CS 188
Spring 2023

Final Review: Search/Logic Solutions

Q1. Power Pellets

Consider a Pacman game where Pacman can eat 3 types of pellets:

- Normal pellets (n-pellets), which are worth one point.
- Decaying pellets (d-pellets), which are worth $max(0, 5 - t)$ points, where t is time.
- Growing pellets (g-pellets), which are worth t points, where t is time.

The location and type of each pellet is fixed. The pellet's point value stops changing once eaten. For example, if Pacman eats one g-pellet at $t = 1$ and one d-pellet at $t = 2$, Pacman will have won $1 + 3 = 4$ points.

Pacman needs to find a path to win at least 10 points but he wants to minimize distance travelled. The cost between states is equal to distance travelled.

(a) Which of the following must be including for a minimum, sufficient state space?

[✓ Pacman's location]
☐ Location and type of each pellet
☐ How far Pacman has travelled
[✓ Current time]
☐ How many pellets Pacman has eaten and the point value of each eaten pellet
[✓ Total points Pacman has won]
[✓ Which pellets Pacman has eaten]

A state space should include which pellets are left on the board, the current value of pellets, Pacman's location, and the total points collected so far. With this in mind:
(1) The starting location and type of each pellet are not included in the state space as this is something that does not change during the search. This is analogous to how the walls of a Pacman board are not included in the state space.
(2) How far Pacman has travelled does not need to be explicitly tracked by the state, since this will be reflected in the cost of a path.
(3) Pacman does need the current time to determine the value of pellets on the board.
(4) The number of pellets Pacman has eaten is extraneous.
(5) Pacman must track the total number of points won for the goal test.
(6) Pacman must know which pellets remain on the board, which is the complement of the pellets he has eaten.

(b) Which of the following are admissible heuristics? Let x be the number of points won so far.

[✓ Distance to closest pellet, except if in the goal state, in which case the heuristic value is 0.]
☐ Distance needed to win $10 - x$ points, determining the value of all pellets as if they were n-pellets.
[✓ Distance needed to win $10 - x$ points, determining the value of all pellets as if they were g-pellets (i.e. all pellet values will be t.)]
☐ Distance needed to win $10 - x$ points, determining the value of all pellets as if they were d-pellets (i.e. all pellet values will be $max(0, 5 - t)$.
☐ Distance needed to win $10 - x$ points assuming all pellets maintain current point value (g-pellets stop

increasing in value and d-pellets stop decreasing in value)

☐ None of the above

(1) Admissible; to get 10 points Pacman will always have to travel at least as far as the distance to the closest pellet, so this will always be an underestimate.
(2) Not admissible; if all the pellets are actually g-pellets, assuming they are n-pellets will lead to Pacman collecting more pellets in more locations, and thus travel further.
(3) Ambiguous; if pellets are n-pellets or d-pellets, Pacman will generally have to go further, except at the beginning of the game when d-pellets are worth more, in which case this heuristic will over-estimate the cost to the goal. However, if Pacman is allowed to stay in place with no cost, then this heuristic is admissable because the heuristic will instead calculate all pellet values as 10. This option was ignored in scoring.
(4) Not admissible; if pellets are n-pellets or g-pellets, Pacman would have an overestimate.
(5) Not admissible; if pellets are g-pellets, then using the current pellet value might lead Pacman to collect more locations, and thus travel further than necesarry.

(c) Instead of finding a path which minimizes distance, Pacman would like to find a path which minimizes the following:

$$C_{new} = a * t + b * d$$

where t is the amount of time elapsed, d is the distance travelled, and a and b are non-negative constants such that $a + b = 1$. Pacman knows an admissible heuristic when he is trying to minimize time (i.e. when $a = 1, b = 0$), h_t, and when he is trying to minimize distance, h_d (i.e. when $a = 0, b = 1$).
Which of the following heuristics is guaranteed to be admissible when minimizing C_{new}?

☐ $mean(h_t, h_d)$ 🟥 $min(h_t, h_d)$ ☐ $max(h_t, h_d)$ 🟥 $a * h_t + b * h_d$
☐ None of the above

For this question, think about the inequality $C_{new} = a * t + b * d \geq a * h_t + b * h_d$. We can guarantee a heuristic h_{new} is admissible if $h_{new} \leq a * h_t + b * h_d$
(1) If $a = b$, $0.5 * h_t + 0.5 * h_d$ is not guaranteed to be less than $a * h_t + b * h_d$, so this will not be admissible.
(2) $min(h_t, h_d) = a * min(h_t, h_d) + b * min(h_t, h_d) \leq a * h_t + b * h_d$
(3) $max(h_t, h_d)$ will be greater than $a * h_t + b * h_d$ unless $h_t = h_d$, wo this will not be admissible.
(4) Admissible.

Q2. Disjunctive Normal Form

A sentence is in disjunctive normal form (DNF) if it is the disjunction of conjunctions of literals. For example, the sentence $(A \wedge B \wedge \neg C) \vee (\neg A \wedge C) \vee (B \wedge \neg C)$ is in DNF.

(a) Any propositional logic sentence is logically equivalent to the assertion that some possible world in which it would be true is in fact the case. From this observation, prove that any sentence can be written in DNF.

Each possible world can be expressed as the conjunction of all the literals that hold in the model. The sentence is then equivalent to the disjunction of all these conjunctions, i.e., a DNF expression.

(b) Construct an algorithm that converts any sentence in propositional logic into DNF. (*Hint*: The algorithm is similar to the algorithm for conversion to CNF.)

A trivial conversion algorithm would enumerate all possible models and include terms corresponding to those in which the sentence is true; but this is necessarily exponential-time. We can convert to DNF using the same algorithm as for CNF except that we distribute $\wedge$ over $\vee$ at the end instead of the other way round.

(c) Construct a simple algorithm that takes as input a sentence in DNF and returns a satisfying assignment if one exists, or reports that no satisfying assignment exists.

A DNF expression is satisfiable if it contains at least one term that has no contradictory literals. This can be checked in linear time, or even during the conversion process. Any completion of that term, filling in missing literals, is a model.

(d) Apply the algorithms in the previous two parts to the following set of sentences:

$$\begin{aligned} A &\Rightarrow B \\ B &\Rightarrow C \\ C &\Rightarrow \neg A \end{aligned}$$

The first steps give

$$(\neg A \vee B) \wedge (\neg B \vee C) \wedge (\neg C \vee \neg A) \ .$$

Converting to DNF means taking one literal from each clause, in all possible ways, to generate the terms (8 in all). Choosing each literal corresponds to choosing the truth value of each variable, so the process is very like enumerating all possible models. Here, the first term is $(\neg A \wedge \neg B \wedge \neg C)$, which is clearly satisfiable.

(e) Since the algorithm in (b) is very similar to the algorithm for conversion to CNF, and since the algorithm in (c) is much simpler than any algorithm for solving a set of sentences in CNF, why is this technique not used in automated reasoning?

The problem is that the final step typically results in DNF expressions of exponential size, so we require both exponential time AND exponential space.