6.7950 Fall 2022: - Recitation 2 Handout

1 Combinatorial Optimization via DP

This exercise explores the example briefly seen in lecture: the Traveling Salesman Problem (TSP). It's an example of combinatorial optimization problem that can be solved with some efficiency using DP.

We are given a list of N cities (the set C) and the travel costs between each pair: $c : C^2 \to \mathbb{R}$. We wish to find a plan to visit them, such that it

- Minimizes the total travel cost
- Visits each city exactly once and then returns to the starting city (i.e., a tour)

Note that the choice of starting city is arbitrary for a given tour, as the total cost remains the same for any choice.

Formulate the TSP into a DP problem. Discuss your formulation in terms of its complexity in the context of combinatorial problems in general.

Solution:

Naive formulation

We can define a problem where each state corresponds to the sequences of visited cities so far at time t: $\overline{s}_t = (s_1, s_2, \ldots, s_t)$. Then the problem starts at the first city (s_1) with $s_1 \in C$ and the first action a_1 is the next city $s_2 \in C$ to be visited, resulting in the next state $\overline{s}_2 = (s_1, s_2)$. We can restrict s_2 to belong to $C \setminus \{s_1\}$ in order to prevent the invalid assignment where s_1 is visited twice. This can be done more generally for a sequence of t cities with $\overline{s}_t = (s_1, \ldots, s_t)$ and with each action a_t selecting the next city $a_t \in C \setminus \{\overline{s}_t\}$, where $\{\overline{s}_t\} = \{s_1, \ldots, s_t\}$ is the set of all cities visited at the state $\overline{s}_t$

Each action would then incur a cost equal to the traveling cost between the last city in the state s_t and the one taken by the action $\overline{c}(\overline{s}_t, a_t) = c(\text{last}(\overline{s}_t), a_t)$. At the last state, we force the termination cost to be $\overline{d}_N(\overline{s}_N) = \overline{c}(\overline{s}_N, s_1)$, as the problem requires a complete tour.

This formulation would return a solution to the TSP problem as the state $\overline{s}_t$ contains a minimum length tour. However, solving it this way would be very inefficient as it would be necessary to compute the total costs for all possible sequence of cities, which is on the order of $O(N!)$. Despite its limitations, we could write a backwards DP recursion for the optimal cost-to-go $V(\overline{s}_t) = V(s_1, \ldots, s_t)$ as

$$V_t(s_1, \ldots, s_t) := \begin{cases} \min\limits_{a_t \in C \setminus \{\overline{s}_t\}} [c(s_t, a_t) + V_{t+1}(s_1, \ldots, s_t, a_t)] & t = 1, \ldots, N-1 \\ c(s_N, s_1) & t = N \end{cases}$$

With this method, we are not taking advantage of the fact that we don't need to know the exact ordering of cities, which is what we are going to do next.

Efficient formulation

One could also define as state the set of cities visited so far (i.e., without order) $\hat{S}_t$, plus the current city s_t, so the full state $\overline{s}_t$ would be $\overline{s}_t = (\hat{S}_t, s_t)$. At each non-terminal step, an action $a_t \in C \setminus \hat{S}_t$ with cost $c(s_t, a_t)$ defines the next city to be visited, so that the next state is $\overline{s}_{t_1} = (\hat{S}_t \cup a_t, a_t)$. The terminal cost at state $\overline{s}_N = (C, s_N)$ is then $c(s_N, s_1)$ in order to complete the tour. We might be tempted to reduce the state to only $\hat{S}_t$, but without knowing which city was visited last it's not possible to determine the cost to reach the next city to be visited.

We can then write its backward DP recursion for the cost-to-go $V_t(\overline{s}_t) = V(\hat{S}_t, s_t)$ as

$$V_t(\hat{S}_t, s_t) := \begin{cases} \min\limits_{a_t \in C \setminus \hat{S}_t} \left[c(s_t, a_t) + V_{t+1}(\hat{S}_t \cup a_t, a_t) \right] & t = 1, \dots, N-1 \\ c(s_t, s_1) & t = N \end{cases}$$

In comparison to the naive formulation, a DP solution using this scheme requires the computation of the value function for $O(2^N N)$ states, which is much better than $O(N!)$, but it's still really expensive. This complexity can be computed by considering the $O(2^N)$ possible subsets $\hat{S}_t$ of C and the $O(N)$ possible values for the last visited city s_t. Furthermore, as there's a minimization over $O(N)$ possible actions for each state, the computational complexity is then $O(2^N N^2)$.

