4. Dead Ends

Dead ends are a phenomenon that can cause trouble for FF’s search algorithm: they are states from which the goal is unreachable. We have already come across dead ends previously, in particular Sections 3.4.2 and 3.7 gave examples where dead ends cause enforced hill-climbing to fail. In this chapter we discuss dead ends in detail, considering a number of possibilities to deal with them. We choose a simple safety net solution that has no effect on FF’s performance in case of success, and can often help in case of failure.

The chapter is organized as follows. We first give sufficient (largely syntactical) criteria for recognizing planning tasks that do not contain any dead ends (Section 4.1) — thereby automatically determining “safe” cases. We then describe some algorithmic techniques we have tried for dealing with dead ends, if they arise (Section 4.2). We finally evaluate the algorithmic technique we have chosen to use in FF (Section 4.3). As a preface, we fill in the formal details about dead ends, their relation to FF’s heuristic function, and their effects on the completeness of FF’s search algorithm. Note that there are actually two sources of incompleteness in the FF base architecture: dead ends, which we discuss here, and the helpful actions pruning technique. For the moment, i.e., the more theoretical part of this chapter up to the end of Section 4.1, we completely ignore the pruning technique. In the more practical Sections 4.2 and 4.3, we include the pruning technique into our considerations.

**Definition 4.0.1.** Given a task \((A, I, G)\), with state space \((S, E)\). A state \(s \in S\) is a dead end if \(gd(s) = \infty\).

As formalized in Definition 3.0.3, a heuristic function \(h\) can return \(h(s) = \infty\) for a state \(s\). Taking this as an indication that \(s\) is a dead end, the obvious idea for any forward search algorithm is to remove \(s\) from the search space. This is exactly what is done in FF’s search algorithm, c.f. Section 3.4.1. The technique is only adequate if \(s\) is really a dead end, i.e., if \(h\) is completeness-preserving in the following sense.

**Definition 4.0.2.** Given a task \((A, I, G)\) with state space \((S, E)\) and heuristic \(h\). The heuristic is completeness preserving, if \(h(s) = \infty \Rightarrow gd(s) = \infty\) for all \(s \in S\). With a completeness preserving heuristic, a dead end state \(s\) is recognized if \(h(s) = \infty\) and unrecognized otherwise.

If a heuristic function guarantees to return \(\infty\) only on dead end states, then it makes sense to call these states recognized dead ends as opposed to unrecognized.
dead ends where the heuristic returns a finite value. We will see below that only the unrecognized dead ends can cause trouble for FF’s search algorithm. Let us first state that FF’s heuristic function $h^{FF}$ is, in fact, completeness preserving. This is because of its close relation to the $h^+$ function.

**Proposition 4.0.1.** Let $(A, I, G)$ be a STRIPS task or a negation-free ADL task. The $h^+$ heuristic is completeness preserving.

**Proof.** By Definition 3.0.6, $h^+(s) = \infty$ iff $(A^+, s, G)$ is unsolvable. For STRIPS and negation free ADL tasks any plan for $(A, s, G)$ is also a plan for $(A^+, s, G)$, c.f. Proposition 3.0.1. By contra-position we thus know that $(A, s, G)$ is unsolvable, i.e., $s$ is a dead end.

For ADL, it is a necessary prerequisite here that the task is negation free. We have already seen this in the following example task (described at the beginning of Chapter 3). The initial state is $I = \{g_1\}$, the goal condition is $G = \neg g_1 \land g_2$, and the action set $A$ comprises one action with empty precondition and a single effect ($\top, \{g_2\}, \{g_1\}$). Applying the action in the initial state yields a goal state: $g_2$ is added, and $g_1$ is deleted. The relaxed task, however, is unsolvable: without delete lists there is no way of making $g_1$ false so there is no plan for $(A^+, I, G)$. Thus $h^+(I) = \infty$ although $gd(I) = 1.1$

The $h^{FF}$ function is only defined for propositional ADL tasks. In this case, $h^{FF} = \infty$ if and only if $h^+ = \infty$.

**Proposition 4.0.2.** Let $(A, I, G)$ be a STRIPS task or a propositional ADL task, with state space $(S, E)$. For all $s \in S$, $h^{FF}(s) = \infty$ if and only if $h^+(s) = \infty$.

**Proof.** By Definitions 3.3.1 and 3.6.1, $h^{FF}(s) = \infty$ if and only if Graphplan respectively IPP terminate without finding a plan for $(A^+, s, G)$. As both these planners are sound and complete [8, 64], this is equivalent to unsolvability of $(A^+, s, G)$. This is, in turn, equivalent to $h^+(s) = \infty$ by Definition 3.0.6.

Recall FF’s search algorithm, enforced hill-climbing as described in Section 3.4. Starting from the initial state there is a number of search iterations. In each iteration starting from a state $s$, a complete breadth first search is performed for a state $s'$ with strictly better heuristic value $h(s') < h(s)$. The next iteration starts out from $s'$. Search stops when a goal state, $h(s) = 0$, is reached. States $s''$ with $h(s'') = \infty$ are skipped. This algorithm is complete under the following circumstances.

**Theorem 4.0.1.** Given a solvable task $(A, I, G)$ with state space $(S, E)$ and completeness-preserving heuristic $h$. If no state $s \in S$ is an unrecognized dead end, then enforced hill-climbing will find a solution.

---

1 In the negation free compilation of $(A, I, G)$, the goal condition is $g_1^- \land g_2$ while the action’s effect has the form ($\top, \{g_2, g_1^-\}, \{g_1\}$), so there $h^+(I) = 1$. 