MINI PROJECT 1

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.

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

1 2 | 3 4

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

2 1 | 3 4 or 4 3 | 1 2. All of these are equally valid

Verify that for this problem the total goodness is 4.4.

Basic Algorithms:

1. Heuristic Search: Design a state space and transition function for this problem. Define a heuristic function for evaluating a state. Implement/download 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/download Depth First Branch and Bound (or variants) and measure scalability.
3. Local search: Implement a neighbor function for this problem. Implement/download 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.

Other Exploratory Algorithms/Suggestions/Research Questions:

1. Which variant of local search performs the best? Implement and test several variants.
2. Is there symmetry in the problem? Can we reduce our state space by breaking the symmetry? How?
3. Can we convert the scheduling problem into an instance of Weighted Partial MAXSAT? How? Use this software for solving MAXSAT problems: http://www.uni-ulm.de/fileadmin/website_uni_ulm/iui.inst.190/Mitarbeiter/kuegel/akmaxsat_1.1.tgz
4. Can we convert the scheduling problem into an instance of integer linear programming? How? Use Gurobi as your base ILP solver: http://www.gurobi.com/html/academic.html. It has a free academic license.
5. Can we use an existing constraint processing software to model and solve this problem? Try this software as a start: http://numberjack.ucc.ie/download.
6. Design your own solver for this problem using your other ideas, for example, here is one. What if we first cluster all papers into groups of k papers each so that each group is as coherent as possible? Later we can schedule these papers to minimize conflict. Indeed, this won't be optimal, but this might scale very well.

Implementation:

You are being provided sample code that can take in the input and generate the output in Java. You may choose to not use this code. You may program the software in any language (or even in multiple languages).

Also, you may choose to write the algorithms yourself or use existing software. You could use any piece of existing software if that helps you. Of course, please describe in your paper what software you used and the source. Many basic algorithms from the book are implemented here: http://aima.cs.berkeley.edu/code.html

The sample code can be downloaded from the course website. 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:

java SessionOrganizer inputfile1.txt

You can also use the sample code to test your solution by running:

Java SessionOrganizer inputfile1.txt organization1.txt

where organization1.txt is the organization that your code produced.

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

Timeline

You are to work in teams of up to 2. The project has two milestones.

Milestone 1 (due April 18th): Write a one page document summarizing results of your current implementation and ideas to explore in the future.

Milestone 2 (due May 2nd noon): Write a 4-8 page report describing: (1) your algorithms implemented, (2) the analysis and comparison of the algorithms along scalability, running time and any other parameters. Also note any external resources used and the names of other students you discussed with through the project. Submit the best performing code from your team for the final competition.

Grading (25 points)

You will be graded for the following:

- [5 points]: The writeup for Milestone 1. These are mostly free points and intended to check that you start making progress early on.
- [15 points]: Study ~3 algorithms/optimizations/reformulations for this problem. Use local search with random restarts as your first algorithm. For all your algorithms study their performance in control experiments. [If it is a random algorithm, use several simulations runs for understanding the performance]. For these experiments, vary the size of the problems, and see the behavior and qualities of solutions obtained. These 15 points are awarded based on choice of algorithms, carefulness of evaluation, validity of final results and clarity of the paper. This is like evaluating a research paper.

- [5 points]: Final performance of your best system compared to other teams' software. This is a competition and better performing systems get more points.
- [Extra credit]: If your ideas are particularly creative and novel. Or if your experiments and extremely thorough and detailed.

Submission Instructions

You should submit a zip file containing all relevant parts of the assignment for each milestone. At Milestone 1, this will just be your one page writeup. For Milestone 2 this will be your report, your source code, and a README describing how to run your system. You should submit your assignment via dropbox. We will post a link on the website that you can use to access dropbox.