动态规划算法的各种应用和最新进展

0 引言

20 世纪 50 年代初美国数学家 R.E.Bellman 等在研究多阶段决策过程（Multistep decision process）的优化问题时，提出了著名的最优化原则（Principle of optimality），把多阶段过程转化为一系列单阶段问题进行逐个求解，创立了解决多阶段决策过程的新方法——动态规划。1957 年 Bellman 在普林斯顿大学发表了动态规划的第一本专著[1]。动态规划在最优控制、资源理论、工业工程、经济管理、变分法和马尔可夫过程等方面得到了广泛应用。采用动态规划方法，可以优雅而高效的解决许多贪心算法或分治算法无法解决的问题，例如著名的 Knapsack 问题，因此动态规划正吸引越来越多的人在各自不同的领域作出新的努力。

本文意在讨论动态规划应用场景中的一些典型问题及其最新进展。

1 动态规划算法适用问题

如果一个问题可以考虑用动态规划算法解决，一般而言具有两个重要特点：

a. 最优子结构：为了解决原始问题，可以定义一组子问题。而原始问题的最优解由子问题的最优解构成；
b. 重复子问题：子问题出现大量重复。

2 动态规划算法的设计

在许多问题中使用动态规划要比线性规划或非线性规划更有成效。但是动态规划并不像线性规划或非线性规划那样有固定的解法，它必须依据问题本身的特性，利用灵活的数学技巧来处理。一般而言，设计动态规划算法可以遵循一下三个步骤：

Step 1. 分析动态规划可行性；
Step 2. 定义子问题，写出递归表达式；
Step 3. 建表记录重复子问题的解，递归求得原始问题的解；
Step 4. 分析算法复杂度。

对于大多数问题，第两步是算法核心，较难，需要深入剖析问题特性。

3 动态规划算法的一些应用

3.1 最短路问题与导航系统

最短路问题是应用动态规划方法求解的一个最直观、最生动的场合。用图论的语言来描述最短路问题即是：

给定一个不含有负圈的图$G=(V=\{1,...,n\},E)$，求出所有点对间的的最短路径。

基于动态规划的 Floyd-Warshall 算法给出了一个巧妙的解法。我们以最短路问题为例按照算法设计的一般步骤进行讨论：

Step 1. 从 A 到 B 的最短路径的任意子路径必是该点对间的最短路径，问题具有最优子结构，可以用动态规划算法求解；

Step 2. 定义子问题：

$dist(i,j,k)=$只允许节点$\{1,...,\mathrm{k}\}$作为中间节点的情况下，G中从i到j的最短路

$$dist(i,j,0)=\begin{cases} len(i,j), & 若(i,j)\in E; \\ +\infty, & 若(i,j)\notin E. \end{cases}$$

则：$dist(i,j,n)=G$中从i到j的最短路即为所求。在扩大中间点集纳入点$k+1$时，需要重新考虑此时i、j对的最短距离，不难发现这时只需比较$dist(i,j,k)$与$dist(i,k,k)+dist(k,j,k)$大小即可，递推关系如下：

$$dist(i,j,k+1)=\min\{dist(i,j,k),\ dist(i,k,k)+dist(k,j,k)\}$$

Step 3. 编程求解，见附录。

Step 4. 观察程序，Floyd-Warshall 算法的时间复杂度主要取决于其中对所有节点一个三重循环，因此其时间复杂度为$O(|V|^3)$。

最短路问题在现实中应用广泛。比如，所有的导航系统采用的都是动态规划的方法，将一个“寻找全程最短路线”的问题，分解成一个个寻找局部最短路线的小问题，可以使用户能在几秒钟内找到最佳行车路线。

3.2 文本相似度量方法：LCS 与编辑距离

3.2.1 LCS(Longest common subsequence)

最长公共子序列（LCS）是一个在一个序列集合中（通常为两个序列）用来查找所有序列中最长子序列的问题。它与查找最长公共子串的问题不同的地方是：子序列不需要在原序列中占用连续的位置，但必须是按照在原始序列中从前向后的顺序排列的 。如：

