Massachusetts Institute of Technology

6.034 Artificial Intelligence

Solutions # **6** 6034 Item # **13**

Problem 1

The constraint graph for the `*trivial-map*` problem is illustrated below:

```
A(var 0)--------B(var 1)
   |   \       /  |
   |    E(var 4)  |
   |   /       \  |
D(var 3)--------C(var 2)
```

Each link represents a `not-same` constraint between adjacent regions in the map. Recall that the links in the diagram above are encoded as directed arcs and so we have two arcs for each link in the graph.

Running `min-conflicts-hill-climb` produces a variety of outcomes depending on the random initial assignment. Here's one that worked the first time:

```
Random assignment:
Var: a Value: yellow
Var: b Value: red
Var: c Value: yellow
Var: d Value: red
Var: e Value: blue
There are 0 vars involved in conflicts
Terminated succesfully after 0 iterations.
Var: a Value: yellow
Var: b Value: red
Var: c Value: yellow
Var: d Value: red
Var: e Value: blue
```

Here's another with a "single" violation – B and C are both blue. This shows up as two conflicts since two variables (and two directed arcs) are involved. Variable C is chosen and a value picked that minimizes violations (to zero).

```
Random assignment:
Var: a Value: red
Var: b Value: green
Var: c Value: blue
Var: d Value: blue
```

```
Var: e Value: yellow
There are 2 vars involved in conflicts
2 (c)=blue  :3 (d)=blue 3 (d)=blue
3 (d)=blue  :2 (c)=blue 2 (c)=blue
0: 2 conf. vars, set var 2 (c) from blue to red
There are 0 vars involved in conflicts
Terminated succesfully after 1 iterations.
Var: a Value: red
Var: b Value: green
Var: c Value: red
Var: d Value: blue
Var: e Value: yellow
```

Here's a really bad initial assignment where every variable is involved in some conflict. This one requires three changes to variables (D, C, and A).

```
Random assignment:
Var: a Value: yellow
Var: b Value: yellow
Var: c Value: green
Var: d Value: yellow
Var: e Value: green
There are 5 vars involved in conflicts
0 (a)=yellow  :3 (d)=yellow 1 (b)=yellow 3 (d)=yellow 1 (b)=yellow
1 (b)=yellow  :0 (a)=yellow 0 (a)=yellow
2 (c)=green  :4 (e)=green 4 (e)=green
3 (d)=yellow  :0 (a)=yellow 0 (a)=yellow
4 (e)=green  :2 (c)=green 2 (c)=green
0: 5 conf. vars, set var 3 (d) from yellow to red
There are 4 vars involved in conflicts
0 (a)=yellow  :1 (b)=yellow 1 (b)=yellow
1 (b)=yellow  :0 (a)=yellow 0 (a)=yellow
2 (c)=green  :4 (e)=green 4 (e)=green
4 (e)=green  :2 (c)=green 2 (c)=green
1: 4 conf. vars, set var 2 (c) from green to blue
There are 2 vars involved in conflicts
0 (a)=yellow  :1 (b)=yellow 1 (b)=yellow
1 (b)=yellow  :0 (a)=yellow 0 (a)=yellow
2: 2 conf. vars, set var 0 (a) from yellow to blue
There are 0 vars involved in conflicts
Terminated succesfully after 3 iterations.
Var: a Value: blue
Var: b Value: yellow
Var: c Value: blue
Var: d Value: red
Var: e Value: green
```

