Overview

We report on a machine-checked proof of correctness for Dijkstra’s one-to-all shortest path algorithm. Unlike previous work, we use classic textbook code written in executable C [2].

Challenges: We use CompCert C, which is executable and realistic but also has real-world complications. We prove full functional correctness, and not just program safety.

Solution: We use the Verified Software Toolchain [1] and our Mathematical and Spatial Graph Libraries [4] to establish machine-checked correctness at the C level (3 files, 3k LOC). The CompCert compiler [3] then guarantees this correctness on the executable code.

Key findings: The algorithm suffers from potential overflow issues. The precise bound

\[ \text{MAX} \]

is nontrivial: we show that the intuitive guess fails, and provide a workable refinement.

Workflow

A sketch of how we verify C programs. Note where we integrate with other projects.

The structure of our Mathematical Graph Library. The soundness condition is entirely customizable. Lemmas and properties can be composed, and are automatically inherited.

Instantiating DijkGraph

PreGraph: VType := Z, EType := Z + Z, src := fst, dst := snd,
\[ \forall v. \text{valid}(v, v) \Rightarrow 0 < v \leq \text{SIZE} \]
\[ \forall v, d. \text{valid}(v, (x, d)) \Rightarrow \text{valid}(v, x) \wedge \text{valid}(v, d) \]
LabeledGraph: EType := Z, VType := list EType, GLType := unit,
GeneralGraph: FiniteGraph[\gamma] := \{ v \mid \text{valid}(\gamma, v) \}
\[ \forall i. \text{valid}(\gamma, v) \Rightarrow i = j \Rightarrow \text{elabel}(\gamma, (i, j)) = 0 \]
\[ \forall i. 0 < \text{elabel}(\gamma, (i, j)) \leq \text{MAX}/\text{SIZE} \]

Upper Bound on Path Cost

The longest optimal path has \( \text{SIZE} - 1 \) links, so we set \( \text{elabel}(u) \)’s upper bound to \( \lfloor \text{MAX}/\text{SIZE} \rfloor - 1 \). Consider the following example, where \( \text{MAX} = 7 \) and \( \text{SIZE} = 3 \), and so \( 0 < \text{elabel}(v, e) < 3 \). The check on line 20 overflows when scanning C’s neighbors:

\[ \lfloor \text{MAX}/\text{SIZE} \rfloor \cdot (\text{SIZE}-1) + \lfloor \text{MAX}/\text{SIZE} \rfloor \cdot \text{SIZE} \]

Solution: Conservatively set the upper bound for an individual edge to \( \lfloor \text{MAX}/\text{SIZE} \rfloor \).

New worst case: line 20 calculates
\[ \text{MAX}/\text{SIZE} \cdot (\text{SIZE}-1) + \lfloor \text{MAX}/\text{SIZE} \rfloor \cdot \text{MAX} \]

The true maximum path cost is \( \lfloor \text{MAX}/\text{SIZE} \rfloor \cdot (\text{SIZE}-1) \leq \text{MAX} \).

Code and Specification

```c
1 void dijkstra (int graph[SIZE][SIZE], int src, int *dist, int *prev) {
2     // DijkGraph(\gamma)
3         int pq[SIZE];
4     int i, j, u, cost;
5     for (i = 0; i < SIZE; i++) {
6         dist[i] = INF;
7         prev[i] = INF;
8         pq[i] = INF;
9     }
10     dist[src] = 0;
11     pq[src] = 0;
12     prev[src] = src;
13     // DijkGraph(\gamma) \land
14     // dikc_correct(\gamma, src, prev, dist, priq)
15     while (*pq_emp(pq)) {
16         u = popMin(pq);
17         for (i = 0; i < SIZE; i++) {
18             cost = graph[u][i];
19             if (\text{cost} < INF) {
20                 if (dist[i] > dist[u] + cost) {
21                     dist[i] = dist[u] + cost;
22                     prev[i] = u;
23                     pq[i] = dist[i];
24                 }
25             }
26         }
27      }
28     return;
29 }
30
31     list_rep(\gamma, i) \triangleq \text{data at array graph2mat(\gamma)[i]} /\!
32     \triangleq \text{graph2mat(\gamma)[i];}
33     \text{graph_graph(\gamma)} \triangleq \* v \rightarrow list_rep(\gamma, v)
34     \text{dijk_correct(\gamma, src, prev, dist, priq)} \triangleq 
35     \forall \text{dst}. \exists \text{popped dst}. \text{popped dst} \subseteq \text{path} /\!
36     \text{path2mat}(\gamma, \text{src}, \text{dst}, \text{path}) \wedge 
37     \text{path \text{global optimal}(\gamma, \text{dist}, \text{path})} \wedge 
38     \forall \text{path}. \exists \text{path}2. \text{path}2 \subseteq \text{path} \wedge 
39     \text{cost(path2)} \leq \text{cost(path)} \wedge 
40     \text{popped dst} \subseteq \text{INF} \Rightarrow \text{let m := pred(d) in m \in popped(d) \land} 
41     \forall m'. \in \text{popped(d)}. \text{cost(path2m) = \text{cost}(m, \text{dist})} <
42     \text{cost(path2m) = \text{cost}(m', \text{dist})} <
43     \text{popped dst} \subseteq \text{INF} \Rightarrow \forall m \in \text{popped(d)}. \text{cost(path2m) = \text{cost}(m, \text{dist})} = \text{INF}
```

Key Transformation: Growing the Subgraph

To begin, vertices \( d_{s1} \), \( d_{s2} \), and \( d_{s3} \) obey the first, second, and third clauses of the invariant \( \text{dijk \ correct} \) respectively. Vertex \( u \) obeys the second clause with minimal cost. The invariant is broken when relaxing \( u \)'s neighbors, and reestablished thereafter: \( u \) now obeys the first clause. Eventually no vertices obey the second clause, and we are done.

References