

Lecture Note 3

Stable Matching

draft of September 29, 2026, by [N. Young](#)

© 2018–2026 [Neal E. Young](#). Text: CC BY-NC-SA 4.0; code: MIT; except where noted. See <https://github.com/nealeyoung/algs101/blob/main/LICENSE>.

Contents

3.1	Problem definition	1
3.2	Algorithm definition (Gale-Shapley algorithm)	2
3.3	Bounding the worst-case run-time	3
3.4	Proving that the algorithm is correct	4
3.5	Advanced topic: hospital-optimality	6
3.6	External sources on Stable Matching	8
3.7	Exercises	8
3.7.1	Exercises with solutions	8
3.7.2	Exercises without solutions	10
3.8	Python code	12

We use the 4-step recipe for presenting an algorithm, as described in [Lecture Note 1](#). Below we mostly follow the definitions and analysis from [KT](#) Section 1.1.

3.1 Problem definition

The *Stable Matching* (historically known as *Stable Marriage*) problem is defined as follows.

input: - a positive integer n , implicitly defining n hospitals $(h_1, \dots, h_n)$ and n doctors $(d_1, \dots, d_n)$
 - $2n$ *preference lists*: for each hospital, a permutation of the doctors, and for each doctor, a permutation of the hospitals.

output: A *stable matching*, defined as follows.

A *matching* M is a collection of hospital/doctor pairs such that each hospital or doctor is in at most one pair in M . For each pair (h, d) in M , we say M *matches* hospital h and doctor d , and that h and d are *matched*. The matching M is *perfect* if M matches each hospital and doctor (i.e., each is in one pair in M , so $|M| = n$). In a perfect matching, a pair (h, d) is an *unstable pair* (also known as an *instability*) if h and d prefer each other to their matches in M . A *stable matching* is a perfect matching with no instabilities.

[To build intuition, consider examples. See Section 3.6 for pointers to textbooks that have motivation and examples.]

There can be more than one stable matching. For a given instance (input), there can be more than one stable matching. (For example, suppose each hospital has a different first choice. Then matching each hospital to his first choice is stable. Likewise, if each doctor has a different first choice, matching each doctor to her first choice is stable. Use these observations to build an example with $n = 2$ that has more than one stable matching. *Challenge:* show that, for every $n \geq 2$, there is an instance with at least $2^{n/2}$ distinct stable matchings.)

Problem variants. There are many possible variants of this problem. For example: Allow having more hospitals than doctors. Or allow incomplete preference lists (where a doctor prefers to be unmatched than matched to someone not on their list). What we discuss below extends cleanly to these variants. One can consider a variant without two different types (the “roommates problem”: given n people (where n is even), each with a preference list ordering all $n - 1$ other people, find a stable matching. ([KW slides](#) have an example for this variant that has no stable matching.)

Exercise: Find an algorithm that, given an instance and a perfect matching, outputs all the unstable pairs (if any). It should run in $O(n^2)$ (linear) time. (It helps to precompute arrays A and B such that $A[h][d]$ gives the rank of doctor d in h ’s preference list, while $B[d][h]$ gives the rank of h in d ’s preference list.)

1. Maintain a matching, initially empty.
2. While there is an unmatched hospital who has not made an offer to every doctor:
 - 3.1. Choose any such hospital h .
 - 3.2. Among doctors that hospital h has not previously made an offer to, let d be the one h prefers most.
 - 3.3. Hospital h makes an offer to doctor d , with the following outcome:
 - 3.4. If doctor d is unmatched, d tentatively accepts the offer, and h and d become matched.
 - 3.5. Else, if d prefers his or her current match, say h' , to h , then d rejects h , and stays matched to h' .
 - 3.6. Else, d tentatively accepts the offer: d rejects h' , and becomes matched to h instead of h' .
4. Finally, return the (possibly incomplete) matching formed by the currently matched pairs.

Figure 3.1: The Gale-Shapley (hospitals-propose) algorithm for Stable Matching.

