CPSC 320 2024W1: Assignment 1 Solutions

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.]

   The statement is sometimes true.

   1. An example in which the statement is true, with $n = 2$, is given by the following preference lists. There is a unique stable matching, namely $(e_1, a_1), (e_2, a_2)$, and so all stable matchings have e_2 paired with its lowest-ranked applicant a_2.

e_1	a_1	a_2
e_2	a_1	a_2

a_1	e_1	e_2
a_2	e_2	e_1

   2. An example in which the statement is false, with $n = 2$, is given by the following preference lists. One stable matching has the pairs $(e_1, a_1), (e_2, a_2)$, in which the employers get their top choices. But another stable matching is $(e_1, a_2), (e_2, a_1)$, where the applicants get their top choices.

e_1	a_1	a_2
e_2	a_2	a_1

a_1	e_2	e_1
a_2	e_1	e_2

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.

   The statement is always true.

1. An example in which the statement is true, with $n = 2$, is given by the following preference lists where a_i is the top choice of e_i and vice versa.

e_1	a_1	a_2
e_2	a_2	a_1

a_1	e_1	e_2
a_2	e_2	e_1

2. To see why the statement must be true in general, note that if employer e_i has a_j as its top choice and (e_i, a_j) is in the stable matching produced by Gale-Shapley, and moreover a_j is matched with its top choice, then e_i *must* be a_j's top choice.

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.

   ○ $n \times n!$ ○ $(n!)^n$
   ○ $2n \times n!$ ● $(n!)^{2n}$

   To see why, note that there are $n!$ possible rankings for each employer or applicant. We multiply the possibilities per employer and applicant, of which there are $2n$ in total.

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.

   ○ $(n-1)!$ ○ $((n-1)!)^n$
   ○ $2n \times (n-1)!$ ● $((n-1)!)^{2n}$

   This is because the top choices of each employer and applicant are fixed, and so there are $(n-1)!$ possible rankings of the remaining applicants for each employer, and vice versa. We multiply the possibilities per employer and applicant, of which there are $2n$ in total.

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.

   Since student 1 knows all other students, student 1 must be in their own group with no other students. We can group the remaining students any way we want as long as both groups are non-empty, e.g., put student 2 into a group of their own and put students 3, 4, 5, 6, 7 and 8 into the third group.

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.

   The following clause ensures that at least one of $x_{i,1}$ and $x_{j,1}$ is false:

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

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.

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

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.)

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

   The reason that there are $\Theta(n^2)$ clauses in the worst case is because of the clauses in part 3; there are two clauses for every pair of students who know each other and there are $\Theta(n^2)$ such pairs of students when every student knows every other student.

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.

   Yes. We need to ensure that each student i is in at least one group:

   $(x_{i,1} \vee x_{i,2} \vee x_{i,3})$

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.

   If K is a Yes-instance, then the students can be divided into three non-empty groups so that no student knows anyone in their group. For each student i and each g in the range 1,2, or 3, set $x_{i,g}$ to 1 if student i is in group g and set $x_{i,g}$ to false otherwise. This truth assignment satisfies all of the clauses and so satisfies SAT instance I.

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

   Suppose that there is a truth assignment satisfying instance I. Put student i in group g if and only if $x_{i,g}$ is set to True by this satisfying truth assignment.

   The clauses from part 4 ensure that each student is in at most one group, and the clauses from part 7 ensure that each student is in at least one group. So each student is in exactly one group. The clauses from parts 2 and 3 ensure that no pair of students who know each other (as specified by matrix K) are in the same group, and the clauses from part 5 ensure that all groups are non-empty. So the grouping that results from the truth assignment is a good solution to the TG problem specified by K, meaning that K is a Yes-instance.

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.

We can remove the clauses from part 5, and also from part 4. Let I' be the resulting SAT instance.

To show correctness, we first note that there is no change to the previous argument that if K is a Yes-instance, then there is a truth assignment that satisfies I'.

We also need to show that if I' is satisfiable then K is a Yes-instance of TG'. Suppose that there is a truth assignment satisfying instance I'. Put student i in group 1 if $x_{i,1}$ is set to True by this satisfying truth assignment; otherwise put the student in group 2 if $x_{i,1}$ is set to False and $x_{i,2}$ is set to True. Otherwise both $x_{i,1}$ and $x_{i,2}$ are false, and so by the clause in part 7, $x_{i,3}$ must be true, and we put the student in group 3. This ensures that each student is in exactly one group. Since the clauses from parts 2 and 3 remain, no pair of students in the same group know each other, and so we have a correct solution to TG'.

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, $G = (G_1, G_2, G_3)$.

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

4:     for each possible grouping G = (G1, G2, G3) of the students into three groups do
5:         ▷ assume that the GENERATE-GROUPINGS function below is adapted to enumerate the groupings
6:         check that each of the groups is non-empty
7:         check that any two people in the same group don't know each other
8:         if both of these checks pass then
9:             return the grouping G (and halt)                    ▷ this is a good solution
10:        end if
11:    end for
12:    return "no good solution'
13: end function
14:
15: function GENERATE-GROUPINGS(n,G[1..n],i)
16:    if i = n + 1 then return G[1..n]
17:    else
18:        G[i] ← 1; GENERATE-GROUPINGS(n,G[1..n],i + 1)
19:        G[i] ← 2; GENERATE-GROUPINGS(n,G[1..n],i + 1)
20:        G[i] ← 3; GENERATE-GROUPINGS(n,G[1..n],i + 1)
21:    end if
22: 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], ...

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).

   GENERATE-GROUPINGS produces 3^n groupings, since there are three possible groups that each of the n students can be placed in.

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] ← 1; GENERATE-GROUPINGS-MODIFIED(n,G[1..n],i + 1)
        G[i] ← 3; GENERATE-GROUPINGS-MODIFIED(n,G[1..n],i + 1)
        G[i] ← 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 i from 1 to n − 1 do
        for j from i + 1 to n do
            if G[i] == G[j] and K[i, j] == 1 then
                return "Check fails: Two students in one group know each other"
            end if
        end for
    end for
    return "Check passes: No two students in any group know each other"
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.

The worst-case runtime of the algorithm is $\Theta(n^2)$. The worst case arises when the check passes, in which case the nested **for** loop iterates $\Theta(n^2)$ time and each iteration takes $\Theta(1)$ time.