Programming Project #2:
Expert System Shells and Airplane Scheduling
Due Nov 9, 2pm

In this programming project assignment, you will develop solutions to a simplified airplane scheduling problem: Suppose that you are an aircraft controller at John Wayne Airport (for simplicity, consider only the one jet plane runway). You receive information about incoming flights, including the type of aircraft, the requested or preferred landing time, and a latest possible landing time (by using reserve fuel to slow down or circle). Obviously, you wish to let all planes land as close as possible to their preferred landing time. However, with only one runway, and with safety in mind (you can't land two jumbo jets 60 seconds apart!), you cannot please everyone.

This problem can be viewed as an optimization or constraint satisfaction problem: your goal is to minimize the *cost* in terms of extra minutes of delay over all incoming planes. For simplicity, you will just consider a set of static situations: You have a page of information about a set of incoming flights, and you must produce a schedule. (In the real world, the situation is more *dynamic*: information arrives at different times, and you may need to re-arrange your schedule on-the-fly, as it were.)

As a simple example, suppose you had a situation with four planes as follows:

```
(deffacts schedule
    (smallPlane P1) (flightInfo P1 1430 30 5)
    (bigPlane P2)   (flightInfo P2 1430 35 10)
    (smallPlane P3) (flightInfo P3 1435 20 5)
    (bigPlane P4)   (flightInfo P4 1438 30 10)
)
```

Where planes P1 and P3 are "small", while P2 and P4 are "big", and the "flightInfo" relation shows the preferred arrival time, the amount of extra fuel (in minutes), and the cost per minute delayed for each plane. Suppose you also have a rule that says that all planes (regardless of size) must land at least 5 minutes apart.

We say a solution is *feasible* if it satisfies all of the hard constraints: namely that all planes are on the ground before they run out of fuel, and that all planes land at least five minutes apart from each other. The above "schedule" has thousands of *feasible* solutions (or is it 10s of thousands?). In fact there are two, equal-cost *optimal* solutions. One of these optimal solutions is:

```
(land P1 1435) (land P2 1430) (land P3 1445) (land P4 1440)
```

(Notice that one should choose to land the big planes first, since they are more expensive to delay, but that one can sneak in a little plane between the two big planes.)

The general task:

For this assignment, the general task is to devise an algorithm (with heuristics) that is able to solve most problem specifications. I would expect that you should be able to solve the problem (that is, find the optimal sequence of landings) for up to about 10 planes.

An open problem!

Unfortunately, unlike your first programming assignment (and unlike problem #3), I believe this problem is *open*: there are no known, general-purpose constraint satisfaction algorithms for this task that are polynomial in execution time. As a computer scientist, I really like open problems: that is what research is all about. As students, you might dislike the openness of this assignment, because it means that it is hard for me to provide exact specifications. What I would really like you to do is think about the problem, and do your best for two weeks and hand in whatever you have at that point. Obviously, I do not expect you to completely solve this open problem. However, here are a set of *minimum* requirements:

Minimum Requirements (for approximately a B+ grade):

For this assignment, I would like you to try out the CLIPS production system. You will find executables available on both Unix and PC. I've put the documentation for this system on the shared files at ics-171\Files\Clips, along with the Clips executable. If you'll note, the sample FlightData files (at ics-171\Files\AirplaneSchedules) are Clips-readable. You must:

- Implement a brute-force solution within Clips that does an exhaustive search of the entire space of solutions. Obviously, this solution should not work on bigger problems, but it should solve the two 4-plane problems I've given. Your Clips solution should just be a set of rules – there is no need to use any of the more advanced programming constructs or data structures provided by Clips.
- Implement a more sophisticated solution that uses the following approach. First, generate all possible permutations of plane sequences (there are n! of these, for n planes). Then, for each ordering, there is a way to quickly find the optimal solution for that ordering (if one exists at all). Simply schedule the first plane at its optimal time, and then each plane thereafter as soon as possible, given the hard constraints. You may implement this solution in Java.
- Run your system on the set of example data files provided (FlightData1, FlightData2, etc) and find the optimal schedule for each problem. Where it is feasible, use both the brute force algorithm and the permutation algorithm, so you can compare performance.