Discussion

There are other formulations that could be used, but the TSP is NP-hard, so it's fundamentally a challenging problem to the best of or knowledge, much like many important combinatorial optimization problems. Without further assumptions, there are no known bounds better than $O(2^N N^2)$. In practice, we may want to avoid exact DP algorithms for solving combinatorial optimization and instead favor approximate DP algorithms (ADP). We will see several ADP algorithms in the following lectures (e.g., the rollout algorithm). It is a recent research topic of applying deep reinforcement learning algorithms to solve combinatorial optimization problems, e.g., see the paper Neural Combinatorial Optimization with Reinforcement Learning that uses recurrent neural nets and policy gradient (we will also cover this method in future) to solve the TSP problem. A survey of RL methods for optimization including the TSP and other classical problems can be found in the paper Reinforcement Learning for Combinatorial Optimization: A Survey

2 LQR and control problems

In this section we will explore some aspects of control theory, under which the LQR is one fundamental solution method we have seen in lecture. To properly cover these topics, one would need several lectures in a typical control course, so the goal here is to give a brief and non-rigorous overview of what types of problems are being solved and how some issues are typically handled. The key takeaways are:

- Control problems are situations where we want to find a set of actions (or a policy) to manipulate a dynamical system in order for it evolve in a more desirable manner
- The discrete LQR is a solution to a particular MDP with linear dynamics and quadratic costs
- Even if the LQR uses linear dynamics, it can help tackle problems with nonlinear ones

- Though there are continuous-time versions of LQR and MDPs, you can approximate continuous-time problems with discrete-time ones

2.1 The problem we are trying to solve: what is control?

A control problem typically starts with a dynamical system, that is, a system that evolves over time according to some rules, called the dynamics. It can often be described by a set of states x, a set of control actions u (that is, actions that can be take to influence its behavior) and certain external disturbances w such that the time evolution of x is given by $\dot{x} = f(t, x, u, w)$.

Sometimes we can't observe x directly, but instead have a set of observations y that gives us some information about the system according to some observation model h, such that $y = h(t, x, u, v)$, where v is a source of observation disturbance.

Given this dynamical system, we are interested in minimizing some loss function L over time time interval $[t_0, t_f]$. This loss may be a function of all the previously defined variables: $L = L(x(t), u(t), y(t), v(t), w(t))$.

This may seem like a very abstract problem, but it can represent many practical scenario. For example, a driver control a car through the control actions of turning the wheel, stepping on the gas pedal, switching gears, amongst other things. An aircraft, whether controlled by human or not, needs to adjust another set of control actions in order to fly properly. Other control problems can arise in chemical systems (e.g., how to adjust the temperature and flow of reagents in a mixture), in electronics (e.g., how to regulate the tension in order to keep a current constant) and in many other scenarios.

In that context, we can better understand what the "R" part in LQR means. A regulator is a control system that aims to set certain states to 0 or another fixed value. In the driving example, one may want to keep their speed at a constant value below the speed limit or an aircraft may need to remain at a constant altitude during cruise. This is in contrast with the problem of tracking, where one has a specific trajectory $\hat{x}(t)$ that needs to be followed (e.g., an aircraft making a turn or a holding pattern).

Note that there are many variations of the basic control problem. For example, the dynamics of the system can be unknown in many different ways, either by which set of equations describe them or by which parameters (or range of parameters) can be expected. In some of these situations, good solutions can be found in the control literature, but in others it may be beneficial to explore reinforcement learning methods. Homework 0 had an example with the cartpole problem, where the objective is to keep a pole upright by adjusting the horizontal position of a cart. In this case, classical control literature is more than enough, but the problem serves as a simple benchmark for different RL methods. Thus, even if we are not going to study classical control theory in the rest of the course, it's beneficial to be aware of the connections between the fields and when one would be more applicable than the other.

