Homework 2

Solution 1:

(a) The set of centers of all maximal disks is called the *medial axis*. In path planning, following the medial axis corresponds to computing a *maximum clearance* path (which stays as far away from obstacles as possible).

(b) (i) A heuristic is *admissible* if it is not greater than the graph distance, that is, $h(u) \leq \delta(u, t)$.

(ii) A heuristic is *consistent* if for all vertices u' and u'', $h(u') \leq \delta(u', u'') + h(u'')$

(iii) The heuristic $h(u) = 3 \cdot \text{dist}(u, t)$ is *neither* admissible *nor* consistent (see Fig. 1).

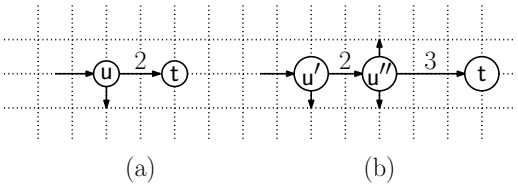

Figure 1: Problem 1(b): A^* heuristic properties.

To see that it is not admissible, consider the graph in Fig. 1(a), or generally any graph where there is an edge between u and t. We have

$$h(u) \;=\; 3 \cdot \text{dist}(u, t) \;=\; 3 \cdot 2 \;>\; 3 \;=\; \delta(u, t).$$

To see that it is not consistent, consider the graph in Fig. 1(b), or generally any graph where the three vertices u', u'', and t are collinear. We have

$$h(u') \;=\; 3 \cdot \text{dist}(u', t) \;=\; 3 \cdot (2 + 3) \;>\; 2 + 3 \cdot \text{dist}(u'', t) \;=\; \delta(u', u'') + h(u'').$$

(c) The *behavior tree* for the hungry student is shown in Fig. 2. The "look in refrigerator" nodes succeed if food is found and fail otherwise. The "try to pay" node succeeds if you have enough money, and fails otherwise.

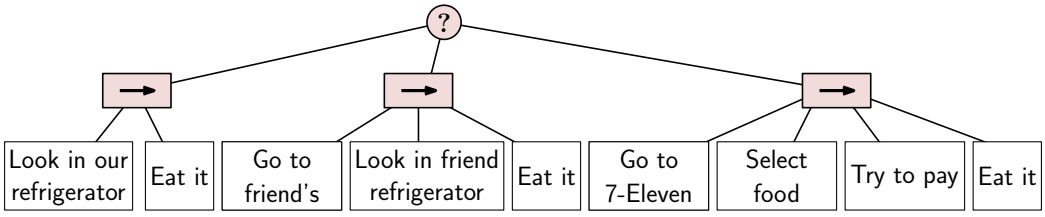

Figure 2: Problem 1(c): Behavior tree.

(d) (i) Sends acknowledgments when packets are received: TCP

(ii) Packets can be ordered through the use of sequence numbers: TCP

(iii) Uses checksums to detect errors in packet transmission: TCP

(iv) Has lower overhead in terms of bandwidth and latency: UDP

(e) According to the lecture, eliminating fog is an example of *Information Exposure*.

Solution 2: The number of triangles is $T(n,h) = n + 2h - 2$. The number of chords is $C(n,h) = n + 3h - 3$.

Extra Credit: Here is a simple proof by a double induction, on n and h. We will build a triangulation up vertex by vertex and hole by hole. (The proof is not fully general, since it builds a specific triangulation, not a general one. A more general proof comes by disassembling an existing triangulation.) For the basis case, we start with a single triangle, that is, $(n,h) = (3,0)$ (see Fig. 3(a)). We have one triangle and no chords. The above formulas yield $T(n,h) = 3 + 0 - 2 = 1$, and $C(n,h) = 3 + 0 - 3 = 0$, as desired.

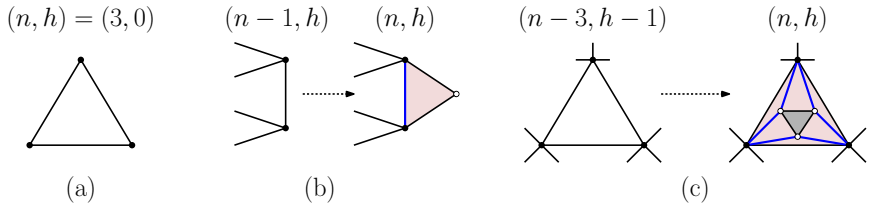

Figure 3: Problem 2: Triangulation size.

