Dynamic Programming

Victor Eijkhout

Notes for CS 594 – Fall 2004

What is dynamic programming?

- Solution technique for minization problems
- Often lower complexity than naive techniques
- Sometimes equivalent to analytical techniques

What is it *not*?

- Black box that will solve your problem
- Way of finding lowest complexity

When dynamic programming?

- Minimization problems
- constraints; especially integer
- sequence of decisions

Decision timing

Description

- Occasions for deciding yes/no
- items with attractiveness $\in [0, 1]$
- Finite set
- no reconsidering
- Question: at any given step, do you choose or pass?
- Objective: maximize expectation

crucial idea

- start from the end:
- step N: no choice left
- expected yield: .5

crucial idea

- start from the end:
- step N: no choice left
- expected yield: .5
- in step $N - 1$: pick if better than .5

yield from $N-1$

- pick if $> .5$: done in .5 of the cases

yield from $N-1$

- pick if $> .5$: done in .5 of the cases
- expected yield .75

yield from $N-1$

- pick if $> .5$: done in .5 of the cases
- expected yield .75
- go on in .5 of the cases

yield from $N-1$

- pick if $> .5$: done in .5 of the cases
- expected yield .75
- go on in .5 of the cases
- expected yield then .5

yield from $N-1$

- pick if $> .5$: done in .5 of the cases
- expected yield .75
- go on in .5 of the cases
- expected yield then .5
- total expected yield: $.5 \times .75 + .5 \times .5 = .625$

at $N-2$

- pick if better than .625

at $N-2$

- pick if better than .625
- happens in .375 of the cases,

at $N - 2$

- pick if better than .625
- happens in .375 of the cases,
- yield in that case $1.625/2$

at $N-2$

- pick if better than .625
- happens in .375 of the cases,
- yield in that case $1.625/2$
- otherwise, .625 yield from later choice
- et cetera

Essential features

- Stages: more or less independent decisions
- Global minimization; solving by subproblems
- Principle of optimality: sub part of total solution is optimal solution of sub problem

Manufacturing problem

Statement

- Total to be produced in given time, variable cost in each time period
- wanted: scheduling
- $$\min_{\sum p_k = S} \sum w_k p_k^2.$$

define concepts

- amount of work to produce s in n steps:

$$v(s|n) = \min_{\sum_{k>N-n} p_k = s} \sum w_k p_k^2$$

- optimal amount $p(s|n)$ at n months from the end

principle of optimality

$$
\begin{aligned}
v(s|n) &= \min_{p_n \le s} \left\{ w_n p_n^2 + \sum_{\substack{k > N-n+1 \\ \sum p_k = s - p_n}} w_k p_k^2 \right\} \\
&= \min_{p_n \le s} \left\{ w_n p_n^2 + v(s - p_n | n-1) \right\}
\end{aligned}
$$

start from the end

- In the last period: $p(s|1) = s$, and $v(s|1) = w_1 s^2$

start from the end

- In the last period: $p(s|1) = s$, and $v(s|1) = w_1 s^2$
- period before:

$$v(s|2) = \min_{p_2}\{w_2 p_2^2 + v(s - p_2|1)\} = \min_{p_2} c(s, p_2)$$

where $c(s, p_2) = w_2 p_2^2 + w_1(s - p_2)^2$.

start from the end

- In the last period: $p(s|1) = s$, and $v(s|1) = w_1 s^2$
- period before:

$$v(s|2) = \min_{p_2}\{w_2 p_2^2 + v(s - p_2|1)\} = \min_{p_2} c(s, p_2)$$

  where $c(s, p_2) = w_2 p_2^2 + w_1(s - p_2)^2$.
- Minimize: $\delta c(s, p_2)/\delta p_2 = 0$,
- then $p(s|2) = w_1 s/(w_1 + w_2)$ and
  $v(s|2) = w_1 w_2 s^2/(w_1 + w_2)$.

general form

- Inductively

$$p(s|n) = \frac{1/w_n}{\sum_{i=1}^{n} 1/w_i} s, \qquad v(s|n) = s^2 \sum_{i=1}^{n} 1/w_i.$$

general form

- Inductively

$$p(s|n) = \frac{1/w_n}{\sum_{i=1}^{n} 1/w_i} s, \qquad v(s|n) = s^2 \sum_{i=1}^{n} 1/w_i.$$

- Variational approach:

$$\sum_k w_k p_k^2 + \lambda(\sum_k p_k - S)$$

Constrained minimization

general form

- Inductively

$$p(s|n) = \frac{1/w_n}{\sum_{i=1}^{n} 1/w_i} s, \qquad v(s|n) = s^2 \sum_{i=1}^{n} 1/w_i.$$

- Variational approach:

$$\sum_k w_k p_k^2 + \lambda(\sum_k p_k - S)$$

  Constrained minimization
- Solve by setting derivatives to p_n and λ to zero.

characteristics

- Stages: time periods
- State: amount of good left to be produced
- Principle of optimality

characteristics

- Stages: time periods
- State: amount of good left to be produced
- Principle of optimality
- Can be solved analytically

characteristics

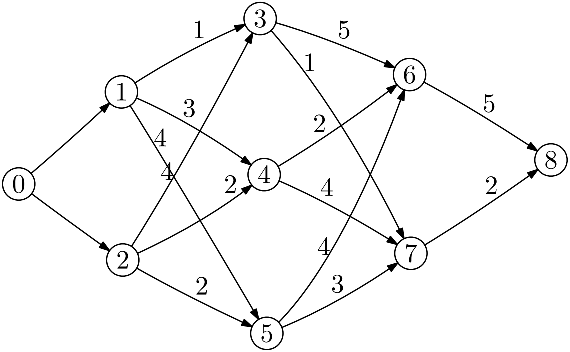
- Stages: time periods
- State: amount of good left to be produced
- Principle of optimality
- Can be solved analytically
- Analytical approach can not deal with integer constraints

