Living with fear stops us taking risks, and if you don't go out on the branch, you're never going to get the best fruit.

Sarah Parish
Question 1 (10 Marks)

Here is the standard B&B best-first algorithm to find an optimal solution for a minimization problem:

```plaintext
compute U
initialize S = {partial solutions resulting from first decision}
while True:
    Choose P in S with minimum L_P
    if P is a complete solution:
        print P
        exit
    else:
        S = S \ {P}
        for each feasible P' that extends P with one more decision,
            compute (L_{P'}, U_{P'})
            if L_{P'} > U, discard P'
            else
                S = S + {P'}
                if U_{P'} < U, U = U_{P'}
        end for each
end while
```

How would you modify this algorithm to find all optimal solutions to the problem?
Solution: the basic idea is to have the algorithm keep going as long as there could still be another optimal solution to be found. We can determine this quite effectively by noting that when the first optimal solution $P$ is generated, its $L_P$ and $U_P$ values will both be equal to the actual value of the optimal solution. This means that the global upper bound $U$ will also be set to this value. This means that when we select a partial solution for expansion, if its $L$ value is $> U$, we can terminate the algorithm – otherwise, we need to keep going because there might be another optimal solution to be found. So the simplest fix to the algorithm is to replace the five lines

```python
while True:
    Choose $P$ in $S$ with minimum $L_P$
    if $P$ is a complete solution:
        print $P$
        exit
```

with

```python
while |$S$| $>$ 0:
    Choose $P$ in $S$ with minimum $L_P$
    if $L_P$ $>$ $U$:
        exit
    if $P$ is a complete solution:
        print $P$
```

Marking:

There are infinitely many ways to correctly modify the given algorithm and students are not necessarily going to do it they way I did, but there are certain key elements of the solution:

- continuing after the first optimal solution is selected from $S$ \hspace{1cm} 2 marks
- ensuring that all optimal solutions are printed \hspace{1cm} 3 marks
- ensuring that only optimal solutions are printed \hspace{1cm} 3 marks
- stopping as soon as no more optimal solutions can exist \hspace{1cm} 2 marks

There is another very subtle point that students may pick up on:

- not crashing if every solution is optimal \hspace{1cm} 2 bonus marks

The “not crashing if every solution is optimal” is why I changed the “while True:” to “while |$S$| $>$ 0” ... if it is left as “while True:” and all solutions are optimal, the algorithm will eventually try to remove a partial solution from an empty set.
Question 2 (40 marks)

Your outstanding work for the National Prison for Disgraced Politicians has made you a hero in Occiput. The Prime Minister has chosen you to arrange her event schedule.

The PM has been invited to attend a variety of festivals around the country, each of which is organized by one of the political parties. Some of the parties have organized multiple events. For each event the organizers are offering you a nice bribe if you agree to schedule the PM’s presence for the entire event.

Obviously you want to select a non-overlapping set of festivals that maximizes your income. However there is a complicating factor: the PM is not willing to show too much favouritism, so she is willing to attend at most two events organized by each party.

Each festival is represented by a 5-tuple: (ID,start date, end date, bribe value, party). Sample data might look like this:

<table>
<thead>
<tr>
<th>ID</th>
<th>Start</th>
<th>End</th>
<th>Bribe</th>
<th>Party</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>6</td>
<td>50</td>
<td>Red</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>8</td>
<td>75</td>
<td>Blue</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>8</td>
<td>20</td>
<td>Red</td>
</tr>
<tr>
<td>4</td>
<td>9</td>
<td>12</td>
<td>100</td>
<td>Yellow</td>
</tr>
<tr>
<td>5</td>
<td>13</td>
<td>15</td>
<td>60</td>
<td>Red</td>
</tr>
</tbody>
</table>

One feasible solution is \{1,3\} – which brings you an income of $70. Another feasible solution is \{4,5\} – with an income to you of $160. Neither of these is optimal for this data. Note that \{1,3,4,5\} is not feasible because it contains three “Red” party events, and \{1,2\} is not feasible because the events overlap.

