

This homework is due Sep 13 in class.

1 Ant robot teams for terrain coverage (20 points)

Consider the problem of designing a team of ant robots to map or monitor a given terrain. We consider the problem of one-time coverage (every cell is visited by at least one robot in the team) which is required for exploration. The robots do not know the terrain, have limited memory and cannot maintain maps of the entire terrain. They do not have the computational power to calculate complete paths. The terrain may change during exploration (rocks or furniture may move around). In addition, these robots are small enough that they may be picked up and moved to a different part of the terrain.

For simplicity we consider terrains modeled as $n \times n$ grids with some squares blocked off as obstacles. We will also assume that every unblocked square is reachable from all other unblocked squares. That is, the terrain modeled as a graph, is fully connected. The goal of the robot team is to visit every unblocked grid square at least once. Consider the following algorithm inspired by pheromone-trail laying behavior of real life ants. Let S denote the set of unblocked grid squares. Let $A(s)$ denote the Manhattan neighbors of square s , i.e., these are the squares that are reachable in one step from the current square going east, west, north or south. Initially, the amount of pheromone in each of the unblocked squares $s \in S$, $u(s)$, is initialized to zero. Each ant then executes the following.

- a. $s = s_{start}$
- b. $a = \operatorname{argmin}_{a \in A(s)} u(\operatorname{succ}(s, a))$ (break ties randomly)
- c. $u(s) = u(\operatorname{succ}(s, a)) + 1$
- d. Execute a and move to square $\operatorname{succ}(s, a)$
- e. Go back to step b.

Each ant has sensors to read the pheromone levels u at all neighboring squares accurately. The ants have effectors to increment the pheromone level in the square they are currently on. They move reliably to the intended square, i.e., step d. works without any error or failure. By laying trails, ants implicitly coordinate their exploration of the grid. Ant teams are fault tolerant (the terrain can be covered even if some ants fail), and they exploit parallelism (a group of ants can cover terrain faster than a single ant).

In this problem you need to show that the above algorithm, when executed by a team of $m \geq 1$ robots,

- (10 points) will in a finite amount of time, visit every square in a fully connected terrain graph S at least once.
- (10 points) will achieve the property above even if the robots are moved by an external agency, or the terrain changes during exploration.

You may assume that all robots in a team can occupy the same square at once and that they do not perceive each other as obstacles. You can also assume that the robots move in a specific, fixed, sequential order.

2 Optimal compact disk packing (20 points)

Have you seen the TV ads for themed compact disks like the “best of the 80s” usually aired in the wee hours of the morning? I have always wondered how they decide which selections to include in those CD sets. Well, it turns out we can model this problem as an optimization problem. You have a set of CDs to fill from a large collection of songs which belong to a chosen theme. Each song has a popularity score associated with it. Your goal is to select songs for that theme from the given collection to maximize the sum of popularity scores. Obviously, you are limited by the size of each CD. We can mathematically formulate this problem as follows: given a set S of n songs each of size l_i and popularity score v_i , $1 \leq i \leq n$, and a CD with storage capacity C , how do we select a subset T of songs from S which maximizes $\sum_{j \in T} v_j$ such that $\sum_{j \in T} l_j \leq C$. Assume that the sizes $l_i, 1 \leq i \leq n$ and CD capacity C are positive integers.

- (15 points) Present a solution to this constrained optimization problem using dynamic programming. What is the time and space complexity of your solution?
- (5 points) Present a heuristic solution to the problem with lower time and space complexity than dynamic programming.

3 Analyzing Search Algorithms (30 points)

Consider a search space where we might use A^* to find shortest paths between given initial and goal states. We have a cost function $g(n)$ that evaluates the distance from the start node to a node n in the search space, and an admissible heuristic h that estimates the distance between a node n and the closest goal. Assume that the resulting function $f(n) = g(n) + h(n)$ is *strictly monotonic*, that is, if node n' is a successor of n , then $f(n') > f(n)$.

We will consider a popular alternative to A^* called depth-first branch-and-bound search (BBS). This algorithm is also used to find the shortest path from an initial state to a goal state. Unlike A^* , however, BBS is a depth-first search. Thus, it is only used when it is reasonable to assume that the search tree is finite, i.e., there are no infinite paths in the search tree. BBS maintains the f value of the best goal node found so far. It does a depth-first search, backtracking whenever it encounters a node whose value is larger than its current bound (i.e., f value of the best goal node found so far). The algorithm is described below. For simplicity of presentation, the procedure only finds the cost of the optimal goal node; it does not store the path to the goal node. The algorithm has a single global variable called `upper_bound` initialized to ∞ . It is invoked as `BBS(initial_state)`. The search terminates when all paths have been explored or pruned; the cost returned by BBS is the final value of `upper_bound`.

```
BBS(node) {
    // pruning
    if (f(node) >= upper_bound) {
        return;
```

```

}
// goal test
if (isGoal(node)) {
    upper_bound = min(upper_bound, f(node));
} else { // DFS
    for each s in successors(node){
        BBS(s);
    }
}
return;
}

```

- a. (5 points) Show that BBS is complete.
- b. (5 points) Show that BBS is optimal.
- c. (10 points) Show that BBS will never prune a node that A* expands. We say that a node n is pruned if the search backtracks before reaching n . Assume that both algorithms use the same h function.
- d. (10 points) In (c) you showed that A* never expands more nodes than BBS. However, BBS is widely used in industry. State two reasons why BBS might be preferred to A*.

