ASSIGNMENT 1: AUTOMATED CONFERENCE SCHEDULER

Goal: The goal of this assignment is to take a complex new problem and formulate and solve it as search. Formulation as search is an integral skill of AI that will come in handy whenever you are faced with a new problem. Heuristic search will allow you to find optimal solutions. Local search may not find the optimal solution, but is usually able to find good solutions for really large problems.

Scenario: Optimization of Conference Schedule.

A conference has n papers accepted. Our job is to organize them in a best possible schedule. The schedule has p parallel sessions at a given time. Each session has k papers. And there are a total of t time slots. We can assume that n = t.p.k. For example, in Figure below t =2, p = 3 and k =4.

Papers 1,2,3,4	Papers 5,6,7,8	Papers 9,10,11,12
Papers 13,14,15,16	Papers 17,18,19,20	Papers 21,22,23,24

We first define the characteristics of a good schedule. For any good schedule most people should feel no conflict about which session to attend. That is, (1) all papers in one session should be related to a single theme. And (2) all papers in parallel sessions should be as far away as possible to avoid conflict.

To operationalize this intuition let us assume we are given a function representing the distance between two papers: d(p1,p2), such that d is between 0 and 1. We can similarly define a similarity between two papers s(p1,p2) = 1-d(p1,p2).

Now we can define the goodness of a schedule as follows

Sum(similarities of all pairs of papers in a session) + C.Sum(distances of all pairs of papers in parallel sessions).

In our example, the goodness will be computed as

s(1,2) + s(1,3) + s(1,4) + s(2,3) + s(2,4) + s(3,4) + s(5,6) +s(5,7)+s(5,8)+s(6,7)+s(6,8)+s(7,8) +…….

+ C[d(1,5)+d(1,6)+…d(1,11)+d(1,12) + d(2,5) … + d(2,12) + ….. + d(8,12) + d(13,17)+…]

The constant C trades off the importance of semantic coherence of one session versus reducing conflict across parallel sessions.

Our goal is to find a schedule with the maximum goodness.

Input:

The first line has total processing time available in minutes.
The second line has k: the number of papers per session
The third line has p: the number of parallel sessions
The fourth line has t: the number of time slots.
(This implies that number of papers is pkt)
The fifth line has C: the tradeoff parameter
Starting sixth line we have space separated list of distances between a paper and all others. Note that d(x,y) = d(y,x). Also, all d(x,x) = 0.

Here is a sample input

```
5
2
2
1
1
0    0.4   0.8   1
0.4   0    0.6   0.7
0.8   0.6   0    0.3
1     0.7   0.3   0
```

Output:

Your algorithm should return the max-goodness schedule found in the desired time limit.

The output format is: space separated list of paper ids, where a session is separated by bars. And all papers in a time slot in a different line.

For this problem above the optimal solution is p1 and p2 in one session; and p3-p4 in other. It will be represented as

0 1 | 2 3

Notice that there are other equivalent ways to represent this same solution. Example:

1 0 | 2 3 or 3 2 | 0 1 or 2 3 | 1 0 (etc). All of these are equally valid

Verify that for this problem the total goodness is 4.4.

Basic Algorithms you can try:

1. Heuristic Search: Design a state space and transition function for this problem. Define a heuristic function for evaluating a state. Implement A* (or variants) and measure quality of solutions found (and scalability). If heuristic is admissible – quality is optimal but algorithm may be slower. Test a couple of heuristics if you can design them.
2. Branch and Bound: Design a state space and transition function for this problem. Optionally define a heuristic function for evaluating a partial schedule. Also, optionally, define a ranking to pick the next successor. Implement Depth First Branch and Bound (or variants) and measure scalability.
3. Local search: Implement a neighbor function for this problem. Implement local search (or variants). Measure of quality of best solution found as a function of time or other model parameters.

Recommended: start with local search as your base algorithm.