英文单词 algorithm 与 alignment 的一个最长公共子序列为 algt，从图中也可看出 LCS 问题可能有多个解。

对给定序列 X、Y, $|X|=m$; $|Y|=n$，若采用暴力搜索的方法，时间复杂度为 $O(n2^m)$。注意到最长子序列问题是有最优子结构的，可将子问题定义为：寻找序列 X、Y 的前缀对的最长子序列。具体的，以 X_i、Y_j 表示 X、Y 的长度分别为 i、j 的前缀；用 $c(i,j)$ 表示 X_i、Y_j 的 LCS 的长度，则 $c(m,n)$ 为 X、Y 的 LCS，即为所求。递推关系为：

$$c(i,j)=\begin{cases}0 & \text{if } i=0 \text{ or } j=0\\ c(i-1,j-1)+1 & \text{if } i,j>0 \text{ and } x_i=y_j,\\ \max(c(i,j-1),c(i-1,j)) & \text{if } i,j>0 \text{ and } x_i\neq y_j\end{cases}$$

不难看出，使用动态规划求解原问题的时间复杂度为 $O(mn)$。直观上，我们可以认为 X 和 Y 的 LCS 越长，那么 X 与 Y 越相似，因此通过恰当的定义，可以将 LCS 用作文本间的相似度量。LCS 是数据比较程序，比如 Diff 工具，和生物信息学应用的基础。它也被广泛地应用在版本控制，比如 Git 用来调和文件之间的改变。

3.2.2 编辑距离

编辑距离最早由俄罗斯科学家 Levenshtein 提出，又称为“Levenshtein 距离”。其定义为：给定两个字符串 X 和 Y，将 X 通过删除、插入和替换三种操作转换为 Y 所需要的最少操作次数。编辑距离衡量的是两个字符串之间的差异程度，差异程度越小，相似程度越大。

采用动态规划解决编辑距离的思路跟 LCS 相似，子问题取做 X 和 Y 的前缀的个编辑距离，以 $E(i,j)$ 表示。注意在分析递推关系时注意分情况讨论即可。写出递推关系如下：

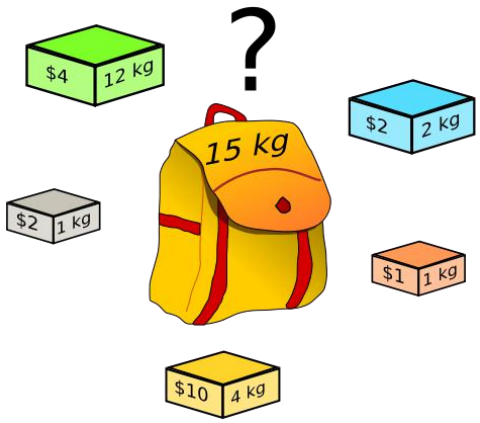
$$E(i,j)=\min\{1+E(i-1,j),1+E(i,j-1),diff(i,j)+E(i-1,j-1)\}$$

$$diff(i,j)=\begin{cases}0, & if\ x_i=y_j;\\ 1, & otherwise.\end{cases}\qquad E(i,0)=i,\quad E(0,j)=j$$

编辑距离常见于生物 DNA 序列分析、模糊匹配、拼写检查等场景中，在语音识别、图像处理领域也有使用编辑距离的场景。

3.3 背包问题和最优配载

背包问题是一中组合优化的 NP 完全问题。给定一组物品，每种物品都有自己的重量和价格，在限定的总重量内，如何选择，才能使得物品的总价格最高。问题名称来源于如何选择最合适的物品放置于给定背包中。背包问题可以使用贪婪算法求解，但是由于贪婪算法本身的特点，不一定求得最优解，使用动态规划可以弥补这一问题。



3.3.1 01 背包问题

01 背包问题是最基础的背包问题，又称为不重复的背包问题。即给定一个容量为 W 的背包，物品条目 $1,...,n$，其中第 i 个物品重 w_i，价格为 v_i。寻找集合 $S \subseteq \{1,...,n\}$，使：$\sum_{i \in S} w_i \le W$，且最大化 $\sum_{i \in S} v_i$。