3.2 Algorithm definition (Gale-Shapley algorithm)

First consider the brute-force algorithm: *Enumerate all $n!$ possible perfect matchings. For each, check if it is stable, and if so halt and return that matching.* This algorithm takes at least $\Omega(n!n^2)$ time, which is super-exponential, way too slow to be practical for large n . Considering that modern CPUs have GHz clocks—they do billions (10^9) of operations per second—this algorithm would currently require more than on the order of $n!n^2/10^9$ seconds, over a year already for $n \approx 17$.

Instead, Gale and Shapley proposed the following algorithm. It is conventionally described as an iterative process that maintains a matching. The matching is initially empty. In each iteration, the algorithm chooses any unmatched hospital. That hospital makes an offer to some doctor—the doctor that it prefers the most, among doctors that it has not previously made offers to. The doctor (tentatively) accepts the offer and becomes matched to that hospital if that new match is better for the doctor than the doctor's current match (or if the doctor is not yet matched). The process halts when every hospital is either matched or has made offers to all doctors. Pseudo-code is shown in Figure 3.1.

Next we consider the following questions:

1. Does this algorithm always terminate?
2. Does it always return a perfect matching?
3. Does it always return a stable matching? (That is, is it correct?)
4. How can we implement this algorithm efficiently? What run time can we achieve?

For intuition, trace the execution of the algorithm on this input:¹

hospitals	1st	2nd	3rd	4th	5th	doctors	1st	2nd	3rd	4th	5th
A	V	W	X	Y	Z	V	B	C	D	E	A
B	W	X	Y	V	Z	W	C	D	E	A	B
C	X	Y	V	W	Z	X	D	E	A	B	C
D	Y	V	X	W	Z	Y	E	A	B	C	D
E	V	X	W	Y	Z	Z	A	B	C	D	E

(continued)

¹source: KW slides

3.3 Bounding the worst-case run-time

Does the algorithm have to terminate? Why? We start by bounding the number of iterations.

Lemma 3.1 (e.g. [KT](#) p. 7 (1.3)). *In any execution of the algorithm, there are at most n^2 iterations.*

Proof (long form).

1. Consider an arbitrary execution of the algorithm.
2. In each iteration, one hospital makes an offer to one doctor.
3. By Lines 3.2 and 3.3 of the algorithm, no hospital makes an offer to the same doctor twice.
4. By two steps above, there are at most n^2 iterations (one for each hospital/doctor pair). □

(Question about long-form proofs: which of the four types of block does the above proof use?)

The algorithm can be implemented so that each iteration takes constant time. (How? Of course, we have to keep track of the current matching, and, for each hospital h , which doctor h should next make an offer to. This is easy to do. The main challenge is to somehow preprocess the input so that, given any doctor and two hospitals, one can check in $O(1)$ time which hospital the doctor prefers. To do this, precompute an array A such that $A[d][h]$ is the rank of h in d 's preference list. See, e.g., [KT](#) Section 2.3.) This and the lemma imply that the algorithm can be implemented to run in $O(n^2)$ time. Note that the input has size $\Theta(n^2)$. We conclude that the algorithm can be implemented to run in time *linear* in the input size:

Lemma 3.2. *The Gale-Shapley algorithm can be implemented to run in time linear in the input size.*

Python code: [stable_matching.py](#) (Section 3.8). It implements the Gale-Shapley algorithm and, for comparison, exhaustive search. Running it shows that Gale-Shapley is much faster than exhaustive search.

(continued)

3.4 Proving that the algorithm is correct

Recall that an algorithm is *correct* if, for every possible input, the algorithm returns a correct output (in this case, a stable matching). We prove this next. We start with some useful observations and a utility lemma. Consider any execution of the algorithm.

Observation 3.1 (KT p.7 (1.1)). *Consider any doctor d . Doctor d is not matched until the first iteration in which d receives an offer. After that d always remains matched, and in each iteration, d either keeps their current match, or rejects that match and becomes matched to a hospital that d prefers more.*

Observation 3.1 follows by inspection of Lines 3.4–3.6 of the algorithm.

