Hamilton **J**acobi **B**ellman **D**ynamic **P**rogramming & RL notation to HJB natives

- Cost To Go
- **H**amilton **J**acobi **B**ellman equation
- **D**ynamic **P**rograming
- DP - RL notation switch

Cost To Go

- Consider a system that obeys $\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t), t)$ starting from $\mathbf{x}_0$ and ending when condition $\psi(\mathbf{x}, t_f) = 0$ is met.
- The cost of the system's path (and control signal) is

$$J(\mathbf{x}(t_0), \mathbf{u}, t_0) = \phi(\mathbf{x}(t_f), t_f) + \int_{t_0}^{t_f} L(\mathbf{x}, \mathbf{u}, \tau) d\tau$$

- The *Cost To Go* at time t along the path is

$$J(\mathbf{x}(t), \mathbf{u}, t) = \phi(\mathbf{x}(t_f), t_f) + \int_{t}^{t_f} L(\mathbf{x}, \mathbf{u}, \tau) d\tau$$

- Consider neighboring optimal paths (starting from different $\mathbf{x}_0$)

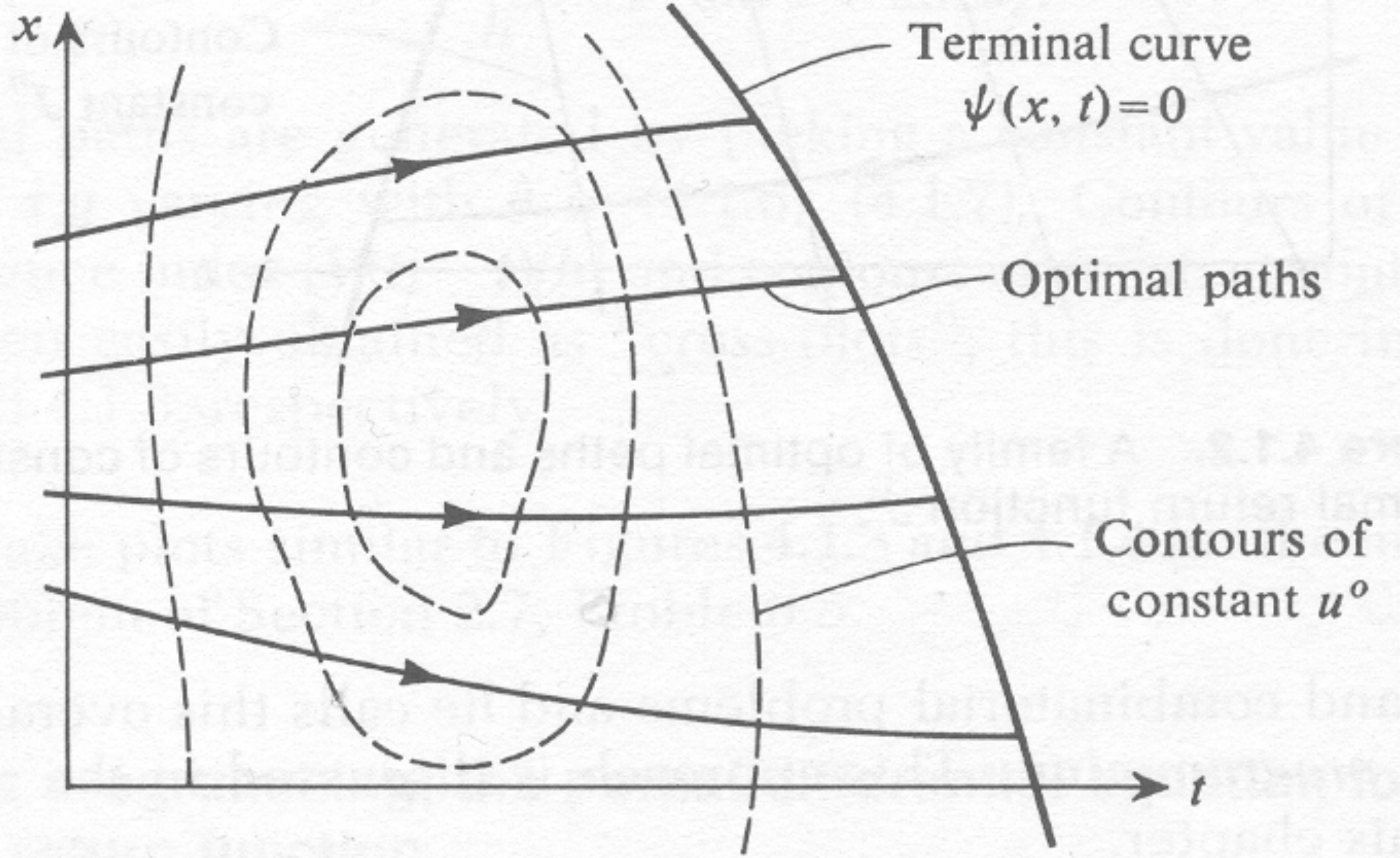
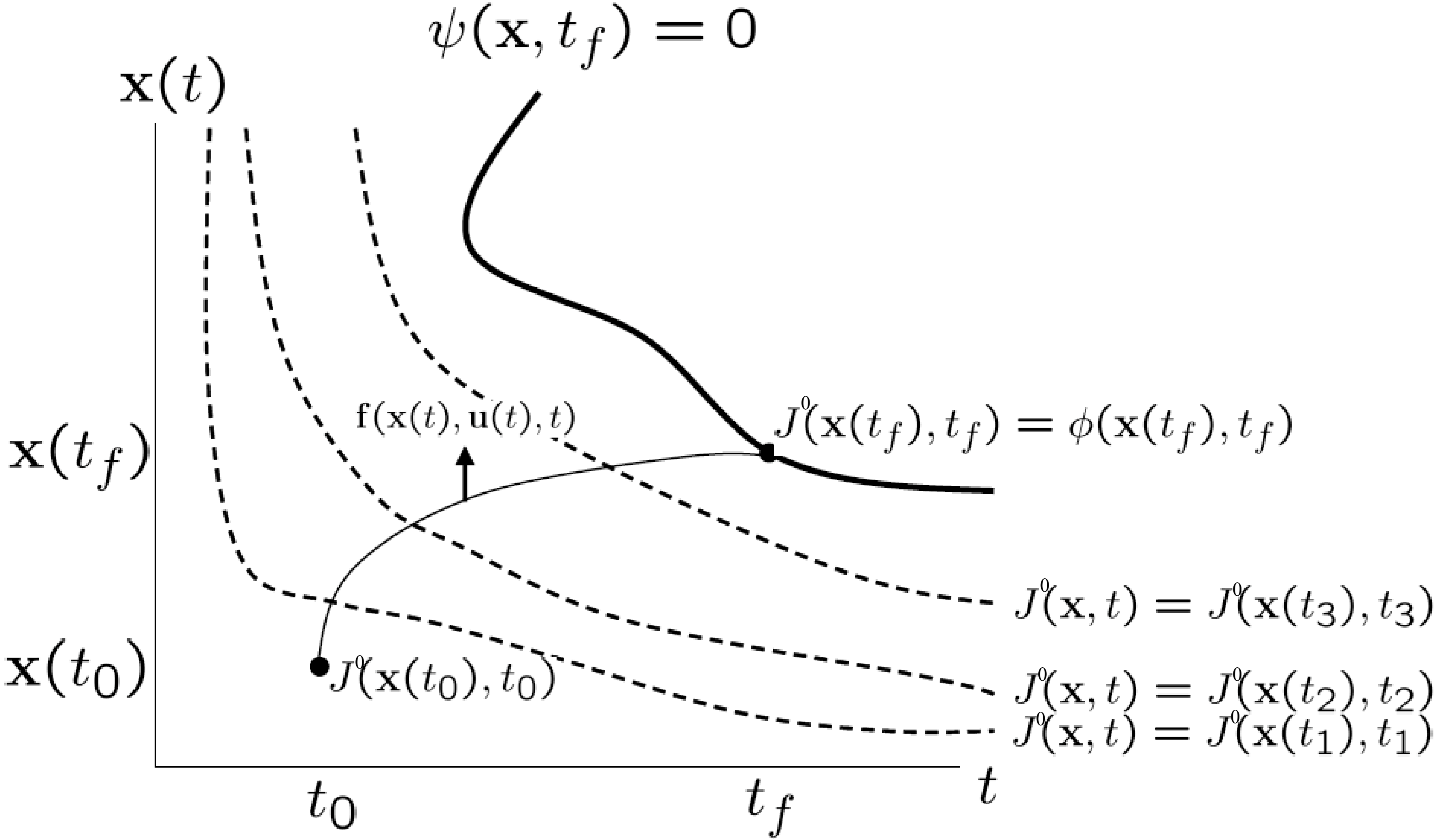


