CPSC 320 2024W1: Assignment 1

Hansel Poe

September 20, 2024

This assignment is due **Friday, September 20 at 7 PM**. Late submissions will not be accepted. Please follow these guidelines:

- Prepare your solution using LaTeXand submit a pdf file. Easiest will be to submit using the .tex file provided. For questions where you need to select a circle, you can simply change `\fillinMCmath` to `\fillinMCmathsoln` .
- Enclose each paragraph of your solution with `\begin{soln}Your solution here...\end{soln}`. Your solution will then appear in dark blue, making it a lot easier for TAs to find what you wrote.
- Submit the assignment via GradeScope at `https://gradescope.ca`. Invitations to the class Gradescope page will be sent sometime in the week before the assignment deadline. Your group must make a **single** submission via one group member's account, marking all other group members in that submission **using GradeScope's interface**.
- After uploading to Gradescope, link each question with the page of your pdf containing your solution. For instructions on this, see `https://canvas.ubc.ca/courses/153809/pages/how-to-submit-assignments-on-gradescope`.

Before we begin, a few notes on pseudocode throughout CPSC 320: Your pseudocode should communicate your algorithm clearly, concisely, correctly, and without irrelevant detail. Reasonable use of plain English is fine in such pseudocode. You should envision your audience as a capable CPSC 320 student unfamiliar with the problem you are solving. You may **neither** include what we consider to be irrelevant coding details **nor** assume that we understand the particular language you chose. (So, for example, do not write `#include <iostream>` at the start of your pseudocode, and avoid language-specific notation like C/C++/Java's ternary (question-mark-colon) operator.)

Remember also to **justify/explain your answers**. We understand that gauging how much justification to provide can be tricky. Inevitably, judgment is applied by both student and grader as to how much is enough, or too much, and it's frustrating for all concerned when judgments don't align. Justifications/explanations need not be long or formal, but should be clear and specific (referring back to lines of pseudocode, for example). Proofs should be a bit more formal.

On the plus side, if you choose an incorrect answer when selecting an option but your reasoning shows partial understanding, you might get more marks than if no justification is provided. And the effort you expend in writing down the justification will hopefully help you gain deeper understanding and may well help you converge to the right selection :).

Ok, time to get started...

Group Members

Please list the CWLs of all group members here (even if you are submitting by yourself). We will deduct a mark if this is incorrect or missing.

- hpoe01

1 Statement on collaboration and use of resources

To develop good practices in doing homeworks, citing resources and acknowledging input from others, please complete the following. To shade a bubble below, replace the LaTeX command `\fillinMCmath` by `\fillinMCmathsoln`. This question is worth 2 marks.

1. All group members have read and followed the guidelines for groupwork on assignments given on the syllabus (see `https://canvas.ubc.ca/courses/153809/assignments/syllabus`, under Academic Conduct -> Assignments).

   [✓ Yes] No

2. We used the following resources (list books, online sources, etc. that you consulted):

3. One or more of us consulted with course staff during office hours.

   [✓ Yes] No

4. One or more of us collaborated with other CPSC 320 students; none of us took written notes during our consultations and we took at least a half-hour break afterwards.

   [✓ Yes] No

   If yes, please list their name(s) here:

   - Jessica Patricia

5. One or more of us collaborated with or consulted others outside of CPSC 320; none of us took written notes during our consultations and we took at least a half-hour break afterwards.

   Yes [✓ No]

   If yes, please list their name(s) here:

2 SMP Extreme True or False

Each of the following problems concerns a SMP scenario, and a statement about that scenario. Recall that an instance of size n of the Stable Matching Problem (SMP) has n employers and n applicants, and can be specified using $2n$ preference (or ranking) lists — one for each employer and one for each applicant.

Each statement may be **always** true, **sometimes** true, or **never** true. Select the best of these three choices and then:

- If the statement is **always** true, (a) give, and very briefly explain, an example instance in which it is true and (b) prove that it is always true.
- If the statement is **never** true, (a) give, and very briefly explain, an example instance in which it is false and (b) prove that it is always false.
- If the statement is **sometimes** true, (a) give, and very briefly explain, an example in which it is true and (b) give and very briefly explain an example instance in which it is false.

