Success Facilitation Survey: Unit 1.

CSC 242

May 2004

*Write your **NAME** legibly on this paper. Open book, open notes.*

You're in a startup company working on a **shortest-time** algorithm for driving in Rochester to bundle with a global positioning system. What path from A to B is shortest in time? You want to extend heuristic search to deal with "chance nodes" like traffic lights.

- **1.** What is an admissible heuristic without "chance nodes?"
- CB's quick answer: Everyone's favorite, the straight line distance from where you are to the goal, divided by the maximum speed limit, say 30 mph.
- **2.** Can you do better? I.e. can you write another admissible heuristic that dominates the first one?
- CBans: Run a quick shortest path algorithm like Floyd from where you are to the goal, use that distance instead of crow-flight distance. It would under-estimate since turns take time, there would be traffic lights, maybe ignore one-way signs, etc.
- **3.** How do you incorporate chance nodes into your heurisic and what are the issues?
- CBans: For every forseeable chance situation (forseeable ones would be traffic lights on the route, or schools at certain times of day, or garbage trucks on certain days), add an "operation" that takes some random amount of time distributed in some known distribution. Then one can do an "expectiA*" algorithm if you compute the expected time cost at each such spot.

  One Fatal problem: if you get really lucky there will be no such delays and the only to under-estimate the delays is to ignore them! So it seems to me if there are no chance delays of guaranteed greater-than-zero value, you can miss an optimal route. Easy to imagine wanting to avoid a shorter route with lots of traffic lights....but what if they were all green for you?

  So one would need to abandon "optimality" and go for something like "expected goodness" I guess.

You find a function purporting to give you a very accurate time to drive from point a to b in Rochester. It starts out

$$T(a,b) = (x_0 - x_3) + (x_1 - x_4) + \log(x_6^{\sin(x_1/x_2)}) + x_0^2 x_5^{3/2} + \sqrt{x_2} \mid \frac{x_1}{x_0 x_2} \mid -...$$

and goes on for some time. $T(a, b)$ is the time from a to b, Variables $x_0, ..., x_{10}$ stand for things like "longitude of a", "latitude of b", "distance on one-way roads", "number of traffic lights", "time of day", "recent snowfall in centimeters", etc.

You want to find the optimum conditions (values of variables) for the fastest drive from some fixed location a_{start} to a fixed destination b_{goal}.

- **4.** How would you set this problem up for genetic search? *I.e.* what are your representation and your mutation and crossover functions?

- CB Ans. Very straightforward, I don't see any reason not to have just 11 numbers representing the 11 variables as the representation. crossover could treat this 11x64 (say) bit string as one thing, with crossover happening anywhere ....snip both strings in the same spot and glue them together. This would be a bit bizarre since you could snip a floating point number in half and you'd pretty massively disturb the mapping of bits onto variables. More natural is swapping variable values (64 bits at a time) so individual values get swapped. Then one can do the clip-in-the-middle-somewhere approach or swap pairs at random thru the list, whatever. Mutation could be adding or subtracting small bits to the numbers. I like the idea of actually doing gradient search from the current state which would mutate the numbers all toward a better solution. But one wants some random mutation too so maybe do a sort of simulated annealing type of mutation that is likely to improve but possibly just kick you away from current position.

- **5.** Nowadays with Redtooth(TM) technology, all the computers in your home, cellphones, PDA, radio, etc. can be hooked together. How would you parallelize your genetic algorithm computation of $T(a, b)$ onto say five computers?

- CBAns:

  Lots of ideas possible here, but basically it seems the most natural one is just to do "the same thing" on all computers and then every N generations pick some number of best representatives from each computer's search (or a stochastic sampling of representatives, with more good ones and some bad ones for genetic diversit). Ship these best representatives over to the other computers where they now start competing with current populations.