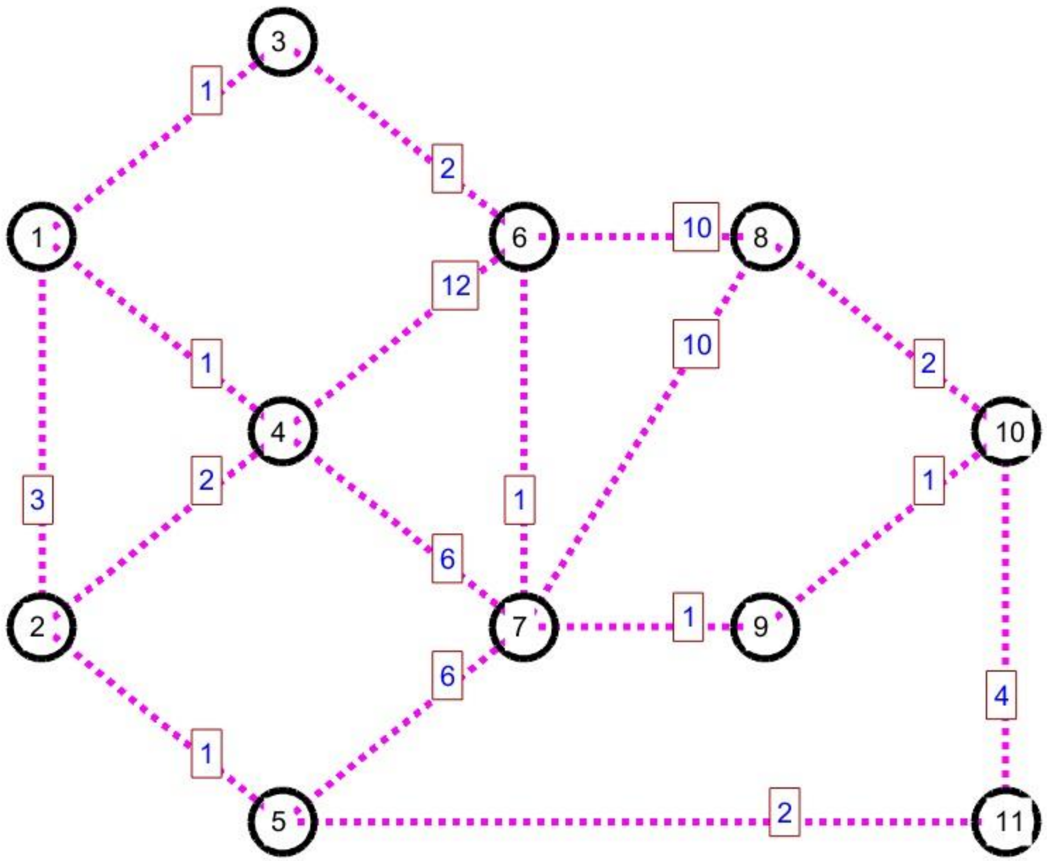


We need to consider some extra information: Cost of each possible transition, as it can be seen in this example (numbers in square boxes).



In general, we express it by a transition cost function $L(x,u)$

Which may be function of the current state and the action.

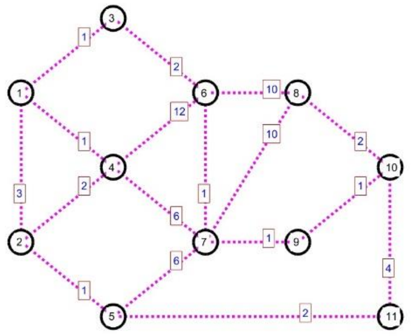


We can also use a different convention, $L(x, x')$

In which we define it as a function of the current state and the resulting state due to applying an associated transition.