Figure 4.1.1. A family of optimal paths and contours of constant optimal control u^o.

Finding these paths may be important for knowing the optimal control when the system is perturbed.

- Define by J^0 the cost-to-go along an optimal path:

$$J^0(\mathbf{x}(t), t) = \min_{\mathbf{u}} \left\{ \phi(\mathbf{x}(t_f), t_f) + \int_t^{t_f} L(\mathbf{x}, \mathbf{u}, \tau) d\tau \right\}$$



Contours of optimal cost-to-go along an optimal path

- At any point along the path - the rest of the path is optimal.

- Minimum time ship path in linearly varying current - single trajectory

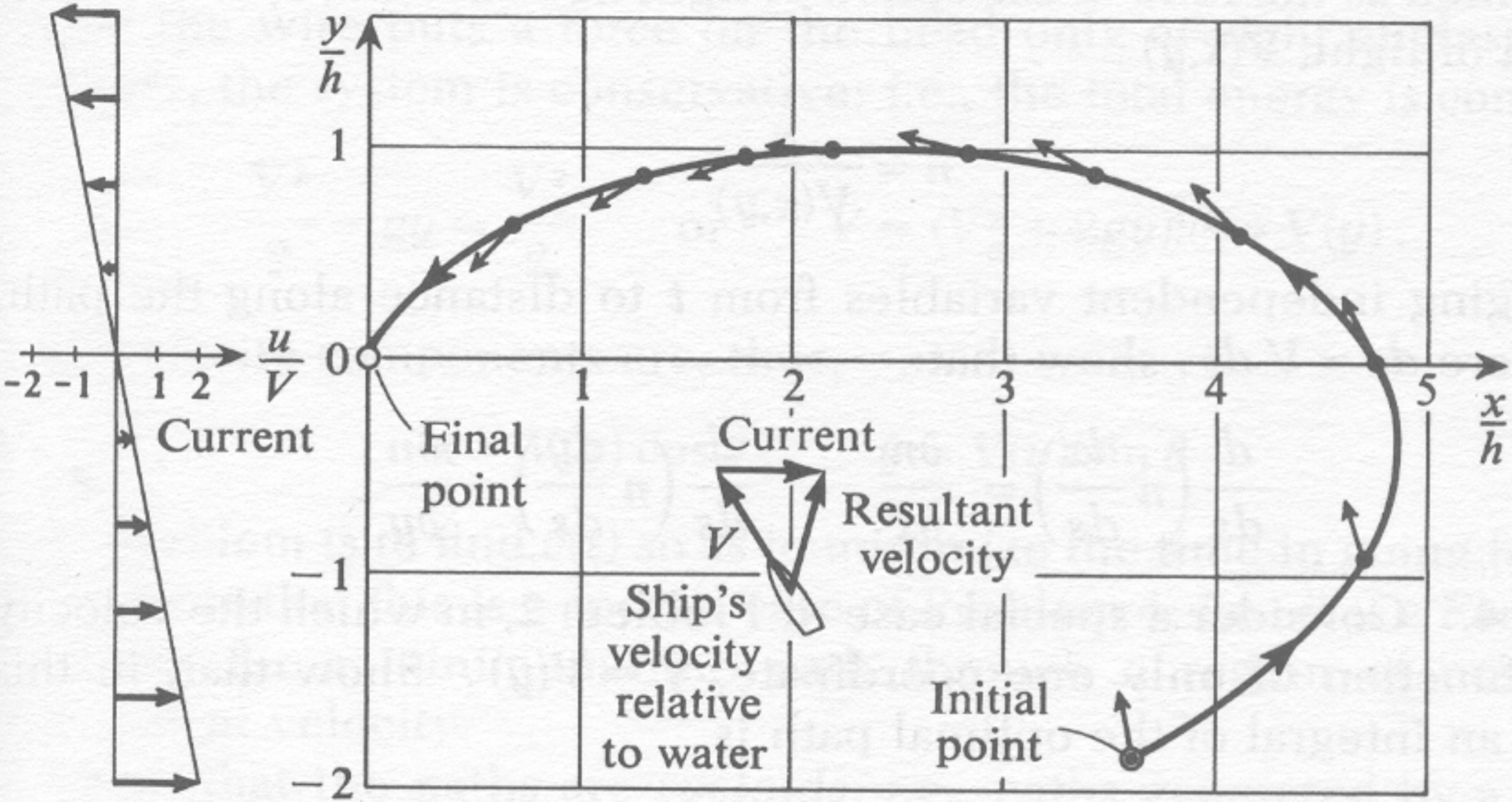


Figure 2.7.2. A minimum-time path through a region of linearly increasing current.