使用动态规划求解，子问题及递推关系如下表示：

$$K(w,j) = \{\text{容量限制为}w\text{且仅允许放入使用物品}1,...,j\text{的情况下的最优值}$$

$$= \begin{cases} K(w,j-1); & if \ \ w_j > w; \\ \max\{K(w,j-1),\ K(w-w_j,j-1)+v_j & otherwise. \end{cases}$$

3.3.2 完全背包问题

完全背包问题又称为带有重复的背包问题，在上述问题的基础上，假定每件物品有无限件可以用。完全背包问题的求解甚至更为简单，因为此时不需要考虑当前物品是否在前一个状态中使用过，只需要一个一维数组便可以实现求解。子问题及递推条件：

$$K(w,j) = \{\text{容量限制为}w\text{的情况下的最优值}$$

$$= \max_{i:w_i \le w} K(w-w_i)+v_i$$

另外，背包问题有多种变形形式：多重背包问题、分组背包问题、有依赖的背包问题、二次背包问题等，需要根据实际约束进行相应的算法修改与设计。背包问题经常出现在商业资源配置、组合数学、计算复杂性理论、密码学和应用数学等领域。

3.4 动态规划与最优控制

动态系统指一个随时间变化的系统。根据时间变化的方式：离散或连续，可将动态系统分为分为离散时间系统或连续时间系统。寻求动态系统最优控制理论的方法是当今发展起来的一个数学新分支——最优控制理论。最优控制理论研究的主要问题是：根据已建立的被控对象的时域数学模型或频域数学模型，选择一个容许的控制律，使得被控对象按预定要求运行，并使给定的某一性能指标达到最优值。从数学的观点看，最优控制问题是求解一类带有约束条件的泛函极值问题，属于变分学的理论范畴。现代变分理论最常用的两种方法是极小值原理和动态规划。

回顾 Bellman 的最优化原则：多级决策过程的最优策略具有这样的性质：不论初始状态和初始决策如何，当把其中任一状态再作为初始状态时，就后一过程而言，以后诸决策必构成最优策略。

以连续系统为例，最优化原则可表述为：若对于初始时刻t_0和初始状态$x(t_0)$，$u^*(t)$和$x^*(t)$是所论系统的最优控制和最优轨线，则对于时刻$t_1(t_1>t_0)$和相应状态$x(t_1)$来说，$u^*(t),t\in[t_1,t_f]$和$x^*(t),t\in[t_1,t_f]$仍是所论系统向后的最优控制和最优轨线。

我们介绍一个简单的线性离散系统的动态规划求解[2]。设离散系统方程

$$x(k+1)=x(k)+u(k),\quad x(0)=0,\quad x(4)=1,\quad u(k)\in\Omega=\{-1,0,+1\}$$。

$$J=\sum_{k=1}^{3}[x^2(k)+u^2(k)]$$

为性能指标。求使性能指标为极小的最优控制序列$u^*(k),k=0,1,2,3,$以及最优轨线$u^*(k),k=0,1,2,3,4$。

将此最优控制问题视为多阶段决策过程，可采用动态规划求解。定义：

$$J(i)=\{\text{自终止时刻回溯}i\text{个时段内的反向阶段最优性能指标}$$

则有$J(0)=0,\ J(1)=\min_{u(3)\in\Omega}\{x^2(3)+u^2(3)+J(0)\}$, $J(4)$即为所求的系统最优指标，一般的：