Observation 3.2 (KT p.7 (1.2)). *Fix any hospital h . Let $d'_1, d'_2, \dots, d'_k$ be the doctors h makes offers to over the course of the execution, in the order that h makes offers to them. Then $d'_1, d'_2, \dots, d'_k$ is a prefix of h 's preference list. In other words, h prefers d'_1 to d'_2 , and d'_2 to d'_3 , and so on, and h prefers d'_k to each doctor that h does not make an offer to.*

Observation 3.2 follows by inspection of Line 3.2 of the algorithm.

We need one more utility lemma before we prove correctness.

Lemma 3.3 (KT p.8 (1.5)). *Any execution of the algorithm returns a perfect matching.*

Proof (long form).

1. Consider an arbitrary execution of the algorithm.
2. By Lemma 3.1, it terminates, so (by inspection of the algorithm) returns some matching M .
- 3.1. Suppose for contradiction that at least one hospital or doctor is unmatched in M .
- 3.2. M matches each hospital to at most one doctor, and vice versa.
- 3.3. Also, there are n hospitals and n doctors.
- 3.4. So, there must be both an unmatched hospital, say, h , and an unmatched doctors, say, d .
- 3.5. By Line 2 of the algorithm (the termination condition), h made an offer to d at some point.
- 3.6. So, by Observation 3.1, d remains matched at the end (so is matched in M).
- 3.7. This is a contradiction, since d is not matched in M (per Step 3.4 above).
4. By Block 3, each person is matched in M . That is, M is a perfect matching. □

(continued)

To finish we show that the perfect matching is stable.

Lemma 3.4 (correctness; [KT](#) p. 8 (1.6)). *Any execution of the algorithm returns a stable matching.*

Proof (long form).

1. Consider an arbitrary execution of the algorithm.
2. Let M be the perfect matching it returns (by Lemma 3.3).
 - 3.1. Consider any hospital/doctor pair (h, d) . We will show that (h, d) is not an unstable pair.
 - 3.2. Let doctor d' and hospital h' be their matches in M . (They exist, since M is perfect.)
 - 3.3. Note that $d' \neq d$ and $h' \neq h$ (otherwise h and d are a matched pair, so are not an unstable pair).
 - 3.4.1. Case 1. First consider the case that h never made an offer to d during the execution.
 - 3.4.2. Hospital h did make an offer to doctor d' , so, by Observation 3.2, h prefers d' over d .
 - 3.4.3. So (h, d) is not an unstable pair.
 - 3.5.1. Case 2. Otherwise h did make an offer to d during the execution.
 - 3.5.2. By alg. Lines 3.3–3.6, just after that, doctor d was matched to h or a hospital d prefers more.
 - 3.5.3. By Obs. 3.1, d prefers his/her final match h' at least as much, so (h, d) is not an unstable pair.
 - 3.6. Case 1 or Case 2 holds, so (h, d) is not an unstable pair.
4. By Block 3, there are no unstable pairs, so M is stable. □

We've proven that the algorithm runs in linear time and is correct.

(continued)

3.5 Advanced topic: hospital-optimality

We consider another interesting aspect. (See also [KT](#) p.9, “Extensions”.)

First we define the following terms. Fix any Stable Matching instance, say, I .

- Doctor d and hospital h are *feasible* for each other if there exists *any* stable matching (for the instance I) that matches the pair (h, d) . Otherwise we say d is infeasible for h (and vice versa).
- The *best* feasible doctor for a hospital h is the doctor, among the feasible doctors for h , whom h prefers most. Let $\text{best}(h)$ denote this doctor.
- A stable matching is *hospital-optimal* if it matches each hospital h to $\text{best}(h)$.

The only possible hospital-optimal matching is $M = \{(h, \text{best}(h)) : h \in \{1, \dots, n\}\}$. Note that a-priori, this M might not be a matching. (For example, it could be that two different hospitals have the same best feasible doctor d . No matching can match them both to d , so if this happens, there is no hospital-optimal matching.) Also, a-priori, the hospital-optimal matching might not be stable!

