Introduction to Deep Reinforcement Learning

Jill-Jênn Vie

November 14, 2025

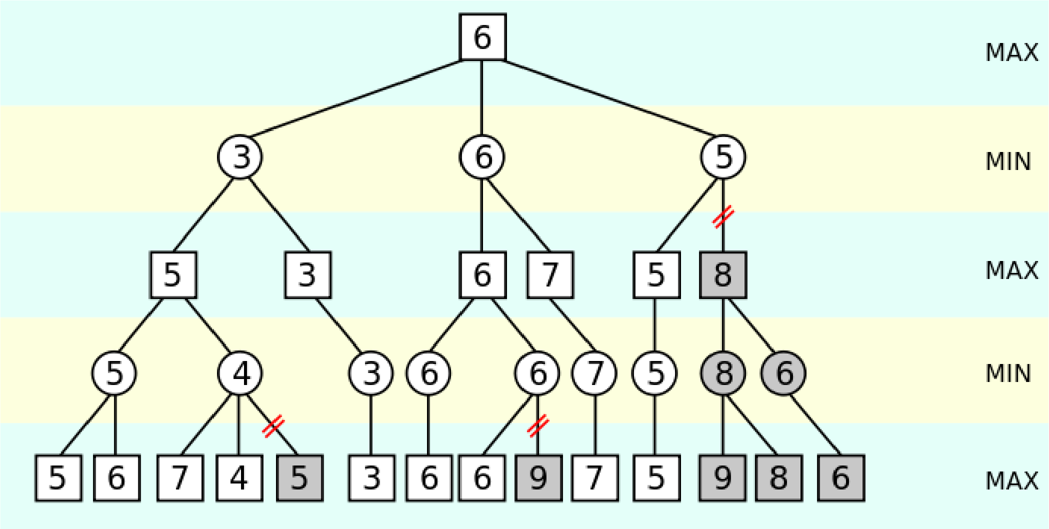
Reinforcement learning

- Train an agent from interaction with an environment
- Actions from the agent impact the data collected

Applications

- Robotics, self-driving cars
- Games
- Recommender systems
- LLMs like ChatGPT

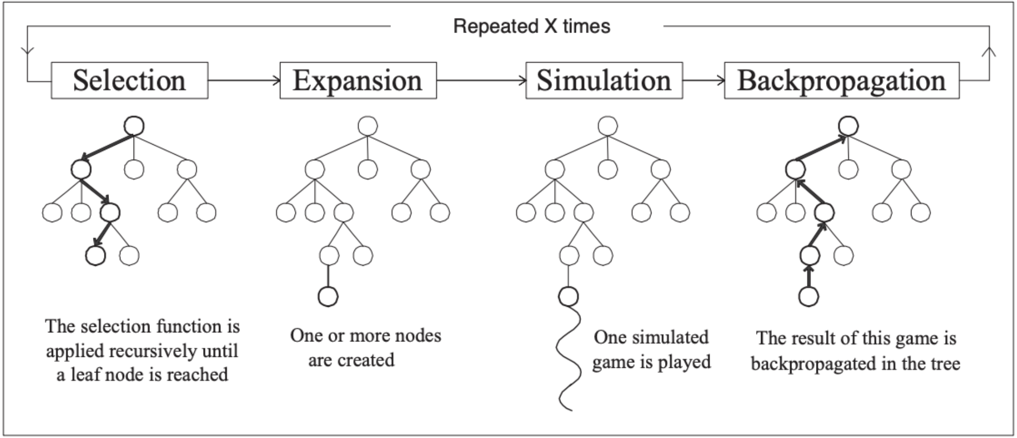
Games on trees or DAGs: minimax (von Neumann, 1928; McCarthy, 1958)



Monte Carlo Tree Search (coined by Rémi Coulom, Inria Lille, in 2006)