$$J(i)=\min_{u(4-i)\in\Omega}\{x(4-i)^2+u^2(4-i)+J(i-1)\}$$

求解时注意讨论所有的可能情况，最终解得：

$$u^*(k)=\{0,0,0,1\}$$
$$x^*(k)=\{0,0,0,0,1\}$$
$$J^*=J(4)=1$$

事实上，由动态规划的最优化原则可以推导出最优控制问题解的必要性条件——最大值原理，因此两者之间有着密切关系，详细过程可参阅[3]。

3.5 动态规划解数学问题

考虑一个矩阵连乘问题：给定 $d_{i-1},...,d_i\in N^+$ 和 n 个矩阵 $A_1,...,A_n$，其中 $\dim(A_i)=d_{i-1}\times d_i$，考虑到矩阵乘法满足结合律，找出花费最小的结合方法，注意到：$d_0\times d_1$ 的矩阵与 $d_1\times d_2$ 的矩阵进行乘法操作需要花费 $d_0\cdot d_1\cdot d_2$ 次标量乘法操作。如 $A_{50\times 20}B_{20\times 1}C_{1\times 10}D_{10\times 100}$，不难发现最优的方法为：$(A_{50\times 20}B_{20\times 1})(C_{1\times 10}D_{10\times 100})$，最优值为 7000。

定义 $C(i,j)=\{M(i:j)$的最优乘法运算数，其中$M(i:j)=M_iM_{i+1}...M_j, i<j\}$

则有

$$\begin{cases}C(i,i)=0\\ C(i,j)=\min\limits_{i\le k\le j}\{C(i,k)+C(k+1,j)+m_{i-1}m_km_j\}\end{cases}$$

最终求得 $C(1,n)$ 即为原问题的解。

除了矩阵连乘问题，动态规划还可用于计算二项式系数、某些不等式的证明、求函数的最佳平方逼近、求解线性规划问题等数学问题中。

动态规划应用还有许多典型问题，如旅行商问题、加权间隔调度、生产计划、马尔可夫决策过程等，限于篇幅，不再一一展开。

4 动态规划最新进展

4.1 自适应动态规划

非线性系统的最优控制动态规划方法需要求解 HJB(Hamilton-Jacobi-Bellman) 方程来得到最优控制策略，而偏微分方程的求解是一件非常困难的事情。长期以来，在使用动态规划方法求解最优控制问题的尝试中，出现了“维数灾难”的问题。

自适应动态规划(Adaptive dynamic programming,ADP)是最优控制领域新兴起的一种近似最优方法，最早有 Werbos 提出[4]。主要思想是利用函数近似结构（神经网络、模糊理论）来近似 HJB 方程的解，采用离线迭代或在线更新的方法，来近似系统的最优控制策略，从而能够有效的解决非线性系统的优化控制问题。

近年来，ADP 方法获得了广泛的关注，也产生了一系列的同义词，如：自适应评价设计、启发式动态规划、增强学习等，可以看出 ADP 方法是结合了学习理论和运筹学的一种最优控制方法。2006 年美国科学基金会组织的“2006 NSF Workshop and Outreach Tutorials on Approximate Dynamic Programming”研讨会上，建议将该方法统称为“Adaptive/Approximate dynamic programming”。

ADP 算法经历了一个由离线迭代到在线实现的发展过程，其理论研究主要涉及到稳定性分析和收敛性证明。ADP 算法具有独特的结构和算法，其作为 DP 方法的近似解法，克服了 DP 方法的局限性，更适合应用在强耦合、强非线性、高复杂性的系统中。目前，ADP 方法在电力系统、智能交通控制、飞行器、通讯系统等已有一些成功的应用，并且正以较快的速度蓬勃发展。

ADP 方法作为一种新型的近似最优求解算法，对其研究刚刚起步。新型 ADP 算法的提出、在线自适应算法的完善、基于输出反馈的 ADP 方法研究、ADP 方法实际应用等都有待于进一步研究。