Here are the problems:

1. [2 points] Let I be any instance of SMP in which a_j is the lowest ranked applicant of employer e_i.

   Statement: If e_i and a_j are paired in *some* stable matching for I, then all stable matchings for I must have e_i and a_j paired. [Note: This statement is about *all* stable matchings, not just those produced by any specific algorithm such as the Gale-Shapley algorithm.]

   Sometimes true Consider the case where we have only one applicant and employer who prefers each other. They are each other's only and lowest ranked preference. Thus, there is only one stable matching and that matching is where e_i is matched with a_j. On the other hand, we may have a case where $n = 2$ with preference lists as follows:

   $$\begin{array}{ll} e_1 : a_2\, a_1 & a_1 : e_1\, e_2 \\ e_2 : a_1\, a_2 & a_2 : e_2\, e_1 \end{array}$$

   In this case, we have two stable matchings: $\{(e_1, a_1), (e_2, a_2)\}$ and $\{(e_1, a_2), (e_2, a_1)\}$. The first one has e_1 paired with its lowest ranked applicant, a_1. However, the second matching doesn't have the two paired.

2. [2 points] Let I be an instance of SMP, where the Gale-Shapley algorithm produces a stable matching for I in which all employers AND all applicants get their top-ranked choice.

   Statement: If applicant a_j is the top choice of employer e_i in instance I, employer e_i is also the top choice of applicant a_j in instance I.

   Always true From our assumption, we must have that for each pair (e_i, a_j) in our matching, a_j is e_i's most preferred candidate and vice versa. This is sufficient for our condition.

3 Counting Matchings

1. [2 points] How many different SMP instances of size n are there in total? Check one, and provide a short justification of your answer.

   ○ $n \times n!$ ○ $(n!)^n$
   ○ $2n \times n!$ [✓ $(n!)^{2n}$]

   Each applicant or employer has $n!$ possible preference list (each list is just a permutation of either applicants or employers). We have $2n$ total applicants and employers. Thus, we have $(n!)^{2n}$ different SMP instances

2. [2 points] Of the total number of different SMP instances of size n, how many are such that e_i is the top choice of a_i, and a_i is the top choice of e_i, for all i from 1 to n? Check one, and provide a short justification of your answer.

   ○ $(n-1)!$ ○ $((n-1)!)^n$
   ○ $2n \times (n-1)!$ [✓ $((n-1)!)^{2n}$]

   For each employer e_i, we have $1 \cdot (n-1) \cdot (n-2) \cdot \ldots 1 = (n-1)!$ possible number of preference lists, note that the top rank is reserved for a_i. Applicants also have the same possible number of preference lists (For as given a_i, their top rank is reserved for e_i). In total, there are $2n$ employers and applicants, so we have $(n-1)!^{2n}$ possible SMP instances.

4 Tour Planning

You're organizing a back-to-school social tour for CS students. Because the major is extremely popular, the students will be partitioned into three groups. You want to determine whether there's a *good solution*, where no student knows anyone else in their group (other than themselves), so as to maximize the opportunity for people to meet new people. Moreover, each of the three groups should be non-empty, although groups need not be of the same size. We'll call this the TG (Tour Grouping) problem.

You have at your disposal a handy matrix $K[1..n][1..n]$, where $n \geq 3$ is the number of students. Entry $K[i, j]$ is 1 if student i knows student j, and is 0 otherwise. The matrix is symmetric, i.e., $K[i, j] = K[j, i]$. (Entry $K[i, i]$ is always 1, for $1 \leq i \leq n$.)

For example, if the number n of students is eight, and the matrix K is as on the left below, then you can put students 1, 2, and 6 in one group, students 3 and 4 in another group, and 5, 7, and 8 in the third. So for this instance of the problem, the answer is Yes (there is a good solution). However, if all entries in the matrix K are equal to 1, then there clearly is no good solution so the answer is No.