Problem 2

Here's an edited transcript of one of the failures. What happens is that it falls into a "local minimum," a labeling such that any individual change leaves the number of conflicts about the same. In order to get out of this hole, it would have to change a lot of the labels in a coordinate fashion at once. But, it gets no information as to how to do this by changing one label at a time.

```
Random assignment:
Var: a Value: (neg neg neg)
Var: b Value: (in out pos)
Var: c Value: (pos pos neg)
Var: d Value: (pos pos neg)
Var: e Value: (out in)
Var: f Value: (in pos)
Var: g Value: (neg in)
0: 7 conf. vars, set var 6 (g) from (neg in) to (out neg)
1: 7 conf. vars, set var 1 (b) from (in out pos) to (in out pos)
2: 7 conf. vars, set var 6 (g) from (out neg) to (out in)
3: 7 conf. vars, set var 3 (d) from (pos pos neg) to (pos pos neg)
4: 7 conf. vars, set var 0 (a) from (neg neg neg) to (neg neg neg)
5: 7 conf. vars, set var 2 (c) from (pos pos neg) to (pos pos neg)
6: 7 conf. vars, set var 5 (f) from (in pos) to (in pos)
7: 7 conf. vars, set var 0 (a) from (neg neg neg) to (neg neg neg)
8: 7 conf. vars, set var 6 (g) from (out in) to (in pos)
9: 7 conf. vars, set var 1 (b) from (in out pos) to (pos pos neg)
10: 6 conf. vars, set var 6 (g) from (in pos) to (in pos)
11: 6 conf. vars, set var 4 (e) from (out in) to (in pos)

;; At this point it is settled into a local minimum.  Everything that follows
;; is just flipping the labels for E, F and G between two values
;; (in pos) and (pos out).  The values of A, B, C, D are not changed.

12: 6 conf. vars, set var 5 (f) from (in pos) to (in pos)
13: 6 conf. vars, set var 2 (c) from (pos pos neg) to (pos pos neg)
14: 6 conf. vars, set var 4 (e) from (in pos) to (in pos)
15: 6 conf. vars, set var 3 (d) from (pos pos neg) to (pos pos neg)
16: 6 conf. vars, set var 1 (b) from (pos pos neg) to (pos pos neg)
17: 6 conf. vars, set var 6 (g) from (in pos) to (pos out)
18: 5 conf. vars, set var 5 (f) from (in pos) to (in pos)
19: 5 conf. vars, set var 5 (f) from (in pos) to (pos out)
20: 5 conf. vars, set var 3 (d) from (pos pos neg) to (pos pos neg)
21: 5 conf. vars, set var 2 (c) from (pos pos neg) to (pos pos neg)
22: 5 conf. vars, set var 3 (d) from (pos pos neg) to (pos pos neg)
23: 5 conf. vars, set var 4 (e) from (in pos) to (in pos)
24: 5 conf. vars, set var 4 (e) from (in pos) to (in pos)
25: 5 conf. vars, set var 5 (f) from (pos out) to (in pos)
26: 5 conf. vars, set var 2 (c) from (pos pos neg) to (pos pos neg)
27: 5 conf. vars, set var 2 (c) from (pos pos neg) to (pos pos neg)
28: 5 conf. vars, set var 2 (c) from (pos pos neg) to (pos pos neg)
29: 5 conf. vars, set var 5 (f) from (in pos) to (in pos)
30: 5 conf. vars, set var 3 (d) from (pos pos neg) to (pos pos neg)
31: 5 conf. vars, set var 4 (e) from (in pos) to (pos out)
32: 5 conf. vars, set var 4 (e) from (pos out) to (pos out)
    ... much more of the same ...
483: 5 conf. vars, set var 1 (b) from (pos pos neg) to (pos pos neg)
484: 5 conf. vars, set var 4 (e) from (pos out) to (pos out)
485: 5 conf. vars, set var 3 (d) from (pos pos neg) to (pos pos neg)
486: 5 conf. vars, set var 3 (d) from (pos pos neg) to (pos pos neg)
487: 5 conf. vars, set var 3 (d) from (pos pos neg) to (pos pos neg)
488: 5 conf. vars, set var 6 (g) from (pos out) to (pos out)
489: 5 conf. vars, set var 4 (e) from (pos out) to (pos out)
490: 5 conf. vars, set var 6 (g) from (pos out) to (pos out)
491: 5 conf. vars, set var 1 (b) from (pos pos neg) to (pos pos neg)
492: 5 conf. vars, set var 1 (b) from (pos pos neg) to (pos pos neg)
493: 5 conf. vars, set var 6 (g) from (pos out) to (in pos)
494: 5 conf. vars, set var 6 (g) from (in pos) to (pos out)
```

```
495: 5 conf. vars, set var 6 (g) from (pos out) to (in pos)
496: 5 conf. vars, set var 6 (g) from (in pos) to (in pos)
497: 5 conf. vars, set var 4 (e) from (pos out) to (in pos)
498: 6 conf. vars, set var 6 (g) from (in pos) to (pos out)
499: 5 conf. vars, set var 3 (d) from (pos pos neg) to (pos pos neg)
Terminated after 500 iterations with 5 conflicted vars.
Var: a Value: (neg neg neg)
Var: b Value: (pos pos neg)
Var: c Value: (pos pos neg)
Var: d Value: (pos pos neg)
Var: e Value: (in pos)
Var: f Value: (in pos)
Var: g Value: (pos out)
```

Problem 3

The basic idea of weighting the conflicts is to try to avoid loops like the one we saw above. The idea is that if we repeatedly get the same arcs violated, then their weight will go up. When we consider choosing a value, we pick the one that minimizes the **weighted** violations. So, violations to these "difficult" constraints will be more expensive and we will choose values that fix those constraints preferentially. However, this strategy is not guaranteed. It is still possible to fall into loops where the weights of some set of arcs continues to rise and we still do not get anywhere. However, this weighting strategy helps in the great majority of the cases.

Problem 4

Part A

The thing that make line-labeling easy for backtracking and hard for the hill-climbing method is that the variables (at least within one object) are tightly coupled. That is, making a change to one junction's labels tends to require changing most of the labels on the object. If you think about it, changing a label on a junction is like "flipping the object inside out." These tight constraints are great for backtracking since they quickly cut out possibilities once a choice is made.

This is also the reason that course scheduling is hard, moving one course can have effects that propagate a long way. However, one can do much better with course scheduling and hill climbing by starting with a plausible schedule (see below).

Part B

Generally, in a problem like labeling the cube, the hill climbing method falls into a loop beyond the first few cycles. So, adding more cycles does nothing. Starting at another random initial assignment is much more likely to succeed. The odds of it failing two or more times in a row from random starts drops very fast (as the product of the failure probability).

Part C (Optional)

One general way of picking an initial assignment is to pick some order for the variables and assign them one of the values that minimize the number of conflicts with the previously assigned values.

In some cases, even simpler heuristics help, for example, in n-queens pick random starting positions but try to start each queen on its own row. This is a weak version of the more general method above and it produces a slight improvement.