$$L(x, x') \geq 0,$$

$$\left(L(x, x') = \infty, \text{ means invalid transition}\right)$$



Now, we want to obtain a sequence of control actions

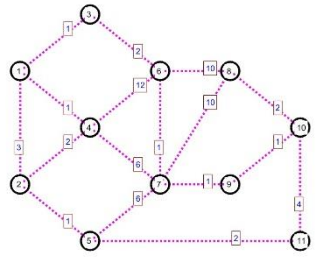
$$\{u_i\}_{i=0}^{N-1}$$

which will solve these constraints

$$x(i+1) = f\left(x(i), u_i\right)$$
$$u_i \in U\left(x(i)\right)$$

$$x(0) = x_{start}$$
$$x(N) = x_{goal}$$

$$\{u_i\}_{i=0}^{N-1}$$



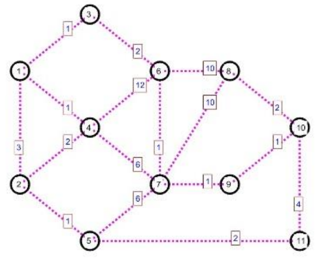
As there may be many solutions, now we add the condition that the sequence of actions must minimize the cost function

$$Cost = \sum_{i=0}^{N-1} L\left(x(i), u_i\right)$$

Always respecting these constraints

$$x(i+1) = f\left(x(i), u_i\right)$$
$$u_i \in U\left(x(i)\right)$$

$$x(0) = x_{start}$$
$$x(N) = x_{goal}$$



We want to obtain a sequence of control actions

$$\left\{u_i\right\}_{i=0}^{N-1}$$

Which will minimize the cost function $Cost = Cost\left(\left\{u_i\right\}_{i=0}^{N-1}\right) = \sum_{i=0}^{N-1} L\left(x(i), u_i\right)$

Respecting the constraints

$$\left\{\begin{array}{c} x(i+1) = f\left(x(i), u_i\right) \\ u_i \in U\left(x(i)\right) \\ \\ x(0) = x_{start}\,; \quad x(N) = x_{goal} \end{array}\right\}$$

We express it as:

$$\left\{u_i^*\right\}_{i=0}^{N^*-1} = \underset{\substack{\left\{u_i\right\}_{i=0}^{N-1}, N, \\ x(N)=x_{goal}}}{\operatorname{argmin}} \left\{Cost\left(\left\{u_i\right\}_{i=0}^{N-1}\right)\right\} = \underset{\substack{\left\{u_i\right\}_{i=0}^{N-1}, N \\ x(N)=x_{goal}}}{\operatorname{argmin}} \left\{\sum_{i=0}^{N-1} L\left(x(i), u_i\right)\right\}$$

$$\left\{u_i^*\right\}_{i=0}^{N^*-1} = \underset{\substack{\left\{u_i\right\}_{i=0}^{N-1}, N \\ x(N)=x_{goal}}}{\operatorname{argmin}} \left\{ \sum_{i=0}^{N-1} L\left(x(i), u_i\right) \right\}$$

$$\left\{ \begin{array}{c} x(i+1) = f\left(x(i), u_i\right) \\ u_i \in U\left(x(i)\right) \\ \\ x(0) = x_{start}; \quad x(N) = x_{goal} \end{array} \right\}$$

The solution of this minimization problem

$$\left\{u_i^*\right\}_{i=0}^{N^*-1}$$

In which we obtain an optimal sequence of control actions.

$$\left\{u_i^*\right\}_{i=0}^{N^*-1} = \underset{\substack{\left\{u_i\right\}_{i=0}^{N-1},N \\ x(N)=x_{goal}}}{\operatorname{argmin}} \left\{\sum_{i=0}^{N-1} L\left(x(i),u_i\right)\right\}$$

$$\left\{\begin{array}{c} x(i+1) = f\left(x(i),u_i\right) \\ u_i \in U\left(x(i)\right) \\ \\ x(0) = x_{start}; \quad x(N) = x_{goal} \end{array}\right\}$$

The solution of this minimization problem

In which we obtain an optimal sequence of control actions. $\left\{u_i^*\right\}_{i=0}^{N^*-1}$

