Reinforcement Learning	March 6, 2019
Recitation 2	
Lecturer:Yishay Mansour	*TA: Lee Cohen*

Part 1: Deterministic Dynamic Programming Algorithms[1]

Question 1: Maximum contiguous sum

Given a (long) sequence of n real numbers $a_1, a_2, ..., a_n$ (both positive and negative). Find

$$V^* = \max_{1 \leqslant i \leqslant j \leqslant n} \sum_{l=i}^{j} a_l$$

Solution 1:

1. Naive approach - exhaustive search, needs to examine $O(n^2)$ sums.
2. DP Solution - the DP solution has linear time. Let

$$M_k = \max_{1 \leqslant i \leqslant k} \sum_{l=i}^{k} a_l$$

denote the maximal sum over all contiguous subsequences that end exactly at a_k. Then $M_1 = a_1$ and $M_k = max\{M_{k-1} + a_k, a_k\}$. We may compute M_k recursively for $k = 2 : n$.
The required solution is given by $V^* = max\{M_1, M_2, ..., M_n\}$

Question 2: An integer knapsack problem

Given a knapsack (bag) of capacity $C > 0$, and a set of n items with respective sizes $S_1, S_2, ..., S_n$ and values (worth) $V_1, V_2, ..., V_n$.
The sizes are positive and integer-valued. Fill the knapsack to maximize the total value. That is, find the subset $A \subset \{1, 2, ..., n\}$ of items that maximize

$$\Sigma_{i \in A} V_i,$$

[1]Taken from "046194 - Learning and Planning in Dynamical Systems" by Shie Manor©

subject to $\Sigma_{i\in A} S_i < C$

Note that the number of item subsets is 2^n. Moreover this problem is NP-complete for "large C", namely, when the integers are n bits long.

Solution 2:

Let $M(i,k)$ be maximal value for filling exactly capacity k with items from the set $1:i$.
If the capacity k cannot be matched by any such subset, set $M(i,k) = -\infty$. Also set $M(0,0) = 0$ and, $M(0,k) = -\infty$ for $k \geqslant 1$.
Then $M(i,k) = max\{M(i-1,k), M(i-1,k-S_i) + V_i\}$ which can be computed recursively for $i = 1:n, k = 1:C$. The required value is obtained by $M^* = \max_{0\leqslant k\leqslant C} M(n,k)$.
The running time of this algorithm is $O(nC)$.
We note that the recursive computation of $M(n,k)$ requires $O(C)$ space. To obtain the indices of the terms in the optimal subset some additional book-keeping is needed, which requires $O(nC)$ space.

Question 3: Longest Common Subsequence

Given Two sequences (or strings) $X(1:m), Y(1:n)$. Find a subsequence of X is the string that remains after deleting some number (zero or more) of elements of X.
We wish to find the longest common subsequence (LCS) of X and Y, namely, a sequence of maximal length that is a subsequence of both X and Y.
For Example:

$$X = \underline{A}V\underline{B}V\underline{A}M\underline{CD}$$
$$Y = \underline{A}Z\underline{B}Q\underline{AC}L\underline{D}$$

Solution 3:

Let $c(i,j)$ denote the length of an LCS of the prefix subsequences $X(1:i), Y(1:j)$. Set $c(i,j) = 0$ if $i = 0$ or $j = 0$.
Then, for $i, j > 0$,

$$c(i,j) = \begin{cases} c(i-1,j-1)+1 & x(i) = y(j) \\ max\{c(i,j-1), c(i-1,j)\} & x(i) \neq y(j) \end{cases}$$

We can now compute $c(i,j)$ recursively, using a row-first or column-first order. Computing $c(m,n)$ requires $O(mn)$ steps.