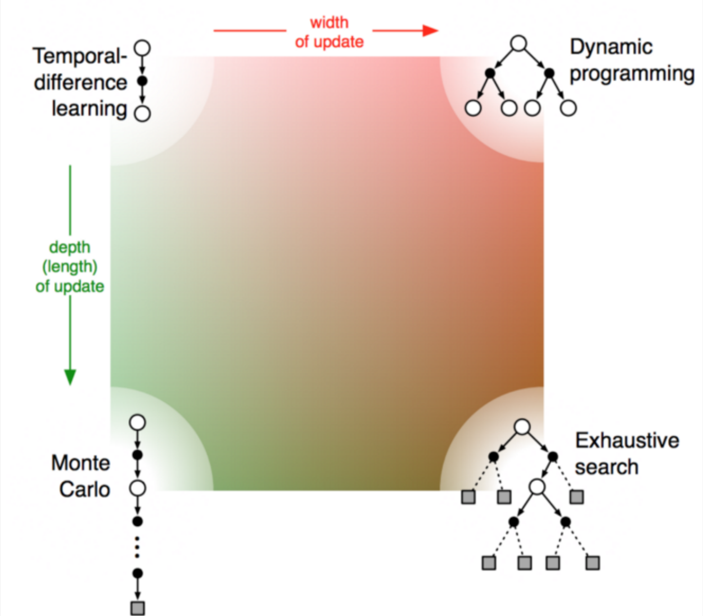
1987: Bruce Abramson, minimax search with an expected-outcome model
2012: Cameron Browne et al. "A survey of monte carlo tree search methods."
2015: AlphaGo combines MCTS and deep learning and beats professional human Go player

Image source: Chaslot et al. (2008)



Algorithm/Discovery	Nature Paper Title	Core Breakthrough
Atari (DQN)	*Human-level control through deep reinforcement learning* (2015)	First time an agent combined **deep neural networks** with **Q-learning** (a model-free method) to achieve human-level performance across a wide range of visually complex tasks (Atari games).
AlphaGo	*Mastering the game of Go with deep neural networks and tree search* (2016)	First AI to defeat a top professional human Go player. It combined **Supervised Learning** (pre-training on human games) with **Reinforcement Learning** (self-play) and **Monte Carlo Tree Search (MCTS)**.
AlphaGo Zero	*Mastering the game of Go without human knowledge* (2017)	**Eliminated human data**. Trained purely through **self-play** (tabula rasa), achieving stronger performance than AlphaGo. Simplified the network architecture and used a **single neural network** for both policy and value.
MuZero	*Mastering Atari, Go, Chess and Shogi by planning with a learned model* (2020)	**Eliminated knowledge of environment rules**. Masters board games and Atari games by **learning a compressed model** of the environment's dynamics, focusing only on predicting **reward, value, and policy**—the elements essential for planning.

Various aspects of RL (Sutton & Barto)



In RL, episodes are sampled according to a policy

Agent observes state S_t chooses action A_t
You can control the agent: policy $\pi(a|s) = \pi(A_t = a|S_t = s)$

$$S_t \to A_t \to R_t \to S_{t+1} \to A_{t+1} \to \cdots \to R_T$$

This is called one *episode* of learning.

In RL, episodes are sampled according to a policy

Agent observes state S_t chooses action A_t
You can control the agent: policy $\pi(a|s) = \pi(A_t = a|S_t = s)$

Environment replies with reward R_t and new state S_{t+1}
You cannot control the environment: $p(s', r|s, a) = p(S_{t+1} = s', R_t = r|S_t = s, A_t = a)$

$$S_t \to A_t \to R_t \to S_{t+1} \to A_{t+1} \to \cdots \to R_T$$

This is called one *episode* of learning.

In RL, episodes are sampled according to a policy

Agent observes state S_{t+1} chooses action A_{t+1}
You can control the agent: policy $\pi(a|s) = \pi(A_t = a|S_t = s)$

Environment replies with reward R_t and new state S_{t+1}
You cannot control the environment: $p(s', r|s, a) = p(S_{t+1} = s', R_t = r|S_t = s, A_t = a)$

$$S_t \to A_t \to R_t \to S_{t+1} \to A_{t+1} \to \cdots \to R_T$$

This is called one *episode* of learning.

Objectives

Let $\gamma \in (0, 1)$ be a discount factor

The goal is to optimize the *discounted return*:

$$G_t = R_{t+1} + \gamma R_{t+2} + \ldots = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k \quad \text{where } T \text{ can be } \infty$$

The *value function* v_π for a policy π is $v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$

The *action-value function* q_π for a policy π is $q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a]$

What can be done with these functions?