- Optimal paths and contours of constant **u**

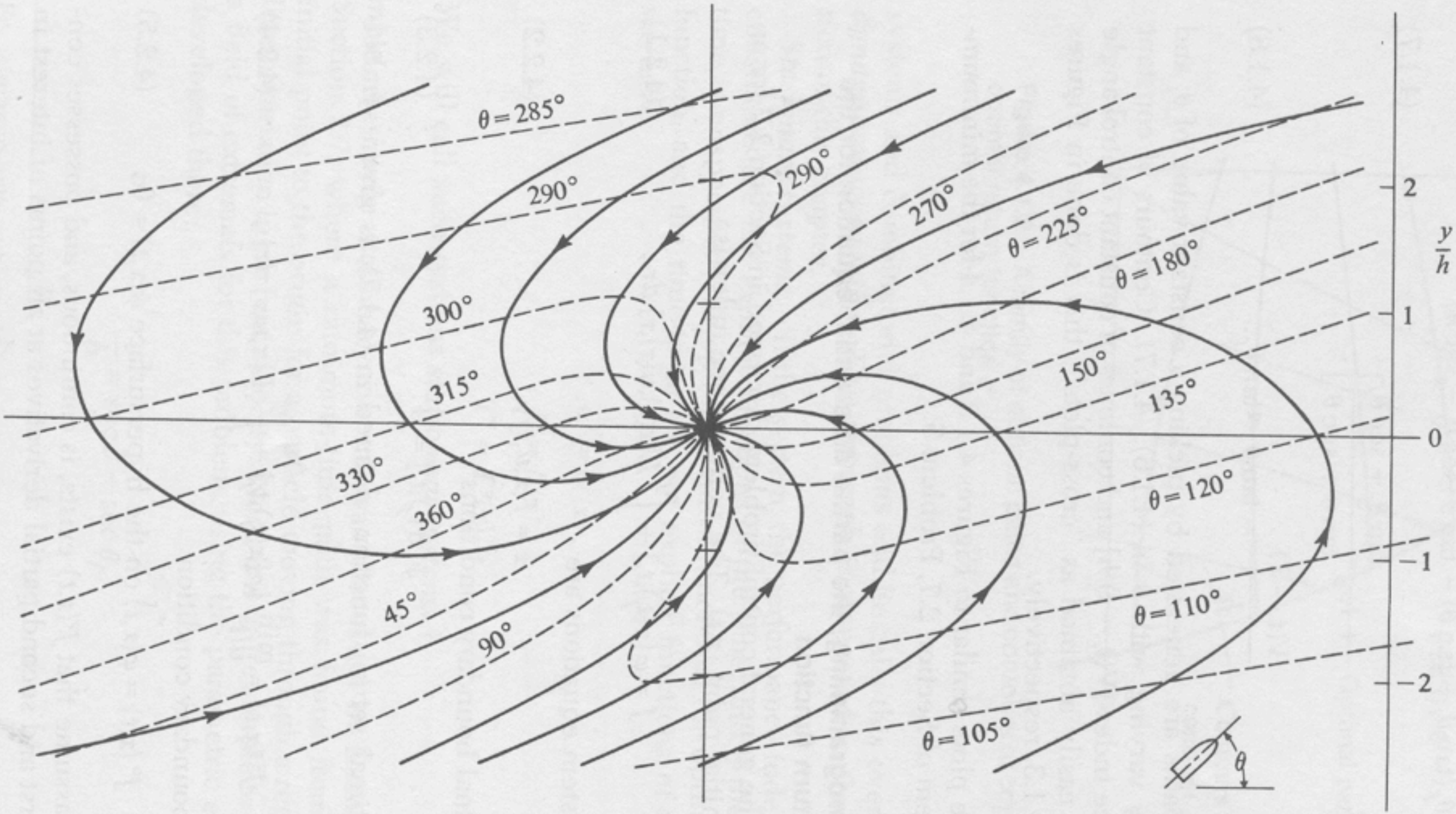


Figure 4.1.3. Minimum-time ship paths with linear variation in current and contours of *constant heading angle.*

- Optimal paths and contours of constant cost-to-go

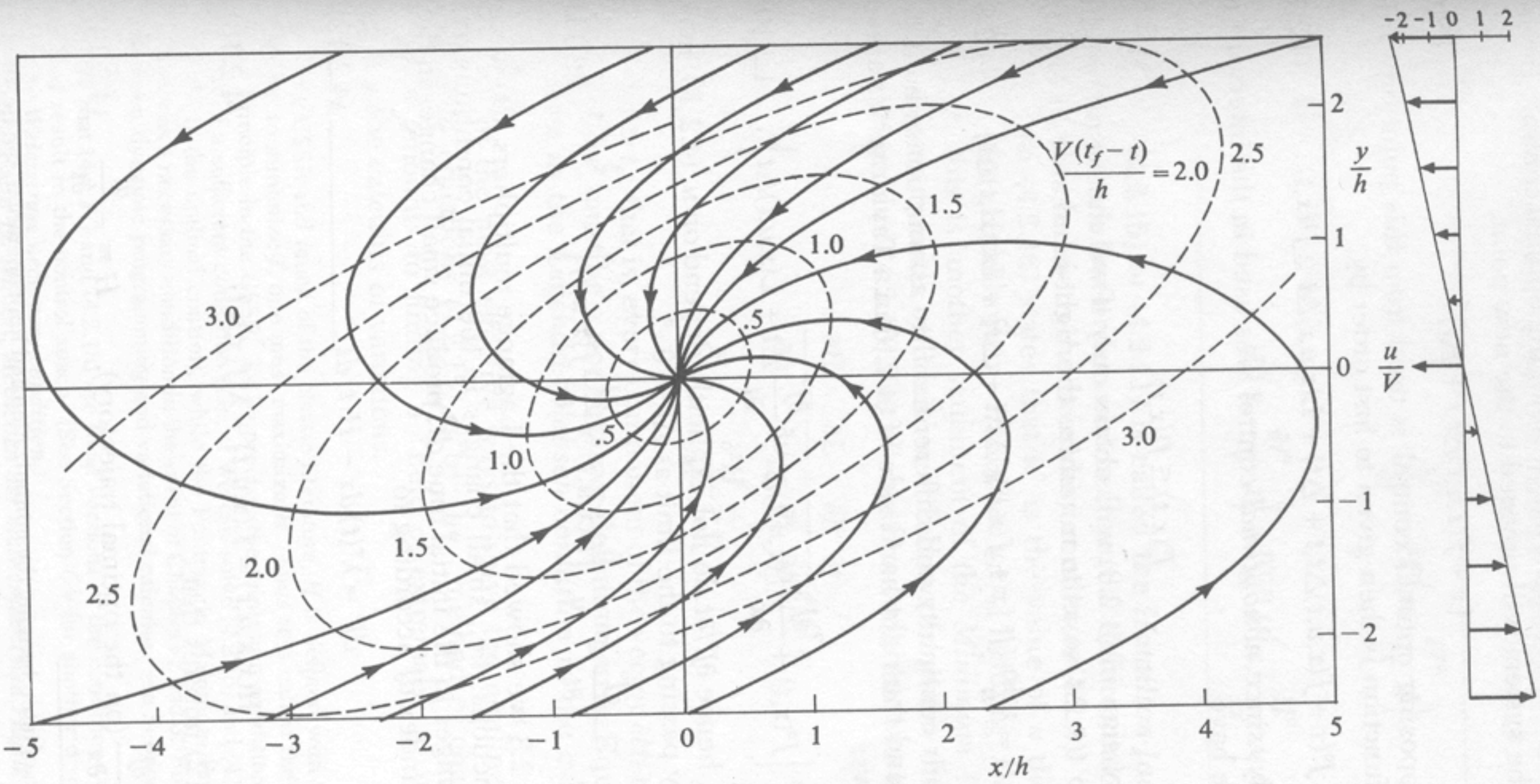


Figure 4.1.4. Minimum-time ship paths with linear variation in current and contours of *constant time-to-go*.

HJB equation derivation

- Suppose that at time t a possibly non-optimal control, $\mathbf{u}(t)$ is applied for a short time Δt and from then on optimal control is applied. Define the cost of this to be

$$\begin{aligned} J^1(\mathbf{x}, \mathbf{u}(t), t) &= \int_t^{t+\Delta t} L\left(\mathbf{x}(\tau), \mathbf{u}(\tau), \tau\right) d\tau + \underbrace{\min_{\mathbf{u}} \left\{ \phi\left(\mathbf{x}(t_f), t_f\right) + \int_{t+\Delta t}^{t_f} L\left(\mathbf{x}(\tau), \mathbf{u}(\tau), \tau\right) d\tau \right\}}_{J^0(\mathbf{x}(t+\Delta t), t+\Delta t)} \\ &= J^0(\mathbf{x}(t+\Delta t), t+\Delta t) + \int_t^{t+\Delta t} L\left(\mathbf{x}(\tau), \mathbf{u}(\tau), \tau\right) d\tau \end{aligned}$$

- Conversely, we can now define J^0 recursively

$$\begin{aligned} J^0(\mathbf{x}, t) &= \min_{\mathbf{u}(t)} J^1(\mathbf{x}, \mathbf{u}(t), t) \\ &= \min_{\mathbf{u}(t)} \left\{ J^0(\mathbf{x}(t+\Delta t), t+\Delta t) + \int_t^{t+\Delta t} L\left(\mathbf{x}(\tau), \mathbf{u}(\tau), \tau\right) d\tau \right\} \end{aligned}$$

$$J^0(\mathbf{x},t) = \min_{\mathbf{u}(t)} \left\{ J^0(\mathbf{x}(t+\Delta t), t+\Delta t) + \int_t^{t+\Delta t} L\left(\mathbf{x}(\tau), \mathbf{u}(\tau), \tau\right) d\tau \right\}$$