Design a Branch & Bound algorithm to find a feasible solution that maximizes your income for any instance of this problem (not just the example). Your solution should include

- Your objective function – ie how you will compute the value of a feasible solution
- Your decision sequence – if you decide to re-order the list of festivals, explain why
- Calculation of the initial Global Upper Bound
- Calculation of Cost-so-far
- Calculation of Guaranteed-future-cost
- Calculation of Feasible-future-cost
Solutions which demonstrate careful thought about the problem will earn more marks than trivial solutions.

You are not required to write any code – describing how you will compute the various values is sufficient.

I have provided this page and the next for your answer – but don’t feel that you have to fill two pages!
Solution:

**Objective Function (5 marks)**

Minimize the sum of the bribes for the events not included in the PM’s schedule.

**Decision Sequence (10 marks)**

Weak answer: leave the list in whatever order it is given, then every decision is yes/no (subject to feasibility) for the next event in the list. (6 marks for this answer)

Strong answer: prepare to use a Greedy Heuristic for computing feasible future costs by sorting the events. A variety of sort orders are plausible:

- descending order of bribe/duration, so the first event considered is the one that gives the highest value per day
- duration, so shorter events come first in the list
- end time of the event, as in the standard activity selection problem
- descending order of bribe value, so the most valuable events are considered first

Regardless, once the list of events has been ordered, the sequence of decisions can consist of a yes/no decision for each event in the list. (10 marks for this answer)

Another equally valid decision sequence, with the events sorted by start time, is to repeatedly decide which event will be selected next. (10 marks for this answer)
**Initial Global Upper Bound** (10 marks)

Trivial answer: sum of all bribes (3 marks for this answer)

Acceptable answer: choose a random feasible solution and evaluate its cost (6 marks for this answer)

Strong answer: use a greedy heuristic such as

- largest bribe/duration ratio first
- earliest end time first
- shortest duration first
- largest bribe first

etc

when building a feasible solution (10 marks for this answer)

Stronger answer: find feasible solutions using several different heuristics, and choose the best one. (11 marks for this answer)

**Cost-so-far** (5 marks)

For a partial solution P, the cost-so-far is the sum of the bribes for the events that are explicitly excluded by decisions in P

**Guaranteed-future-cost** (10 marks)

Trivial answer: guaranteed-future-cost = 0 (3 marks for this answer)

Acceptable answer: guaranteed-future-cost = sum of all remaining events that overlap with events already selected. (6 marks for this answer)

Strong answer: guaranteed-future-cost = the acceptable answer, plus, after eliminating all events that overlap with selected events, look at sets of events that are restricted based on party affiliation. For example, if k “Red” events have been selected, where 0 <= k <= 2, then at most x = k – 2 more “Red” events can be selected. If there are n > x remaining “Red” events after all the time-overlapping events have been eliminated, then we can add the bribes for the n-x lowest-valued remaining “Red” events to our guaranteed-future-cost.

Or

Strong answer: guaranteed-future-cost = the acceptable answer, plus, after eliminating all events that overlap with selected events, look for disjoint pairs of events that overlap with each other. We can add the lower of the two bribes in each pair to guaranteed-future-cost.

(10 marks for either strong answer, or something similar)

Industrial strength answer: guaranteed-future-cost = the acceptable answer, plus a combination of both
strong answers. This is best done by subtracting the guaranteed costs off the events in the various sets that are identified: this allows both techniques to be applied without any double-counting of future costs. (12 marks for this answer)

**Feasible-future-cost: (10 marks)**

*Trivial answer: sum of the bribe values of all remaining events*  
(3 marks for this answer)

*Acceptable answer: randomly select some of the remaining events that form a feasible solution, and evaluate its cost.*  
(6 marks for this answer)

*Strong answer: use a greedy heuristic to try to find a good feasible solution (as described under Global Upper Bound)*  
(10 marks for this answer)

*Stronger answer: use a variety of methods to generate feasible solutions and choose the best one.*  
(11 marks for this answer)