Next we show that, not only does the hospital-optimal matching exist for every instance, but the Gale-Shapley algorithm (in the hospitals-propose form) always returns it! The proof requires a careful use of induction. It takes some work to get the underlying intuition!

Consider any execution of the algorithm on any input. We first prove two utility lemmas.

Lemma 3.5. *Suppose a doctor d rejects some feasible hospital h in favor of another hospital h' (Line 3.5 or 3.6 of the algorithm). Then h' has a feasible doctor d' that h' prefers to d .*

Proof (long form).

1. Let d , h , and h' be as in the statement of the lemma.
2. Let M be a stable matching that matches h to d . (M exists by def'n of “feasible”.)
3. M is stable, so (h', d) is not an unstable pair for M .
4. Doctor d rejects h in favor of h' , so d prefers h' to h (algorithm Line 3.5 or 3.6).
5. But (h', d) is not an unstable pair for M , so hospital h' must prefer its match in M , say d' , to d .
6. So d' is a feasible doctor for h' that h' prefers to d . □

Lemma 3.6. *During the execution of the algorithm, each hospital is rejected only by its infeasible doctors.*

Proof (long form).

1. Suppose for contradiction that some hospital is rejected by one of its feasible doctors.
2. Let t be the *first* time that this happens. Let h and d be the hospital and doctor in question.
3. Let h' be the hospital in favor of which d rejects h , i.e., the hospital d is matched to just after time t . Note that $h' \neq h$.
4. By Lemma 3.5, h' has a feasible doctor, say d' , that h' prefers to d .
5. Hospital h' made an offer to d (at or before time t), but prefers d' more.
6. So (by Obs. 3.2), h' previously made an offer to d' .
7. So hospital h' must have been rejected by d' before making an offer to d .
8. This contradicts that t is the first time that a hospital was rejected by one of its feasible doctors. □

From Lemma 3.6, hospital-optimality of the final matching M follows easily.

Lemma 3.7 (hospital optimality; KT p. 10 (1.7)). *The matching output by the algorithm is hospital-optimal.*

Proof (long form).

1. Let M be the matching output by the algorithm.
- 2.1. Consider any hospital h . Let d be its matched doctor in M .
- 2.2. M is stable, so, by definition of feasible, d is feasible for h .
- 2.3. By Observation 3.2, h was rejected by every doctor that h prefers to d .
- 2.4. By this and Lemma 3.6, every doctor that h prefers to d is infeasible for h .
- 2.5. So, d , who is h 's doctor in M , is h 's most-preferred feasible doctor. That is, $d = \text{best}(h)$.
3. By Block 2, M matches each hospital h to $\text{best}(h)$. That is, M is hospital-optimal. □

We've proved that there is always a hospital-optimal matching, and the algorithm always outputs it. One consequence is that, for any instance, every possible execution of the algorithm yields the same matching. This was not obvious a-priori, as different executions can choose different hospitals in each iteration.

Exercise 3.5 is to show that the algorithm's matching is *doctor-pessimal*.

(continued)

3.6 External sources on Stable Matching

- [KT](#) Sections 1.1 and 2.3 (on implementation) and [KW slides](#) (for those chapters)
- [JE Chapter 4](#) Section 4.5; [LLM](#) (MIT Math for CS text) Section 6.4
- Lecture video (U. Vazirani)²
- Can the participants gain an advantage by misrepresenting their preferences?³

3.7 Exercises

3.7.1 Exercises with solutions

Exercise 3.1. (*Unstable pairs*) Consider this instance of Stable Matching with $n = 3$ (with hospitals X, Y, Z and doctors A, B, C).

$X:$	A	B	C
$Y:$	B	C	A
$Z:$	C	A	B

$A:$	Z	Y	X
$B:$	X	Y	Z
$C:$	X	Z	Y

List all pairs that are unstable for this input and matching $\{(X, B), (Y, C), (Z, A)\}$.

Solution. (Z, C) is the only unstable pair.