Stagecoach problem

Statement

Several routes from beginning to end

Cost of travel insurance:



Objective: minimize cost

data in python

```
table = [ [0, 5, 4, 0, 0, 0, 0, 0, 0], # first stage: 0
          [0, 0, 0, 1, 3, 4, 0, 0, 0], # second: 1 & #2
          [0, 0, 0, 4, 2, 2, 0, 0, 0],
          [0, 0, 0, 0, 0, 0, 5, 1, 0], # third: 3, #4, #5
          [0, 0, 0, 0, 0, 0, 2, 4, 0],
          [0, 0, 0, 0, 0, 0, 4, 3, 0],
          [0, 0, 0, 0, 0, 0, 0, 0, 5], # fourth: 6 & #7
          [0, 0, 0, 0, 0, 0, 0, 0, 2]
        ]
final = len(table);
```

recursive formulation

- In the final city, the cost is zero;
- Otherwise minimum over all cities reachable
  of the cost of the next leg plus the minimum cost from that city.
  (principle of optimality)

recursive formulation

- In the final city, the cost is zero;
- Otherwise minimum over all cities reachable
  of the cost of the next leg plus the minimum cost from that city.
  (principle of optimality)
- wrong to code it recursively

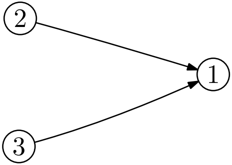
recursive solution in python

```
def cost_from(n):
    # if you're at the end, it's free
    if n==final: return 0
    # otherwise range over cities you can reach
    # and keep minimum value
    val = 0
    for m in range(n+1,final+1):
        local_cost = table[n][m]
        if local_cost>0:
            # if there is a connection from here,
            # compute the minimum cost
            local_cost += cost_from(m)
            if val==0 or local_cost<val:
                val = local_cost
    return val
print "recursive minimum cost is",cost_from(0)
```

backward

characteristic

- Overlapping subproblems



- `cost_from(1)` computed twice
- Cost: N cities, S stages of L each: $O(L^S)$

dynamic programming

- Compute minimum cost $f_n(x_n)$
  of traveling from step n, starting in x_n (state variable)
- Formally, $f_k(s)$ minimum cost for traveling from stage k starting in city s. Then

$$f_{k-1}(s) = \min_t \{c_{st} + f_k(t)\}$$

  where c_{st} cost of traveling from city s to t.

backward dynamic solution in python

```
cost = (final+1)*[0] # initialization
# compute cost backwards
for t in range(final-1,-1,-1):
    # computing cost from t
    for i in range(final+1):
        local_cost = table[t][i]
        if local_cost==0: continue
        local_cost += cost[i]
        if cost[t]==0 or local_cost<cost[t]:
            cost[t] = local_cost
print "minimum cost:",cost[0]
```

recursive forward

analysis

- Running time $O(N \cdot L)$ or $O(L^2 S)$
- compare L^S for recursive

Forward solution

- Backward: $f_n(x)$ cost from x in n steps to the end
- Forward: $f_n(x)$ cost in n steps to x

$$f_n(t) = \min_{s<t}\{c_{st} + f_{n-1}(s)\}$$

- sometimes more appropriate
- same complexity

forward dynamic solution in python

```
cost = (final+1)*[0]
for t in range(final):
    for i in range(final+1):
        local_cost = table[t][i]
        if local_cost == 0: continue
        cost_to_here = cost[t]
        newcost = cost_to_here+local_cost
        if cost[i]==0 or newcost<cost[i]:
            cost[i] = newcost
print "cost",cost[final]
```

backward

Traveling salesman

Problem statement

- Cities as stages?

Problem statement

- Cities as stages?
- No ordering

Problem statement

- Cities as stages?
- No ordering
- Stage n : having n cities left

Problem statement

- Cities as stages?
- No ordering
- Stage n : having n cities left
- State: combination of cities to visit plus current city
- Cost formula to 0 (both start/end)

$$\begin{aligned} C(\{\,\},f) &= a_{f0} \quad \text{for } f = 1,2,3,\ldots \\ C(S,f) &= \min_{m\in S} a_{fm} + [C(S-m,m)] \end{aligned}$$

backward implementation

```
def shortest_path(start,through,lev):
    if len(through)==0:
        return table[start][0]
    l = 0
    for dest in through:
        left = through[:]; left.remove(dest)
        ll = table[start][dest]+shortest_path(dest,left,lev+1)
        if l==0 or ll<l:
            l = ll
    return l
to_visit = range(1,ntowns);
s = shortest_path(0,to_visit,0)
```

(recursive; need to be improved)

Discussion

Characteristics

Stages sequence of choices

Stepwise solution solution by successive subproblems

State cost function has a state parameter, description of work left &c

Overlapping subproblems

Principle of optimality This is the property that the restriction of a global solution to a subset of the stages is also an optimal solution for that subproblem.

Principle of optimality

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

derivation

Maximize $\sum_i^N g_i(x_i)$ under $\sum_i x_i = X$, $x_i \geq 0$
Call this $f_N(X)$, then

$$\begin{aligned}
f_N(X) &= \max_{\sum_i^N x_i = X} \sum_i^N g_i(x_i) \\
&= \max_{x_N < X} \left\{ g_N(x_N) + \max_{\sum_i^{N-1} x_i = X - x_N} \sum_i^{N-1} g_i(x_i) \right\} \\
&= \max_{x_N < X} \{ g_N(x_N) + f_{N-1}(X - x_N) \}
\end{aligned}$$