CS358: Applied Algorithms

Assignment 3: MIPs and ILPs (due 11/16/24)

Instructor: Sam McCauley

Instructions

All submissions are to be done through github. This process is detailed in the handout "Handing In Assignments" on the course website. Answers to the questions below should be submitted by editing this document. All places where you are expected to fill in solution are marked in comments with "FILL IN." You will also create one `.lp` file to fill in: `clerks.lp`. The assignment will not have a leaderboard or any automated testing.

Please contact me at srm2@williams.edu if you have any questions or find any problems with the materials.

This is an assignment (as defined on the syllabus). This means that **all work must be done alone.** Please do not discuss solutions with any other students, even at a high level. You should not look up answers, hints, or even code libraries on the internet, and you should not use AI to obtain answers. This assignment was designed to be completed with no external resources, beyond those explicitly linked in the midterm. (Class resources, such as slides, notes, the GLPK manual, and your previous assignment submissions, are of course acceptable, as are basic resources such as looking up debugging information.)

For this assignment, each question begins a new page. Make sure you answer all five questions!

Some problems may seem a bit harder than others. As always, if you're unsure of a 100% correct solution, make sure to write down the ideas you have for partial credit.

Problem 1 (20 points)**.** Let's generalize the router assignment problem we saw in class to the case with multiple routers.

There are n roommates living in a single-floor house; roommate i is located at coordinates (x_i, y_i). You want to place k routers in the house. Let the distance between roommate i and a router at (x, y) be defined as $|x_i - x| + |y_i - y|$.

You want to place the routers under the following constraints:

- Each roommate must be assigned to exactly one router
- The router i assigned to a roommate r must satisfy that the distance between router i and roommate r is at most 10.

Your goal is to minimize the sum, over all roommates, of the distance between the roommate and the router they are assigned to.

To be clear, you receive as input n, k, and the coordinates of all n roommates: $(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)$.

Give an ILP or MIP for this problem and **prove that it is correct**. You do not need to bound its size and construction time, but they should be polynomial in n and k.

Solution.

Problem 2 (20 points)**.** The following is a fairly classic bin packing problem. Let's solve it using a branch and bound approach. This question will not have any LP, ILP, or MIP in it—it's just about how branch and bound itself works.[1]

Problem Statement: You have n items with weights $W = w_1, \ldots, w_n$. You have m bins with capacities $C = c_1, \ldots c_m$; these are given in *decreasing* order. Your job is to assign items to bins, with the restriction that the total weight of all items assigned to bin i is at most c_i.

The goal is to minimize the number of bins that have any item assigned to them. (To rephrase in case it helps with clarity: your goal is to maximize the number of empty bins with no item assigned.)

You may assume that, without loss of generality, a solution that uses k bins uses bins $1, \ldots k$ (you don't want to "skip over" bins to use a later bin—that wouldn't make sense since the later bins are smaller).

A Recursive Approach: The following recursive algorithm solves this problem optimally by considering every possible solution (recall that $V \setminus \{v\}$ means the set resulting when removing v from V):

```
1  BinPacking(V, i, w):
2      if |V| = 0 or i > m then:
3          return i
4      bestSoFar ← m + 1
5      for all v ∈ V:
6          if w + w_v ≤ c_i then:
7              bestSoFar ← min{bestSoFar, BinPacking(V \ {v}, i, w + w_v)}
8      bestSoFar ← min{bestSoFar, BinPacking(V, i + 1, 0)}
9      return bestSoFar
```

For any set of items V, any $i \in \{1, \ldots m\}$, and any w, BinPacking(V, i, w) returns the smallest j such that all items in V can be stored in bins $\{i, i+1, \ldots, j\}$ (where bin i already has weight w of items stored in it); if no such j exists then BinPacking returns $m + 1$.[2]

Therefore, we can solve this problem by calling BinPacking$(W, 1, 0)$.

Developing Branch and Bound: The remainder of this question is in two parts.

(a) Give a *simple* method LowerBound(V, i, w) that gives a lower bound on the number of bins necessary from $i, i+1, \ldots, m$ in order to store all items from V given that bin i already has w worth of items in it. Your method should be efficient and nontrivial; in particular, it should satisfy the following constraints:

- (Efficient): LowerBound(V, i, w) should run in at most $O(|V| \log |V|)$ time; of course this means that $O(|V|)$ time is also OK. (The time should be $O(1)$ if $|V| = 0$.)

[1] That said—yes, in general one could absolutely solve this with an ILP.

[2] I won't include the proof of correctness—you may take as given that this is true.

