Review: Verified Dijkstra's Algorithm

Qingyi He
qingyi.he@epfl.ch

Yicong Luo
yicong.luo@epfl.ch

December 28, 2022

1 Introduction

Say a few general words about the general context of the paper you chose. Explain why the topic is of interest, or where it can be applied. If it is about a piece of software or artifact, give a description of it. State the main result of the paper and why it is new or how it improves on previous state of knowledge. You can cite references using, for example [1] and make a succint presentation of the organisation of your report.

2 Preliminaries

Something about graphs and the algorithm.

The Dijkstra's algorithm:

```
def Dijkstra(Graph, source) =
  for vertex v in Graph do
    if v == source then
      dist[v] <- INFINITY
    else
      dist[v] <- 0
    add v to Q

  while Q is not empty do
    u <- vertex in Q with min dist[u]
    remove u from Q
    for neighbor v of u in Q do
      alt <- dist[u] + Graph.Edges(u, v)
      if alt < dist[v] then
        dist[v] <- alt

  return dist
```

3 Body

To build a verified dijkstra algoritm in lisa, the first thing we need to do is to translate this algorithm to a functional way.

3.1 Functional Graph

We represents graphs using associate lists of each vertex and its edges to its nearby edges with its distance (or weight).

```
case class Graph(graph:
    List[(Int, List[(Int, Distance)])])
```

We also define the "valid graph", to avoid lisa generating invalid graphs (e.g. vertexes that have non-zero distance to itself).

```
def validGraph(
  graph: List[(Int, List[(Int, Distance)])]
): Boolean =
  noDuplicates(graph) &&
  graph.forall(e => noDuplicates(e._2)) &&
  graph.forall(e =>
    e._2.forall((i, _) => graph.get(i) != None())) &&
  graph.forall(n =>
    n._2.forall(z => 0.toDist <= z._2)) &&
  graph.forall { case (n, a) =>
    a.get(n) match {
    case None()  => true
    case Some(d) => d == 0.toDist
    }
  }
```

3.2 The Algotirhm

The (functional) dijkstra's algorithm resembles the original version, which we frist generate the list of distances `Q`, then get into the main loop.

```
// type Node = (Int, Distance)
def dijkstra(source: Int): List[Node] = {
    require(graph.get(source) != None())
    val Q = prepare(source)
    iterate(Nil[Node](), Q)
}
```

3.2.1 Representing distance

Distance used in Dijistra algorithm, a distance is either infinite or a non-negative number. However, such data structure is not built-in in Scala, so we need to define the following.

```
sealed abstract class Distance
case object Inf extends Distance
case class Real(i: BigInt) extends Distance {
    require(i >= 0)
}
```

We also need to define addition and comparsion bewtween Distance, which is just integer addditon and comparsion with infinity.

3.2.2 A verified `getMin`

One important step in the dijkstra algorithm is extracting the node with minimial distance to the source, so we need to build a pure function that return the node that is closest to the source and the rest of nodes.

```
def getMin(l: List[Node]): (Node, List[Node]) = {
  require(l != Nil())
  l match
    case Cons(h, t) => getMinAux(h, t, Nil[Node]())
} ensuring (res =>
  res._2.size == l.size - 1 &&
    res._2.content ++ Set(res._1) == l.content &&
    res._2.map(_._1).content ++ Set(res._1._1) ==
      l.map(_._1).content &&
    res._2.forall(n => res._1._2 <= n._2)
)
```

Here we would call a helper function `getMinAux` which carries the min value, the traversed list and the rest to get the result. Ensuring that the remaining list is 1 shorter, the set of content of the remaining list with min node equals the original content of the original list. Finally the most important property is the all the nodes in the remaining list should have a longer distance than the we min one we get.

In this way, we ensure that the `getMin` function is correctly implemented.

