Graduate Artificial Intelligence 15-780

Homework 3: *Heuristics in Search*

Out on February 15
Due on March 1

Problem 1: Extending Your SAT Solver (35 points)

In the last homework, you constructed both a SAT Solver and a random *3SAT* generator. In this problem you will extend your SAT solver, which we'll call S.

- As in the last homework, generate 25 random *3SAT* instances for $n = 50$ and $c = \{150, 170, \ldots, 350\}$. These 275 problem instances will be your *testing battery*.
- Implement trivial backjumping into your SAT solver. *Trivial* means that your SAT solver should jump back to the most recently set variable in a node's conflict set only. Call this SAT solver S_T.
- Replace the implementation of trivial backjumping in S_T with conflict-directed backjumping. Call this SAT solver S_B.
- Augment your SAT solver with conflict-directed backjumping S_B with clause learning. Call this SAT solver S_C.
- **Extra Credit - 15 points**. GRASP (Marques-Silva and Sakallah 1999, available at `http://goo.gl/krcIN`), is a SAT solver that uses advanced notions of clause learning and backjumping. Implement the GRASP solver and call it S_G.
- Time $\{S, S_T, S_B, S_C, S_G\}$ against your testing battery.

You should hand in plots, tables, and discussions of the performance of the SAT solvers you made. Which solver ran the best? Why do you think this is? Did c make a difference in your results?

We do not require you to submit source code, but we reserve the right to ask for it. (If you do choose to do the extra credit, please attach a well-commented printout of your implementation to the problem set.)

Problem 2: You're the Only Ten I See (65 points)

Every state receives a certain number of delegates to the US Congress based on their population. *Congressional districting* is the process of dividing a state into districts, each of which will be represented by a single congressperson living in that district. The US Code mandates that districts should be contiguous and made as equal as possible in terms of population size. In this problem, you will use local search to divide the state of Tennessee into contiguous, compactly-shaped districts of approximately equal population.

The basic unit of area in this problem will be *tracts*, which are contiguous groups of residences where about five thousand people live. For districting in practice it is possible to go down to the level of individual blocks to make districts as equal-sized as possible. However for this problem the coarseness of the tract-level data we are using means a good solution will see a discrepancy between district sizes of a few thousand people.

The 2000 Census counted 5,689,283 people living in Tennessee, and the state was allocated nine districts. Consequently the average number of people per district is $\bar{p} \equiv 632,142.56$.

Data files:

Tennessee had 1,261 census tracts in the 2000 Census. The relevant information for each tract is given in two files, `tracts.txt` and `neighbors.txt`.

`tracts.txt` has 1,261 lines of the form

```
id population x y
```

Where $id \in \{0, \ldots, 1260\}$ is a unique integer ID for each tract, $population$ is the number of people living in the census tract, and x and y are the longitude and latitude of the center of the tract. (These values will be useful for visualizing your solutions.)

`neighbors.txt` has the adjacency information for the tracts. It has 1,261 lines of the form

```
id neighbor neighbor neighbor ...
```

or, in other words, a list of an integer ID followed by the integer ID of every tract adjacent to it. **This is the adjacency information to use in this problem. Do not use a custom solution based on the longitude and latitude values.**

Introduction (15 points):

As an introduction to local search, you will compare hill climbing and stochastic hill climbing. Algorithms 1 and 2 give the pseudocode for the maximization versions of these search methods.

```
current = StartNode();
best = current;
for Some number of iterations do
    neighbors = GetNeighbors(current);
    sort neighbors by score descending;
    current = neighbors.pop;
    if score(current) > score(best) then
        best = current;
    end
end
return best;
```

Algorithm 1: Hill Climbing

```
current = StartNode();
best = current;
for Some number of iterations do
    neighbors = GetNeighbors(current);
    sort neighbors by score descending;
    while rand() < δ and !neighbors.empty do
        current = neighbors.pop;
    end
    if score(current) > score(best) then
        best = current;
    end
end
return best;
```

Algorithm 2: Stochastic Hill Climbing

There are several functions that define these algorithms that are left for you to write.

- StartNode() should return a random start node. One solution for this is to assign every tract to a single district, and then re-assign a random collection of 8 tracts to each other district, while taking care that your assignments will not violate contiguity of the large district.
- GetNeighbors($node$) should return the local neighborhood of a search node. One solution for this is to find the smallest district and allocate it one of its adjoining tracts.
- score should return the score of a node. For this part, use the *negative variance of district populations*, i.e., letting $p(d_i)$ represent the population of district d_i then the score of a node is

$$-(1/9)\sum_i (p(d_i) - \bar{p})^2.$$

  With this objective function, the upper bound on node score is zero.