By applying this optimal sequence, we reach the goal, at a cost (which is optimal)

$$C^*\left(x_{start}; x_{goal}\right) = \min_{\substack{\left\{u_i\right\}_{i=0}^{N-1},N \\ x(0)=x_{start};\, x(N)=x_{goal}}} \left\{\sum_{i=0}^{N-1} L\left(x(i),u_i\right)\right\}$$

$$\left\{u_i^*\right\}_{i=0}^{N^*-1} = \underset{\substack{\left\{u_i\right\}_{i=0}^{N-1}, N \\ x(N)=x_{goal}}}{\operatorname{argmin}} \left\{ \sum_{i=0}^{N-1} L\left(x(i), u_i\right) \right\}$$

This is not an easy optimization, as that dimension of the space of solutions can be very high (N). A standard optimizer would find it too expensive.

$$\left\{u_i^*\right\}_{i=0}^{N^*-1} = \underset{\substack{\left\{u_i\right\}_{i=0}^{N-1}, N \\ x(N)=x_{goal}}}{\operatorname{argmin}} \left\{ \sum_{i=0}^{N-1} L\left(x(i), u_i\right) \right\}$$

$$\left\{u_i^*\right\}_{i=0}^{N^*-1} = \underset{\substack{\left\{u_i\right\}_{i=0}^{N-1},N \\ x(N)=x_{goal}}}{\operatorname{argmin}} \left\{\sum_{i=0}^{N-1} L\left(x(i),u_i\right)\right\}$$

This is not an easy optimization, as that dimension of the space if solutions can be very high (N). A standard optimizer would find it too expensive.

Suppose we had just 2 possible control actions. The number of solutions, by exhaustive search would be

$$2^N$$

What would happen if we needed N =30? Or N=100?

(in the mining truck example, we may need sequences of 300 or more steps, depending on the context and on the initial and goal states, and the cardinality of U is well higher then 2)

$$\left\{u_i^*\right\}_{i=0}^{N^*-1} = \underset{\substack{\left\{u_i\right\}_{i=0}^{N-1},N \\ x(N)=x_{goal}}}{\operatorname{argmin}} \left\{ \sum_{i=0}^{N-1} L\left(x(i), u_i\right) \right\}$$

The “The principle of Optimality” (by   Richard E. Bellman) will help us.

$$\left\{u_i^*\right\}_{i=0}^{N^*-1} = \underset{\substack{\left\{u_i\right\}_{i=0}^{N-1}, N \\ x(N)=x_{goal}}}{\operatorname{argmin}} \left\{ \sum_{i=0}^{N-1} L\left(x(i), u_i\right) \right\}$$

Bellman's Principle of Optimality

$$C^*\left(x; x_{goal}\right) = \min_{u \in U(x)} \left(C^*\left(z; x_{goal}\right) + L\left(x, u\right) \right)$$

$$z = f\left(x, u\right)$$

$$x \in \Omega$$

$$u \in U\left(x\right)$$

In which $C^*(\boldsymbol{x}; x_{goal})$ is the optimal cost to reach the goal from state ***x*** (via an optimal sequence of actions)

Using Bellman's Principle of Optimality

$$C^*\left(x; x_{goal}\right) = \min_{u \in U(x)} \left(C^*\left(z; x_{goal}\right) + L(x,u) \right)$$

$$z = f(x,u)$$

$$u \in U(x)$$

If I aim to reach x_{goal} and my current state is x,

the best decision is: $u^*\left(x; x_{goal}\right)$

$$u^*\left(x; x_{goal}\right) = \operatorname*{argmin}_{u \in U(x)} \left(C^*\left(z; x_{goal}\right) + L(x,u) \right)$$

