|  |
| --- |
| *Algorithm: Improved A with Jump Point Search (JPS)*\* |
| Input: Start node *S*, Goal node *G*, Grid map *M* |
| Output: Optimal path from *S* to *G* |
|  |
| 1. Initialize: |
| * + Open list ← {*S*} |
| * + Closed list ← ∅ |
| * + *g*(*S*) ← 0 |
| * + *f* (*S*) ← *h*(*S*) |
| 1. While Open list is not empty do: |
| 1. Select node *n* from Open list with the smallest *f* (*n*). |
| 1. If *n* = *G*: |
| 1. Return the reconstructed path from *S* to *G*. |
| 1. Remove *n* from Open list and add to Closed list. |
| 1. For each neighbor *m* of *n*: |
| 1. Perform Jump Point Search (JPS) from *m* to determine the next valid jump point. |
| 1. If *m* ∈ Closed list then continue. |
| 1. Tentative cost *gtentative* ← *g*(*n*) + *cost*(*n*, *m*). |
| 1. If *m* ∈/ Open list then: |
| 1. Add *m* to Open list. |
| 1. Else if *gtentative* ≥ *g*(*m*) then continue. |
| 1. Update: |
| 1. *g*(*m*) ← *gtentative*. |
| 16. *f* (*m*) ← *g*(*m*) + *h*(*m*). |
| 1. Parent of *m* ← *n*. |
| 1. End While |
| 1. If Open list is empty then: |
| 1. Return "Path not found". |
| 1. End |

**Key Features of Jump Point Search (JPS):**

Pruning unnecessary nodes: JPS skips redundant nodes in straight paths and diagonal expansions.

Optimized neighbor selection: Only relevant nodes (jump points) are explored to reduce the search space and improve efficiency (Jurnal Urut-24) (Jurnal Urut-5) .