- Under most conditions, for small Δt we can make the following (1st degree) approximation:

$$\int_t^{t+\Delta t} L\left(\mathbf{x}(\tau), \mathbf{u}(\tau), \tau\right) d\tau = L\left(\mathbf{x}(\tau), \mathbf{u}(\tau), \tau\right) \Delta t$$

- Plugging them into the recursive definition of J^0 we get

$$J^0(\mathbf{x},t) \quad \approx \quad \min_{\mathbf{u}(t)} \left\{ \underbrace{J^0\left(\mathbf{x}(t+\Delta t), t+\Delta t\right)}_{(\star)} + L(\mathbf{x}, \mathbf{u}(t), t)\Delta t \right\}$$

- Next we take a first order approximation of (*)

$$J^0\left(\mathbf{x},t\right) \quad = \quad \min_{\mathbf{u}(t)} \left\{ J^0\left(\mathbf{x},t\right) + \frac{dJ^0}{dt}\Delta t + L(\mathbf{x}, \mathbf{u}(t), t)\Delta t \right\}$$

$$J^0(\mathbf{x},t) = \min_{\mathbf{u}(t)} \left\{ J^0(\mathbf{x},t) + \frac{dJ^0}{dt}\Delta t + L(\mathbf{x},\mathbf{u}(t),t)\Delta t \right\}$$

- Rewriting as sum of partial derivatives:

$$= J^0(\mathbf{x},t) + \min_{\mathbf{u}(t)} \left\{ \frac{\partial J^0}{\partial \mathbf{x}}\frac{\partial \mathbf{x}}{\partial t}\Delta t + \frac{\partial J^0}{\partial t}\Delta t + L(\mathbf{x},\mathbf{u}(t),t)\Delta t \right\}$$

$$= J^0(\mathbf{x},t) + \frac{\partial J^0}{\partial t}\Delta t + \min_{\mathbf{u}(t)} \left\{ \frac{\partial J^0}{\partial \mathbf{x}}\mathbf{f}(\mathbf{x},\mathbf{u}(t),t)\Delta t + L(\mathbf{x},\mathbf{u}(t),t)\Delta t \right\}$$

- Finally, omit J0(x,t) from both sides and get

$$-\frac{\partial J^0}{\partial t} = \min_{\mathbf{u}(t)} \left\{ L(\mathbf{x},\mathbf{u}(t),t) + \frac{\partial J^0}{\partial \mathbf{x}}\mathbf{f}(\mathbf{x},\mathbf{u}(t),t) \right\}$$

with border conditions at end

$$J^0(\mathbf{x},t) = \phi(\mathbf{x},t)$$

$$\psi(\mathbf{x},t) = 0$$

known as the HJB equations

$$-\frac{\partial J^0}{\partial t} = \min_{\mathbf{u}(t)} \left\{ L(\mathbf{x}, \mathbf{u}(t), t) + \frac{\partial J^0}{\partial \mathbf{x}} \mathbf{f}(\mathbf{x}, \mathbf{u}(t), t) \right\}$$

- This is a first order non-linear partial differential equation.
- In words: change in cost along small part of optimal path is the sum of the current loss and the change in loss due to the change in state value.
- Not easily solved in general.

Discrete HJB = Dynamic Programming

- Suppose that time, states and control signal are all discrete:

$$x_{t+1} = f\left(x_t, u_t, t\right)$$

$$x \in \mathcal{X} = \{1, \ldots, |\mathcal{X}|\} \qquad u \in \mathcal{U} = \{1, \ldots, |\mathcal{U}|\}$$

- Cost-to-go on optimal path

$$V^0(x_k, k) = \min_{\mathbf{u}} \left\{ \phi(x_N, N) + \sum_{t=k}^{N-1} L(x_t, u_t, t) \right\}$$

- Recursively

$$V^0(x_k, k) = \min_{u_k} \left\{ L(x_k, u_k, k) + V^0\left(f(x_k, u_k, k), k+1\right) \right\}$$

- Suppose we know f(x,u,t) ,x_1, x_N and N (number of time steps). We want to find the optimal control and the cost for getting from x_1 to x_N. We can do so with *dynamic programming* as follows.

- Define $\mathbf{V} \in R^{|X| \times N}$ such that

$$\mathbf{V}_{ij} = V^0(x_j = i, j)$$

  is the cost-to-go from state *i* at time *j*.

- Filling this matrix is done recursively

$$\begin{aligned} \mathbf{V}_{iN} &= \begin{cases} \phi(x_N, N) & i = x_N \\ \infty & else \end{cases} \\ \mathbf{V}_{i,k} &= \min_{u_k} \left\{ L(x_i, u_k, k) + \mathbf{V}_{f(x_i, u_k, k), k+1} \right\} \end{aligned}$$

- Easy to see that when finished $V_{i,k}$ is the cost-to-go at time *k* from state *i*

- If $V_{i,k} = \infty$ then the is no path of length N-k+1 from state *i* to state x_N.

- Cost of filling the matrix is O(N |X| |U|)

- To remember the optimal control signal, fill

$$\mathbf{U}_{i,k} = \arg\min_{u} \left\{ L(x_i, u_k, k) + \mathbf{V}_{f(x_i,u_k,k),k+1} \right\}$$

- Optimal path can be remembered as $\mathbf{X}_{i,k} = \mathrm{f}(x_i, \mathbf{U}_{i,k}, k)$ or reconstructed from **U** in a forward pass (hence known as a forward backward method).

- If, instead of initial and end states we know conditions that they must obey

$$\psi(i, N) = 0$$

$$\chi(i, 1) = 0$$

we simply change the initialization of **V** to

$$\mathbf{V}_{i,N} = \begin{cases} \phi(i, N) & \psi(i, N) = 0 \\ \infty & else \end{cases}$$