5 附录

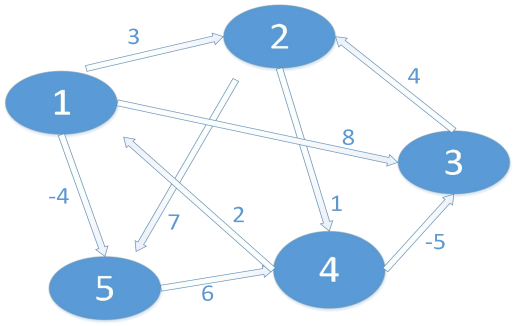
一 最短路问题

```
function [S, P] = APSP(A)
% APSP All-pairs shortest paths Using Floyd-Warshall algorithm
% Given: G: a graph without negative cycles;
% A is the weighted adjacent matrix of n*n s.t.
% A(i,j) = {len(i,j), if (i,j) is a length of G
% A(i,j) = Inf, o.w.
% Return S:n*n = the length of a shortest G-path from i to j.
n = length(A);
dist = zeros(n, n, n+1);
PI = dist;   %记录求解过程中的前驱结点
dist(:, :, 1) = A;
for i = 1:n
      for j = 1:n
```

```
            if A(i, j) ~= 0 && A(i, j) ~= Inf
                PI(i, j, 1) = i;
            else
                PI(i, j, 1) = nan;
            end
        end
end         %初始化 PI 矩阵

for k = 1:n
        for i = 1:n
            for j = 1:n
                if dist(i, j, k) <=    dist(i, k, k)+dist(k, j, k)
                    dist(i, j, k+1) = dist(i, j, k);
                    PI(i, j, k+1) = PI(i, j, k);
                else
                    dist(i, j, k+1) = dist(i, k, k)+dist(k, j, k);
                    PI(i, j, k+1) = PI(k, j, k);
                end
            end
        end
end
S = dist(:, :, n+1);
P = PI(:, :, n+1);          %根据前驱节点矩阵可以写出对应的路径
end

% Example
```




```
% >>A= [0, 3, 8, Inf, -4;
```

```
%         Inf, 0, Inf, 1, 7;
%         Inf, 4, 0, Inf, Inf;
%         2,Inf, -5, 0, Inf;
%         Inf, Inf, Inf, 6, 0];
%
% >>[S, P] = APSP(A)
%
%S =
%
%          0        1       -3        2       -4
%          3        0       -4        1       -1
%          7        4        0        5        3
%          2       -1       -5        0       -2
%          8        5        1        6        0
%
% P =
%
%       NaN        3        4        5        1
%          4     NaN        4        2        1
%          4        3     NaN        2        1
%          4        3        4     NaN        1
%          4        3        4        5     NaN
%
%最终，从 1 到 2 的最短路径为：  1→5→4→3→2.
```

二 最长公共子序列问题

```
function [c, b, lcs] = LCS_Length(X, Y)
% Longest common subsequence of X and Y.
%  正向使用动态规划求解并记录过程值
m = length(X);
n = length(Y);
c = zeros(m+1, n+1);
b = cell(m+1, n+1);
```

```
for i = 1:m
    for j = 1:n
        if X(i) == Y(j)
            c(i+1, j+1) = c(i, j) + 1;
            b{i+1, j+1} = 'leftup';
        elseif c(i, j+1) >= c(i+1, j)
            c(i+1, j+1) = c(i, j+1);
            b{i+1, j+1} = 'up';
        else
            c(i+1, j+1) = c(i+1, j);
            b{i+1, j+1} = 'left';
        end
    end
end
%回溯并返回 LCS
lcs = [];
i = m+1;
j = n+1;
while c(i, j) > 0
    if strcmp(b{i, j}, 'up')
        i = i-1;
        continue
    end
    if strcmp(b{i, j}, 'left')
        j = j-1;
        continue
    end
    if strcmp(b{i, j}, 'leftup')
        lcs = [X(i-1), lcs];
        i = i-1;
        j = j-1;
        continue
    end
end
```

```
end

% example
% >>X = 'president';
% >>Y = 'providence';
% >>[c, b, lcsxy] = LCS_Length(X, Y)
% 输出结果：
% lcsxy =
%
% priden
```

三 编辑距离

```
function [E, edcost] = editdistance(X, Y)
%Editdistance of X and Y(Insert\Delete\Substitute).
m = length(X);
n = length(Y);
E = zeros(m+1, n+1);
E(:, 1) = 0:m;
E(1, :) = 0:n;
for i = 1:m
    for j = 1:n
        if X(i) == Y(j)
            E(i+1, j+1) = min([1+E(i, j+1), 1+E(i+1, j), E(i, j)]);
        else
            E(i+1, j+1) = min([1+E(i, j+1), 1+E(i+1, j), 1+E(i, j)]);
        end
    end
end
edcost = E(m+1, n+1);
return
end
%example
%>> X = 'snowy';
```

```
%>> Y = 'sunny';
%>> [E, edcostxy] = editdistance(X, Y);
% 输出结果：
%
% edcostxy =
%
%           3
```

