CSE 6140 - Homework 4

Alexander Winkles

Problem 1

1. We will first design a branch-and-bound algorithm for minimum set cover. Consider a universe of elements U and a collection of sets S such that $\bigcup_{x \in S} x = U$.

   (a) Our subproblem will be a subset of S that we check to see if $\bigcup_{x \in S} x = U$.

   (b) To chose a subproblem to expand, we pick an element of S (possibly based on size or arbitrarily) and remove it.

   (c) We expand our subproblem by removing one set $y \in S$ and checking if $\bigcup_{x \in S - \{y\}} x = U$.

   (d) An appropriate lower bound will be the size of S each step.

   With all of this in mind, our algorithm is this:

```
procedure BnB_SetCover($U, S$)
    $P \leftarrow \{S\}$                                  ▷ Assigns $S$ to active subproblems
    Sol $\leftarrow 0$                                     ▷ Assigns initial solution
    bestval $\leftarrow \infty$                            ▷ Assigns initial bound
    while $P \neq \emptyset$ do
        choose $T \in P$                                   ▷ Chooses a subproblem
        $P \leftarrow P - \{T\}$                           ▷ Removes subproblem from set
        for $i \in T$ do                                   ▷ Expands current subproblem into smaller ones
            $T_i = T - \{i\}$
        for $T_i$ do
            $K \leftarrow \bigcup_{x \in T_i} x$
            if $K = U$ & $|T_i| <$ bestval then            ▷ Checks if subproblem is a solution
                Sol $\leftarrow T_i$
                bestval $\leftarrow |T_i|$
            if $lowerbound(T_i) \leq$ bestval & $|T_i| \neq \emptyset$ then   ▷ Adds back to subproblem set
                $P \leftarrow P \cup T_i$
    return Sol
```

   I believe this algorithm will work well for minimum set cover problems, as it does a good job of pruning solutions that are not minimal.

2. Now we will design a greedy heuristic.

```
procedure APPROX_SETCOVER($U, S$)
    $P \leftarrow \{\}$
    for $s_i \in S$ do
        if $(x \in s_i) \notin P$ then
            $P \leftarrow P \cup \{s_i\}$
            break if statement
        if $\bigcup P = U$ then
            return $P$
    return NO SOLUTION
```

This approximation algorithm goes through every $s_i \in S$ and checks if it has an element that is not found in any subset within P. If a new element is found, s_i is added to P. This process is continued until either a solution is found or all s_i have been checked, in which case there is no solution. Because it searches through the entire set S, it is guaranteed to find a set cover (if one exists), though it is unlikely to be the minimum. Technically, there are two for loops nested within the **for** $s_i \in S$ loop. The first one will have a worst case of $\mathcal{O}(|s_i||P|)$ while the second will have a worst case of $\mathcal{O}(|P||U|)$. Suppose $|U| = n$, then this becomes $\mathcal{O}(n^2)$. Additionally, suppose that $|s_i| = n$ in a worst case scenario, then our overall time complexity will be $\mathcal{O}(n^3)$. Obviously this is not an ideal running time, but it is polynomial and guarantees a solution.

3. For a Local Search solution to this problem, a possible scoring function would be how many elements in the universe are found in a candidate solution. A neighborhood for this problem would be either adding a new subset, removing a new subset, or swapping a subset with one that is not in the candidate solution. This thus has $\mathcal{O}(n^3)$ neighbors, one n per each move as we could have a set cover with no subsets and thus have to add a subset, we could have all the subsets and attempt to remove one, or we could be somewhere in between and swap subsets. Using Tabu memory would be beneficial as it would avoid cycles and help our algorithm not be stuck in a local optima. The Tabu memory would remember subsets that have already been checked in the candidate solution, which would reduce repeats.

Problem 2

1. In this greedy strategy, we take the containers in order and add them to trucks. Consider $W = \{0.5, 1.0, 0.5\}$. By the algorithm provided, this will require 3 trucks, as the first truck will take 0.5, then since 1.0 will overload it, 1.0 will go to truck 2, which cannot fit any more, so the last 0.5 will go to truck three. However, the optimal number of trucks for this is 2, one that holds both 0.5 and the other that holds the 1.0.