and select the initial state to be

$$x_1 = \min_{i} \mathbf{V}_{i,1} \quad s.t.\ \chi(i, 1) = 0$$

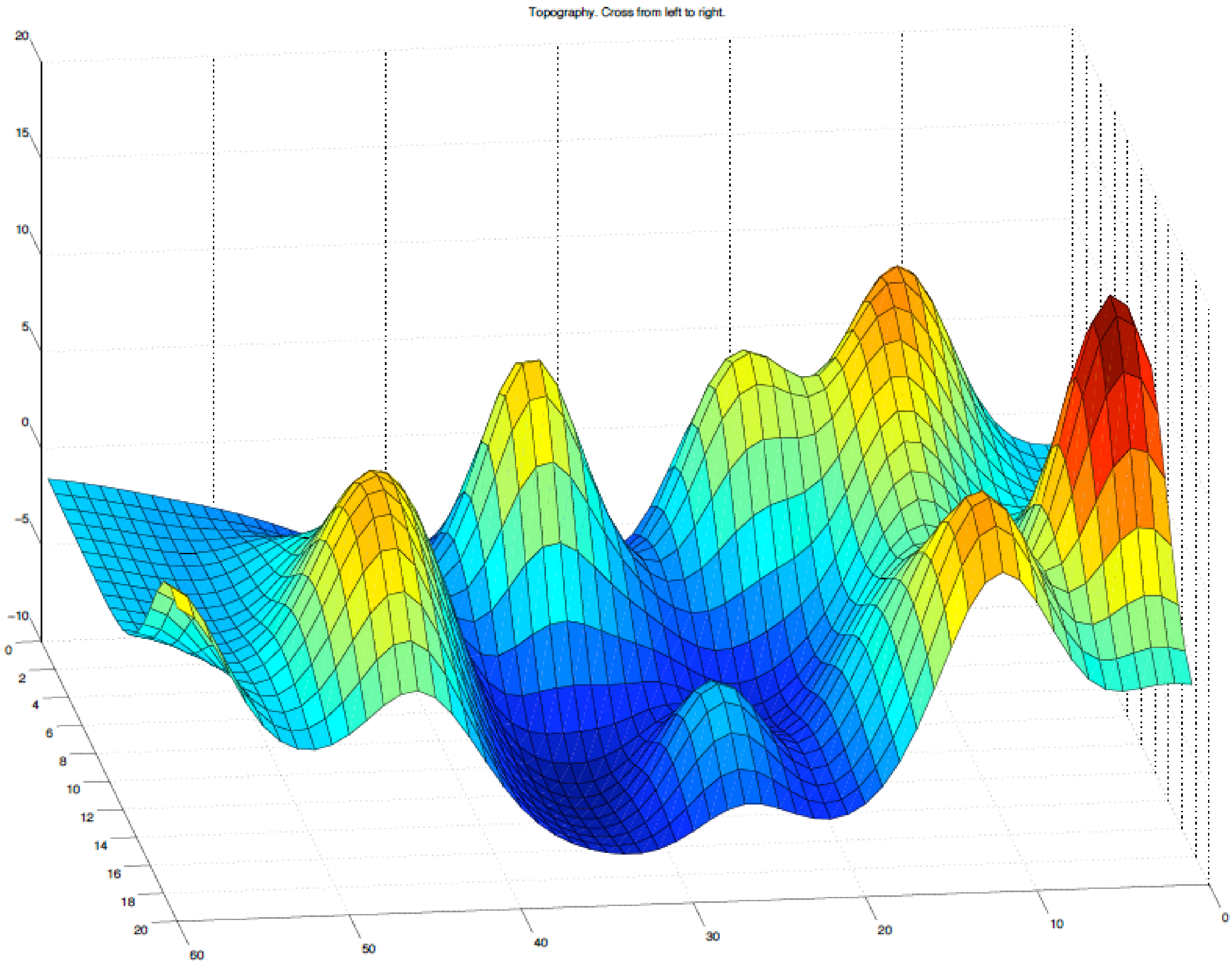
- If the run time (N) is not known, the depth of the recursion (width of **V**) depends on L(x,u,t). For any N, if a path of length N exists (such that initial and final conditions are met) then there is an L that would make the optimal path to be of length N.
- The initial state and time are chosen as

$$x_{init} = \min_{i,j} \mathbf{V}_{i,j} \quad s.t.\ \chi(i,j) = 0$$

- DP of length N ensures optimal paths of length $\leq$ N
- If L > 0 and is not a function of time (just *x*) then N $\leq$ |X|
- You can only do it if you know *f* and L.
- Complexity can be exponential in the number of state dimensions (but linear in the number of states).
    - Still much cheaper that considering all possible controls on length N (of which there are $|U|^N$).

DP example - Bicycle Navigation

- Given a topographical map as a matrix **M** (**M**$_{ij}$ is the height at i,j)



- At each time step you can move one square (8 options, inc. diagonals).
- Find a path that start at left had side of map and ends at the right hand side.
- The loss you pay at each step is

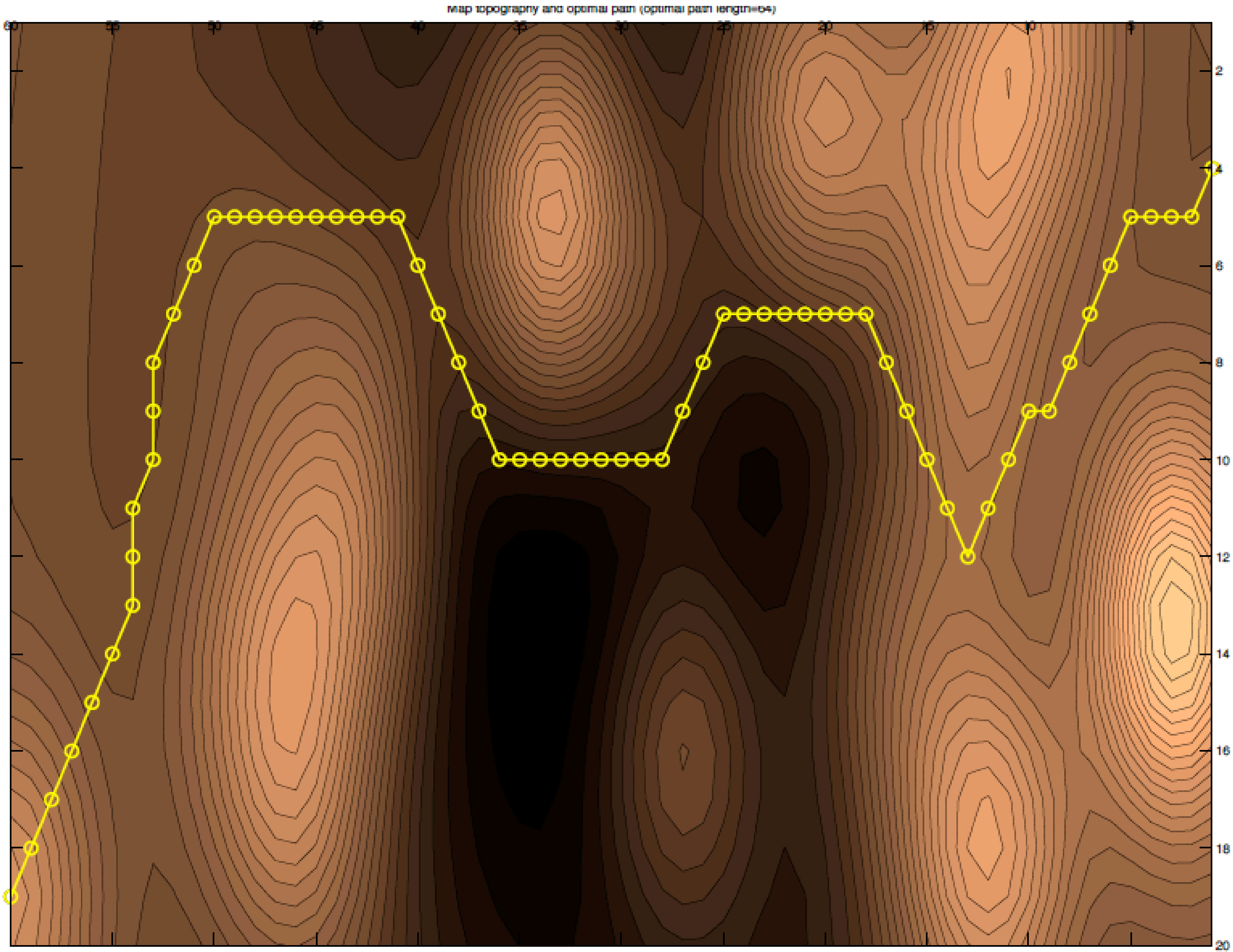
$$L = 0.1 * \textit{step-size} + \textit{height-diff} * (1 + 0.5\ \mathrm{sign}(\textit{height-diff}))$$

where *step-size* is 1 or sqrt(2), *height-diff* is the difference in heigh (values of **M**).

