Better bounding

Contents

1 Recall

1.1 Traveling Salesman Problem

Given a complete graph $G = (V, E)$ with $V = \{1, \ldots, n\}$ and a cost function

$$\text{cost} : E \to \mathbb{Z}_{>0}$$

Find a Hamiltonian cycle X of G such that

$$\text{cost}(X) = \sum_{e \in X} \text{cost}(e)$$

is minimized.

1.2 Naive backtracking solution

Algorithm: NaiveTravelingSalesman

```
global: X, OptC, OptX, C
input: l
if l=n
    C = cost([x(0),...,x(n-1)] (as a cycle)
    if C < OptC
        OptC = C
        OptX = [x(0),...,x(n-1)]
    return
if l=0
    C(l)={0}
if l=1
    C(l)={1,...,n-1}
else
    C(l)=C(l-1)-{x(l-1)}
for x(l) in C(l)
    NaiveTravelingSalesman(l+1)
```

1.3 Min Cost Bound

Definition. Given a graph and edge costs as above, and given $x \in V$, $W \subseteq V$, $W \neq \emptyset$ define

$$b(x, W) = \min\{\text{cost}(x, y) : y \in W\}$$

Proposition. *Given a graph and edge costs as above. Let* $X' = [x(0), \ldots, x(n-1)]$ *be the ninimim cost Hamiltonian cycle which extends* $[x(0), \ldots, x(l-1)]$ *(with* $l < n$*). Then*

$$\textit{cost}(X') \geq \sum_{i=0}^{l-2} \textit{cost}(x_i, x_{i+1}) + b(x_{l-1}, Y) + \sum_{y \in Y} b(y, Y \cup \{x_0\})$$

where $Y = V \setminus \{x_0, \ldots, x_{l-1}\}$

Algorithm: MinCostBoundTravelingSalesman

```
global: X, OptC, OptX, C
input: l
if l=n
    D = cost([x(0),...,x(n-1)] (as a cycle)
    if D < OptC
        OptC = D
        OptX = [x(0),...,x(n-1)]
    return
if l=0
    C(l)={0}
else
    if l=1
        C(l)={1,...,n-1}
    else
        C(l)=C(l-1)-{x(l-1)}
    B = MinCostBound(x(0),...,x(l-1))
for x(l) in C(l)
    if B >= OptC
        return
    MinCostBoundTravelingSalesman(l+1)
```

2 Even better bounding functions

(see Donald Kreher and Douglas Stinson, *Combinatorial Algorithms*, sections 4.6 and 4.7 for a reference on this material)

2.1 A better bounding function for the Traveling salesman problem

Definition. Given an instance of the Traveling salesman problem with graph G, let M be the matrix whose $i.j$ entry is ∞ if $i = j$ and is the cost of the edge $\{i, j\}$ otherwise.

For example if we have the graph with vertices $0, 1, 2, 3$ and weights

edge	cost
01	3
02	5
03	8
12	2
13	7
23	8

then

$$M = \begin{bmatrix} \infty & 3 & 5 & 8 \\ 3 & \infty & 2 & 7 \\ 5 & 2 & \infty & 6 \\ 8 & 7 & 6 & \infty \end{bmatrix}$$

Here is an algorithm which finds the smallest element in each row, and subtracts that amount from each entry in the row, and then does the same thing on the columns. The output is the sum of all the smallest elements.

Algorithm: Reduce

```
input: M (m by m matrix)
val = 0
for i from 0 to m-1
    min = M(i,0)
    for j from 1 to m-1
        if M(i,j) < min
            min = M(i,j)
    for j from 0 to m-1
        M(i,j) = M(i,j) - min
    val = val + min
for j from 0 to m-1
    min = M(0,j)
    for i from 1 to m-1
        if M(i,j) < min
            min = M(i,j)
    for i from 0 to m-1
        M(i,j) = M(i,j) - min
    val = val + min
return val
```

Definition. With M as in the previous definition, the cost of M is the output of the `Reduce` algorithm.