2. Let $W = \sum_{i=1}^{n} w_i$ be the total weight of all the containers and let t^* be the minimum number of trucks required. Let t be the solution given by the greedy algorithm. We know that $W \leq t^*$, because otherwise we would need additional trucks to carry all of the containers.

3. Now suppose that the algorithm uses an even number of trucks consecutively, so $t = 2z$. Because there are two of them, the lowest bound on their weight is $\frac{1}{2}$, because if it was less then the containers could be fit on one

truck rather than two. Thus from this we see that $t \leq 2W$ using the fact that $t = 2z$.

4. Combing the above parts, we see that $t \leq 2W \leq 2t^*$, thus $t \leq 2t^*$, proving that the algorithm has an approximation ratio of 2.

5. Suppose now that there are an odd number of trucks, so $t = 2z+1$. By a similar logic from above, the minimum weight of $z \geq \frac{1}{2}$, for otherwise the weight could be consolidated on fewer trucks. Thus we find that $t \leq 2W+1$. Combining this with above results shows $t \leq 2W+1 \leq 2t^*+1$. Thus the algorithm is still a 2-approximation algorithm, with $\epsilon = 1$.

6. Suppose for contradiction that more than one truck is less than half full. This means that there are two trucks, called t_1, t_2 respectively, each with less than 0.5 tons of container weight. But by construction of the algorithm, the weight being placed in t_2 implies that the weight could not be placed in t_1 without overloading it. This implies that the weight in t_1 is greater than 0.5 tons, as the weight placed in t_2 is less than. This contradicts the assumption that t_1 is less than half full, which completes our proof.

7. The smallest number of containers a truck can hold is 1. Thus in a worst case scenario, ie each container has weight $w_i \in (0.5, 1]$, Consider each container to have the smallest of these weights. Then it will be the case that $t \approx 2W$, as each truck is missing about W weight in this scenario. If the containers all weigh more than that, the approximation slowly approaches $t \approx 1$. If the containers weigh less, then multiple containers can be places on individual trucks, which likewise reduces the approximation. Thus $t \leq \lceil 2W \rceil$ are required.

8. By the above statement, we know that $t \leq \lceil 2W \rceil$. Additionally we know that $W \leq t^*$. Combining these gives us $t \leq 2t^*$ as desired. x

Problem 3

1. For the independent set problem, assign each vertex a variable x_i, where $x_i \in \{0,1\}$. Then the integer linear programming solution is as follows:

$$\begin{aligned} \text{minimize} \quad & \sum x_i \\ \text{subject to} \quad & x_i + x_j \leq 1, (i,j) \in E \\ & x \in \{0,1\} \end{aligned}$$

   Thus, if there are n vertices and m edges, then there will be n variables and m constraints.

2. For this facility location problem, we can frame it as an integer linear programming problem as follows, where $y_i = 1$ represents a facility being opened and $x_{ij} = 1$ represents client j being assigned to facility i:

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{n} f_i y_i + \sum_{i=1}^{n} \sum_{j=1}^{m} c_{ij} x_{ij} \\ \text{subject to} \quad & \sum_{i=1}^{n} x_{ij} = 1 \text{ for } j = 1, 2, ..., m \\ & x_{ij} \leq y_i \text{ for } i = 1, ..., n; j = 1, ..., m \\ & x_{ij}, y_i \in \{0,1\} \end{aligned}$$

   This has m constrains and $mn + n$ variables.

3. We frame the sudoku problem as an integer linear programming problem as follows. Let $x_{ijk} \in \{0,1\}$ represent if the number k is on row i and column j of the $n^2 \times n^2$ grid. Then:

$$
\begin{aligned}
&\sum_{j=1}^{n^2} x_{ijk} = 1 \\
&\sum_{i=1}^{n^2} x_{ijk} = 1 \\
&\sum_{i=1}^{n} \sum_{j=1}^{n} x_{(i+U),(j+V),k} = 1 \\
&x_{ijk} \in \{0,1\} \\
&U, V \in \{0, n, 2n, ..., (n-1)(n)\}
\end{aligned}
$$

This uses $n^2 \times n^2$ variables and has three forms of constraints. The first line represents that each k can only appear in each row once, the second line represents that each k can only appear in each column once, and the third line represents that each k can only be found in each sub $n \times n$ grid once.