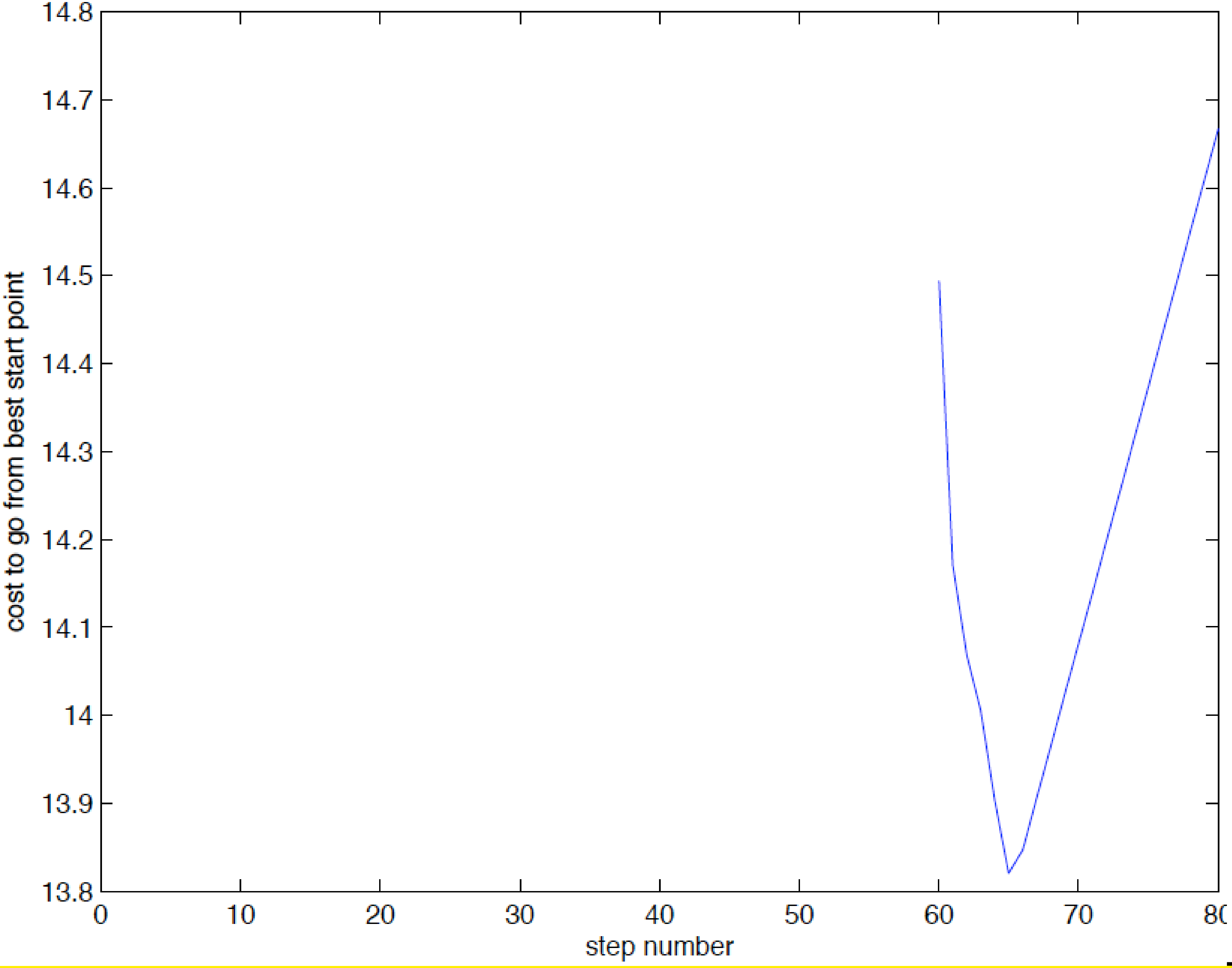
  - Going straight costs you 0.1s*tep-size*
  - Going up - pay 1.5 height difference
  - Going down - you gain 0.5 height difference (Loss can be negative)

- We define the state to be from 1 to the size of **M** (width x height).
- The end condition is ψ(i,N) = 0 if *i* is a position on the rhs of the map
- The initial condition χ(i,1) = 0 if i is on the lhs of the map

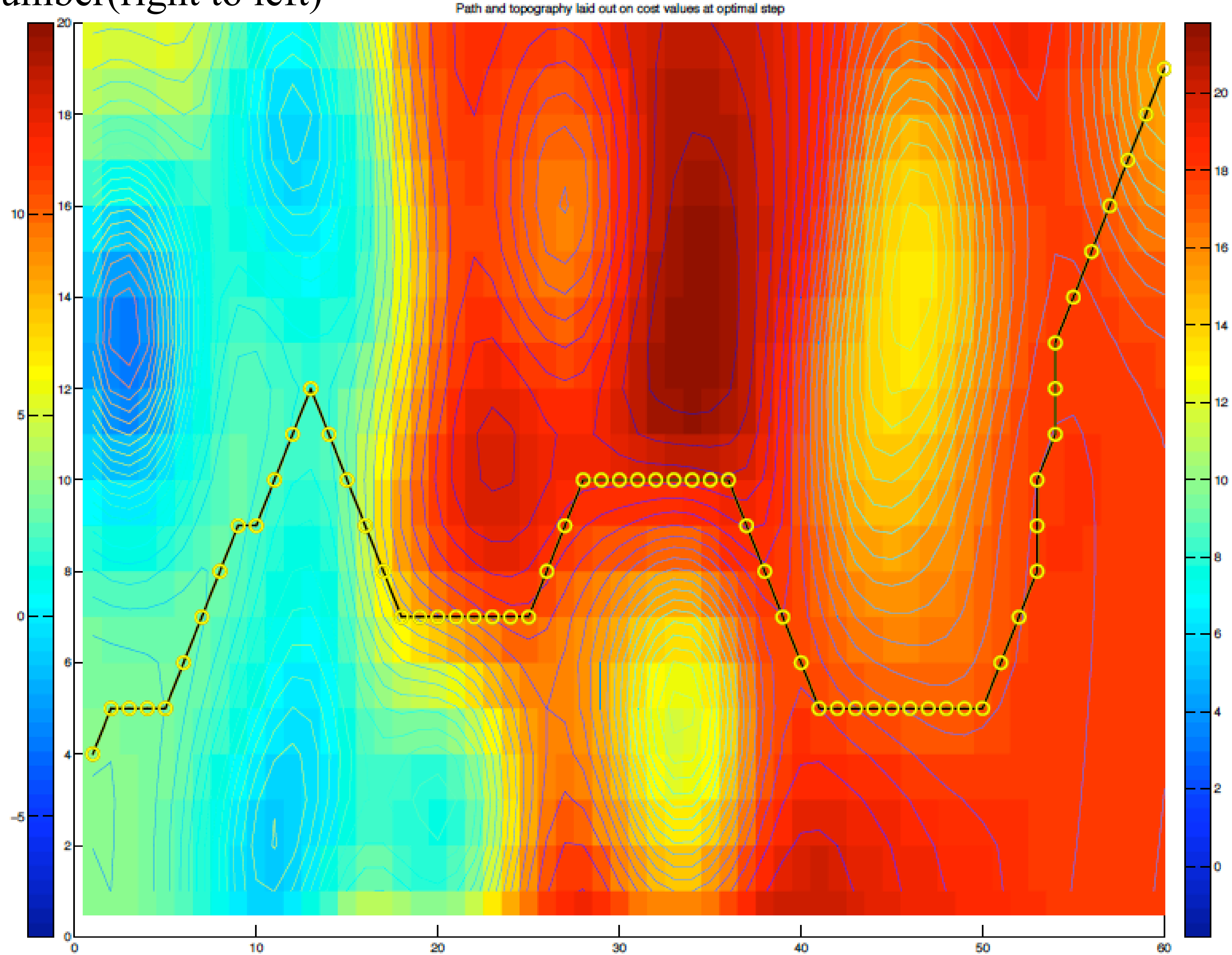
The topography an the optimal path (left to right)



