Computational Biology: Assignment #5

Due on Monday, Mar 24, 2014

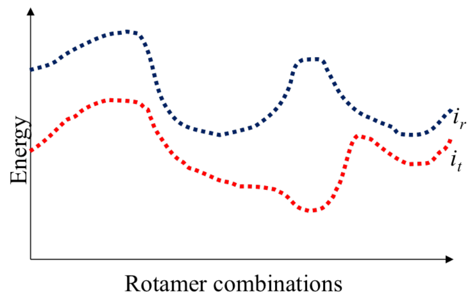
Jianyang Zeng 1:30pm

Weiyi Chen

Problem 1

Dead End Elimination (DEE) Pruning

(1) Prove the theorem



Assume by contradiction that rotamer i_r is part of optimal solution in some case, namely there exist some combination of j such that

$$\sum_j [E(i_r, j_s) - E(i_t, j_s)] < 0$$

This implies that

$$\min_s \sum_j [E(i_r, j_s) - E(i_t, j_s)] < 0$$

However, it is easy to figure out

$$\min_s \sum_j [E(i_r, j_s) - E(i_t, j_s)] \geq \sum_j \min_s [E(i_r, j_s) - E(i_t, j_s)]$$

This is because in the left part, the combination of j is defined before deriving minimal, which is possibly to exist. But in the right part of the inequality, it is deriving the minimal of each j_s before summing up. So there're some cases, like different j_s in each element of the sum is not possible to exist at the same time. In other words, cases in the right part cover the left part, so it's able to get a smaller value.
Since

$$\sum_j \min_s [E(i_r, j_s) - E(i_t, j_s)] > 0$$

We have

$$\min_s \sum_j [E(i_r, j_s) - E(i_t, j_s)] > 0$$

which contradicts to our assumption.

(2) Analyze the time complexity

Let N be the number of amino acid positions, as above, and let p be the number of rotamers at each position (this is usually, but not necessarily, constant over all positions). Given this model, it is clear that the DEE algorithm is guaranteed to find the optimal solution; that is, it is a global optimization process. The single-rotamer search scales quadratically in time with total number of rotamers. The pair search scales cubically and is the slowest part of the algorithm (aside from energy calculations). This is a dramatic improvement over the brute-force enumeration which scales as $O(p^N)$.

Problem 2

A* Search

Problem 2 continued on next page. . .

(1) Prove the theorem

If the heuristic function h is admissible, meaning that it never overestimates the actual minimal cost of reaching the goal, then A* is itself admissible (or optimal) if we do not use a closed set. If a closed set is used, then h must also be monotonic (or consistent) for A* to be optimal. This is to prove, problems with a consistent heuristics cannot be solved any better than A* can solve them by algorithms with the same information.
Defined by a quadruple: $I = (G, s, \Gamma, h)$, where

- G is the graph representation of the problem
- s is the start node in the graph
- Γ is the set of goal nodes
- h is a heuristic that any algorithm run on this instance will use

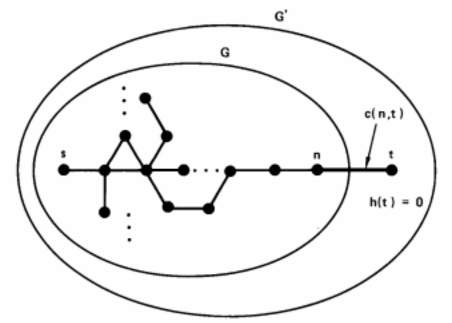
Given a set of problem instance $I = (G, S, \Gamma, h)$, and assuming n is surely expanded by A*, then there exists a path P_{s-n} s.t.

$$\forall n' \in P_{s-n}, g(n') + h(n') < C*$$

where

- $P_{n_i - n_j}$ is a path in G between the node n_i and n_j
- $g(n)$ is the sum of the branch costs along the current path of pointers from n to s
- $f(.)$ is the evaluation function de?ned over partial paths, i.e., to to each node n along a given path $P = s_1, n_1, n_2, ..., n$ we assign the value $f_P(n)$ which is shorthand notation for $f(s, n_1, n_2, ..., n)$
- C* is the cost of the cheapest solution path

Let B be an algorithm compatible with A* that halts with cost C* in I. Assume that B does not expand n. We can then create G'. What we do here is setup a contradiction: For B to be better than A*, it must skip some node n that A* visits. We assume B does this, and setup a new graph G' that will introduce a contradiction.



where we have added a goal node t to G. The costs of t is given by $h(t) = 0$ and the edge from n to t is given as $c = h(n) + \Delta$ where:

$$\Delta = \frac{1}{2}(C^* - D) > 0$$

$$D = \max\{f(n') | n' \in N_{g+h}^{G^*}\}$$

This creates a new path P* whose cost is $\leq C^* - \Delta$ yet is still consistent (and admissible) on the new I'. Here, we setup a new node in G0 and make sure it has the proper costs associated with it so that the new G' obeys all the rules that the old G did.
It remains to be proven that h is consistent on I' for the new node t in G' (this is trivially true for all the

previous nodes since the h values of all the nodes in G remain unchanged). This is done by establishing $h(n') \leq k(n', t) \forall n' \in G$. At any node n' we should also have

$$h(n') > k(n', n) + c = k(n', n) + h(n) + \Delta$$

We show that by using the above weight for the $k(n', t)$ edge, the new graph G' remains consistent. Thus A* will ?nd the new path P* (which costs $C^* - \Delta$) because

$$f(t) = g(n) + c = f(n) + \Delta = D + \Delta = C^* - \Delta < C^*$$

So t is reachable from s by a $C^* - \Delta$ bounded path, so it will be selected.
But algorithm B must behave the same as if it were running on I, halting with cost $C^* > C^* - \Delta$ This is a contradiction that B is both admissible on I, and avoids the expansion of n.
We have a contradiction, because after creating G', which has a shortest path goal node attached to n (which A* finds properly), B is unable to find a shortest path and therefore must not be admissible.

(2) What is the relationship between Dijkstra and A* search algorithms?

The A* algorithm is a generalization of Dijkstra's algorithm that cuts down on the size of the subgraph that must be explored, if additional information is available that provides a lower bound on the "distance" to the target. This approach can be viewed from the perspective of linear programming: there is a natural linear program for computing shortest paths, and solutions to its dual linear program are feasible if and only if they form a consistent heuristic (speaking roughly, since the sign conventions differ from place to place in the literature). This consistent heuristic defines a non-negative reduced cost and A* is essentially running Dijkstra's algorithm with these reduced costs.