CSE6140 - Fall 2014
Computational Science & Engineering (CSE) Algorithms

Homework 4 – Solutions

1 Branch and Bound (15 points)

Devise a branch-and-bound algorithm for the Minimum SET COVER problem.

Given a set of elements $\mathcal{U} = \{u_1, u_2, \ldots, u_m\}$ and a set of subsets of $\mathcal{U}$, $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$, $S_i \subseteq \mathcal{U}, \forall i = 1, \ldots, n$, find a subset $\mathcal{S}'$ of $\mathcal{S}$ such that the union of the selected subsets covers all elements, $\bigcup_{S_i \in \mathcal{S}'} S_i = \mathcal{U}$ and the number of selected subsets, $|\mathcal{S}'|$, is minimized.

We will design a branch-and-bound method, where the partial solutions (2 pts) will be of the form $X_1, X_2, ..., X_n$, where for each subset up to i we decide whether S_i is inside ($X_i = 1$) or outside ($X_i = 0$) of the set cover $\mathcal{S}'$ or still undecided ($X_i = N/A$), starting with the partial solution with all undecided, $\forall i, X_i = N/A$.

Remark: *You should define your problem with suitable notation. Throughout the answer introduce variables and notations whenever needed.*

(a) What is a subproblem? (2 pts)
Answer. Given a partial solution $X_1, X_2, \ldots, X_n$ (as described above), consider the subsets that are selected for inclusion in the SET COVER according to this partial solution $\hat{S} = \{S_j | X_j = 1, j = 1, \ldots, n\}$ and the set of elements that are covered by subsets selected in the partial solution, i.e. $\hat{U} = \bigcup_{S_j \in \hat{S}} S_j$. The corresponding subproblem will be a smaller instance of the SET COVER problem with universe $\bar{U} = U \backslash \hat{U}$ adjusted by removing the elements that have been covered by at least one subset selected in our partial solution. We will in addition have a set of subsets derived from the original subsets for which no decision has been made yet, i.e. $\bar{\mathcal{S}} = \{S_j \backslash \hat{U} | X_j = N/A\}$.

Remark: *Subproblem should include both $\bar{\mathcal{S}}$ and $\bar{\mathcal{U}}$. Not only one of them.*

(b) How do you choose a subproblem to expand? (1 pts)
Answer. We can pick the subproblem in which the number of (remaining, uncovered) elements is minimal, i.e. with minimal $|\bar{U}|$. Alternative, you can choose from the queue using the lower bound on the objective value for the subproblems.

(c) How do you expand a subproblem? (1 pts)
Answer. We expand a subproblem by selecting all element i for which $X_i = N/A$ and derive from each two sub-problems corresponding to setting X_i to 0 and 1 respectively.

(d) What is an appropriate lower bound? (1 pts)
An appropriate lower bound for a partial solution X' is $L_{X'} = \sum_{j=1...i|X_j=1} 1$, i.e. the number of subsets used so far.

Remark: *The number of remaining uncovered elements is NOT a lower bound!*

PART 2: Outline a simple greedy heuristic for the SET COVER problem (2 pts), and explain why it finds a valid solution (1 pts) and its running time (2 pts).

Answer. We assume the SET COVER instance is feasible, i.e. there is some subset of subsets S_i that can cover all elements in the universe (otherwise we can quickly check this as a preprocessing step by taking the union of all subsets). The idea is that at each iteration we will choose to add to our SET COVER the subset that covers the biggest number of remaining uncovered elements from the universe.

Algorithm 1: GreedyMinSetCover

Input: a set of elements $\mathcal{U} = \{u_1, u_2, ..., u_m\}$, a set of subsets $\mathcal{S} = \{S_1, S_2, ..., S_n\}, S_i \subseteq \mathcal{U}, \forall i = 1...m$
Output: a feasible solution $\mathcal{S}' \subseteq \mathcal{S}$
$\mathcal{S}' = \emptyset$
$\mathcal{U}' = \mathcal{U}$
while $\mathcal{U}' \neq \emptyset$ **do**
  $S_{\text{add}} = \text{argmax}_{S_i \in \mathcal{S} \setminus \mathcal{S}'} |S_i \cap \mathcal{U}'|$
  $\mathcal{S}' = \mathcal{S}' \cup S_{\text{add}}$
  $\mathcal{U}' = \mathcal{U}' \setminus S_{\text{add}}$
end

Remark: *Algorithm should be presented clearly with suitable notation.*

Validity: The algorithm is valid because it does not terminate until all elements of the universe have been covered.