Policy evaluation

Given policy π, compute value v_π

$$v_\pi(s) = \mathbb{E}_a \; q_\pi(s, a) = \sum_a \pi(a|s) q_\pi(s, a)$$

Policy improvement

Given value v_π on states, improve policy π
i.e. pick best action a: $\operatorname{argmax}_a q_\pi(s, a)$

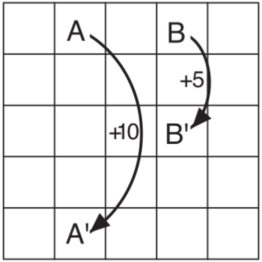
$$q_\pi(s, a) = \mathbb{E}_\pi[r + \gamma v_\pi(s')] = \sum_{r,s'} p(r, s'|s, a)[r + \gamma v_\pi(s')]$$

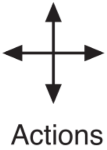
A first example: policy evaluation on Gridworld

$$S \in \{(i,j)\}_{1 \le i,j \le 5}$$

$$A \in \{\leftarrow, \rightarrow, \uparrow, \downarrow\}$$

$$R = \begin{cases} 10 \text{ if leaving from A} \\ 5 \text{ if leaving from B} \\ -1 \text{ if hitting a wall} \\ 0 \text{ otherwise.} \end{cases}$$

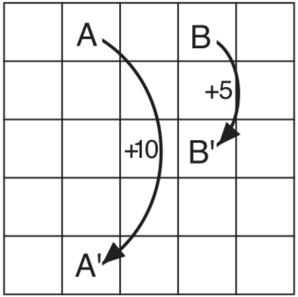




3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

Gridworld $\qquad \pi_{\text{uniform}} = (1/4, 1/4, 1/4, 1/4) \qquad v_{\pi_{\text{uniform}}}$

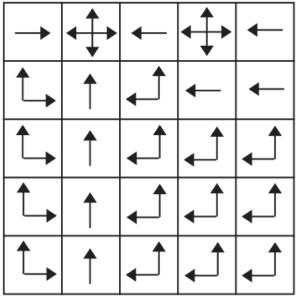
A first example: policy improvement on Gridworld



Gridworld

22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7

v_*



π_*

Richard Bellman's principle of optimality (1952) for dynamic programming

An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.

Bellman's equation

$$\underbrace{V(s)}_{\text{value}} = \max_a \underbrace{R(s,a)}_{\text{reward}} + \gamma V(\underbrace{T(s,a)}_{\text{transition}})$$

$$v_{\pi^*}(s) = \max_a \mathbb{E}_{\pi^*}[r + \gamma v_{\pi^*}(s')]$$

Rings a bell?

Richard Bellman's principle of optimality (1952) for dynamic programming

An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.

Bellman's equation

$$\underbrace{V(s)}_{\text{value}} = \max_a \underbrace{R(s,a)}_{\text{reward}} + \gamma V(\underbrace{T(s,a)}_{\text{transition}})$$

$$v_{\pi^*}(s) = \max_a \mathbb{E}_{\pi^*}[r + \gamma v_{\pi^*}(s')]$$

Rings a bell?

Bellman-Ford in shortest paths

$$V(u) = \max_v -w_{u,v} + V(v) \quad V = -d \quad \gamma = 1$$

Bellman-Ford

```
const int oo = 1e9;
int n, m;
int dist[N];
vector<tuple<int, int, int>> edge;

bool bellmanford (int u0) {
        fill_n (dist, n, +oo);
        dist[u0] = 0;
        bool stable = false;
        for (int t = 0; t < n && !stable; t++) {
                stable = true;
                for (auto[u, v, c] : edge) if (dist[u] < +oo && dist[u] + c < dist[v]) {
                        dist[v] = dist[u] + c;
                        stable = false;
                }
        }
        return stable;
}
```

Repeat at most $|V|$ times

For each edge $(u, v) \in E$

$d(v) = \min(d(v), d(u) + w_{uv})$

$d_k[v]$ = shortest length from source to v using at most k edges

Value iteration

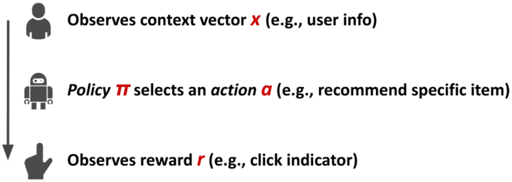
Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
| $\Delta \leftarrow 0$
| Loop for each $s \in \mathcal{S}$:
| $v \leftarrow V(s)$
| $V(s) \leftarrow \max_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma V(s')\big]$
| $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
$\pi(s) = \operatorname{argmax}_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma V(s')\big]$

