



Figure 5.13: All of these depict high-dimensional obstacle regions in C-space. (a) The search must involve some sort of multi-resolution aspect, otherwise, that algorithm may explore too many points within a cavity. (b) Sometimes the problem is like a bug trap, in which case bidirectional search can help. (c) For a double bug trap, multi-directional search may be needed. (d) This example is hard to solve even for multi-directional search.

**Unidirectional (single-tree) methods:** In this case, the planning appears very similar to discrete forward search, which was given in Figure 2.4. The main difference between algorithms in this category is how they implement the VSM and LPM. Figure 5.13b shows a *bug trap*<sup>9</sup> example for which forward-search algorithms would have great trouble; however, the problem might not be difficult for backward search, if the planner incorporates some kind of greedy, best-first behavior. This example, again in high dimensions, can be considered as a kind of “bug trap.” To leave the trap, a path must be found from  $q_I$  into the narrow opening. Imagine a fly buzzing around through the high-dimensional trap. The escape opening might not look too difficult in two dimensions, but if it has a small range with respect to each configuration parameter, it is nearly impossible to find the opening. The tip of the “volcano” would be astronomically small compared to the rest of the bug trap. Examples such as this provide some motivation for bidirectional

<sup>9</sup>This principle is actually used in real life to trap flying bugs. This analogy was suggested by James O’Brien in a discussion with James Kuffner.