Sample Code:

You are being provided sample code that can take in the input and generate the output in C++. You may choose to not use this code. You may program the software in any of C++, Java or Python. The versions of the compilers that will be used to test your code are

JAVA: java version "1.7.0_79" (OpenJDK)
Python 2.7.3
g++ 4.8.1

The sample code (written in cpp) can be downloaded here. This code reads in an input file, organizes the papers according to the order they were read in, and scores the organization. If you choose to use the sample code, you should replace SessionOrganizer.organizePapers() (and add any supporting methods or classes that you need). You can compile and run the sample code as follows:

make
./main <input_filename> <output_filename>

We have even supplied formatchecker.jar which checks the format of the output file generated and gives the score (if the format is valid). To run the formatchecker, use the following command:
java -jar formatchecker.jar <input_filename> <organization_filename>

We have provided three input files representing easy problems. We recommend you experiment with other problems as well.

In addition to sample code and formatchecker.jar, we are also providing a sample submission file (2017CSZ9999.zip) as well which adheres to the submission guidelines mentioned in the next section.

What to submit?

1. Submit your code in a .zip file named in the format **<EntryNo>.zip.** If there are two members in your team it should be called <EntryNo1>_<EntryNo2>.zip. Make sure that when we run "unzip yourfile.zip", a folder with the name 'yourfile' is created and it contains at least the following files in addition to your source code -

    a) compile.sh - it should create all the binaries that you will be using in run.sh. Keep it empty if your code does not require any compilation, eg. in case of python program. Time taken by compile.sh will not be taken into account while evaluating your submission.
    b) run.sh
    c) writeup.txt

    You will be penalized for any submissions that do not conform to this requirement.
    Your code must compile and run on our VMs. They run amd64 Linux version ubuntu 12.04. You are already provided information on the compilers on those VMs. These configurations are similar to GCL machines like 'todi' (have a RAM of 16 GB)

    Your run.sh should take 2 inputs, someinputfile.txt and someoutputfile.txt. It should read the first input as input and output the answer in a file named someoutputfile.txt

    ./run.sh input.txt output.txt (please note any name could be given to the two files).

2. The writeup.txt should have two lines as follows

First line should be just a number between 1 and 3. Number 1 means C++. Number 2 means Java and Number 3 means Python.
Second line should mention names of all students you discussed/collaborated with (see guidelines on collaboration vs. cheating on the course home page). If you never discussed the assignment with anyone else say None.
After these first two lines you are welcome to write something about your code, though this is not necessary.

Code verification before submission: Your submission will be auto-graded. This means that it is absolutely essential to make sure that your code follows the input/output specifications of the assignment. Failure to follow any instruction will incur significant penalty.
We shall be generating a log report for every submission within 12 hours of submission. This log will let you know if your submission followed the assignment instructions (format checker, scripts for compilation & execution, file naming conventions etc.). Hence, you will get an opportunity to resubmit the assignment within half a day of making an inappropriate submission. However, please note that the late penalty as specified on the course web page will still apply for resubmissions beyond the due date. Exact details of log report generation will be notified on Piazza soon.
Also, note that the log report is an additional utility in an experimental stage. In case the log report is not generated, or the sample cases fail to check for some other specification of the assignment, appropriate penalty for not adhering to the input/output specifications of the assignment will still apply at the time of evaluation on real test cases.

Evaluation Criteria: Performance, i.e., quality of the best schedules found by your method on a variety of test inputs. Extra credit may be given to standout performers.

What is allowed? What is not?

1. You may work in teams of two or by yourself. We do not expect a different quality of assignment for 2 people teams. At the same time, please spare us the details in case your team cannot function smoothly. Recall, that you can't repeat a team in the course. If you are short of partners, our recommendation is that this assignment is quite straightforward and a partner should not be required.
2. You cannot use built-in libraries/implementations for search or scheduling algorithms.
3. While you are allowed one of three languages we recommend that you use C++ since it produces the most efficient code. This assignment requires you to produce the best solution within a given time constraint.
4. You must not discuss this assignment with anyone outside the class. **Make sure you mention the names in your write-up in case you discuss with anyone from within the class outside your team.** Please read academic integrity guidelines on the course home page and follow them carefully.
5. Please do not search the Web for solutions to the problem.
6. Your code will be automatically evaluated. You get a zero if your output is not automatically parsable.
7. We will run plagiarism detection software. Any team found guilty will be awarded a suitable penalty as per IIT rules.