Remark: *For this part it suffices to show the algorithm returns a feasible solution. You don't need to find the approximation factor.*

Complexity: The time complexity is $O(m^2 n)$: at each iteration, in line 4 we do $O(mn)$ work to get the intersection of each remaining set (at most n such sets) with the remaining elements of the universe (at most m elements) (line 4); lines 5-6 require work linear in m and n. The loop is executed at most $min(n, m)$ times since at every iteration we cover at least one additional element from the universe (for a total of at most n such additions) and since at every iteration we add one more of the subsets (for a total of at most m such additions). The overall complexity is $O(\min(m, n)mn)$, or $O(m^2 n)$ when the universe size is smaller than the number of subsets we are given.

Remark: *The question doesn't ask you to find the optimal algorithm or the optimal way of implementing that greedy algorithm. A reasonable approach accompanied with a correct complexity computation is better than a fast algorithm with wrong complexity analysis.*

PART 3: Imagine you wanted to use a local search method to solve Minimum SET COVER such as Simulated Annealing or Iterated Local Search. Imagine a candidate solution is a subset of the sets that might or might not cover all elements in the Universe set $\mathcal{U}$.

- What could be a possible scoring function for such candidate solutions? (2 pts)
  Answer. Let $\mathcal{S}'$ be a candidate solution. A possible scoring function $g : 2^{\mathcal{S}} \to \mathbb{R}$ is:

$$g(\mathcal{S}') = \frac{|(\bigcup_{S_i \in \mathcal{S}'} S_i) \cap \mathcal{U}|}{|\mathcal{S}'|}$$

  In words, the score of a candidate solution is the ratio of the number of elements of the universe that its subsets cover, to the number of subsets it includes. Such a scoring function favors solutions that cover many elements with only a few subsets.

- What would be a Neighborhood (or Moves) you would consider using for your local search to move from one candidate solution to other 'nearby' solutions? How many potential neighbors can a candidate solution have under your Neighborhood (using Big-Oh)? (2 pts)
  Answer. One neighborhood we can adopt is that of the candidate solutions that differ by exactly one subset, in that we either add a new subset to our current solution, or we remove one of the subsets already included in our current solution. Since there are n subsets in total and each is either considered for addition or removal in relation to the current solution, the size of the Neighborhood is $O(n)$.

  Remark: *Note that this is a* **local** *search algorithm! Considering all* 2^n *possible neighbors is not a local neighborhood!*

- Why would you consider adding Tabu Memory and what would be remembered in your Tabu Memory? (1 pts)
  Answer. Tabu Memory is useful in avoiding local minima by forbidding repeated solutions. One implementation of Tabu Memory would remember the k last subsets that were added or removed in the latest k solutions. This forces the algorithm to explore "new" options by choosing solutions which include subsets that were not recently explored.

2 Approximation Algorithms–Truck Loading (12 pts)

(a) Provide an example of a set of weights for which your algorithm does *not* use the minimum possible number of trucks. State the number of trucks used by the algorithm for this example, and the true minimum number of trucks needed.

Answer (1 point): Suppose there are 3 containers with weights (in tons) $w_1 = 0.5, w_2 = 0.8, w_3 = 0.5$. The proposed algorithm uses 3 trucks, whereas these containers can actually be loaded onto two trucks.

(b) Let $W = \sum_{i=1}^{n} w_i$ be the total weight of all n containers. Let t^* be the minimum number of trucks you need to rent to transport the n containers. Provide a lower bound for t^*, and argue in support of your bound.

Answer (1 point): If we need to transport W tons of containers, we need trucks with total capacity at least W. Since each truck has a capacity of 1 ton, this means we need at least W trucks. Therefore, W is a lower bound for t^*.

(c) Suppose your greedy solution uses an even number of trucks, $t = 2z$ where z is a positive integer. You consider these $2z$ trucks in consecutive pairs. What is a lower bound for the weight of containers in each consecutive pair? Argue in support of your bound.