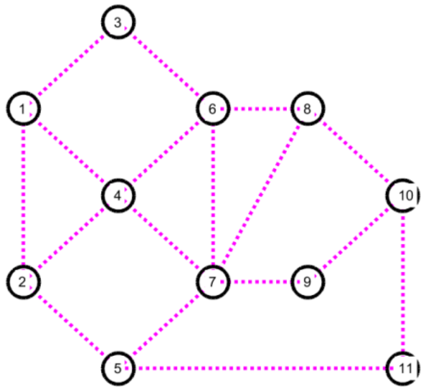
$$z = f(x,u)$$

If I aim to reach x_{goal} and my current state is x,

the best decision is: $u^*\left(x; x_{goal}\right)$

$$u^*\left(x; x_{goal}\right) = \underset{u \in U(x)}{\operatorname{argmin}}\left(C^*\left(z; x_{goal}\right) + L(x,u)\right)$$

$$z = f(x,u)$$



Very easy and cheap, for a given graph, if we knew the needed $C^*\left(z; x_{goal}\right)$

Bellman’s Principle of Optimality

$$C^*\left(x;x_{goal}\right)=\min_{u\in U(x)}\left(C^*\left(z;x_{goal}\right)+L\left(x,u\right)\right)$$

$$z=f\left(x,u\right)$$

$$u\in U\left(x\right)$$

But how do we know $C^*\left(z;x_{goal}\right)$?

$$\left\{u_i^*\right\}_{i=0}^{N^*-1} = \underset{\substack{\left\{u_i\right\}_{i=0}^{N-1},N \\ x(N)=x_{goal}}}{\operatorname{argmin}} \left\{ \sum_{i=0}^{N-1} L\left(x(i),u_i\right) \right\}$$

Bellman's Principle of Optimality

$$C^*\left(x;x_{goal}\right) = \min_{u \in U(x)} \left(C^*\left(z;x_{goal}\right) + L(x,u) \right)$$

$$z = f(x,u)$$

$$u \in U(x)$$

If we had $C^*\left(x;x_{goal}\right) \quad \forall \quad x \quad \in \quad \Omega$

Obtaining an optimal control action would be trivial.

So, now, we need an efficient method to obtain $C^*\left(x;x_{goal}\right)$

$$\left\{u_i^*\right\}_{i=0}^{N^*-1} = \underset{\substack{\left\{u_i\right\}_{i=0}^{N-1},N \\ x(N)=x_{goal}}}{\operatorname{argmin}} \left\{ \sum_{i=0}^{N-1} L\left(x(i),u_i\right) \right\}$$

The Dijkstra's Algorithm:

The method analyzes all the possible states for obtaining the optimal cost function C*. One way to do it is as follows.

First, we assign *C** at the goal state, *x*=x_{goal} ; we set *C**(x_{goal})=0.
This goal state is pushed into a Priority Queue (Priority Queue: we discuss this concept in class).

After this initialization step, the method operates in the following way (expressed through pseudo code):

% Initialize the cost for all the states *Cost(x) = "INFINITE" for all x*
% Priority queue is initially empty

```
Cost( $x_{goal}$ )=0;              % GOAL state has cost=0
Queue.Insert( $x_{goal}$ ,0);      % push this item into queue
                                   % "insert"
```