Particular case: $\gamma = 0$ is the setting of contextual bandits



$$V(\pi) = \mathbb{E}r = \int_s \int_a \int_r \overbrace{p(s)}^{\text{observe student context } s} \; \overbrace{\pi(a \mid s)}^{\text{select activity } a \text{ using policy}} \; \underbrace{p(r \mid s, a) r}_{\text{observe reward } r} \, ds \, da \, dr$$

Online learning of classifiers: if the classifier looks at an image x picks a class y it may receive a reward of 1 if it picked the right class, -1 otherwise

ChatGPT: given a prompt x it should choose an answer y and gets a reward $r(x, y)$ that it estimates itself from preferences

Reinforcement learning from human feedback: InstructGPT, ChatGPT

1. Collect demonstration data, and train a supervised policy $\pi_0(y|x)$ (based on GPT-3)
2. Collect comparison data, train a reward model (Elo rating from “only” 50k pairs)

$$\text{loss}(\theta) = -\mathbb{E}_{(x,y_w,y_\ell)\sim D} \log \underbrace{\sigma(r_\theta(x, y_k) - r_\theta(x, y_\ell))}_{\text{Pr(”answer } y_k \text{ is preferred to } y_\ell\text{)”}}$$

3. Optimize a policy against the reward model using PPO.

$$\text{objective}(\phi) = \mathbb{E}_{(x,y)\sim\pi_\phi} r_\theta(x, y) - \beta \text{KL}(\pi_\phi, \pi_0)$$

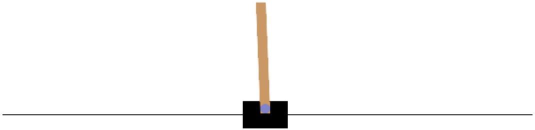
Ouyang, Wu, Jiang, Almeida, Wainwright, Mishkin, ... & Lowe (NeurIPS 2022). Training language models to follow instructions with human feedback.

A recent paper suggests to drop part 3 (RL) and focus on part 2 (supervised reward model):

Rafailov, Sharma, Mitchell, Ermon, Manning and Finn (arXiv 2023.06). Direct Preference Optimization: Your Language Model is Secretly a Reward Model.

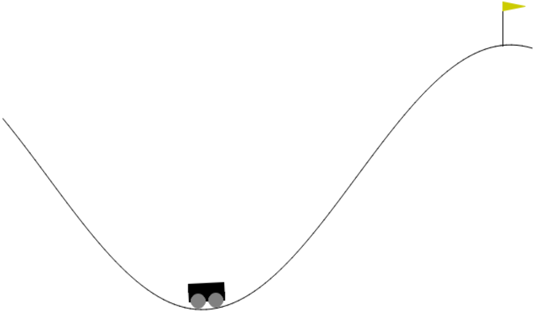
Practical

CartPole-v1



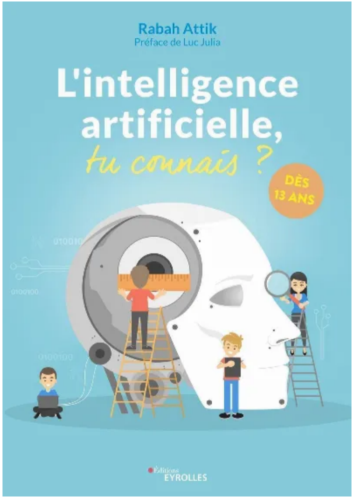
$S = (x, \dot{x}, \theta, \dot{\theta}) \in \mathbb{R}^4$
$A \in \{\leftarrow, \rightarrow\}$
$R = 1$

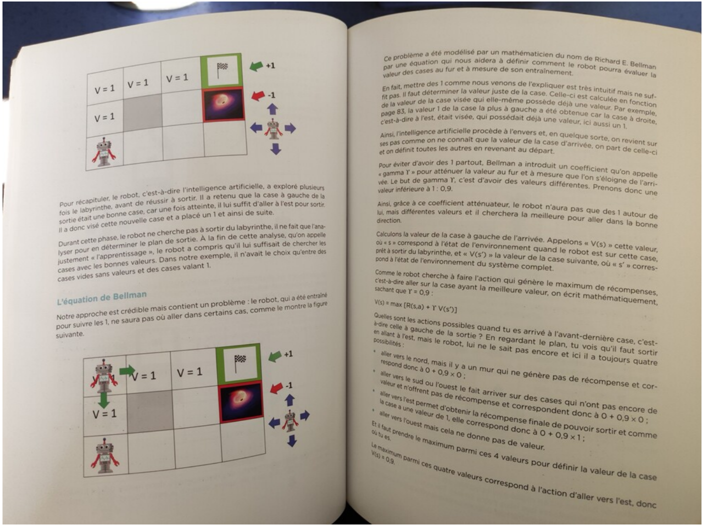
MountainCar-v2



$S = (x, \dot{x}) \in \mathbb{R}^2$
$A \in \{\leftarrow, 0, \rightarrow\}$
$R = -1$