(Each of the hospitals get its second choice, so a pair (h, d) can be unstable only if doctor d is hospital h 's first choice. So the only potentially unstable pairs are (X, A) , (Y, B) , and (Z, C) . These are the pairs (h, d) where hospital h prefers doctor d to its matched doctor. Among these, pairs (X, A) and (Y, B) are not unstable, because A and B are matched to their first choices. But pair (Z, C) is unstable, because C is matched to his or her last choice.)

Exercise 3.2. (*Gale-Shapley practice*) Consider executing the Gale-Shapley algorithm (hospitals-propose, as described in the lecture notes) on the input from Exercise 3.1. Can there be more than three iterations before the algorithm halts?

Solution. No. Each hospital has a different first choice, so after the first three iterations, all hospitals are matched.

Exercise 3.3. (*Gale-Shapley practice. Similar to the example on Page 173 of JE Chapter 4.*) Show an execution of the Gale-Shapley (hospitals-propose) algorithm on the input from Exercise 3.1 modified so that the hospitals are A, B, C and the doctors are X, Y, Z .

Solution.

1. Hospital A makes an offer to doctor Z . Z accepts. Current matching is $\{(A, Z)\}$.
2. Hospital B makes an offer to doctor X . X accepts. Current matching is $\{(A, Z), (B, X)\}$.
3. Hospital C makes an offer to doctor X . X rejects C . Matching is unchanged.
4. Hospital C makes an offer to doctor Z . Z accepts C and rejects A . Current matching is $\{(B, X), (C, Z)\}$.
5. Hospital A makes an offer to doctor Y . Y accepts. Final matching is $\{(A, Y), (B, X), (C, Z)\}$.

Exercise 3.4. (*Practice with another algorithm*) In 1951, before Gale and Shapley proposed their algorithm, a different algorithm was in use. For instances with $n = 3$, that algorithm is shown in Figure 3.2. What is the output of that algorithm on the input from Exercise 3.1?

²https://archive.org/details/ucberkeley_webcast_8qhxIAWp-zk

³<https://math.stackexchange.com/questions/2775243>

1. For each pair (i, j) in $\langle (1, 1), (1, 2), (2, 1), (1, 3), (3, 1), (2, 2), (2, 3), (3, 2), (3, 3) \rangle$, do:
 - 1.1. Consider those hospital/doctor pairs (h, d) such that d is h 's i th choice, h is d 's j th choice, and neither h nor d are yet matched. For each such pair (h, d) , match h to d .
2. Return the matched pairs.

Figure 3.2: In 1951, before Gale and Shapley proposed their algorithm, a different algorithm was in use. For instances with $n = 3$, the algorithm is shown above.

Solution. First, here is the input in the form of a table of the ranks (i, j) for each hospital/doctor pair:

	A	B	C
X	(1, 3)	(2, 1)	(3, 1)
Y	(3, 2)	(1, 2)	(2, 3)
Z	(2, 1)	(3, 3)	(1, 2)

(For example, X ranks A first, while A ranks X third, so the entry for hospital X and doctor A is $(1, 3)$.)

In the first iteration, when $(i, j) = (1, 1)$, the algorithm doesn't match any pairs (because there are none with those ranks). In the second iteration, when $(i, j) = (1, 2)$, the algorithm matches Y to B and Z to C . In the third iteration, when $(i, j) = (2, 1)$, the algorithm doesn't match any pairs (because B and Z are already matched). In the fourth iteration, when $(i, j) = (1, 3)$, the algorithm matches X to A . After that, no more matches are made (because everybody is already matched).

The algorithm returns matching $\{(X, A), (Y, B), (Z, C)\}$. In this case, each hospital is matched to his first choice, so cannot be in any unstable pair. So there are no unstable pairs, and the matching is stable.