Answer (1 point): Consider the first pair of loaded trucks. We know that the total weight of containers loaded onto these two trucks must be strictly greater than 1 ton, because if it were less than 1 ton, the second truck would not be loaded.

(d) Using (b) and (c), prove an approximation ratio of 2 for your algorithm.

Answer (2 points): If we used z pairs of trucks, and the total weight of containers carried by each pair is strictly greater than 1 ton (from part (c)), then we loaded strictly greater than z tons of containers. In other words, $\frac{t}{2} = z < W$. From part (b) we have $W \leq t^*$. Together, this gives $\frac{t}{2} < t^*$ or $t < 2t^*$, which gives our approximation ratio of 2.

(e) Prove the same approximation ratio for the case when your greedy approach uses an odd number of trucks, $t = 2z + 1$.

Answer (2 points): We still know that the first $2z$ trucks must carry at least z tons of containers, so $z < W$. We also know that W is a lower bound for the optimal number of trucks needed, so $t^* \geq z + 1$. Together, this gives us $2t^* \geq 2z + 2 > 2z + 1 = t$.

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

Answer (1 point): Assume for the sake of contradiction there are two trucks loaded to less than half full using this algorithm. Consider the moment when the second such truck is loaded with a container for the first time. This container must have weight strictly less than 0.5 tons in order for this truck to be less than half full. Since we failed to fit this container on one of the previously loaded trucks, it must be that each of the previously loaded trucks is filled to a weight greater than 0.5 tons. However, this is a contradiction because we supposed that there were *two* trucks loaded to less than half full.

(g) Show that the number of trucks you would use with the second strategy is never more than $\lceil 2W \rceil$.

Answer (2 points): We can write $2W = z + r$ where z is an integer and $0 \leq r < 1$. There are two cases:

- if $r = 0$

  In this case, $\lceil 2W \rceil = 2W = z$. Suppose we actually use more than $\lceil 2W \rceil$ trucks, i.e. $t > z$. This means we use at least $z + 1$ trucks. Since there is at most 1 truck that is less than half full, we must have at least $z \times 0.5$ tons of containers, i.e. $W > z \times 0.5$ or $2W > z$. This is a contradiction.

- if $r > 0$

  Either $\lceil 2W \rceil = z + 1$ or $\lceil 2W \rceil = z + 2$. In the first case, suppose we have used more than $z + 1$ trucks, i.e. $t \geq z + 2$. At most 1 of these trucks is less than half full, so we have $W > (z + 1) \times 0.5$ or $2W > z + 1$, which is a contradiction with $\lceil 2W \rceil = z + 1$. In the second case, suppose we have used more than $z + 2$ trucks, i.e. $t \geq z + 3$. Again, at most 1 of these is less than half full, so that $W > (z + 2) * 0.5$ or $2W > z + 2$. This contradicts $\lceil 2W \rceil = z + 2$.

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

Answer (2 points):

From part (f): $t \leq \lceil 2W \rceil$.
From part (b): $\lceil W \rceil \leq t^*$, so $2\lceil W \rceil \leq 2t^*$
Lastly, $\lceil 2W \rceil \leq 2\lceil W \rceil$
So, $t \leq 2t^*$

3 Modeling with ILP (15 points)

a) 5 points

Independent set: Given an undirected graph $G = (V; E)$, find a maximum independent set, that is, a maximum subset of vertices in which no two vertices are adjacent.

We define a binary decision variable x_i corresponding to each vertex i. It is set to 1 if the vertex is in the independent set and 0 otherwise.

$$x_i = \begin{cases} 1 & \text{if vertex } i \text{ is selected, } i \in [1, \ldots, n] \\ 0 & \text{otherwise} \end{cases}$$

We want to maximize the number of vertices we select, i.e.:

$$\max \sum_{i=1}^{n} x_i \tag{1}$$

We are subject to the constraint that each selected vertex cannot be adjacent to another selected vertex; in other words, each edge can have at most one endpoint belonging to the independent set:

$$x_i + x_j \leq 1 \quad \forall (i,j) \in E \tag{2}$$

Lastly, we must ensure that x_i is binary:

$$x_i \in 0, 1 \tag{3}$$

- Number of variables = $|V|$
- Number of constraints = $|E| + |V|$

b) 5 points

Facility location: We are given n potential facility locations and a list of m clients who need to be serviced from these locations. There is a fixed cost f_j of opening a facility at location j, while there is a cost c_{ij} of serving client i from facility j. The goal is to select a set of facility locations and assign each client to one facility, while minimizing the total cost.