Part 2: Finding Minimum Mean Cycle[2]

Let $G(V, E)$ be a directed strongly connected graph with cost function $c : E \to \mathbb{R}$. Let s be an arbitrary start vertex of G. For each vertex $v \in V$, let $d_k(v)$ be the minimum cost of any path of length exactly k from s to v, or ∞ if no such path exists. Let μ^* be the *min average cost cycle*. Karp proves the following:

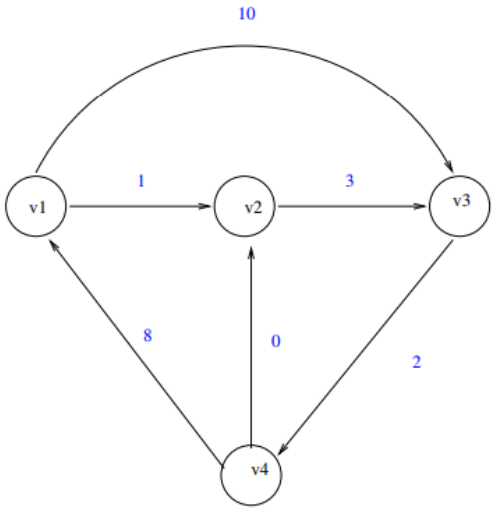
Theorem 1

$$\mu^* = \min_{v \in V} \max_{0 \le k \le n-1} \frac{d_n(v) - d_k(v)}{n - k},$$

where we define $\infty - \infty$ *as* ∞.

Now, $d_k(v)$ can be found in $O(|V| \cdot |E|)$ time for all $v \in V$ and all $k \in \{0, 1 \ldots, n\}$ using a dynamic programming algorithm that assigns $d_k(v)$ to an entry of the table indexed by (k, v). This gives an $O(|V| \cdot |E|)$ algorithm for finding the value of the minimum cycle mean, by Theorem 1. For each $v \in V$ and $\{k | 0 < k \le n\}$, the dynamic programming algorithm can assign a backpointer from (k, v) to an entry $(k - 1, w)$, giving the predecessor w on a path of length k and weight $d_k(v)$ from s to v. This allows a minimum weight path of length k from s to v to be reconstructed, by following backpointers.

Example 1 [3] *Minimum mean cost cycle: Suppose we wish to find the minimum mean cost cycle from the initial node* $s = v_1$.



First We initialize d_0 *to* ∞ *for all vertices other than* v_1. *We can compute* $d_k(v)$ *for all*

[2] Taken from "A Note On Finding Minimum Mean Cycle", Mmamu Chaturvedi and Ross M. McConnell, IPL 2017

[3] Optimization II by Cliff Stein, Spring 2016, Columbia university

vertices v and all $k \leq n$ in by iterating the recurrence:

$$d_k(w) = \min_{(v,w)\in E} d_{k-1}(v) + c(v,w)$$

Solution:

	d_0	d_1	d_2	d_3	d_4	$\max(d_n(v) - d_k(v))/(n-k)$
v_1	0	∞	∞	20	14	7/2
v_2	∞	1	∞	12	6	5/3
v_3	∞	10	4	∞	15	11/2
v_4	∞	∞	12	6	∞	∞

Where $\mu^* = \frac{5}{3}$ as this is the minimal value for all $v \in V$.

Finding a cycle of minimum mean

Let a minimizer be a vertex v such that $\mu^* = \max_{0\leq k\leq n-1} \frac{d_n(v)-d_k(v)}{n-k}$, and let a minimizing pair be a minimizer v and integer k such that $0 \leq k \leq n-1$ and k maximizes $\frac{d_n(v)-d_k(v)}{n-k}$.
Karp suggests the following for finding a cycle of minimum mean weight: "If the actual cycle yielding the minimum cycle mean is desired, it can be computed by selecting the minimizing [pair] v and k, finding a minimum-weight edge progression (path) of length n from s to v, and extracting a cycle of length $n-k$ occurring within that edge progression".
For the above example, the cycle is (v_2, v_3, v_4), which is indeed the cycle with a minimum mean cost. Karp does not supply a proof that this procedure is correct, and next we show a counterexample.

Counterexample to Karp's Algorithm

In the figure, the minimum cycle mean is 1, which is the mean weight of the cycles (s, a, b) and (a, c, d, e, f). The value of $d_n(g) = d_8(g)$ is 9, given by the path $(s, a, c, d, e, f, a, c, g)$, and the value of $d_6(g)$ is 7, given by (s, a, b, s, a, c, g). Since $[d_8(g) - d_6(g)]/(8-6) = 1$, which is the minimum cycle mean, g and 6 are a minimizing pair. There is supposed to be a cycle of length $8-6=2$ on the path $(s, a, c, d, e, f, a, c, g)$, but there is no cycle of length 2 in the graph.