Exercise 3.5. (*prove Doctor-pessimality*) Section 3.5 shows that the Gale-Shapley algorithm (with hospitals proposing) outputs a hospital-optimal matching. In this exercise you will show that the matching is *doctor-pessimal*, defined as follows (in the terminology from Section 3.5). Fix any Stable Matching instance I . For any doctor d , let $\text{worst}(d)$ denote d 's least-preferred feasible hospital. A stable matching is *doctor-pessimal* if it matches each doctor d to $\text{worst}(d)$. (In this case the matching must be $\{(\text{worst}(d), d) : d \in \{1, 2, \dots, n\}\}$, but note that a-priori this set of n pairs might not be a matching, or might not be stable.)

Prove the following lemma. Hint: use hospital-optimality (Lemma 3.7).

Lemma 3.8 (doctor pessimality; KT p.11 (1.8)). *The matching output by the Gale-Shapley algorithm (hospitals-propose) is doctor-pessimal.*

Solution.

Proof (long form).

1. Assume for contradiction that some execution outputs a matching M that isn't doctor pessimal.
2. Let (h, d) be a matched pair in M such that $h \neq \text{worst}(d)$. Let $h' = \text{worst}(d)$.
3. Let H' be a stable matching that matches h' and d . (H' exists by definition of $h' = \text{worst}(d)$.)
4. By Lemma 3.7, M is hospital-optimal, so $d = \text{best}(h)$, and h prefers d to its matched doctor in H' .
5. Since H' is stable, d must prefer his or her match in H' (that is h') to h .
6. Both h and h' are feasible for d , so this contradicts that $h' = \text{worst}(d)$. □

Exercise 3.6. (*Implement Gale-Shapley*) Do the following Hacker-Rank challenge (implement the algorithm): <https://www.hackerrank.com/neaileyong-algs101-stable-matching>.

Solution. Python code: `stable_matching.py` (Section 3.8).

External exercises with solutions

- **KT:** Page 19, Solved Exercises 1 (reducing another problem to Stable Matching) and 2 (a variant of Stable Matching).

3.7.2 Exercises without solutions

Exercise 3.7. (*Stable matching definition*) Describe a specific instance I of Stable Matching with $n = 3$ that has only one stable matching. Describe the stable matching. Explain clearly why it is stable. Explain clearly why no other matching is.

Exercise 3.8. (*Another algorithm, proof problem 1*) Does the algorithm in Exercise 3.4 return a *perfect* matching for every input with $n = 3$? If so, give a long-form proof that it does. If not, give a counter-example: describe a particular Stable Matching instance with $n = 3$ and show that the algorithm does not give a perfect matching for that instance.

Exercise 3.9. (*Another algorithm, proof problem 2*) Consider the following lemma and proof about the algorithm in Exercise 3.4:

Lemma 3.9. *For any execution of this algorithm, if it returns a perfect matching M , then M is stable.*

Proof (long form).

1. Consider any execution of the algorithm that returns a perfect matching M .
 - 2.1. Consider any pair (h, d) .
 - 2.2.1. Suppose for contradiction that (h, d) is unstable for M .
 - 2.2.2. Let d' and h' be the respective matches of h and d in M (they exist, as M is perfect).
 - 2.2.3. Because (h, d) is an unstable pair, h ranks d above d' .
 - 2.2.4. So, by inspection of the alg. (Line 1), it considered the pair (h, d) before the pair (h, d') .
 - 2.2.5. So, when the algorithm considered the pair (h, d) , hospital h was not yet matched.
 - 2.2.6. Likewise, since (h, d) is unstable for M , doctor d ranks h above h' .
 - 2.2.7. So, by inspection of the alg. (Line 1), it considered the pair (h, d) before the pair (h', d) .
 - 2.2.8. So, when the algorithm considered the pair (h, d) , doctor d was not yet matched.
 - 2.2.9. By Steps 2.2.5 and 2.2.8, when the algorithm considered (h, d) , neither h nor d were matched.
 - 2.2.10. So, by inspection of the algorithm, it must have matched h to d in that iteration.
 - 2.2.11. But once a pair is matched, it stays matched, so (h, d) is in M .
 - 2.2.12. This is a contradiction, as no matched pair is unstable.
 - 2.3. By Block 2.2, (h, d) is not unstable for M .
 3. By Block 2, no pair is unstable for M . That is, M is stable. □

What is the first line in this proof that makes an assertion that is not necessarily true? Describe an example (an input and an execution of the algorithm on that input) for which the assertion fails.