4 Designing Search Algorithms: Protein Folding (20 points)

Proteins, which are initially synthesized in the cell as long chains of amino acid building blocks, automatically fold down into more compact working shapes. Exactly how they do this, and how they do it quickly and consistently, is poorly understood. The problem is so notoriously difficult that even dramatic simplifications are difficult to solve.¹

One stripped-down version of protein-folding is the so-called H/P (hydrophobic/polar) lattice model introduced by Lau and Dill in 1989, which we will use in this problem. In the H/P model, there are only two kinds of amino acids: hydrophobic amino acids (red in figures below) and hydrophilic or polar amino acids (blue in figures below). We will represent amino acid sequences in the binary H/P alphabet, using 1 to denote hydrophobic residues and 0 for the polar ones. For example, the amino acid chain in Figure 1 is represented by the sequence [0, 1, 1, 1, 1, 0].

Since proteins exist in a watery environment, the hydrophobic amino acids prefer to cling to each other, compact and apart from the water as much as possible. We call a folded configuration of the protein a conformation. We will consider conformations in two dimensions, treating amino acids as links in a chain. Each link may bend by a multiple of 90 degrees, but the distance between links may not change (and is taken to be 1). No self-intersections are permitted, and the final conformation is constrained to lie on a single plane. A conformation of a protein is thus a self-avoiding walk on a two-dimensional square lattice or a Manhattan grid.

We represent the conformation of an amino acid chain of length n by a sequence of length $n - 1$ as follows. We use complex numbers 1 for east, i for north, -1 for west, and $-i$ for south. Each element

¹Some resources on the real science of protein folding are <http://folding.stanford.edu/science.html> and <http://www.people.virginia.edu/~rjh9u/protfold.html>.

k of the conformation represents the direction of the link joining the k^{th} and $k + 1^{st}$ amino acids in the chain. The straight line conformation of the chain to the left in Figure 1 is represented by the sequence $[1, 1, 1, 1, 1]$, the middle conformation is $[1, 1, i, i, i]$ while the conformation to the right is $[1, 1, i, -1, -1]$.

We can calculate the locations of the individual proteins on the amino acid chain from its H/P sequence and its conformation. The chain $[0, 1, 1, 1, 1, 0]$ with conformation $[1, 1, 1, 1, 1]$ has its residues at location vector $[0, 1, 2, 3, 4, 5]$, while the same chain with conformation $[1, 1, i, i, i]$ has residues at $[0, 1, 2, 2 + i, 2 + 2i, 2 + 3i]$.

Each conformation of the protein sequence is associated with a free energy. The free energy of a conformation is calculated by adding the distances between all pairs of hydrophobic amino acids. The free energy associated with the conformation on the left of Figure 1 is 10, with the conformation in the center is $5 + \sqrt{2} + \sqrt{5}$ and that of the right is $4 + 2\sqrt{2}$. Given an amino acid sequence of a protein in the H/P alphabet, the task is to determine an *optimal* conformation; that is, one with the lowest free energy.

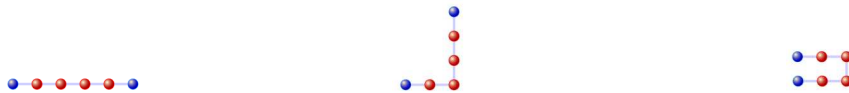


Figure 1: The amino acid chain to the left gets folded in two steps into an optimal conformation.

Figure 4 shows near-optimal folds for three interesting amino acid chains.

- (5 points) Formulate the problem of finding the minimal energy configuration of an amino acid sequence as a state space search problem. That is, identify the state space, the initial states and goal states, the successor function, and the path cost function.
- (5 points) What family of search algorithms are suitable for solving the problem of finding minimal free energy configurations? Why?
- (10 points) Design an effective search algorithm for finding good solutions to this folding problem. Present your algorithm in pseudocode. What is the time and space complexity of your algorithm? Is your algorithm guaranteed to find an optimal solution? Explain your answers.

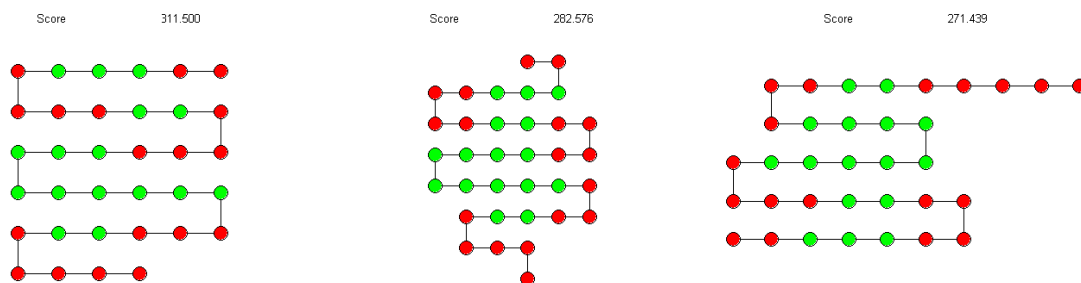


Figure 2: Three examples of folded amino acid chains.