References





Pour récapituler, le robot, c'est-à-dire l'intelligence artificielle, a exploré plusieurs fois le labyrinthe, avant de réussir à sortir. Il a retenu que la case à gauche de la sortie était une bonne case, car une fois atteinte, il lui suffit d'aller à l'est pour sortir. Il a donc visé cette nouvelle case et a placé un 1 et ainsi de suite.

Durant cette phase, le robot ne cherche pas à sortir du labyrinthe, il ne fait que l'analyser pour en déterminer le plan de sortie. À la fin de cette analyse, qu'on appelle justement « l'apprentissage », le robot a compris qu'il lui suffisait de chercher les cases avec les bonnes valeurs. Dans notre exemple, il n'avait le choix qu'entre des cases vides sans valeurs et des cases valant 1.

L'équation de Bellman

Notre approche est crédible mais contient un problème : le robot, qui a été entraîné pour suivre les 1, ne saura pas où aller dans certains cas, comme le montre la figure suivante.

Ce problème a été modélisé par un mathématicien du nom de Richard E. Bellman par une équation qui nous aidera à définir comment le robot pourra évaluer la valeur des cases au fur et à mesure de son entraînement.

En fait, mettre des 1 comme nous venons de l'expliquer est très intuitif mais ne suffit pas. Il faut déterminer la valeur juste de la case. Celle-ci est calculée en fonction de la valeur de la case visée qui elle-même possède déjà une valeur. Par exemple, page 83, la valeur 1 de la case la plus à gauche a été obtenue car la case à droite, c'est-à-dire à l'est, était visée, qui possédait déjà une valeur, ici aussi un 1.

Ainsi, l'intelligence artificielle procède à l'envers et, en quelque sorte, on revient sur ses pas comme on ne connait que la valeur de la case d'arrivée, on part de celle-ci et on définit toutes les autres en revenant au départ.

Pour éviter d'avoir des 1 partout, Bellman a introduit un coefficient qu'on appelle « gamma ϒ » pour atténuer la valeur au fur et à mesure que l'on s'éloigne de l'arrivée. Le but de gamma ϒ, c'est d'avoir des valeurs différentes. Prenons donc une valeur inférieure à 1 : 0,9.

Ainsi, grâce à ce coefficient atténuateur, le robot n'aura pas que des 1 autour de lui, mais différentes valeurs et il cherchera la meilleure pour aller dans la bonne direction.

Calculons la valeur de la case à gauche de l'arrivée. Appelons « V(s) » cette valeur, où « s » correspond à l'état de l'environnement quand le robot est sur cette case, prêt à sortir du labyrinthe, et « V(s') » la valeur de la case suivante, où « s' » correspond à l'état de l'environnement du système complet.

Comme le robot cherche à faire l'action qui génère le maximum de récompenses, c'est-à-dire aller sur la case ayant la meilleure valeur, on écrit mathématiquement, sachant que $\Upsilon = 0{,}9$:

$$V(s) = \max\,[R(s,a) + \Upsilon\, V(s')]$$

Quelles sont les actions possibles quand tu es arrivé à l'avant-dernière case, c'est-à-dire celle à gauche de la sortie ? En regardant le plan, tu vois qu'il faut sortir en allant à l'est, mais le robot, lui ne le sait pas encore et ici il a toujours quatre possibilités :

- aller vers le nord, mais il y a un mur qui ne génère pas de récompense et correspond donc à $0 + 0{,}9 \times 0$;
- aller vers le sud ou l'ouest le fait arriver sur des cases qui n'ont pas encore de valeur et n'offrent pas de récompense et correspondent donc à $0 + 0{,}9 \times 0$;
- aller vers l'est permet d'obtenir la récompense finale de pouvoir sortir et comme la case a une valeur de 1, elle correspond donc à $0 + 0{,}9 \times 1$;
- aller vers l'ouest mais cela ne donne pas de valeur.