Regarding contiguity. A *contiguous* district forms a connected component, where vertices are tracts and edges are based on the given adjacencies. Your solution to the problem *must* be contiguous. You are very strongly advised to never consider non-contiguous solutions in your search. Your solution should include a function that checks to see if a prospective collection of tracts would form a contiguous district.

Implement Hill Climbing and Stochastic Hill Climbing with $\delta = .75$, using the starting node/neighborhood suggestions we have made or come up with your own. Regardless, be sure to fully describe the details of your

implementation. Plot the mean and median scores of the best node found from 101 runs over their first 500 steps. How similar are the means and medians? Why? Which algorithm performs better? Why do you think that is? Are there prominent differences between Hill Climbing and Stochastic Hill Climbing in terms of which neighbors are expanded? Did you notice any problems with local optima in Hill Climbing? Why do you think that is?

Competition (25 points):

Consider the more-sophisticated objective function

$$\text{obj}(\vec{d}) \equiv -100 \sum_i \left(\frac{|p(d_i) - \bar{p}|}{\bar{p}} \right)^2 + \sum_i cc(d_i)$$

Where cc is the *clustering coefficient* of a district, the fraction of possible adjacencies between the tracts of d_i. Formally, let $|t_j|$ be the number of adjacent tracts to tract j, and let $|t_j^i|$ be the number of adjacent tracts to tract j in district i. Then

$$cc(d_i) = \frac{\sum_{t_j \in d_i} |t_j^i|}{\sum_{t_j \in d_i} |t_j|}$$

Find a solution to the districting problem with maximum score for this new objective by using local search. You should feel free to update any part of the local search method used in the previous part: the starting node, the neighborhood definition, or the neighbor selection routine. Here are some ideas you might consider:

- Find good starting nodes by separating out the initial tracts of a district based on geographic, adjacency, or some kind of population-weighted distance.
- Expand your neighborhood definition to include swaps and multiple simultaneous moves.
- Try a more-advanced search method like *Tabu search* (which keeps a fixed-length queue of recently-visited nodes and will not revisit those nodes again) or *Beam search* (which considers a set of nodes at each step, and where the nodes for the next iteration are taken from the union of the neighborhoods of the nodes in the current step).

Submit your solution by putting a file named `districts.txt` into your dropbox. The file `districts.txt` should consist of 1,261 lines of the form

```
id district
```

where *id* is the unique tract integer id and *district* is an integer 1 through 9.

For this part of the problem you will be graded based on the evaluation of the objective function obj on the solution you submit. Let $s = \text{obj}(\vec{d})$ be the score for the solution you submit:

- If $s \geq 7$, you will receive full credit.
- If $5 \leq s < 7$, you will receive 20 points.
- If $0 \leq s < 5$, you will receive 15 points.
- If $-10 \leq s < 0$, you will receive 10 points.
- If $s < -10$, or if your solution does not have contiguous districts, you will not receive any points for this section.

Additionally, the **top five submissions** in terms of score will receive $6 - rank$ extra credit points. The TAs were able to quickly code up a solution with a score of about 7.7; we expect the winning score to be around 8.

Your solutions should be entirely algorithmic. Attach your source code for this part of the problem to your homework. Clearly note your implementations of StartNode() and GetNeighbors($node$).

Writeup (25 points):

In addition to your writeup from your hill climbing implementation, discuss how you solved the optimization problem. We are as much interested in what worked as what didn't. Which of the three parts of the local search algorithm (start node, neighborhood definition, selection criteria) did you find was most important to improving your solution quality?

Also, please consider the following questions in your writeup. We do not expect exact solutions to them.

- What is the size of the search space (How many nodes are there)?
- How hard (from a complexity standpoint) is this problem?
- How would you translate this problem into a SAT instance?
- California is the most populous state; it has 53 congressional districts. How well do you think your approach would scale to districting California?
- Why do you think we did not want you to look at non-contiguous prospective solutions?
- Can you think of ways other than the clustering coefficient to measure the compactness of a district? If all the tracts are roughly the same size and shape, can you come up with a district that has very high clustering coefficient but that would not be considered compact?

Extra Credit Come up with a good way of visualizing your solution. The most innovative and attractive solution will receive **5 points** of extra credit, and the second- and third-best will receive **3 points** of extra credit. Attempting any kind of visualization will earn **1 point** of extra credit.