	1	2	3	4	5	6	7	8
1	1	0	0	1	1	0	0	0
2	0	1	1	1	0	0	1	0
3	0	1	1	0	1	1	1	1
4	1	1	0	1	0	0	1	1
5	1	0	1	0	1	0	0	0
6	0	0	1	0	0	1	1	0
7	0	1	1	1	0	1	1	0
8	0	0	1	1	0	0	0	1

	1	2	3	4	5	6	7	8
1	1	1	1	1	1	1	1	1
2	1	1	0	0	0	0	0	0
3	1	0	1	0	0	0	0	0
4	1	0	0	1	0	0	0	0
5	1	0	0	0	1	0	0	0
6	1	0	0	0	0	1	0	0
7	1	0	0	0	0	0	1	0
8	1	0	0	0	0	0	0	1

Follow the steps below to show how to reduce the TG problem to the SAT problem. That is, given an instance $K[1..n][1..n]$ of the TG problem, construct an instance I of SAT, such that instance K is a Yes-instance of TG if and only if I is a Yes-instance of SAT.

Your reduction will have three variables, $x_{i,1}$, $x_{i,2}$, and $x_{i,3}$ for each student i. The rough idea is that if there is a satisying assignment to I in which $x_{i,1}$ is true, then student i is in group 1; if $x_{i,2}$ is true, then student i is in group 2; and if $x_{i,3}$ is true, then student i is in group 3.

1. [2 points] For the example matrix K given above on the right, where again there are eight students, give one good solution to the TG problem.

   $G_1 = \{1\}, G_2 = \{2, 3, 4\}, G_3 = \{5, 6, 7, 8\}$ where G_i is group i of students

2. [2 points] Now, on to the reduction to SAT. If $K[i, j] = 1$ (student i knows student j) then we must ensure that group 1 does not contain both student i and student j. Give a clause to ensure this.

   $(\bar{x}_{i,1} \vee \bar{x}_{j,1})$

3. [2 points] Similarly, if $K[i, j] = 1$ (student i knows student j) then we must ensure that group 2 does not contain both students, and group 3 does not contain both students. Give clauses to ensure this.

   $(\bar{x}_{i,2} \vee \bar{x}_{j,2}) \wedge (\bar{x}_{i,3} \vee \bar{x}_{j,3})$

4. [2 points] We also don't want student i to be in two groups simultaneously. Give clauses to ensure this.

$$(\bar{x}_{i,1} \vee \bar{x}_{i,2}) \wedge (\bar{x}_{i,1} \vee \bar{x}_{i,3}) \wedge (\bar{x}_{i,2} \vee \bar{x}_{i,3})$$

5. [2 points] We must ensure that each group is non-empty. Give clauses to ensure this.

$$(x_{1,1} \vee x_{2,1} \vee \cdots \vee x_{n,1}) \wedge (x_{1,2} \vee x_{2,2} \vee \cdots \vee x_{n,2}) \wedge (x_{1,3} \vee x_{2,3} \vee \cdots \vee x_{n,3})$$

6. [2 points] Let $c(n)$ be the total number of clauses specified in parts 2 to 5 of this question *in the worst case.* (Consider, for example, how many clauses there are when all entries in matrix K are 1.) Choose all that apply. (No justification needed.)

○ $c(n) = O(n)$	○ $c(n) = \Theta(n)$	● $c(n) = \Omega(n)$
● $c(n) = O(n^2)$	● $c(n) = \Theta(n^2)$	● $c(n) = \Omega(n^2)$

7. [2 points] Are any additional constraints needed? If so, describe the constraints and corresponding clauses, and otherwise simply indicate that no further clauses are needed. No

8. [3 points] Let I be the SAT instance that is the conjunction of all of the clauses from your previous parts (and no other clause). Show that if K is a Yes-instance of TG then I is satisfiable.

First, we want to prove that the clauses $(\bar{x}_{i,l} \vee \bar{x}_{j,l})$ for all $l = 1, 2, 3$ evaluate to true for any student i and j that knows each other. From our assumption, we know that for any student i and j, such that $K[i, j] = 1$, they are not in the same group. For any group l, this implies two cases:

- Neither i and j are in the group, this will produce true for all variables in the clause
- One of i or j is not the group, this will produce true for one variable in the clause

In both cases, the clauses evaluate to true

Next, we want to show that the clauses $(\bar{x}_{i,1} \vee \bar{x}_{i,2}) \wedge (\bar{x}_{i,1} \vee \bar{x}_{i,3}) \wedge (\bar{x}_{i,2} \vee \bar{x}_{i,3})$ evaluate to true for any i. Our K is a yes instance, thus it must produce partitions that are disjoint. Each student i cannot be in more than one group. There are three possible cases, let us take a look at one:

- i is in group one and not any other. Variables $\bar{x}_{i,1}$ is false, but $\bar{x}_{i,2}$ and $\bar{x}_{i,3}$ are both true. Each clause is a disjunction and contains either $\bar{x}_{i,2}$ or $\bar{x}_{i,3}$ so all clauses must evaluate to true

The same logic can be applied to the two other cases where i is in group two and three.