- (Nontrivial):[3] The value of LowerBound(V, i, w) should depend in some way on V.

To be clear: I am not looking for a specific answer here; there are many possible solutions for LowerBound(V, i, w) that will receive full credit.

Explain your solution: why is it a lower bound? (A brief explanation is sufficient; a proof is not necessary.)

Solution.

(b) Use LowerBound(V, i, w) to create an algorithm BinPackingWithBounds that avoids some of the recursive subcalls made by BinPacking. Your algorithm should be a modification of the BinPacking algorithm (to this end, the code for BinPacking is included below; modify it to obtain your solution).

Prove that your BinPackingWithBounds algorithm will give the correct answer. (You may assume what was already mentioned above: that LowerBound correctly lower bounds the subproblem solution, and that BinPacking is correct.)

Hint: There must be some optimal solution to each recursive BinPackingWithBounds call. Prove that your algorithm will always make a recursive call containing this solution. The remainder of the proof follows from this.

Solution.

```
1  BinPacking(V, i, w):
2      if |V| = 0 or i > m then:
3          return i
4      bestSoFar ← m + 1
5      for all v ∈ V:
6          if w + w_v ≤ c_i then:
7              bestSoFar ← min{bestSoFar, BinPacking(V \ {v}, i, w + w_v)}
8      bestSoFar ← min{bestSoFar, BinPacking(V, i + 1, 0)}
9      return bestSoFar
```

[3] This is to rule out trivial solutions like "LowerBound$(V, i, w) = 1$ for all V, i, w" or "LowerBound$(V, i, w) = i$ for all V, i, w"—these solutions are lower bounds, but they will not help in part *b*.

Problem 3 (20 points)**.** A bank is open from 9:00 to 15:00. During each hour of the day, the number of clerks required is shown in the following table:

Time Period	No. of Clerks
9:00–10:00	4
10:00–11:00	3
11:00–12:00	4
12:00–13:00	6
13:00–14:00	5
14:00–15:00	6

The bank can hire full-time and part-time clerks. Full-time clerks work from 9:00 to 15:00 except for a one-hour lunch break, which is either from 12:00–13:00 or from 13:00–14:00 (the bank decides the time at which each clerk takes their lunch break). The clerks are paid \$8 per hour; they receive payment for their lunch break. Part-time clerks work for three consecutive hours and the bank specifies the start time for each of them. Part-time clerks are paid \$6 per hour. No more than five part time clerks can be hired. The goal is to minimize the total cost of the clerks hired.

Solve this problem using GLPK, in the file `clerks.lp`. Write below: how many full-time employees are hired in your solution, and what are their lunch breaks? **You do not need to write an ILP/MIP here; just the .lp file and the full time employees written below is sufficient.**

Solution.

Problem 4 (20 points)**.** The following is a variant of the Travelling Salesman Problem.

Let's say you have a collection of n points; for each pair of points i and j there is a cost c_{ij} to travel from i to j.

In addition, k *target points* are identified (they are a subset of the n points above).

Your goal is to find the shortest cycle that visits all k of the target points. Your cycle may also visit points that are not target points—the only requirement is that at least the k target points *must* be visited. To be clear, the solution must be a "cycle": the cycle must start and stop in the same place, and the cycle cannot visit any point more than once. (In particular, each of the target points must be visited *exactly* once; each of the $n - k$ points that are not target points can be visited *at most* once.)

The input consists of: n, k, the n^2 values for c_{ij}, and a list of k target points.

Give an MIP or ILP for this problem. Its size and construction time should be polynomial in n. **You do not need to prove that it is correct or bound the size or construction time** (though the proof may be a good exercise to verify your solution).

Solution.

Problem 5 (20 points)**.** You are given a set of n strings, each of length m.

Your goal is to find the longest string s such that s is a subsequence of all n input strings. That is to say: for every input string i, the characters of s should all appear in i (in order, but not necessarily consecutively).

For example, if the input is:

```
tomorrownamaste
tomatosteerings
antimasterheads
tastetastetaste
```

Then the solution is `taste`. (The red highlighted characters in the below can be used to verify that this is, in fact, a solution; I believe there is no longer solution.)

```
tomorrownamaste
tomatosteerings
antimasterheads
tastetastetaste
```

Give an MIP or ILP to solve this problem; its size and construction time should be polynomial in n and m. **You do not need to prove that it is correct or bound the size or construction time** (though the proof may be a good exercise to verify your solution).

Hint: if the solution has length k, then there are exactly k characters from each input string that are matched to the solution string (here I'm referring to the red characters in the above). This may simplify your objective function.

Solution.