Exercise 3.10. (*Another algorithm, proof problem 3*) Does the algorithm in Exercise 3.4 return a *stable* matching for every input with $n = 3$? If so, prove it. If not, give a counter-example: describe a particular Stable Matching instance with $n = 3$ and show that the algorithm does not give a stable matching for that instance.

Exercise 3.11. (*Gale-Shapley proof problem*) Prove or disprove the following assertion:

In any execution of the hospitals-propose Stable Matching algorithm, there is some doctor that does not receive an offer before the last iteration. (Note: we mean the algorithm described in the lecture notes—in each iteration, one hospital makes an offer to one doctor.)

If the assertion is false, describe a counter-example: a particular instance, and an execution of the algorithm on that instance, in which every doctor receives an offer before the start of the last iteration. If the assertion

is true, give a long-form proof. You may reuse without proof any observations or lemmas from the lecture notes (but cite the observation or lemma by number when you do, to let the reader know).

Exercise 3.12. (*Gale-Shapley run-time*) Let I be any instance of the Stable Matching problem in which all n hospitals have the exact same preferences. As a function of n , how many iterations will any execution of the Gale-Shapley algorithm (hospitals-propose, as described in the lecture notes) do on input I ? Explain your answer clearly.

Exercise 3.13. (*Gale-Shapley proof problem*) Prove or disprove the following assertion:

In every execution of the hospitals-propose Stable Matching algorithm, there is at most one hospital that makes offers to every doctor. (Note: we mean the algorithm described in the lecture notes—in each iteration, one hospital makes an offer to one doctor.)

If the assertion is false, describe a counter-example: a particular instance, and an execution of the algorithm on that instance, in which at least two hospitals make offers to every doctor. If the assertion is true, give a long-form proof. You may reuse without proof any observations or lemmas from the lecture notes (but cite the observation or lemma by number when you do, to let the reader know).

External exercises without solutions

- [JE](#) Chapter 4 (Page 179), Exercises 9–13
- [KT](#) (Page 22), Exercises 1–8
- [LLM](#) (2018) Page 207, Problems 6.16–6.32

3.8 Python code

This section lists the Python code for this lecture note. Click on a file's name to open the file on GitHub. Copying code from GitHub is more reliable than copying it from this PDF, since some PDF viewers drop the leading spaces (the indentation) of copied lines.