Finally, we want to show that $(x_{1,1} \vee x_{2,1} \vee \cdots \vee x_{n,1}) \wedge (x_{1,2} \vee x_{2,2} \vee \cdots \vee x_{n,2}) \wedge (x_{1,3} \vee x_{2,3} \vee \cdots \vee x_{n,3})$ is true. Now our yes K instance guarantees that not a single group is empty. This means that for any $l = 1, 2, 3$ in $(x_{1,1} \vee x_{2,1} \vee \cdots \vee x_{n,1})$ there exists at least one i such that the variable $x_{i,l}$ is true, thus evaluating the whole disjunction as true

All clauses have been proven to be true, therefore I is satisfiable $Q.E.D$

9. [3 points] Now show that if I is satisfiable then K is a Yes-instance of TG.

   First, we will show that any two students i and j who knows each other are not in the same group. Because I is satisfiable, then the clauses $(\bar{x}_{i,l} \vee \bar{x}_{j,l})$ for any $l = 1, 2, 3$ are true. This means that for any l, either none of $i or j$ is in the group or only one of them is in the group. In either case, $i and j$ are not in the same group.

   Next, we will prove that no student can be in more than one group. Because I is satisfiable, the clauses $(\bar{x}_{i,1} \vee \bar{x}_{i,2}) \wedge (\bar{x}_{i,1} \vee \bar{x}_{i,3}) \wedge (\bar{x}_{i,2} \vee \bar{x}_{i,3})$ must be true. Proof by contrapositive, assume that a student i is in two groups. We will divide into cases:

   - i is in group 1 and 2, then $(\bar{x}_{i,1} \vee \bar{x}_{i,2})$ will evaluate to false.
   - i is in group 1 and 3, then $(\bar{x}_{i,1} \vee \bar{x}_{i,3})$ will evaluate to false.
   - i is in group 2 and 3, then $(\bar{x}_{i,1} \vee \bar{x}_{i,3})$ will evaluate to false
   - i is in all three groups, any clause mentioned above will be false

   In all cases, our group of clauses is false

   Finally, we want to prove that not a single group can be empty. We know that for all $l = 1, 2, 3$ $(x_{1,l} \vee x_{2,l} \vee \cdots \vee x_{n,l})$ is true. This implies that at least one student i must be in the group. Thus no group can be empty.

   All conditions of Yes-K instance is satisfied. therefore K is a Yes-instance. $Q.E.D$

10. [4 points] Finally suppose that we remove the requirement that all groups must be non-empty. We'll refer to this variant of the original problem as TG'. Which of the clauses included above can be removed, while still ensuring that the reduction is correct? Remove as many as possible and justify your answer.

    $l = 1, 2, 3$ $(x_{1,l} \vee x_{2,l} \vee \cdots \vee x_{n,l})$

5 A Brute Force Approach to Tour Planning

Here is pseudocode for a brute force algorithm that either outputs a good solution to the TG problem described above, if one exists, or else outputs "no good solution". The algorithm generates all possible ways to group students into three groups, and checks whether any such grouping is a good solution. This brute force algorithm could use or adapt the subroutine called GENERATE-GROUPINGS to enumerate all possible ways to put the students into three groups, $G = (G_1, G_2, G_3)$.

```
function TOUR-PLANNING-BRUTE-FORCE($n$, $K[1..n][1..n]$)
    ▷ $n \geq 3$ is the number of students
    ▷ entries of $K$ are either 0 or 1, and $K$ is symmetric

    for each possible grouping $G = (G_1, G_2, G_3)$ of the students into three groups do
        ▷ assume that the GENERATE-GROUPINGS function below is adapted to enumerate the groupings
        check that each of the groups is non-empty
        check that any two people in the same group don't know each other
        if both of these checks pass then
            return the grouping $G$ (and halt)                    ▷ this is a good solution
        end if
    end for
    return "no good solution'
end function

function GENERATE-GROUPINGS($n$,$G[1..n]$,$i$)
    if $i = n + 1$ then return $G[1..n]$
    else
        $G[i] \leftarrow 1$; GENERATE-GROUPINGS($n$,$G[1..n]$,$i + 1$)
        $G[i] \leftarrow 2$; GENERATE-GROUPINGS($n$,$G[1..n]$,$i + 1$)
        $G[i] \leftarrow 3$; GENERATE-GROUPINGS($n$,$G[1..n]$,$i + 1$)
    end if
end function
```

