CS430 Final Project

Peter Hadlaw, Henry Winn

Algorithm Design and Pseudocode

1. Read jobs from text file.
    1. Time Complexity: O(n) (n = # jobs)
2. Build adjacency matrix from jobs where an edge between jobs represents an overlapping of jobs.
    1. Time Complexity: $O(n^2)$ because each job is compared to every other job
    2. During this, count the number of inbound edges for every node
    3. Could be optimized slightly by adding both directions when an edge is found because edges are not directed
3. Group jobs by the number of edges
    1. Time complexity: This is done while building the adjacency, so no time is added
4. Iterate over all jobs in each connectedness set in order of connectedness level. (e.x. jobs that are connected to fewer other jobs are iterated over before jobs that are more connected).
    1. For each job, add to a machine's schedule depending on if the jobs already in the machine's schedule don't share an edge with the job you are trying to add.
        1. If they share an edge (checked in O(1) time using adj. matrix), the jobs overlap and the current job cannot be added to the machine's schedule.
        2. In this case, move on to the next machine. If all machines are checked and not can be used, move onto the next job
    2. Time Complexity: O(nm) because in the worst case is that a job has to check every scheduled job on every machine

Psuedocode:

```
jobs = []
f = open(inputfile)
#1
for f.readline as line:
        jobs.append(process(readline))

matrix = [][]
```

```
for jobs as index,job:
        for jobs as other_index,other_job:
                if job.intersects(other_job):
                        matrix[index][other_index] = True
                        job.inbound_edges += 1
        connectedness_groups[job.inbound_edges].append(job)

for connectedness_groups as group:
        for group as job:
                for machines as machine:
                        for machine.schedule as scheduled_job:
                                if matrix[job.i][scheduled_job.i] == None:
                                        machine.schedule.append(job)
                                        break
```

Proof of Correctness by Contradiction

To prove our algorithm is correct, we will begin by assuming that choosing first from a higher connected group of jobs would yield a better (more jobs scheduled) solution.

Suppose we have m machines that we will be assigning n jobs to. We will denote each set of equally connected jobs as S[i] where S[i+1] is the set of jobs each with one more edge connecting to it than any job in S[i]. We will also assume that all jobs in S belong to one connected graph for simplicity's sake.

If we compare the set S[i+1] to S[i], S[i+1] must be connect to all jobs in S[i] because the graph is connected. By scheduling from S[i+1], we eliminate the possibility that some jobs from S[i] will be able to be selected. Since S[i] must contain multiple jobs that are not directly connected, we lose the opportunity to use those unconnected jobs by using a single connecting job. Because the multiple unconnected jobs are greater in number than the single highly-connected job, it is impossible that using a more highly connected job would ever yield more jobs scheduled than using less connected jobs.