We define the following binary decision variables:

$$x_{ij} = \begin{cases} 1 & \text{if client } i \text{ is served via facility } j,\ i \in [1, \ldots, m],\ j \in [1, \ldots, n] \\ 0 & \text{otherwise} \end{cases}$$

$$y_j = \begin{cases} 1 & \text{if facility } j \text{ is opened} \\ 0 & \text{otherwise} \end{cases}$$

We want to minimize the total cost, which includes the cost of servicing each client from the facility they are assigned to and the cost of opening the facilities.

$$\min \sum_{i=1}^{m} \sum_{j=1}^{n} x_{ij} c_{ij} + \sum_{j=1}^{n} y_j f_j \tag{4}$$

We need to ensure that each client is served by exactly one facility:

$$\sum_{j=1}^{n} x_{ij} = 1 \quad \forall i \in [1, \ldots, m] \tag{5}$$

If a client is assigned to a facility, then that facility must be open:

$$\sum_{i=1}^{m} x_{ij} \leq y_j * m \quad \forall j \in [1, \ldots, n] \tag{6}$$

Again, we need to ensure the decision variables are assigned binary values:

$$x_{ij} \in \{0,1\} \quad \forall i \in [1,\ldots,m],\ \forall j \in [1,\ldots,n] \tag{7}$$

$$y_j \in \{0,1\} \quad \forall j \in [1,\ldots,n] \tag{8}$$

- Number of variables $= mn + n$
- Number of constraints $= m + n + mn + n$

c) 5 points

The Sudoku problem: Given is an $n^2 \times n^2$ matrix (for some positive integer n), which contains some matrix entries pre-filled with integral values between 1 and n^2. The task is to fill the remaining entries with integers from $\{1, 2, ..., n^2\}$, such that each row, each column, and each of the n^2 major $n \times n$ submatrices contains each integer 1 through n^2 exactly once.

For $i = 1,\ldots,n^2$, and $j = 1,\ldots,n^2$ and $k = 1,\ldots,n^2$ we define the variable x_{ij}^k. If element at row i and column j of the matrix is set to k then $x_{ij}^k = 1$ and it is 0 otherwise.

First, note that there is no objective function to maximize or minimize. We can just set a "dummy" objective such as:

$$\max 1 \tag{9}$$

Next, each cell must be assigned exactly one value:

$$\sum_{k=1}^{n^2} x_{ij}^k = 1 \quad \forall i,j \in [1,\ldots,n^2] \tag{10}$$

Each row i must contain exactly one k:

$$\sum_{j=1}^{n^2} x_{ij}^k = 1 \quad \forall i,k \in [1,\ldots,n^2] \tag{11}$$

Each column j must contain exactly one k:

$$\sum_{i=1}^{n^2} x_{ij}^k = 1 \quad \forall j,k \in [1,\ldots,n^2] \tag{12}$$

Each submatrix (r,s) must contain exactly one k:

$$\sum_{i=(r-1)n+1}^{i=rn} \sum_{j=(s-1)n+1}^{j=sn} x_{ij}^k = 1 \quad \forall k \in [1,\ldots,n^2],\ r,s \in [1,\ldots,n] \tag{13}$$

Given pre-filled cells with values we represent as $p_{i,j}^k = 1$ if cell (i, j) is pre-filled with value k, 0 if it is not pre-filled:

$$x_{i,j}^k \geq p_{i,j}^k \quad \forall i, j, k \in [1, \ldots, n^2] \tag{14}$$

Lastly, make sure each decision variable gets a binary value:

$$x_{ij}^k = \{0, 1\} \quad \forall i, j, k \in [1, \ldots, n^2] \tag{15}$$

- Number of variables = n^6
- Number of constraints = $n^4 + n^4 + n^4 + n^4 + n^6 + n^6$