Karp's proof of Theorem 1 shows that for some minimizing pair v and k, and some path W of weight $d_n(v)$ from s to v, the last $n-k$ edges of W are a cycle of minimum mean weight. In Figure 1, d and 3 are such a pair; the last $n-3=5$ edges of $(s, a, c, d, e, f, a, c, d)$ are a cycle of minimum mean weight. For v, the proof uses a vertex that lies on a cycle of minimum mean weight, and shows that there exists a path from s to v of length n and weight $d_n(v)$ such that the last $n-k$ edges of the path are a cycle of minimum mean weight. However, these conditions do not apply for all minimizing pairs. The example of g in Figure

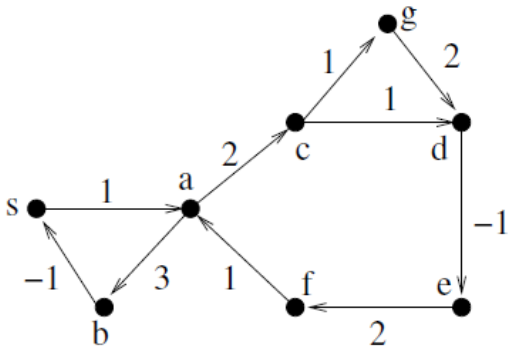


Figure 1: Counterexample to Karp's algorithm for constructing a minimum mean weight cycle.

1, explained above, shows that a minimizer need not lie on a cycle of minimum cycle mean. Even for a vertex v such that the assumption applies, it is not true for every minimizing pair that v is a part of: in the figure, $(d, 6)$ is a minimizing pair for which the assumption applies, but $(d, 7)$, given by $(s, a, c, d, e, f, a, c, d)$ and (s, a, b, s, a, c, d), is one where it is not.

Naive solution

One fix would be to apply his suggested algorithm to each minimizing pair, since the assumptions apply to at least one of them. However, even when the assumptions apply to a given minimizing pair (v, k), the existence of more than one cycle of minimum mean weight can mean that there is more than one minimum-weight path of length n from s to v. It may be the case that not all of them satisfy the required conditions, and the dynamic programming algorithm might find one that does not. There is a way around this, but there are $\Theta(|V|^2)$ minimizing pairs in the worst case, and some care is needed to keep the time bound of this approach to $O(|V| \cdot |E|)$. Next we show more efficient solution.

(Efficient) Alternative for Karp's Algorithm

Lemma 1 *Let v be a vertex such that there exists k, where v and k are a minimizing pair. Every cycle on the length n path from s to v of weight $d_n(v)$ is a cycle of minimum mean weight.*

Proof 1 *Let W be a length n path from s to v of weight $d_n(v)$. Subtracting μ^* from the weight of every edge of G reduces the mean weight of every cycle and path by μ^* in the resulting graph G'. The cycles of minimum mean weight in G are those that have weight 0 in G', if v and k are a minimizing pair, they remain one in G', and W remains a minimum-weight path of length n from s to v in G'. It suffices to show that every cycle on W has weight 0 in G'.*

Suppose there's a cycle of positive weight on W. Omission of the cycles on W results in a path P from s to v of weight $w < d_n(v)$ in G'. Let $k' = |P|$. In G', $d_{k'}(v) < d_n(v)$, and $\frac{d_n(v)-d_{k'}(v)}{n-k'} > 0$, which is the minimum cycle mean of G', contradicting that v must be a minimizer in G' by Theorem 1.

If v is a minimizer, then following backpointers from (n, v) of the table gives a path of weight $d_n(v)$ from s to v. Since it is longer than $n - 1$, it has a cycle, and by the lemma, every cycle on it is a cycle of minimum mean weight. By traversing backpointers marking vertices visited by the path until a previously marked vertex w is encountered, a cycle of minimum mean weight can be identified in $O(|V|)$ time.