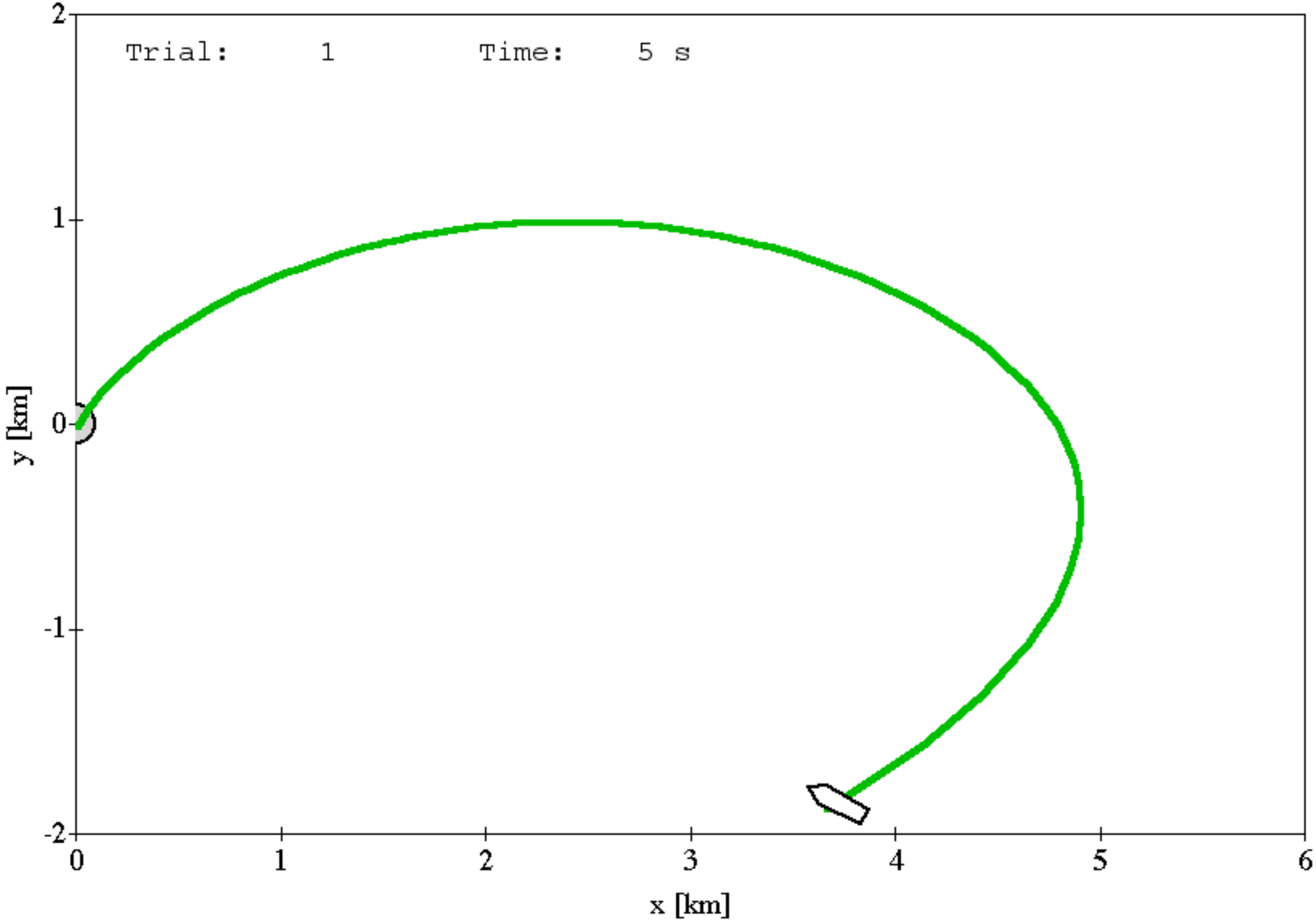
The cost as a function of the number of steps



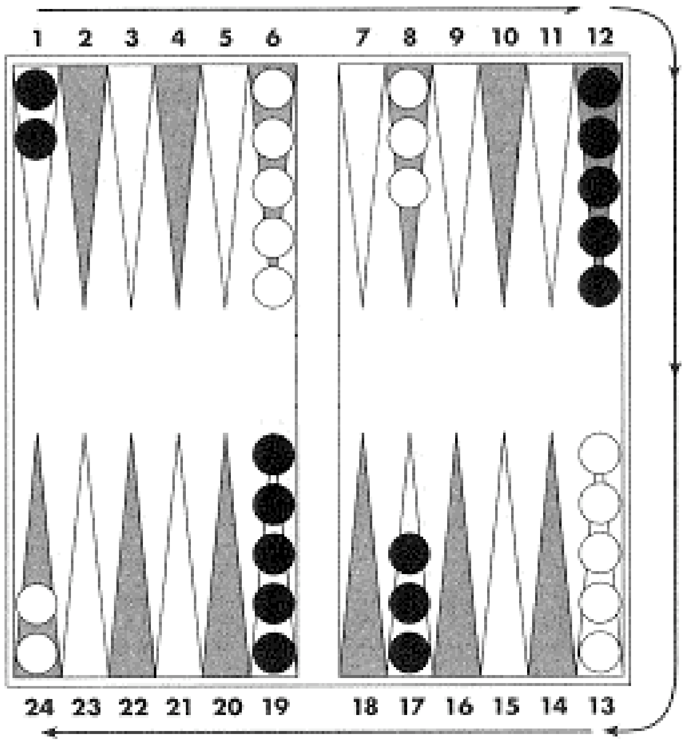
topography as contours and optimal path, laid on cost at optimal step number(right to left)



- Greedy partial DP



RL Motivation



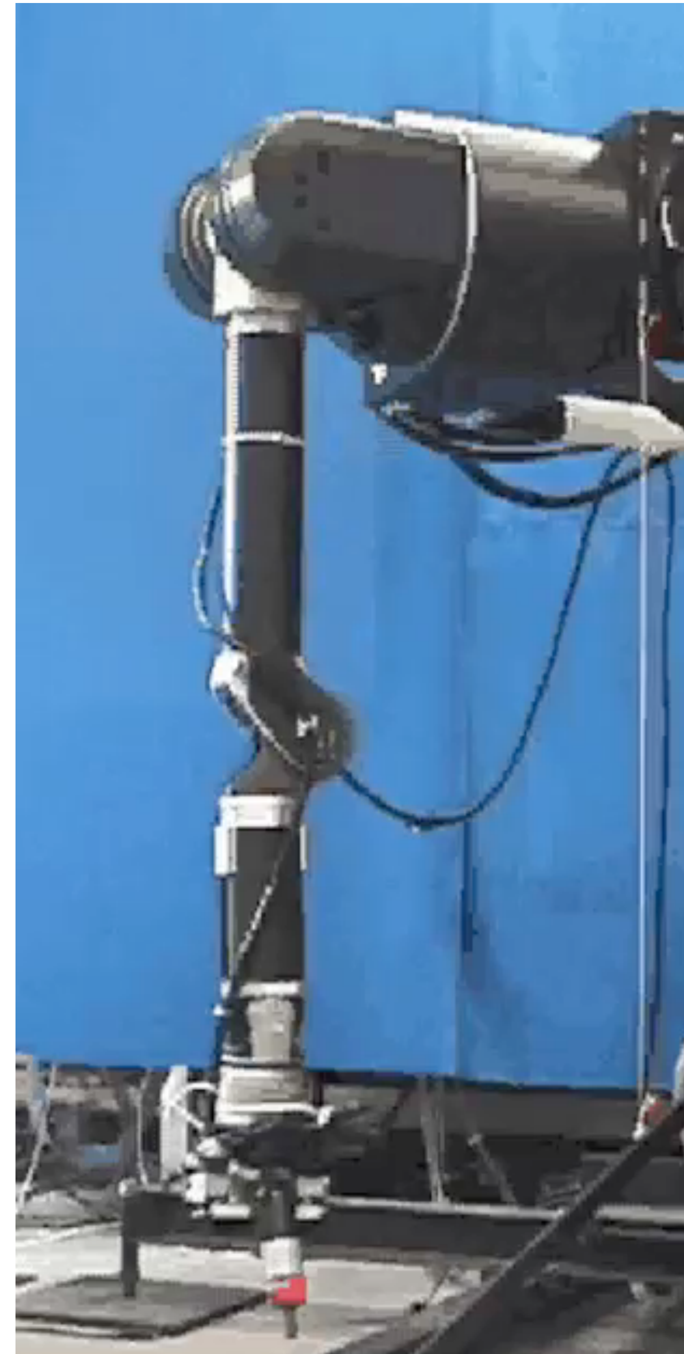
- TD-Gammon
- Created in early 90's
- Did not have any prior knowledge other than the rules of the game
- Used RL (and neural network design) to become world champion