Et il faut prendre le maximum parmi ces 4 valeurs pour définir la valeur de la case où tu es.

Le maximum parmi ces quatre valeurs correspond à l'action d'aller vers l'est, donc $V(s) = 0{,}9$.

Code une voiture autonome avec Python

Dans cet exercice, tu vas jouer en ligne et entraîner une voiture autonome. Comment vas-tu procéder ? Avant le début de la course sur le circuit, tu pourras activer un enregistrement qui récupérera les multiples images de la vidéo de ton jeu qui serviront à générer un modèle basé sur un réseau de neurones à convolution. Te rappelles-tu des réseaux spécialisés pour la reconnaissance d'images ?

Nous nous appuierons sur les outils développés par NVIDIA, une société américaine spécialisée dans la fabrication de cartes graphiques qui développe depuis 2017 de nouvelles cartes pour l'intelligence artificielle.

Les voitures autonomes sont l'un des domaines de recherche et d'activité les plus dynamiques. Ce qui semblait relever de la

This basic equation can be used to study optimal trajectories, satellite launching, geodesics, paths of minimum length through discrete networks, r-th shortest paths through networks, soft landing on the moon and other planets, chemical process control, reactor control, and many other processes and problems.

Questions of computational solution will be discussed in the next chapter.

4.17 The Brachistochrone

As an example of a problem in the calculus of variations which can be explicitly resolved, either by classical techniques, or by means of the functional equation technique which we shall pursue here, let us consider the problem of determining a curve connecting the two points P and Q with the property that a particle sliding along this curve under the influence of gravity will arrive at Q in a minimum time.

Without loss of generality, let P be the origin, and let the axes be oriented as indicated below.

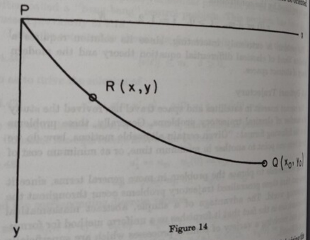


Figure 14

Omitting the gravitational constant, the problem is that of minimizing the functional

$$J(y) = \int_0^{x_0} \left(\frac{1+y'^2}{y}\right)^{1/2} dx, \tag{4.56}$$

over all curves satisfying the end conditions

$$y(0) = 0, \quad y(x_0) = y_0. \tag{4.57}$$

To treat the problem by dynamic programming techniques, we let $f(x, y)$ denote the minimum time required to go from the generic point $R(x, y)$ to (x_0, y_0). The equation in (4.56) is in this case

$$f(x, y) = \operatorname*{Min}_{y'} \left[\left(\frac{1+y'^2}{y}\right)^{1/2} \Delta + f(x+\Delta, y+y'\Delta) + O(\Delta^2) \right].^{*} \tag{4.58}$$

Passing to the limit as $\Delta \to 0$, this yields the nonlinear partial differential equation

$$0 = \operatorname*{Min}_{y'} \left[\left(\frac{1+y'^2}{y}\right)^{1/2} + f_x + y'f_y \right], \tag{4.59}$$

equivalent to the two equations

$$\begin{aligned} &(a) \quad 0 = \left(\frac{1+y'^2}{y}\right)^{1/2} + f_x + y'f_y, \\ &(b) \quad 0 = \frac{y'}{[y(1+y'^2)]^{1/2}} + f_y. \end{aligned} \tag{4.60}$$

To eliminate the function f, and thus obtain an equation for y', we differentiate (4.60b) with respect to x and take the partial derivative of (4.60a) with respect to y, obtaining

$$\begin{aligned} &(a) \quad \frac{d}{dx}\left[\frac{y'}{(y(1+y'^2))^{1/2}}\right] + f_{yx} + f_{yy}y' = 0, \\ &(b) \quad -\tfrac{1}{2}(1+y'^2)^{1/2}y^{-3/2} + f_{xy} + f_{yy}y' = 0. \end{aligned} \tag{4.61}$$

Thus

$$\frac{d}{dx}\left[\frac{y'}{(y(1+y'^2))^{1/2}}\right] + \tfrac{1}{2}(1+y'^2)y^{-3/2} = 0. \tag{4.62}$$

A first integral is

$$\left(\frac{1+y'^2}{y}\right)^{1/2} - \frac{y'^2}{[(1+y'^2)y]^{1/2}} = k, \tag{4.63}$$

or

$$\frac{1}{[y(1+y'^2)]^{1/2}} = k. \tag{4.64}$$

This is equivalent to Snell's law for the propagation of light. It is interesting to interpret this well-known physical principle as an optimal policy in a multistage decision process.

