# 22C:153 Programming Assignment

**Due: Friday, 5/9**

**Introduction.** This programming assignment aims to give you some practice in trying to experimentally analyze a bunch of heuristics for a problem known to be NP-hard. One of the heuristics involves formulating the problem as a 0-1 integer program, solving the corresponding LP-relaxation, and then rounding possible fractional values. This will give you some experience with using LP solvers, and more generally it will provide some hands-on experience with LP-based techniques which are becoming more and more popular as a tool of choice to solve combinatorial optimization problems. The problem, called *broadcast scheduling* will be described in the following section. This problem was shown to be NP-hard (Erlebach and Hall, SODA 2002) only recently and absolutely nothing is known about solving this problem approximately.

**The Broadcast Scheduling problem.** The *broadcast scheduling* problem can be informally described as follows. There are $n$ items called *pages* and there are a bunch of clients that make requests for some of these pages from time to time. There is a server that can broadcast pages. A page that is broadcast satisfies *all* outstanding requests that have not yet been satisfied. The goal is to schedule the broadcasts so as to minimize the total wait time of the requests.

More precisely, suppose that $P = \{1, 2, \ldots, n\}$ is the set of pages. Also suppose that all events take place at time steps $1, 2, \ldots, T$ for some positive integer $T$. At each time step $t$, $1 \leq t \leq T$, there are 0 or more requests for each page $p \in P$. If a page $p$ is requested at time $t$ and if $t'$ is the earliest time step *after* $t$ at which $p$ is broadcast, then the *wait time* for the request for $p$ at time $t$ is $(t' - t)$. The broadcast scheduling problem is to pick a page from $P$ to broadcast at each time step, so as to minimize the total wait time (over all requests for pages).

The description of the problem and the heuristics we will consider, all come from the paper "Algorithms for Minimizing Response Time in Broadcast Scheduling" by Gandhi, Khuller, Kim, and Wan (IPCO 2002; can be downloaded from `http://www.cs.umd.edu/~gandhi/pubs.html`). Henceforth, we'll call this the GKKW paper. Before you actually start any implementation, you should at least read Sections 1, 2, 3, and 8 of the paper.

**Heuristics.** In Section 8 of the GKKW paper, the authors describe experiments with two classes of heuristics. The class of *greedy heuristics* are heuristics which determine which page to broadcast at each time step based on some local measure of how costly each choice is. For example, let $N_t^p$ be the number of requests for page $p$ which are not yet satisfied at time step $t$. A large value for $N_t^p$ provides strong support for broadcasting $p$ at time $t$. So we could use the following heuristic:

At each time step $t$, broadcast a page $p$ with largest $N_t^p$.

This turns out to be somewhat simplistic and you should read the GKKW paper for a more sophisticated version of this heuristic.

The second class of heuristics considered in the GKKW paper are *LP-based heuristics.* An LP-based heuristic, typically, involves two steps: (i) solving the LP-relaxation of the 0-1 IP formulation of the broadcast scheduling problem and (ii) rounding any fractional solution values, either deterministically or in a randomized fashion to get an integral solution. As the tables at the end of the GKKW paper show, deterministic rounding outperforms randomized rounding and the greedy heuristic.

**Your Tasks.** For this assignment you are required to expand on the experiments described in the GKKW paper in order to answer the following questions. Given the fact, that we don't know much about this problem, the questions are somewhat open ended.

1. In the experiments in the GKKW paper, the greedy heuristic performs poorly as compared to LP-based techniques. It is not hard to come up with greedy heuristics that are more sophisticated than what is used in the GKKW paper. The open question is whether these heuristics will perform any better than the simple greedy heuristic in the paper. Specifically, it seems that the greedy heuristic does not pay enough attention to future requests. Can you come up with greedy heuristics that perform as well (or better) than the LP-based heuristics? The motivation for this is the fact that even though an LP can be solved in polynomial time, this is relatively slow and so should be avoided, if possible.

2. In the experiments described in the GKKW paper, the greedy heuristics perform about 20% worse than OPT, in the worst case. Are there examples, on which the greedy heuristics perform much worse – say , 100% worse than OPT? If you have trouble devising or generating such examples, can you identify why the greedy heuristic might perform quite well relative to OPT, even in the worst case. In fact, it would be even better if you could prove that the greedy heuristic always produced a solution within $c$ times OPT, for some constant $c$. Of course, doing this would solve an open problem and would yield an easily publishable paper.

3. Let $\text{OPT}_L$ denote the cost of an optimal solution of the LP relaxation. Clearly, $\text{OPT}_L \leq \text{OPT}$, but the GKKW paper does not say much about how large the gap between OPT and $\text{OPT}_L$ might be. Specifically, how bad can the ratio $\text{OPT}/\text{OPT}_L$ be? It could be that this ratio can be quite large in the worst case, and in that case it would be nice to see an example for which this gap is large. Alternately, it could be that this ratio is always quite small and in that case, it would be quite nice to see a proof that this is true.

To be able to perform these experiments and say something intelligent in connection to the above questions, you need to have three pieces of code implemented.

1. Code that finds OPT. This would be a simple branch-and-bound type algorithm, that essentially scans through a large number of possible solutions before picking OPT.

2. Code that implements the greedy heuristic.

3. Code that implements the LP-based heuristics. This involves using an LP-solver. There are a number of public-domain LP solvers. I have used LINDO in the past successfully. A trial version of LINDO can be downloaded from `http://www.lindo.com/table/opsyst.html`. I recommend this, but you are welcome any other tools you can find including the LP solvers that are part of packages such as Mathematica and Matlab.

Turn in a write-up, at most 4 pages in length. This should contain a brief description of how your experiments were performed followed by a more detailed section addressing the above issues. The programming aspect of this project is not intensive, however you should give yourself plenty of time to perform experiments, tweak inputs, and interpret the results you get. So ideally, you should get the program done quickly and then spend the rest of the semester performing experiments and understanding the results.