2.2 LQR as a special case of MDP

For the purpose of this course, we are interested in solving MDP problems of the form

$$\begin{aligned} &\text{maximize } \mathbb{E}\left[\sum_{t=1}^{N} r_t(x_t, u_t, x_{t+1})\right] \\ &\text{subject to } x_{t+1} = f(x_t, u_t, w_t) \end{aligned}$$

It can be verified that the LQR problem is a special case of that problem where we take

$$f(x_t, u_t, w_t) = Ax_t + Bu_t + w_t \text{ (linear dynamics)}$$
$$-r_t(x_t, u_t, x_{t+1}) = \frac{1}{2}x_t^\top Q_t x_t + \frac{1}{2}u_t^\top R u_t \text{ (quadratic costs)}$$

For many MDPs, we won't be able to find an analytical solutions. The LQR problem (like the inventory control problem we have seen in the previous recitation) is one such example, so it's worth investigating it a bit further.

2.3 Nonlinear to linear dynamics

The nonlinear dynamics, if they are differentiable, can be approximated by using its partial derivatives around a points $p_0 = (x_0, u_0, w_0)$

$$\dot{x} \approx f(x_0, u_0, w_0) + \left(\frac{\partial f}{\partial x}\right)_{p_0}(x - x_0) + \left(\frac{\partial f}{\partial u}\right)_{p_0}(u - u_0) + \left(\frac{\partial f}{\partial w}\right)_{p_0}(w - w_0) + \left(\frac{\partial f}{\partial x}\right)_{p_0} \tag{1}$$

If we pick p_0 to be an equilibrium point $f(x_0, u_0, w_0) = 0$ and define the deviations from this points as $\overline{x} = x - x_0$, $\overline{u} = u - u_0$ and $\overline{w} = w - w_0$, then the dynamics become linear with additive noise:

$$\dot{\overline{x}} = \frac{\partial f}{\partial x}\overline{x} + \frac{\partial f}{\partial u}\overline{u} + \frac{\partial f}{\partial w}\overline{w} \tag{2}$$

Note that, since $\overline{x}$, $\overline{u}$ and $\overline{w}$ are in general vectors, then the partial derivatives $\frac{\partial f}{\partial x} = A_c$, $\frac{\partial f}{\partial u} = B_c$ and $\frac{\partial f}{\partial w} = W_c$ are matrices and perhaps it's easier to visualize the dynamics in a more familiar form as

$$\dot{\overline{x}} = A_c\overline{x} + B_c\overline{u} + W_c\overline{w}$$

where the subscript were added to the matrices to emphasize the fact they correspond to the continuous-time version of the problem. For notation convenience, one often drops any symbol that would indicate that $\overline{x}$ is actually a deviation from an equilibrium point in most situations, as the components of the linear approximation are written many times. This can be confusing at first, but it's important to remember as it better contextualizes the value $x = 0$, the target of a regulator, as deviations from any nominal value.

Though this approximation is only good in the neighborhood of p_0, it can be done for many points inside a region of interest (e.g., if you discretize the state-space) or along a nominal trajectory (e.g., for an aircraft doing an aerobatic maneuver). There also techniques, such as nonlinear dynamic inversion, that "cancel" the nonlinear parts, leaving a linear system to be controlled. The key takeaway is that many nonlinear control problems can be tackled with the use of linear controllers.

2.4 The weight matrices

For the LQR problem, the Q and R matrices penalize, respectively, deviations in the state and control actions. Effectively, they balance how much error in the states the system is allowed to have and how much control effort is used. A good choice for them is a problem-specific task, although there a good rules of thumb, such as Bryson's rule (where, among other things, each variable is normalized by their expected range). Let's try to build some intuitions by look at the case of a diagonal matrix Q_d

$$Q_d = \begin{bmatrix} q_{11} & & & \\ & q_{22} & & \\ & & \ddots & \\ & & & q_{nn} \end{bmatrix} \tag{3}$$