Continuing the previous example, the rows contribute $3+2+2+6 = 13$ to the cost leaving the matrix

$$\begin{bmatrix} \infty & 0 & 2 & 5 \\ 1 & \infty & 0 & 5 \\ 3 & 0 & \infty & 4 \\ 2 & 1 & 0 & \infty \end{bmatrix}$$

and so the columns contribute $1+0+1+4 = 6$ to the cost, for a total cost of 19.

This matrix cost is a bounding function for the Traveling salesman problem:

Proposition. *Let M be the matrix for an instance of the Traveling salesman problem and let G be the graph of the instance. Let X be any Hamiltonian cycle of G, then*

$$\texttt{Reduce}(M) \leq \textit{cost}(X)$$

Where we recall from last time that the cost of a Hamiltonian cycle is the sum of the costs of the edges making it up.

Proof. Let $X = [x_0, \ldots, x_{n-1}]$ be a Hamiltonian cycle gives as a list of vertices and let $x_n = x_0$. Then

$$\text{cost}(X) = M(x_0, x_1) + M(x_1, x_2) + \cdots + M(x_{n-2}, x_{n-1}) + M(x_{n-1}, x_n)$$

which uses one entry from each row and column of M.

Let

$$r_i = \min\{M(i,j) : 0 \leq j \leq n-1\}$$
$$c_j = \min\{M(i,j) - r_i : 0 \leq i \leq n-1\}$$

Then the cost of M is

$$\sum_{i=0}^{n-1} (r_i + c_i)$$

and

$$r_{x_i} + x_{x_{i+1}} \leq M(x_i, x_{i+1})$$

so

$$\texttt{Reduce}(M) \leq \text{cost}(X)$$

□

In the running example the Hamiltonian cycles are

Hamiltonian cycle	cost
0123	3+2+6+8 = 19
0132	3+7+6+5 = 21
0213	5+2+7+8 = 22

while the cost of M we calculated above to be 18, so we see this bound isn't tight but it is pretty good.

Next let us write the bounding function based on `Reduce`, and then the backtrack algorithm using it:

Algorithm: ReduceBound

```
  input M (m by m matrix), X, G=(V,E)
  if m=n
       return cost(X) (as a Hamiltonian cycle)

  (build the M for the smaller problem)
  M'(0,0) = infinity
  Y = V - {x(0), ... , x(m-1)}
  j=1
  for y in Y
      M'(0,j) = M(x(m-1),y)
      j = j+1
  i=1
  for x in y
      M'(i,0) = M(x, x(0))
      j=1
      for y in Y
          M'(i,j) = M(x,y)
          j = j+1
      i = i+1
ans = Reduce(M', [x(0),...,x(m-1)], V)

(add in the cost of the partial solution)
for i from 1 to m-1
    ans = ans + M(x(i-1), x(i))

return ans
```

Algorithm: ReduceTravelingSalesman

```
global: X, OptC, OptX, C
input: l
if l=n
    D = cost([x(0),...,x(n-1)] (as a cycle)
    if D < OptC
        OptC = D
        OptX = [x(0),...,x(n-1)]
    return
if l=0
    C(l)={0}
else
    if l=1
        C(l)={1,...,n-1}
    else
        C(l)=C(l-1)-{x(l-1)}
    B = ReduceBound(x(0),...,x(l-1))
for x(l) in C(l)
    if B >= OptC
        return
    ReduceTravelingSalesman(l+1)
```

2.2 Branch and Bound

As we noticed in the knapsack problem, the order in which we recurse on the elements of C_ℓ can make a big difference. We can use the bounding function to make a good choice. When we are at node X in the state space tree, calculate $B(X')$ for every child X' of X. For a maximizing problem (like the knapsack problem), make the recursive calls in decreasing order of $B(X')$. For a minimizing problem (like the traveling salesman problem), make the recursive calls in increasing order of $B(X')$. This way we are likely to get the most pruning.