There are two ways to add to the triangulation. First, we can add a new "ear" by attaching a new triangle to an edge of the boundary of the polygon (see Fig. 3(b)) or the boundary of some hole. This results in the addition of one new vertex, no new holes, one new chord, and one new triangle. Let n denote the number of vertices *after* the addition. Applying the induction hypothesis we have

$$\begin{aligned} T(n,h) &= 1 + T(n-1,h) = 1 + ((n-1) + 2h - 2) = n + 2h - 2 \\ C(n,h) &= 1 + C(n-1,h) = 1 + ((n-1) + 3h - 3) = n + 3h - 3, \end{aligned}$$

as desired.

Second, we insert a new hole. The most trivial example of a hole is a single triangle added in the interior of some existing triangle and connecting it to the three vertices of the triangle (see Fig. 3(c)). This yields three new vertices, one new hole, six new chords, and a net increase of five triangles (six created and one destroyed). Letting n and h denote the number of vertices and holes *after* the addition, we have

$$\begin{aligned} T(n,h) &= 5 + T(n-3,h-1) = 5 + ((n-3) + 2(h-1) - 2) = n + 2h - 2 \\ C(n,h) &= 6 + C(n-3,h-1) = 6 + ((n-3) + 3(h-1) - 3) = n + 3h - 3, \end{aligned}$$

as desired.

Solution 3: We will solve this using two nested loops. The first loop enumerates the edges that are incident to v. For each such edge e_1, we then enumerate the remaining edges on this face until returning to e_1.prev (or equivalently, if you prefer, the twin of e_1.onext). To do this, we simply follow next pointers around the face. See the code block below.

List the edges in the link of e

```
e1 = eStart = v.incident;   // e1 runs through the edges incident to v
do {
  e2 = e1.next;             // e2 runs through the edges on the face to the left of e1
  while (e2 != e1.prev) {
    add e2 to L;
    e2 = e2.next;           // advance to next edge CCW about e1's left face
  }
  e1 = e1.onext;            // advance to next edge CCW about v
} while (e1 != eStart);
return L;
```

Solution 4:

(a) We can compute the C-obstacle for an obstacle by "scraping" the robot R around the boundary of the obstacle and drawing the path traced by the reference point (see Fig. 4(a)). The C-obstacles for O_1 and O_2 are shown in Fig. 4(b).

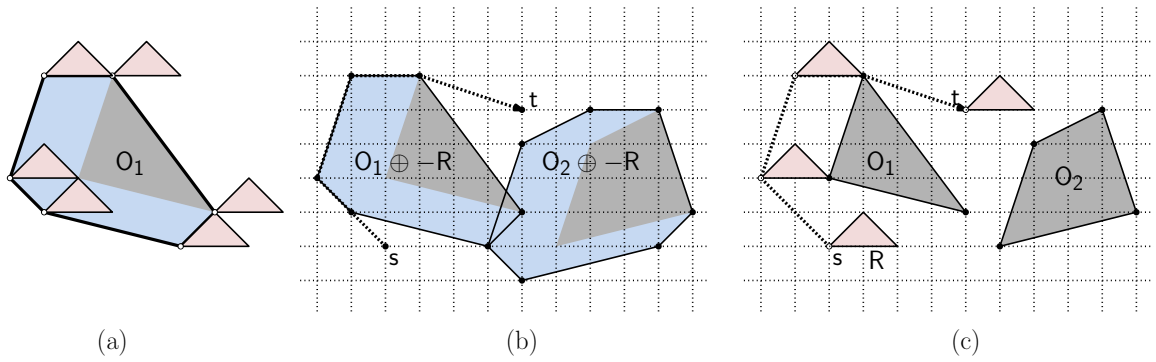

Figure 4: Problem 2: Building configuration obstacles.

(b) Because the two C-obstacles overlap, there is no path between them. The shortest path goes around to the left (see the dotted line in Fig. 4(c)).