3.2.3 Preprocessing

The first stage of the algorithm is to generate a list of vertexes with there initial distance to the source node, which every vertex should have a distance of inifinty and the source itself has distance 0.

```
def prepare(start: Int): List[Node] = {
    require(graph.get(start) != None())
    prepareAux(graph, start)
} ensuring (res => prepareProp(res, graph, start))
```

This accomplished in the `prepareAux` function, which is a simple mapping of the vertexes in the graph to the corresponding distance (either infinty or 0).

```
def prepareProp(
    res: List[Node],
    graph: List[(Int, List[Node])],
    start: Int
): Boolean =
  res.size == graph.size &&
    res.map(_._1).content ==
      graph.map(_._1).content &&
    res.map(_._1) == graph.map(_._1) &&
    (res.get(start) match {
      case Some(d) => d == Real(0)
      case None()  => res.forall(_._2 == Inf)
    })
```

3.2.4 Main loop

The main loop also resembles the original version, where we extract the vertex `h` with minimial distance to the source then update all the distances of the vertexes left using `h`.

```
def iterate(
  seen: List[Node],
  future: List[Node]
): List[Node] = {
  /* decreases and requires */
  future match
    case Nil() => seen
    case fu @ Cons(_, _) =>
      val (h, t) = getMin(fu)
      iterate(h :: seen, iterOnce(h, t))
} /* ensurings */
```

The updates are accomplished by the `iterOnce` function. It ensures that the nodes in the updated list will have a shorter distance to the source and all of their distances are larger than `cur`, as otherwise `cur` will no longer be the vertex that is shortest in the list. This help us verify that the updates of the distances are implemented correctly.

```
def iterOnce(cur: Node, rest: List[Node]): List[Node] = {
  decreases(rest.size)
  require(rest.forall(n => cur._2 <= n._2))
  rest match {
      case Nil()       => Nil()
      case Cons(h, t) =>
        Cons(updateDist(cur, h), iterOnce(cur, t))
  }
} ensuring (res =>
  res.forall(n => cur._2 <= n._2) &&
  res.map(_._1).content ==
    rest.map(_._1).content &&
  res.size == rest.size &&
  res.zip(rest)
    .forall((y, x) => y._1 == x._1 && y._2 <= x._2)
)
```

This is just a simple `map` over the list `rest`, applying function `updateDist`, which is simply defined as the following.

```
def updateDist(cur: Node, tar: Node): Node = {
  require(cur._2 <= tar._2)
  val nd = cur._2 + distance(cur._1, tar._1)
  (tar._1, if nd <= tar._2 then nd else tar._2)
} ensuring (res => res._2 <= tar._2 && cur._2 <= res._2)
```

The `ensuring` part verified that it the node returned will always have as shorter or equal distance to the source.

4 Future Work

However, we are not able make it pass one of the most important properties of the dijkstra's algoritm, which is the "triangle inequality", that is, if we choose two random vertexes in the resulting list (of distances), we should have the distance of vertex `m` to the source should be equal or smaller than the distance of sum of first going to vertex `n` then go to vertex `m`, otherwise the distance of `m` should be the smaller (which shows a faulty implementation).

```
def itInv(seen: List[Node]): Boolean =
  seen.forall { case (m, d0) =>
    seen.forall { case (n, d) =>
      n == m ||
      (graph.get(n).flatMap(_.get(m)) match
        case None      => true
        case Some(d1) => d0 <= d1 + d
      )
    }
  }
```

Such invariant should hold for `seen` or `res` in the main loop `iterate`. However, we cannot just *require* or *ensure* such property in lisa, since it will timeout as expected.

To help lisa verify such property, we still need other lemmas. For example, whenever we add a vertex to the `seen` nodes, we know that the newly added vertex should always have a longer distance to the source vertex, otherwise we havn't extracted the vertex with min distance last time. But to verify this, we need to we would need extra constrains in our `getMin` function, as it is deeply tied to the min node we have extracted.

Unfortunately, even though we have attempted, we did not get it pass lisa.

5 Conclusion

Recall briefly what the paper achieves, and how it is new. Express your critical skil on the paper and explain what you think are the strong and weak points of the paper. Also tell how you could potentially use the paper's results in your future project. You can also suggest further work or extensions to the paper.

References

[1] Bibliography management in LaTeX. https://overleaf.com/learn/latex/Bibliography_management_in_LaTeX.