For the traveling salesman the algorithm will look like

Algorithm: BranchandBoundTravelingSalesman

```
global: X, OptC, OptX, C
input: l
if l=n
    D = cost([x(0),...,x(n-1)] (as a cycle)
    if D < OptC
        OptC = D
        OptX = [x(0),...,x(n-1)]
    return
if l=0
    C(l)=[0]
else
    if l=1
        C(l)=[1,...,n-1]
    else
        C(l)=C(l-1) with x(l-1) removed
B(l) = [];
for x in C(l)
    append bound(x(0),...,x(l-1),x) to B(l)  (use your favorite bounding function)
sort B(l) in increasing order
put C(l) in the corresponding order
for i from 0 to |C(l)|-1
    if B(l)(i) >= OptC
        return
    x(l) = C(l)(i)
    ReduceTravelingSalesman(l+1)
```

Kreher and Stinson ran the naive traveling salesman, the bounded traveling salesman with both the min cost bound and the reduce bound, and the branch and bound traveling salesman with both bounds, on random instances of the problem with edges costs random integers between 0 and 100. The results were (table from p134 and p143). Algorithm 4.10 is the naive backtrack, Algorithm 4.13 is the bounded backtrack and Algorithm 4.23 is the branch and bound algorithm. The values in the table are the sizes of the state space trees.

n	Algorithm 4.10	Algorithm 4.13		Algorithm 4.23	
		MINCOSTBOUND	REDUCEBOUND	MINCOSTBOUND	REDUCEBOUND
5	65	45	18	25	9
10	986,410	5,199	1,287	490	102
15	236,975,164,805	1,538,773	53,486	128,167	5,078
20	$\approx 3.3 \cdot 10^{17}$	64,259,127	1,326,640	6,105,089	39,035

3 Heuristic search

Sometimes even good bounding is too slow. How can we explore the state space faster, perhaps just finding a close to optimal solution rather than an optimal solution.

Definition.

- The universe $\mathcal{X}$ is a finite set of elements which are possible (not necessarily feasible) solutions
- $X \in \mathcal{X}$ is feasible if it satisfies the constraints of the problem
- $P(X)$ is the profit of X
- A neighbourhood function is a function

$$N : \mathcal{X} \to 2^{\mathcal{X}}$$

  where $2^{\mathcal{X}}$ is the set of all subsets of $\mathcal{X}$.
- Given N, a neighbourhood search is an algorithm, possible randomized, which takes a feasible solution $X \in \mathcal{X}$ and returns either a feasible solution $Y \in N(X) \setminus \{X\}$ or `fail`

The idea is that if we are at a feasible solution, then we consider its neighbourhood, and following the neighbourhood search algorithm we update X to Y (or we fail).

For the knapsack problem we could take $\mathcal{X} = \{0,1\}^n$ the set of all binary strings of length n. A reasonable neighbourhood function would be

$$N(X) = \{Y \in \mathcal{X} : d(X,Y) \leq c\}$$

for some fixed c, where $d(X,Y)$ is the Hamming distance between X and Y.

Possible neighbourhood search strategies include

- Find a feasible solution $Y \in N(X)$ such that $P(Y)$ is maximized. Fail if there are no feasible solutions in $N(X)$.
- Find a feasible solution $Y \in N(X)$ such that $P(Y)$ is maximized. If $P(Y) > P(X)$ return Y, otherwise fail.
- Choose a random solution in $N(X)$ return it if it is feasible, otherwise fail (or perhaps try a fixed number of times).
- Choose a random solution Y in $N(X)$ return it if $P(Y) > P(X)$, otherwise fail (or perhaps try a fixed number of times).

More sophisticated heuristic searches may use a combination of approaches.

3.1 Hill climbing

Hill climbing is the simplest heuristic search. Hill climbing is when, as in points 2 and 4 above, we always choose a solution which increases the profit. For many applications this is too naive because it can get stuck in a local maximum which may be quite far from optimal. The basic shape of the algorithm is as follows

Algorithm: GenericHillClimbing

```
Select a feasible X in the universe
searching = true
while searching
    try to get a feasible solution Y in N(X) with P(Y) > P(X) (randomly or exhaustively)
    if Y = fail
        searching = false
    else
        X = Y
return X
```