If we work out the values of $x^\top Q_d x$, we will notice that it's equal to

$$x^T Q_d x = \sum_{i=1}^{n} q_{ii} x_i^2$$

In other words, we are penalizing the square of the deviations x_i with some weights q_{ii}. This seems like a reasonable approach as minimizing square errors is a standard choice in many problems. The use of a more general matrix $Q \succeq 0$ doesn't fundamentally alter the cost as its eigendecomposition is $Q = M^\top \Lambda M$, for some orthogonal eigenvector matrix M, and then the error term becomes $x^\top Q x = (Mx)^\top \Lambda M x = v^\top \Lambda v$. That means that it's also a quadratic cost with respect to a linear combination v of the errors x using the diagonal matrix of eigenvectors Λ. The same reasoning applies to the R matrix.

2.5 Continuous to discrete time

Let the continuous time dynamics be

$$\dot{x} = h(x(t), u(t), w(t)) \tag{4}$$

then a zero order hold assumes that $x(t)$, $u(t)$ and $w(t)$ maintain constant values x_t, u_t and w_t for a small period of time Δt, which ends in $x(t + \Delta t) = x_{t+1}$. The resulting approximate dynamics can be expressed as

$$h(x_t, u_t, w_t) = \dot{x} \approx \frac{x_{t+1} - x_t}{\Delta t} \tag{5}$$

$$\therefore x_{t+1} = x_t + \Delta_t h(x_t, u_t, w_t) = f(x_t, u_t, w_t) \tag{6}$$

which defines a discrete approximation for the continuous time system. Note that more elaborate approximations can be used, such as treating the continuous variable as linearly increasing instead of fixed at each time interval (naturally, that's called a first order hold).

2.6 An illustrative example: Newton's second law

Given an object of mass m subject to the gravity g and a force F, its position p is subject to the dynamics

$$\ddot{p} = \frac{F}{m} - g \tag{7}$$

We can think of this problem, for example, as a quadcopter trying to hover given a total propeller thrust F. It's convenient to rewrite this second order equation into a system of first order equations with position $x_1 = p$ and velocity $x_2 = \dot{p}$ as

$$\dot{x} = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \left(\frac{F}{m} - g \right) \tag{8}$$

This is not linear yet because of the constant terms proportional to g. So we take the linearization point $(0, 0, mg)$ with input $F = mg + u$, ending up with the linear system

$$\dot{x} = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} x + \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} u \tag{9}$$

To make it discrete in time, we use a zero order hold over the time Δt, thus obtaining

$$x_{t+1} = x_t + \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} x_{t+1}\Delta_t + \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} u_t\Delta_t = \begin{bmatrix} 1 & \Delta_t \\ 0 & 1 \end{bmatrix} x_t + \begin{bmatrix} 0 \\ \frac{\Delta_t}{m} \end{bmatrix} u_t = Ax_t + Bu_t \tag{10}$$

Thus, we now have the matrices A and B for our linear approximation. Assuming an infinite horizon for simplicity, we still need to find reasonable values for Q and R in order to apply the proposed LQR method. For this simple example, we can set representative values as

$$Q = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \; R = \begin{bmatrix} 1 \end{bmatrix} \tag{11}$$

In other words, we will penalize deviations from a nominal position at 0 used for linearization. As the system is controllable (you can check the controllability matrix), the resulting optimal controller will maintain the position at 0 and, by consequence also the velocity. We could go a step further and set $R = \begin{bmatrix} 0 \end{bmatrix}$ and the system would be stabilized at 0 with no control input (otherwise it would deviate from $x = 0$, even if u is not directly penalized).

In this case we set the linearization to be $(0, 0, mg)$, so the system is regulate to stay near the value $x_1 = x_2 = 0$. This was done for simplicity and notational convenience as there's no need to change the name variables x_1 and x_2. Inspired by the quadcopter interpretation of the problem, we could take a hover height h and set the equilibrium points to be $(h, 0, mg)$. We can then proceed analogously to arrive at a regulator that penalizes deviations from $\overline{x}_1 = x_1 - h$ and as a consequence it would keep the vehicle hovering at the desired altitude since having $\overline{x}_1 \to 0$ means that $x_1 \to h$.