and hole on already-linked pieces and moves to change the angle of a joint. Changing one angle may force changes in others, and the changes will vary depending on whether the other pieces are at their joint-angle limit. In general there will be no unique "minimal" solution for a given angle change in terms of the consequent changes to other angles, and some changes may be impossible.

4.3 Here is one simple hill-climbing algorithm:

- Connect all the cities into an arbitrary path.
- Pick two points along the path at random.
- Split the path at those points, producing three pieces.
- Try all six possible ways to connect the three pieces.
- Keep the best one, and reconnect the path accordingly.
- Iterate the steps above until no improvement is observed for a while.

4.4 Code not shown.

4.5 See Figure S4.1 for the adapted algorithm. For states that OR-SEARCH finds a solution for it records the solution found. If it later visits that state again it immediately returns that solution.

When OR-SEARCH fails to find a solution it has to be careful. Whether a state can be solved depends on the path taken to that solution, as we do not allow cycles. So on failure OR-SEARCH records the value of $path$. If a state is which has previously failed when $path$ contained any subset of its present value, OR-SEARCH returns failure.

To avoid repeating sub-solutions we can label all new solutions found, record these labels, then return the label if these states are visited again. Post-processing can prune off unused labels. Alternatively, we can output a direct acyclic graph structure rather than a tree.

See (Bertoli *et al.*, 2001) for further details.

4.6

The question statement describes the required changes in detail, see Figure S4.2 for the modified algorithm. When OR-SEARCH cycles back to a state on $path$ it returns a token $loop$ which means to loop back to the most recent time this state was reached along the path to it. Since $path$ is implicitly stored in the returned plan, there is sufficient information for later processing, or a modified implementation, to replace these with labels.

The plan representation is implicitly augmented to keep track of whether the plan is cyclic (i.e., contains a $loop$) so that OR-SEARCH can prefer acyclic solutions.

AND-SEARCH returns failure if all branches lead directly to a $loop$, as in this case the plan will always loop forever. This is the only case it needs to check as if all branches in a finite plan loop there must be some And-node whose children all immediately loop.

4.7 A sequence of actions is a solution to a belief state problem if it takes every initial physical state to a goal state. We can relax this problem by requiring it take only *some* initial physical state to a goal state. To make this well defined, we'll require that it finds a solution

telegram : @konkurcomputer

www.konkurcomputer.ir

```
function AND-OR-GRAPH-SEARCH(problem) returns a conditional plan, or failure
  OR-SEARCH(problem.INITIAL-STATE, problem, [])

function OR-SEARCH(state, problem, path) returns a conditional plan, or failure
  if problem.GOAL-TEST(state) then return the empty plan
  if state is on path then return loop
  cyclic − plan ← None
  for each action in problem.ACTIONS(state) do
      plan ← AND-SEARCH(RESULTS(state, action), problem, [state | path])
      if plan ≠ failure then
          if plan is acyclic then return [action | plan]
          cyclic − plan ← [action | plan]
  if cyclic − plan ≠ None then return cyclic − plan
  return failure

function AND-SEARCH(states, problem, path) returns a conditional plan, or failure
  loopy ← True
  for each s_i in states do
      plan_i ← OR-SEARCH(s_i, problem, path)
      if plan_i = failure then return failure
      if plan_i ≠ loop then loopy ← False
  if not loopy then
     return [if s_1 then plan_1 else if s_2 then plan_2 else ... if s_{n−1} then plan_{n−1} else plan_n]
  return failure
```

Figure S4.2 AND-OR search with repeated state checking.

solves a belief state b and a belief state b' then it solves the union belief state $b \cup b'$.

b. On expansion of a node, do not add to the frontier any child belief state which is a superset of a previously explored belief state.

c. If you keep a record of previously solved belief states, add a check to the start of OR-search to check whether the belief state passed in is a subset of a previously solved belief state, returning the previous solution in case it is.

4.9

Consider a very simple example: an initial belief state $\{S_1, S_2\}$, actions a and b both leading to goal state G from either initial state, and

$$c(S_1, a, G) = 3\,; \qquad c(S_2, a, G) = 5\,;$$
$$c(S_1, b, G) = 2\,; \qquad c(S_2, b, G) = 6\,.$$

In this case, the solution $[a]$ costs 3 or 5, the solution $[b]$ costs 2 or 6. Neither is "optimal" in any obvious sense.

In some cases, there *will* be an optimal solution. Let us consider just the deterministic case. For this case, we can think of the cost of a plan as a mapping from each initial physical state to the actual cost of executing the plan. In the example above, the cost for $[a]$ is

telegram : @konkurcomputer

www.konkurcomputer.ir