四 背包问题

```
function maxvalue = KnapsackRE(W, weights, values)
%01 背包问题
K = 0:W;
if length(weights) ~= length(values)
    fprintf('Parameter Input Error')
    K = [];
    return
end
n = length(weights);
for w = 1:W
    K(w+1) = 0;
    for i = 1:n
        if weights(i) <= w
            K(w+1) = max(K(w+1), K(w-weights(i)+1) + values(i));
        end
    end
end
maxvalue = K(W+1);
End

% >>KnapsackRE(15, [12, 2, 1, 1, 4], [4, 2, 2, 1, 10])
%
% ans =
%
```

```
%        36

function maxvalue = Knapsack(W, weights, values)
%完全背包问题
if length(weights) ~= length(values)
    fprintf('Parameter Input Error')
    K = [];
    return
end
n = length(weights);
K = zeros(W+1, n+1);
for j = 1:n        %按照 item 的序号进行一次性判断赋值，使结果中不出现重复
    for w = 1:W
        if weights(j) > w
            K(w+1, j+1) = K(w+1, j);
        else
            K(w+1, j+1) = max(K(w+1, j), K(w-weights(j)+1, j)+values(j));
        end
    end
end
maxvalue = K(W+1, n+1);
end

%>>Knapsack(15, [12, 2, 1, 1, 4], [4, 2, 2, 1, 10])
%
% ans =
%
%        15
```

五　矩阵连乘问题

```
function [C, t] = MatrixChainMul( d )
%矩阵连乘问题,d 为 n 个矩阵的维数序列，即 A1 矩阵维数为 d(1)*d(2)... An 的维数为 d(n)*d(n+1)；
n = length(d) - 1;
```

```
C = zeros(n);
t = C;
for s = 1:n-1
    for i = 1:n-s
        j = i+s;
        C(i, j) = Inf;
        for k = i:j-1
            w = C(i, k) + C(k+1, j) + d(i)*d(k+1)*d(j+1);
            if w < C(i, j)
                C(i, j) = w;
                t(i, j) = k;
            end
        end
    end
end
end

%>> MatrixChainMul([50, 20, 1, 10, 100])
%ans =
%
%         0    1000    1500    7000
%         0       0     200    3000
%         0       0       0    1000
%         0       0       0       0
```

6 参考文献

[1] Bellman R E. Dynamic Programming[J]. 1957.

[2] 胡寿松，王执铨，胡维礼. 最优控制理论与系统[M]. 科学出版社, 2005.

[3] 王日爽，徐兵，魏权龄. 应用动态规划[M]. 国防工业出版社, 1987.

[4] Werbos P J. Advanced forecasting methods for global crisis warning and models of intelligence[J]. General Systems Yearbook, 1977, 22(12): 25-38.

[5] Liu D, Wei Q. Policy iteration adaptive dynamic programming algorithm for

discrete-time nonlinear systems[J]. IEEE Transactions on Neural Networks and Learning Systems, 2014, 25(3): 621-634.

[6] Bellman R. Dynamic programming treatment of the travelling salesman problem[J]. Journal of the ACM (JACM), 1962, 9(1): 61-63.

[7] 张化光, 张欣, 罗艳红, 等. 自适应动态规划综述[J]. 自动化学报, 2013, 39(4): 303-311.

[8] 徐昕, 沈栋, 高岩青, 等. 基于马氏决策过程模型的动态系统学习控制: 研究前沿与展望[J]. 自动化学报, 2012, 38(5): 673-687.

[9] 张如, 贾亚朝, 唐贤芳. 浅谈动态规划的几方面应用[J]. 价值工程, 2016, 35(19): 1-3.

[10] 申科. 动态规划算法的原理, 应用和最新进展[J]. 中国电子商情: 通信市场, 2006 (1): 41-43.

[11] 关秀翠. 优化算法的设计.东南大学:B007138,2017.