AMERICAN UNIVERSITY OF ARMENIA
College of Science and Engineering
CS 140 Mechanics
HW03

Hopfield Networks in Unconstrained Examination Timetabling

Preliminary Deadline:	Sunday, May 05 2024, no later than 18:00 **SHARP**
Progress Deadline:	Saturday, May 11 2024, no later than 22:00 **SHARP**
Final Deadline:	Friday, May 17 2024, no later than 22:00 **SHARP**

Or

Final Exam Date:	Saturday, May 18 2024

Submission format:	Repository shared with skhachat@aua.am, lusine_mheryan@edu.aua.am, lilit.melikyan@ysu.am
Text:	D. Kriesel. "A Brief Introduction to Neural Networks" (see Kriesel_Brief_Intro_to_Neural_Networks.pdf in References.zip)
Reading:	Chapter 8, Hopfield Networks, pages 151 – 161

Submission Conditions:

1. This is an individual assignment. Identical or similar submissions / files / results / reports / diagrams etc. will be disqualified – both the source(s) and receiver(s) will collect 0 point.
2. Group work will be accepted only if all group members are explicitly indicated in the submission. The individual contribution of each group member must also be explicitly stated, including all reasons of forming the group.
3. The submission deadlines are rigidly strict. The repository contents will be checked even if the work is not complete
4. Not only precise solutions, but also free-format descriptions of ideas, difficulties, algorithms, simplifications, assumptions, etc. may be submitted.
5. All used external sources must be explicitly acknowledged.
6. **ATTENTION**: The assignment will be discussed during the upcoming PSS / OH. The regular sessions are scheduled for: Friday **May 03** at **14:30** in **313W PAB**, Saturday **May 04** at **16:00** in **416 PAB**, Wednesday **May 08** at **16:00** in **313 PAB**, Friday **May 10** at **14:30** in **313W PAB** and Saturday **May 11** at **16:00** in **416 PAB**.
7. Depending on the results of the Progress Submission, it will be decided after May 11, if HW03 proceeds to its Final Deadline or terminates at the Progress Deadline and the Final Exam is offered on May 18, as scheduled.

Consider the **Unconstrained Examination Timetabling Problem** – given *N* courses, *M* students and the lists S_k of students taking each course ($1 \le k \le N$), distribute the *N* exams within the given number of timeslots *T* such that no student will take more than one exam in each timeslot. The optimal solution is achieved when *T* is minimal.

Consider also the following heuristic algorithm to solve the problem:

1. Specify the number of timeslots *T* and schedule all of *N* exams (enumerated from 1 to *N*) for the same first timeslot (the timeslots are enumerated from 0 to $T - 1$). Obviously, the total number of clashes between the exams is maximal in such a single-timeslot schedule.
2. Check if the first exam has a clash. If so, shift it to the next timeslot. If in the next timeslot it still has a clash, shift it further to the next timeslot. The number of such shifts *s* is specified at the beginning. If the exam still has a clash after *s* shits, leave it there and turn to the next exam.
3. Complete one full iteration by repeating such checks and shifts for all other exams from 2 to *N*.
4. Repeat the iterations *n* times.

This algorithm is implemented in a simple Java GUI application **SchedulingCS140** (see **SchedulingCS140.zip** archive in \Home Works folder of the course Moodle site). The numbers of timeslots *T*, exams *N*, iterations *n* and shifts *s* are specified in the "**Slots:**", "**Courses:**", "**Iters:**" and "**Shifts:**" boxes respectively. The "**Load**" button loads clashes from the specified file. Each exam is represented by a black segment moving horizontally from slot to slot. If having a clash, its motion line is drawn in red, otherwise – in green. Press "**Start**" button to start the scheduling from the initial state with all courses placed in the first timeslot, once "**Slots:**", "**Courses:**", "**Iters:**" and "**Shifts:**" values are specified. Press "**Step**" button to run a single iteration from the current state. Press "**Print**" button to print the current configuration in the Java console.

Task 1.

1.1. Create the repository and share it. Select your individual timetables with the indicated optimal number of timeslots T_{opt} and upload them in the repository:

- **hec-s-92** (T_{opt} = **17** slots) and **car-s-91** (T_{opt} = **28** slots), if the combined number of letters in your first and last names is strictly less than 13;
- **sta-f-83** (T_{opt} = **13** slots) and **uta-s-93** (T_{opt} = **30** slots), if the combined number of letters in your first and last names is 13;
- **ute-s-92** (T_{opt} = **10** slots) and **car-f-92** (T_{opt} = **28** slots), if the combined number of letters in your first and last names is 14;
- **lse-f-91** (T_{opt} = **17** slots) and **tre-s-93** (T_{opt} = **20** slots), if the combined number of letters in your first and last names is 15;
- **yor-f-83** (T_{opt} = **19** slots) and **rye-s-93** (T_{opt} = **21** slots), if the combined number of letters in your first and last names is 16;
- **ear-f-83** (T_{opt} = **22** slots) and **kfu-s-93** (T_{opt} = **19** slots), if the combined number of letters in your first and last names is strictly greater than 16.

1.2. Add in the **TimeTable.java** class of the project ScehdulingCS140 a new JButton “Continue” to continue the scheduling algorithm from the current state. Upload the updated project into the repository.

ATTENTION!!! Complete **Task** 1 by the Preliminary Deadline.

Task 2.

2.1 Implement the **public class Autoassociator**.

2.2 In **public class CourseArray** add a method **public int[] getTimeSlot(int index)** that returns the specified timeslot as an int array of 1 and -1 values. The k-th element is 1, if the k-th course is in the timeslot, and -1 – otherwise.

Task 3. Run the original algorithm for each of the two timetables for different values of the shifts, iterations and, possibly, slots. Observe and summarize in a log file the behavior of the timetable – how these changes affect the number of clashes. Keep the updated log uploaded in the repository.

Task 4. Create an instance of **Autoassociator** as a member of **public class TimeTable**. Train the autoassociator with clash-free timeslots observed in **Task 3**. Save the used timeslots in a log file indicating the number of slots, the shift, the iteration index and the timeslot index.

Task 5. Re-run the algorithm for each of the two timetables, this time interrupting the iterations with unit updates of the timeslots using the trained autoassociator. Try different strategies to minimize the number of clashes. Save each such update instance in a log file.

ATTENTION!!! All tasks from **Task 2** to **Task 5** must be addressed before the Progress Deadline.