1 Branch and Bound & Local Search

Minimum Set Cover: Given a universe U and a family S of subsets of U, a cover is a subfamily C⊆S of sets whose union is U. The input is a pair (U, S) and the task is to find a set covering that uses the fewest sets.

1.

(a) Subproblem:

Given a partial solution (U', S'), U' is the set of elements covered by subsets in S'. S' is the family of subsets used in the partial solution from original subset family S. Thus, the subproblem is finding minimum set cover in a smaller universe U* = U-U' using sets S*=S-S'.

(b) How do you choose a subproblem to expand:

In order to minimize the number of subsets, we need to choose the subproblem which covers the most number of elements in U at each stage.

(c) How do you expand a subproblem:

Branch out two subproblems for each subset that is not used in the existing partial solution. One for adding this subset to solution, the other for not adding this subset to solution.

(d) Appropriate lower bound:

The number of subsets used so far in a partial solution.

2.

(a) Greedy Heuristic:

At each step, choose the set that contains the largest number of uncovered elements.

(b) Why it finds a valid solution:

Input (U, S)
C←∅
While U ≠ 0
    Select $S_i \in S$ that maximizes $S_i \cap U$
    U ← U - S_i
    C ← C ∪ S_i
Return C

Start with an empty set C. Let C contains subsets used in the cover. While there exist remaining uncovered elements, choose a subset S_i from S that covers as many uncovered elements as possible. Remove the covered elements from original universe U and add S_i to C. When all the elements are covered, C contains the subsets from S that covers U and the algorithm terminates.

(c) Running Time:

Assume there are *n* elements and *k* sets. The while loop stops when U becomes an empty set so the while loop runs for *k* times. In each loop finding the S_i which maximizes $S_i \cup U$, it takes time $O(n*k)$. So, the time complexity is $O(n*k*k)$.

3.

(a) What could be a possible scoring function for such candidate solutions:

Given a candidate solution, calculate the number of elements it covers, *n'*, and the number of subsets it uses, *k'*. The scoring function is *n'/k'*. We want to use minimum number of subsets to cover maximum number of elements. So, *n'/k'* larger, the candidate solution better.

(b) What could be a Neighborhood (or Moves)? How many potential neighbors can

a candidate solution have:

Iteratively remove or add a subset at a time. Add a new subset that covers elements which are not covered by the present candidate solution. Remove a subset that covers most elements which have already been covered in a candidate solution.

Assume we have k subsets. A candidate solution can have $O(k)$ potential neighbors.

(c) Why would you consider adding Tabu Memory and what would be remembered in your Tabu memory?

Local search methods have a tendency to become stuck in suboptimal regions or on plateaus where many solutions are equally fit. Tabu search enhances the performance of local search by relaxing its basic rule. First, at each step worsening moves can be accepted if no improving move is available (like when the search is stuck at a strict local minimum). In addition, prohibitions (henceforth the term tabu) are introduced to discourage the search from coming back to previously-visited solutions. Tabu memory can record the visited solutions or user-provided sets of rules.

Short-term Tabu memory records the list of solutions recently found. In the set cover problem, a certain number of added or removed subsets in the last several solutions can be remembered in the Tabu memory. It's helpful for choosing candidate solutions containing subsets that have not been visited.

2 Approximation using Greedy Heuristics

(a) Provide example:

w1=0.8ton, w2=0.3ton, w3=0.8ton, w4=0.3ton. In this greedy algorithm, each container will be uploaded in a truck and 4 trucks will be used. The true minimum number of trucks is 3 where container 2 and container 4 will be uploaded in a same truck.

(b) Lower bound for t*:

The number of trucks will be minimum only if n containers can be placed on $t^*=W/1=W$ trucks and all the trucks are full. This case uses the space of trucks to the maximum ratio. If there are trucks with unused space, the total number of trucks will be greater than W.

(c) Lower bound for the weight of containers in each consecutive pair:

The lower bound for the weight of containers in each consecutive pair is 1 ton.