Here, a grouping of the students is represented as an array $G[1..n]$, where each entry $G[i]$ is either 1, 2, or 3, indicating which group student i is in. For example, if $n = 8$ and $G[1..n]$ is [1,1,2,2,3,1,3,3], this represents the grouping discussed in our example above, with students 1, 2, and 6 in the first group, students 3 and 4 in the second, and 5, 7, and 8 in the third. When $n = 3$, the function call GENERATE-GROUPINGS(3, $G[1..3], 1$) outputs the arrays (representing groupings) in the order

$$[1,1,1], [1,1,2], [1,1,3], [1,2,1], [1,2,2], [1,2,3], [1,3,1], [1,3,2], [1,3,3], [2,1,1], \ldots$$

and so on. (The array G need not be initialized for this function call.) Note that while the groupings output by GENERATE-GROUPINGS ensure that each student is in exactly one group, some groups may be empty. That is, the groupings might not be a *partition* of the students into non-empty groups. So the brute force algorithm includes a check to ensure that if a good solution is output, all groups are indeed non-empty.

1. [2 points] As a function of n, how many groupings are generated by the GENERATE-GROUPINGS algorithm? Briefly explain your answer (one short sentence). Each student has 3 possible groups into which they can be assigned. there are n students. so there are 3^n total groupings

2. [3 points] Now suppose that we change the ordering of the lines to get an alternative algorithm (which also generates all possible groupings):

   function GENERATE-GROUPINGSS-MODIFIED(n, $G[1..n]$,i)

```
        if $i = n + 1$ then return $G[1..n]$
        else
            $G[i] \leftarrow 1$; GENERATE-GROUPINGS-MODIFIED($n$,$G[1..n]$,$i + 1$)
            $G[i] \leftarrow 3$; GENERATE-GROUPINGS-MODIFIED($n$,$G[1..n]$,$i + 1$)
            $G[i] \leftarrow 2$; GENERATE-GROUPINGS-MODIFIED($n$,$G[1..n]$,$i + 1$)
        end if
    end function
```

On the call GENERATE-GROUPINGS-MODIFIED(3,$G[1..3]$, 1), what is the ordering in which arrays are output? Give the first **seven** arrays in the ordering. No justification needed.

$[1, 1, 1], [1, 1, 3], [1, 1, 2], [1, 3, 1], [1, 3, 3], [1, 3, 2], [1, 2, 1]$

3. [4 points] Write pseudocode to check whether a particular grouping satisfies the second check of the TOUR-PLANNING-BRUTE-FORCE function, i.e., that for each group, check that no-one in the group knows anyone else in the group. The algorithm should return "Check fails" if in some group two students know each other, and otherwise should return "Check passes".

```
    function CHECK-WHO-KNOWS-WHO($n$, $G[1..n]$, $K[1..n][1..n]$)

        for each student $i$ in $G[1 \ldots n]$ do
            for each student $j$ in $G[i + 1 \ldots n]$ do
                if $G[i] = G[j]$ then
                    if $K[i, j] = 1$ then return Check fails
                    end if
                end if
            end for
        end for
    return Check passes

        end function
```

4. [2 points] Give a Θ bound on the worst case runtime of your algorithm from part 3, and provide a brief justification.

$\theta(n^2)$

The if statements perform the checking in constant time. This checking is done for each iteration of the inner for loop. Now, the inner for loop shrinks as i grows. In particular, we have : $(n-1), (n-2), \ldots, 0$ iterations as i goes from 1 to n. Using the summation formula, we have $n/2 \cdot (1 + (n - 1))$ total iterations,which is $\theta(n^2)$