The integration can now be readily completed to obtain the standard representation of a brachistochrone in parameteric form:

$$\begin{aligned} y &= (1 - \cos t)/2k^2, \\ x &= c_1 + (t - \sin t)/2k^2. \end{aligned} \tag{4.65}$$

* We have deliberately kept y' as the slope, instead of v, in order to manipulate the Euler equation.

The great drawback of dynamic programming is, as Bellman himself calls it, the "curse of dimensionality." Even recording the solution to a moderately complicated problem involves an enormous amount of storage. If we want only one optimal path from a known initial point, it is wasteful and tedious to find a whole field of extremals; if we need feedback, a perturbation feedback scheme is often quite adequate (see Chapter 6, Neighboring Extremals and the Second Variation).

Derivation of the Euler-Lagrange equations from the Hamilton-Jacobi equation. Consider a particular optimal path and its associated optimal control function. Then we have

$$\frac{d\lambda^T}{dt} \equiv \frac{d}{dt}\left(\frac{\partial J^\circ}{\partial x}\right) = \frac{\partial^2 J^\circ}{\partial x^2}\dot{x} + \frac{\partial^2 J^\circ}{\partial x \partial t}. \tag{4.2.19}$$

Partial differentiation of (4.2.15) with respect to x, considering $u^\circ = u^\circ(x,t)$, gives

$$\frac{\partial^2 J^\circ}{\partial x \partial t} + \frac{\partial L}{\partial x} + \frac{\partial L}{\partial u}\frac{\partial u^\circ}{\partial x} + f^T\frac{\partial^2 J^\circ}{\partial x^2} + \frac{\partial J^\circ}{\partial x}\left(\frac{\partial f}{\partial x} + \frac{\partial f}{\partial u}\frac{\partial u^\circ}{\partial x}\right) = 0. \tag{4.2.20}$$

Now the coefficient of $\partial u^\circ/\partial x$ in (4.2.20) vanishes on an optimal path according to (4.2.17).† Using (4.2.20) in (4.2.19), then, we obtain

$$\frac{d\lambda^T}{dt} = -\lambda^T\frac{\partial f}{\partial x} - \frac{\partial L}{\partial x}, \tag{4.2.21}$$

which, along with (4.2.17), are the Euler-Lagrange equations.

Furthermore, the fact that J° is equal to ϕ on $\psi = 0$ implies that there exists a vector ν such that

$$\left.\frac{\partial J^\circ}{\partial x}\right|_{t_f} = \left(\frac{\partial \phi}{\partial x} + \nu^T\frac{\partial \psi}{\partial x}\right)_{t=t_f} \triangleq \lambda^T(t_f). \tag{4.2.22}$$

In words, the change in cost due to admissible change in state (that is, $d\psi = 0$) is given by a linear combination of the gradient of ϕ with respect to state and the gradients of ψ (constraints) with respect to state (see Section 1.2).

Combinatorial problems. Dynamic programming is especially useful in solving multistage optimization problems in which there are only a small number of possible choices of the control at each

†When there are inequality constraints on the control variable, it can be shown (e.g., by defining a modified Hamiltonian as in Chapter 3 or see Kalman (1963) or Dreyfus (1965)) that the term $(L_u + J^\circ_x f_u)u^\circ_x$ still vanishes.

stage and in which no derivative information is available. Consider a simple example with only two possible choices of control at each stage. We would like to find the path from A to B in Figure 4.2.1,

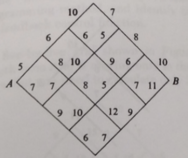


Figure 4.2.1. A minimum-time combinatorial problem. Numbers are times to travel legs of grid.

traveling only to the right (either up-right or down-right at each corner), such that the sum of the numbers on the segments along this path is a minimum. If we consider these numbers to be travel times, we are looking for the minimum-time path.

There are 20 possible routes from A to B, traveling only to the right. It would be tedious to try out all 20 routes. Instead of starting at A and trying different routes to B, we work *backward* from B to find the minimum-time routes to B from each of the 15 corners on the grid, as indicated on Figure 4.2.2.

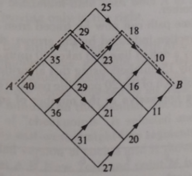


Figure 4.2.2. Dynamic programming solution to the problem of Figure 4.2.1.