In each consecutive pair, the containers in the second truck must weigh more than the remaining containing capability of the first truck. Otherwise the containers in the second truck should be placed in the first truck and there is no need for a pair.

So, in each consecutive pair two trucks must contain more than 1 ton. Thus, the lower bound in a consecutive pair is 1 ton.

(d) Prove an approximation ratio of 2 for your algorithm:

From (c) we know if we apply the greedy algorithm and use $t=2z$ trucks to contain n containers weighting W, each pair of trucks must contain more than 1 ton. z pairs of trucks must contain more than $1*z=t/2$ ton. So, $t/2<W$, $t<2W$. From (b) we know W is a lower bound of the optimum solution, so the approximation ratio for the greedy algorithm is 2.

(e) Prove the same approximation ratio when using t=2z+1 trucks:

If $2z+1>2W$, $z>W-0.5$. z is an integer so $z>=W$. From (d), for the first z pairs of trucks, all trucks can contain more than $1*z>=W$ ton. So, z pair of trucks can contain all the

containers. There is no need for the $(2z+1)^{th}$ truck. It contracts the assumption that we use $2z+1$ trucks, so $2z+1 <= 2W$. The approximation ratio is 2.

(f) Argue that this approach leaves at most one truck less than half full:

If there are two trucks less than half full, it means when you upload containers weighting less than 0.5 ton to a new truck while the first truck having enough space room (more than 0.5 ton) for those containers. But in the algorithm dealing with those containers, instead of using a new truck, we should look through all used trucks to see if there is any trunk with room for these containers and the containers will be loaded on the first truck. The second truck will not be used at this time. It's a contradiction.

So, the only truck which may take less than 0.5 ton is the last truck when all the previous t-1 trucks taking more than 0.5 ton and none of them have enough room for the remaining containers.

(g) Show that the number of trucks you would use with second strategy is never more than ⌈2W⌉:

From (f) we know, the first t-1 trucks all take more than 0.5 ton. We get $0.5*(t-1) < W$. $t<2W+1$. t is an integer. Thus, $t <= \lceil 2W \rceil$.

(h) Prove an approximation ratio of 2 for this strategy:

Lower bound of optimal value: minimum number of trucks $t = W/1 = W$.

From (g) we get $t <= \lceil 2W \rceil$. t is an integer. Thus, t<= 2W, the approximation ratio for this strategy is 2.

3 Modeling with ILP

(a) Independent set problem:

V variable x_i:

Associate x_i with vertex v_i.

(V+E) constraints:

1. $x_i + x_j \leq 1$ for every $(v_i, v_j) \in E$
   ⇒each edge does not have both endpoints with value 1.
2. $x_i = 0$ or 1, for $1 \leq i \leq n$

Formula:

Maximize Σx_i

(b) Facility location problem:

(m+n) variables:

1. $x_i \in \{0,1\}$⇒ denote whether facility i is open
2. $y_{ij} \in \{0,1\}$⇒ denote whether client j is assigned to facility i

2(n+m) constraints:

1. for any j, $\sum_i y_{ij} \geq 1$ ⇒ each client should be assigned to at least one facility
2. for any i and j, $x_i \geq y_{ij}$ ⇒ if a client is assigned to a facility, then that facility must be open
3. $x_i \in \{0,1\}$
4. $y_{ij} \in \{0,1\}$

Formula:

Minimize $\sum_i f_i x_i + \sum_{i,j} c_{ij} y_{ij}$

(c) Sudoku problem:

Consider the $n^2 * n^2$ matrix as n^2 sub matrixes, each with n rows and n column. The position in row i, column j of submatrix k is denoted by $(i, j; k)$.

n^4 variable:

$y_{i,j;k}^{m} = 1 \Rightarrow$ when position $(i, j; k)$ contains number m

$y_{i,j;k}^{m} = 0 \Rightarrow$ when position $(i, j; k)$ does not contain number m

$4n^4$ Constraints:

To make sure each number m appears exactly once in each row, n^4 constraints:

place number m once in the first n rows ⇒

$\sum_{k=1}^{k=n} \sum_{j=1}^{j=n} y_{r,j;k}^{m} = 1$ for r =1, 2…n, m=1, 2…...n^2

place number m once in the $(n+1)^{th}$ row to $2n^{th}$ row ⇒

$\sum_{k=n+1}^{k=2n} \sum_{j=1}^{j=n} y_{r,j;k}^{m} = 1$ for r =1, 2…n, m=1, 2…... n^2

……

place number m once in the $(n^2-n)^{th}$ row to n^{2th} row ⇒

$\sum_{k=n\text{^}2-n}^{k=n\text{^}2} \sum_{j=1}^{j=n} y_{r,j;k}^{m} = 1$ for r =1, 2…n, m=1, 2…... n^2

To make sure each number m appears exactly once in each column, n^4 constraints:

To place number m once in the first n columns ⇒

$\sum_{k\in\{1,n+1,2n+1\ldots n(n-1)+1\}} \sum_{j=1}^{j=n} y_{r,j;k}^{m} = 1$ for r =1, 2…n, m=1, 2…... n^2

To place number m once in the first $(n+1)^{th}$ column to $2n^{th}$ column⇒

$\sum_{k\in\{2,n+2,2n+2\ldots n(n-1)+2\}} \sum_{j=1}^{j=n} y_{r,j;k}^{m} = 1$ for r =1, 2…n, m=1, 2…... n^2

……

To place number m once in the first $(n^2-n)^{th}$ column to n^{2th} column⇒

$\sum_{k\in\{n,2n,3n\ldots n\text{^}2\}} \sum_{j=1}^{j=n} y_{r,j;k}^{m} = 1$ for r =1, 2…n, m=1, 2…... n^2

To make sure each submatrix has only one number m, n^4 constraints:

$\sum_{r=1}^{r=n} \sum_{j=1}^{j=n} y_{r,j;k}^{m} = 1$ for k =1, 2…n^2, m=1, 2…...n^2

To make sure each position in the matrix has a number in it, n^4 constraints:

$\sum_{m=1}^{m=n} y_{r,j;k}^{m} = 1$ for k=1, 2…n^2, r, j=1,2…n

Formula:

$\sum_{k=1}^{k=n}\sum_{j=1}^{j=n} y_{r,j;k}^{m} = 1$ for r =1, 2…n, m=1, 2…...n^2

$\sum_{k=n+1}^{k=2n}\sum_{j=1}^{j=n} y_{r,j;k}^{m} = 1$ for r =1, 2…n, m=1, 2…... n^2

……

$\sum_{k=n\hat{}2-n}^{k=n\hat{}2}\sum_{j=1}^{j=n} y_{r,j;k}^{m} = 1$ for r =1, 2…n, m=1, 2…... n^2

$\sum_{k\in\{1,n+1,2n+1\ldots n(n-1)+1\}}\sum_{j=1}^{j=n} y_{r,j;k}^{m} = 1$ for r =1, 2…n, m=1, 2…... n^2

$\sum_{k\in\{2,n+2,2n+2\ldots n(n-1)+2\}}\sum_{j=1}^{j=n} y_{r,j;k}^{m} = 1$ for r =1, 2…n, m=1, 2…... n^2

……

$\sum_{k\in\{n,2n,3n\ldots n\hat{}2\}}\sum_{j=1}^{j=n} y_{r,j;k}^{m} = 1$ for r =1, 2…n, m=1, 2…... n^2

$\sum_{r=1}^{r=n}\sum_{j=1}^{j=n} y_{r,j;k}^{m} = 1$ for k =1, 2…n^2, m=1, 2…...n^2

$\sum_{m=1}^{m=n} y_{r,j;k}^{m} = 1$ for k=1, 2…n^2, i, j=1,2…n

$y_{r,j;k}^{m} \in \{0, 1\}$