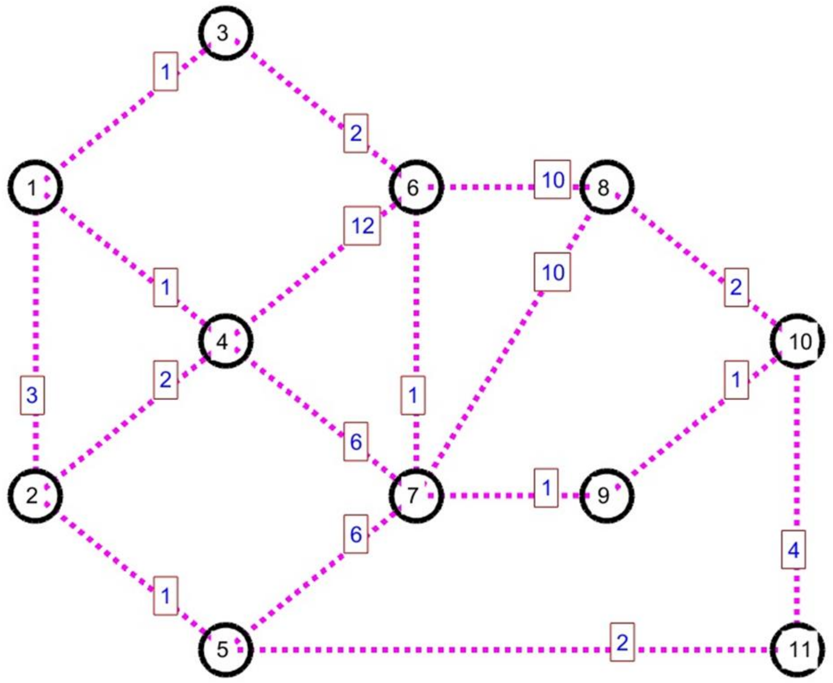
```
Loop, while Queue is not empty; and if I have not reached the "source" state
{           % "POP" : get the queue item that is currently at the top of the priority queue
            x = Queue.GetFirst ( );
            if $x == x_{sourceI}$ {          exit()  ; }  % SUCCESS! , exit loop. We reached the source state!
            % if We do not stop here, we will search the full reachable state domain (in that case, we will get the
            % optimal cost function C*(x) for all the domain, which in certain cases is required)

            % Generate the "children" states of  last "popped" state x
            LOOP: for each state y that could reach state x (via a feasible action u, e.g. that  satisfies x= f(y, u )
            {          %Evaluate the hypothetical cost of y (cost to reach the goal state),  based on the  cost of x and the
transition cost from y to x.

                       ProposedCost _y= Cost(x) + CostOfTransition y_to_x;
                       CurrentCost _y = Cost( y )

                       if ProposedCost < CurrentCost_y
                       {          Cost( y )= ProposedCost_y ;
                                  if  (CurrentCost_y==INFINITE),  {      Queue.Insert( y , ProposedCost_y );  }
                                  else                            {      Queue.Relocate( y , ProposedCost_y)  ; }
                       }
                       else
                       {          % nothing to do:  previous assigned cost to state y is better than proposed cost.
                       }
            }          //End Loop for "children"
}           //End loop popping from priority queue
```

//if we are here is because the queue is empty. This situation means FAILURE; there is no feasible solution for connecting xSource with the goal.
//(OR because we decided to obtain the full C*(X), for all the values of X).

The Dijkstra's Algorithm:

$$\left\{u_i^*\right\}_{i=0}^{N^*-1} = \underset{\substack{\left\{u_i\right\}_{i=0}^{N-1},N \\ x(N)=x_{goal}}}{\operatorname{argmin}} \left\{\sum_{i=0}^{N-1} L\left(x(i),u_i\right)\right\}$$

We see it working for a small case
(and in slow motion, via animation n Matlab)



$$\left\{u_i^*\right\}_{i=0}^{N^*-1} = \underset{\substack{\left\{u_i\right\}_{i=0}^{N-1}, N \\ x(N)=x_{goal}}}{\operatorname{argmin}} \left\{ \sum_{i=0}^{N-1} L\left(x(i), u_i\right) \right\}$$

The Dijkstra's Algorithm:

We see it for planning in a dense map, represented by an occupancy grid, in 2D.

$$\left\{u_i^*\right\}_{i=0}^{N^*-1} = \underset{\substack{\left\{u_i\right\}_{i=0}^{N-1},N \\ x(N)=x_{goal}}}{\operatorname{argmin}} \left\{ \sum_{i=0}^{N-1} L\left(x(i), u_i\right) \right\}$$

The Dijkstra's Algorithm:

We see it in planning , for a mining truck.
(position, heading and forward/reverse)

Planning

The platform moves and keeps performing localization and perception (e.g. mapping).

Re-plan periodically to adapt to changes (real or perceived) in the environment.

The planner is performed periodically for providing an updated optimal path. Of which the first part is actually applied.

We see a simulation. We note how the updated path varies as the belief about the context is refined.

We see it, in simulation, for a mining truck .