[stable_matching.py](#)

```
#!/usr/bin/env python3

from collections.abc import Collection, Iterator, Sequence

# preferences[i] lists, in order of preference, the people that person i ranks
type Preferences = Sequence[Sequence[int]]
type Matching = Collection[tuple[int, int]]

class Person:
    def __init__(self, preferences: Sequence[int]) -> None:
        self.preferences = preferences
        self.ranks = [0] * len(preferences) # allocate array
        for r, p in enumerate(preferences):
            self.ranks[p] = r
        self.partner: int | None = None
        self.partner_iterator = iter(self.preferences)

    def prefers(self, p1: int, p2: int) -> bool:
        return self.ranks[p1] < self.ranks[p2]

def Gale_Shapley(
    hospitals_preferences: Preferences, doctors_preferences: Preferences
) -> set[tuple[int, int]]:
    """The Gale-Shapley algorithm for stable matching"""

    hospitals = tuple(Person(p) for p in hospitals_preferences)
    doctors = tuple(Person(p) for p in doctors_preferences)

    unmatched_hospitals = list(range(len(hospitals)))

    while unmatched_hospitals:
        h = unmatched_hospitals.pop()
        D = doctors[next(hospitals[h].partner_iterator)]

        if D.partner is None:
            D.partner = h
        elif D.prefers(D.partner, h):
            unmatched_hospitals.append(h)
        else:
            unmatched_hospitals.append(D.partner)
            D.partner = h

    return set((D.partner, d) for d, D in enumerate(doctors)) # type: ignore[misc] # all matched

def unstable_pairs(
    hospitals_preferences: Preferences, doctors_preferences: Preferences, M: Matching
) -> Iterator[tuple[int, int]]:
    hospitals = [Person(p) for p in hospitals_preferences]
    doctors = [Person(p) for p in doctors_preferences]
```

```

    for h, d in M:
        hospitals[h].partner = d
        doctors[d].partner = h

    for h, H in enumerate(hospitals):
        for d, D in enumerate(doctors):
            # M is perfect, so H.partner and D.partner aren't None
            if H.prefers(d, H.partner) and D.prefers(h, D.partner): # type: ignore[arg-type]
                yield (h, d)

def is_stable(
    hospitals_preferences: Preferences, doctors_preferences: Preferences, M: Matching
) -> bool:
    return not any(unstable_pairs(hospitals_preferences, doctors_preferences, M))

def exhaustive_search(
    hospitals_preferences: Preferences, doctors_preferences: Preferences
) -> set[tuple[int, int]] | None:
    """Exhaustive search to find a stable matching"""

    from itertools import permutations

    n = len(hospitals_preferences)

    for p in permutations(range(n)):
        M = {(i, p[i]) for i in range(n)}
        if is_stable(hospitals_preferences, doctors_preferences, M):
            return M

    return None

def old_algorithm(
    hospitals_preferences: Preferences, doctors_preferences: Preferences
) -> list[tuple[int, int]]:
    """Old algorithm from LN 3 Fig. 2"""

    n = len(hospitals_preferences)
    assert n == 3

    hospitals = [Person(p) for p in hospitals_preferences]
    doctors = [Person(p) for p in doctors_preferences]

    pairs_by_rank: list[list[list[tuple[int, int]]]] = [[] for _ in range(n)] for _ in range(n)]

    for h, H in enumerate(hospitals):
        for d, D in enumerate(doctors):
            i, j = H.ranks[d], D.ranks[h]
            pairs_by_rank[i][j].append((h, d))

    M: list[tuple[int, int]] = []

    for i, j in ((1, 1), (1, 2), (2, 1), (1, 3), (3, 1), (2, 2), (2, 3), (3, 2), (3, 3)):
        for h, d in pairs_by_rank[i - 1][j - 1]:
            H, D = hospitals[h], doctors[d]
            if H.partner is None and D.partner is None:
                M.append((h, d))
                H.partner = d

```

```

        D.partner = h

    assert len(M) == 3
    return M

def test_Gale_Shapley(n: int = 10) -> None:
    from random import sample

    prefs = [[sample(tuple(range(n)), n) for _ in range(n)] for _ in range(2)]

    M = Gale_Shapley(*prefs)
    assert is_stable(*prefs, M) # type: ignore[call-arg, arg-type] # prefs has 2 items

def test_exhaustive_search(n: int = 5) -> None:
    from random import sample

    prefs = [[sample(tuple(range(n)), n) for _ in range(n)] for _ in range(2)]

    M = exhaustive_search(*prefs)
    assert is_stable(*prefs, M) # type: ignore[call-arg, arg-type] # prefs has 2 items

def test_old_algorithm() -> bool:
    from random import sample

    n = 3

    prefs = [[sample(tuple(range(n)), n) for _ in range(n)] for _ in range(2)]

    M = old_algorithm(*prefs)
    if not is_stable(*prefs, M): # type: ignore[call-arg, arg-type] # prefs has 2 items
        return False
    return True

if __name__ == "__main__":
    print(
        "old_algorithm produced",
        len([stable for stable in [test_old_algorithm() for _ in range(100)] if stable]),
        "stable matchings out of 100",
    )

    for n in range(5, 20):
        test_Gale_Shapley(n)
        print(f"Gale_Shapley({n}) passed")

    for n in range(5, 9): # exhaustive search takes time n!, so keep n small
        test_exhaustive_search(n)
        print(f"exhaustive_search({n}) passed")

```