A state in our recursion represents a partial or complete schedule of tasks.

The heuristic we currently use is:

(Time taken for all processors to have completed their schedule)+(sum of durations for unscheduled tasks)/(number of processors)

This essentially takes the existing schedule and then adds a (slightly more optimistic than is possible) heuristic for best case scenario.

Therefore, we know that whatever full schedule we create from a partial schedule, its length cannot be less than our heuristic.

The heuristic increases monotonically as you go down the tree of states, so as soon as it is too high, we know it will only get higher.

The recursion begins with a full schedule from a greedy algorithm as bestFoundState. bestFoundState is only updated when a full schedule that takes less time is found.

How do we know that we found the BEST solution?  
The recursion carries out a depth-first-search, and as soon as the heuristic for a given node(state) is greater than the best found state, the entire subtree starting from that node is ignored. This is because we know that no full schedule starting from this partial schedule can be faster than the heuristic we have calculated, which is already more than an existing solution we know of.

This means we have considered all leaves of the state tree (all possible schedules) because when we evaluate a node, the heuristic gives us a tight-ish lower bound on all leaves that ultimately come from that node. So if the heuristic is higher than the value of our best schedule so far, no leaf in the subtree from that node will be better than our best so far.

Using this method of bounding, we essentially DFS the tree, aborting each branch as soon as we know it won’t pan out.