Details, details, details:

Stated more formally, this is an *optimization* problem. Given a cost for each plane (C_i), and its preferred landing time ($BestT_i$), you must assign landing times to all planes ($LandT_i$), such that you minimize the function $\Sigma\ C_i\ (LandT_i - BestT_i)$ over all planes. The assignment of times to planes must be subject to two hard constraints: $|\ LandT_i\ - LandT_j\ | > SafetyT$ (the difference between landing times of any two planes *i* and *j* must be some minimum safe time between landings), and $LandT_i$ must be within the predefined domain of plane *i* (since each plane has a finite amount of extra gas).

For this assignment, you may have a single value for $SafetyT$, the times between safe landings. You may ignore the distinction between different-sized aircraft. You may also ignore the usual specification of time as a pair (hours, minutes); instead, assume that time is expressed as "minutes since midnight", and can therefore be subtracted and added without any difficulty. You should also ignore seconds – time is discrete for this assignment and the smallest unit of interval is one minute.

Advanced stuff (for those aiming for As):

There are a number of *heuristic* approaches that can be applied to constraint-satisfaction problems, that might allow your system to handle larger numbers of planes. I haven't implemented these, so I do not know how effective they will be for this particular problem.

- Look ahead pruning: Whenever you assign a landing time to a plane, $LandT_i$, you can reduce the domain of all other planes, by eliminating any possible landing times that conflict with the safety constraint surrounding the time $LandT_i$.
- Cost pruning: Similar to alpha-beta cutoffs, the idea is that once you have one complete and feasible solution with a certain cost, then as you are building alternatives, there is no need to consider partial solutions whose cost already exceeds that of the complete solution.
- Permutation elimination: For the approach that first generates all permutations, you should be able to eliminate many of these as infeasible by a simple analysis of the domain of each plane: If the last possible landing time for plane *i* is **before** the best possible time for plane *j* and if plane *j* is before plane *i* is in your current permutation, than that permutation can be dropped.

For extra work toward that A grade, you can also try eliminating some of the simplifications that I made. For example, use the three classes of planes (small, medium and large) and then make the value of $SafetyT$ depend on the type of plane. I happen to have a friend who is a flight instructor, and he gave me a complex table for safe distance between landings of three different classes of planes. One interesting point is that the safe distance between a big plane and a small plane is different depending on which of the two is in front! Send me email, I can supply this table, and you can try to apply it to the problem.

Also, what happens to the problem if you allow planes to land some small number of minutes **before** their "preferred" time (by accelerating, or somehow cutting corners on the flight path). There should be some cost/min associated with an early landing (but perhaps different than the cost for a late landing). What does this do to the complexity of the algorithms?

What if you have two runways (such as at LAX, SFO, etc.)? Assume that all planes can be landed at either runway without a penalty. Now, you must assign a landing time, and a runway to all planes, and the safety constraint must be satisfied for any two planes that land on a given runway. What does this do to the complexity of the problem? If you implement a solution to this problem, you should increase the density of the planes in the FlightData input files.

Finally, I will also award extra effort points if you implement the permutation algorithm within Clips (rather than Java or C++). Clips includes an entire lisp-like programming language that can be used on the right hand side of the production rules. Ask me for details and hints.

For an A grade, you do not need to address all of these additional features. The general idea is to do something beyond the minimum requirements, and I will make the (subjective) decision as to how much additional effort and results you have demonstrated.

Project report details

For this project, you must include an analysis of the computational cost of various solutions. In particular, under exactly what conditions is the "sophisticated" permutation solution faster than a

brute force solution? At what point (how many planes?) would you predict these algorithms to fail (run out of memory, or take more than 24 hours to run). As with the previous project, your report should describe to what degree any heuristics you use actually improves performance on the specific examples of FlightData you use.

Note that this project doesn't really have much of a user interface: Read data file—produce optimal flight landing schedule, with cost. I would not expect much under this heading of the project report.

Division of labor:

First, a reminder that your team must rotate roles from the previous project. For this project, one obvious way to divide up work between the programmers is to have one person in charge of the Clips implementation of the brute-force exhaustive algorithm, and the other programmer in charge of implementing the permutations algorithm (and this second programmer may use Java). If the permutation algorithm is written in Java, then it really won't share any commonalities with the Clips exhaustive search algorithm. (And you may transform the FlightData input files into a format that is easier to read into Java.)

Reminder: no late projects! You only have 2 weeks for this project.