Program	Hidden Units	Training Games	Opponents	Results
TD-Gam 0.0	40	300,000	Other Programs	Tied for Best
TD-Gam 1.0	80	300,000	Robertie, Magriel, ...	−13 pts / 51 games
TD-Gam 2.0	40	800,000	Var. Grandmasters	−7 pts / 38 games
TD-Gam 2.1	80	1,500,000	Robertie	−1 pts / 40 games
TD-Gam 3.0	80	1,500,000	Kazaros	+6 pts / 20 games

- Heli-control
- High dimensional
- Continuous
- Expensive (uses robust control)

- Air Hockey
  ATR, Japan

- Robot arm movement optimization



DP & RL

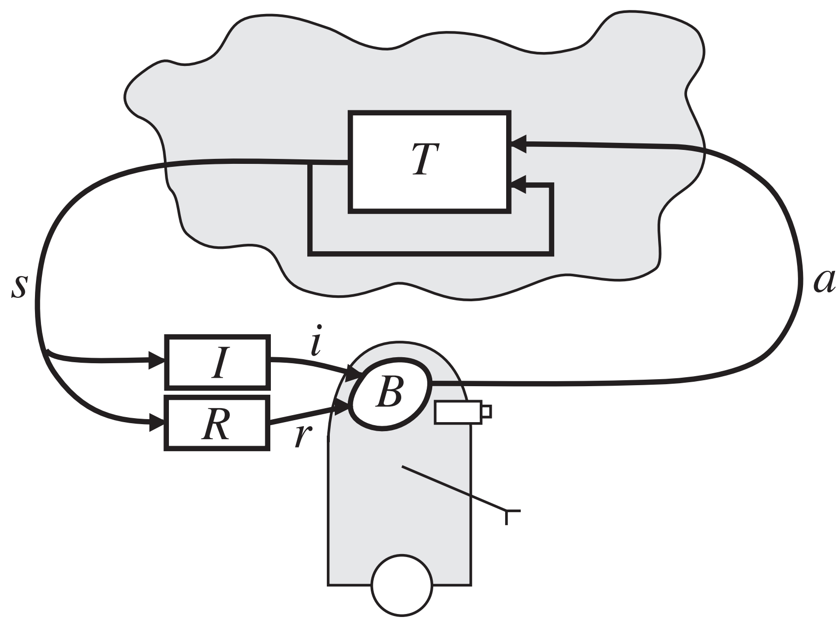
- **R**einforcement **L**earning (RL) is a set of algorithms for learning optimal control.
- Classic RL is for discrete worlds (state, action, time step).
- While in DP - you need to know L and *f* (loss function and dynamics), RL lets you learn them.
- While in DP, we update the value function for every state at each step (at a cost of O(|X| |U|), in RL we usually do partial updates (at lower cost).
- The cost is usually not over a finite number of step but over infinite (discounted) steps.

Control ↔ RL notation change

RL		control	
name	notation	notation	name
state	$s \in S$	$\mathbf{x}$	state
action	$a \in A(s)$	$\mathbf{u}$	control sig.
(stochastic) policy	$\pi(s,a)=\Pr(a\|s)$	$\mathbf{u}(\mathbf{x},t)$	control function
(stochastic) state transition	$P^a_{ss'}$	$\mathrm{f}(\mathbf{x},\mathbf{u},t,\omega)$	state dynamics
(stochastic) reward	r	$-L$	instantaneous loss
cumulative expected reward	V	$-J$, $-V$	cost-to-go

$$P^a_{ss'} = Pr\left\{s_{t+1} = s' \middle| s_t = s, a_t = a\right\}$$

Perception - Action loop



Agent performs action → environment changes → new state is seen by agent (as sensory input *i*) and a reward *r* is generated. The agent's behavior (*B* / policy) might change → next action is performed..

Reward model

- The agent is (usually) concerned with maximizing the cumulative discounted reward:

$$
\begin{aligned}
R_t &= r_{t+1} + \gamma r_{t+2} + \gamma^3 r_{t+1} + \ldots \\
&= r_{t+1} + \gamma R_{t+1}
\end{aligned}
$$

  where $\gamma < 1$. This is a stochastic variable, dependent on policy, reward and state transition distributions

- Suppose that at time *t* the state is *s*, action *a* is performed, the new state is *s'* and reward r_{t+1} is given. The expected reward $E\{r_{t+1}\}$ is denoted

$$
R^a_{ss'} = E_{r_{t+1}} \{ r_{t+1} | s_t = s \; , \; a_t = a \; , \; s_{t+1} = s' \}
$$

- The expected R_t is the expected discounted sum of future rewards. R_t at state *s* under a policy π is called the *value* of *s* (under π)

$$
V^\pi(s) = E_\pi \{ R_t | s_t = s \}
$$

$$
\begin{aligned}
V^{\pi}(s) &= E_{\pi}\left\{R_t | s_t = s\right\} \\
&= E_{\pi}\left\{r_{t+1} + \gamma V\left(s_{t+1}\right) | s_t = s\right\} \\
&= \sum_{a} \pi(s,a) \sum_{s'} P^{a}_{ss'}\left[R^{a}_{ss'} + \gamma V^{\pi}(s')\right]
\end{aligned}
$$

- $V^{\pi}(s)$ is equivalent to $J(\mathbf{x},t)$ in control notation.
- Note that in classic RL reward, state transition probability and policy (i.e. all the distributions) are assumed to be time independent, which is why we denote $V(s)$, not $V(s,t)$.
- As is control, RL learns a policy that maximizes $V(s)$.
- The value of s under optimal control, (previously $J^{0}(\mathbf{x},t)$) is denoted

$$
V^{*}(s) = \max_{\pi} V^{\pi}(s)
$$

- Because the value is time independent a DP solution looks for a "vector" not a "matrix" (the t axis is gone).
- The optimal policy is denoted by $\pi^{*}(s)$.

Q function

- Recall that in control notation $J^1(\mathbf{x},\mathbf{u},t)$ is the cost of performing possibly non-optimal **u** and then performing optimally. In RL we define the Q function to mean the same thing

$$
\begin{aligned}
Q^*(s,a) &= E\left\{r_{t+1} + \gamma V^*(s_{t+1})\right\} \\
&= E\left\{r_{t+1} + \gamma \max_{a'} Q^*(s',a')\right\} \\
&= \sum_{s'} P^a_{ss'}\left[R^a_{ss'} + \gamma \max_{a'} Q^*(s',a')\right]
\end{aligned}
$$

(a.k.a the bellman optimality condition)

- The optimal policy is easy to derive from Q:

$$\pi^*(s) = \arg\max_{a \in A(s)} Q^*(s,a)$$

- We will see later why Q is sometimes necessary (rather than V). E.g. if dynamics function ($P^a_{ss'}$) is not known.