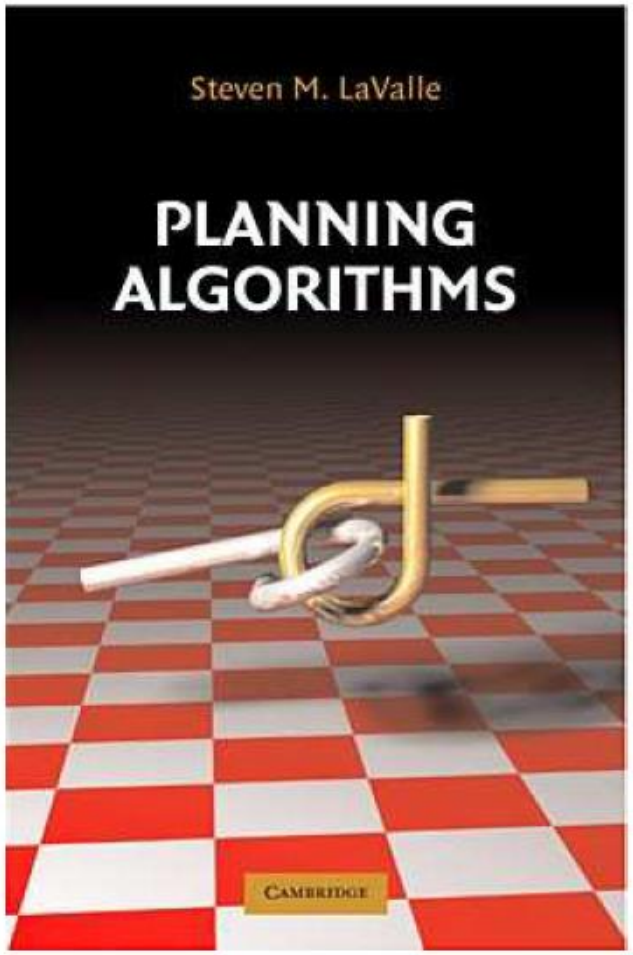
$$\left\{u_i^*\right\}_{i=0}^{N^*-1} = \underset{\substack{\left\{u_i\right\}_{i=0}^{N-1}, N \\ x(N)=x_{goal}}}{\operatorname{argmin}} \left\{ \sum_{i=0}^{N-1} L\left(x(i), u_i\right) \right\}$$

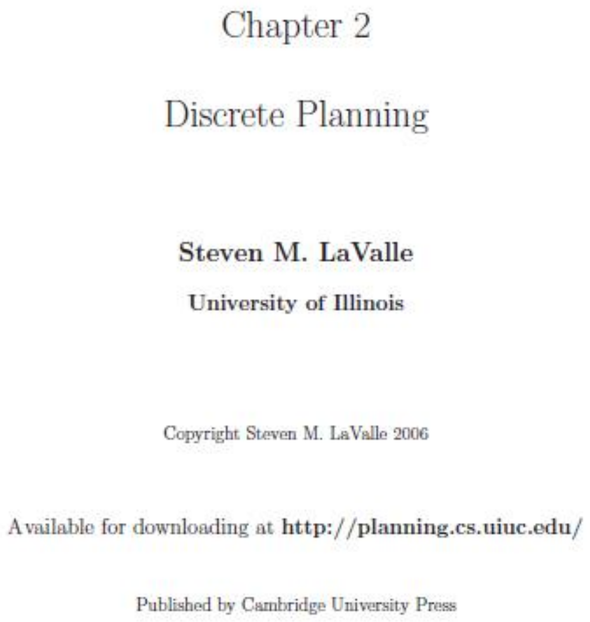
The Dijkstra's Algorithm:

Variants : A*, D*, RRT.

You may try Matlab toolbox (you do not need to implement those).

Good reference for Planning (including Dynamic Programming)





Chapter 2

Discrete Planning

Steven M. LaValle

University of Illinois

Copyright Steven M. LaValle 2006

Available for downloading at **http://planning.cs.uiuc.edu/**

Published by Cambridge University Press

Lecture ends here