Mechanically checked proof on Dijkstra's shortest path algorithm

Qiang Zhang

J Moore

October 13, 2004

Introduction

- Dijkstra's shortest path algorithm: a classical algorithm to find the shortest path between two vertices in a finite graph with non-negative weighted edges
- Directed Finite Graph with non-negative weighted edges
- Correctness of the algorithm: "if both vertices a and b are in the graph g, then the algorithm does return a shortest path from a to b in the graph g"

Algorithm

1. $\lambda(u) \Leftarrow 0$; for each vertex t other than u in V, $\lambda(t) \Leftarrow \infty$; and $T \Leftarrow V$;
2. Let s be a vertex in T such that $\lambda(s)$ is minimum;
3. If $s = v$, stop (or If $T = \{\}$, stop);
4. For every edge from s to t, if $t \in T$ and $\lambda(t) > \lambda(s) + wt(st)$, then $\lambda(t) \Leftarrow \lambda(s) + wt(st)$;
5. $T \Leftarrow T - \{s\}$ and go to step 2.

Formalization

- Graph representation: an association list ((u1 (v1 . w1) (v2 . w2) ...) ...)
- path table pt: ((u . path-from-a-to-u) ...)
- Function returns the result

```
(defun dijkstra-shortest-path (a b g)
  (let ((p (dsp (all-nodes g) (list (cons a (list a))) g)))
    (path b p)))
```

- Function maintains the iteration

```
(defun dsp (ts pt g)
  (cond  ((endp ts) pt)
         (t (let ((u (choose-next ts pt g)))
                (dsp (del u ts)
                     (reassign u (neighbors u g) pt g)
                     g))))))
```

Formalization

1. Let ts be initially all vertices in g;
2. Let pt be initially (list (cons a (list a)));
3. (path n pt) returns the already discovered path associated with n in pt, i.e. initially (path a pt) = (list a) and (path n pt) = nil for all other vertices; and (d n pt g) returns the weight of (path n pt) in g. It is convenient to use NIL as "infinity";
4. Repeat until ts is empty:
   (a) Choose u in ts such that (d u pt g) is minimal;
   (b) for each edge from u to some neighbor v with weight wt, if (d v pt g) > (d u pt g) + wt, then reassign (path v pt) to be (append (path u pt) (list v));
   (c) Delete u from ts.

Traditional Proof

- When a vertex u is chosen by step 4(a), the path associated with u in the path table is the shortest path from the start vertex to u in the graph
- When a vertex u is chosen by step 4(a), for any vertex v chosen after u, the path associated with v in the path table is the shortest path from the start vertex to v through the vertices(i.e. the internal vertices), which are chosen before u

Mechanical Proof

- Main Theorem:

```
(defthm main-theorem
 (implies (and (nodep a g)
               (nodep b g)
               (graphp g))
          (shortest-path a
                         b
                         (dijkstra-shortest-path a b g)
                         g)))
```

- Invariant:

```
(defun inv (ts pt g a)
    (let ((fs (comp-set ts (all-nodes g))))
      (and (prop-ts-node a ts fs pt g)
           (prop-fs-node a fs fs pt g)
           (paths-from-s-table a pt g))))
```

Function details

- (all-nodes g) returns all the nodes in the graph g

```
(defun all-nodes (g)
 (cond ((endp g) nil)
       (t (cons-set (caar g)
                    (my-union (strip-cars (cdar g))
                              (all-nodes (cdr g)))))))
```

- (nodep n g) returns t iff a is a vertex in the graph g

```
(defun nodep (n g) (mem n (all-nodes g)))
```

- (graphp g) returns t iff g is a legal graph:

```
(defun graphp (g)
  (cond ((endp g) (equal g nil))
        ((and (consp (car g))
              (edge-weightsp (cdar g)))
         (graphp (cdr g)))
        (t nil)))
```

Function details

- (edge-weightsp lst) returns t iff lst is a legal list of edges:

```
(defun edge-weightsp (lst)
  (cond  ((endp lst) (equal lst nil))
         ((and (consp (car lst))
               (rationalp (cdar lst))
               (<= 0 (cdar lst))
               (not (assoc (caar lst) (cdr lst))))
          (edge-weightsp (cdr lst)))
         (t nil)))
```

- (comp-set ts s) returns the set deleting ts from s

```
(defun comp-set (ts s)
  (if (endp s) nil
    (if (mem (car s) ts)
        (comp-set ts (cdr s))
        (cons (car s) (comp-set ts (cdr s))))))
```

Function details

- (shortest-path a b p g) returns t iff p is the shortest path from a to b in g

```
(defun-sk shortest-path (a b p g)
    (forall path (implies (path-from-to path a b g)
                          (shorter p path g))))
```

- (paths-from-s-table s pt g) returns t iff for any path in pt, it is associated with a key vertex u, then the path is a path from s to u in g

```
(defun paths-from-s-table (s pt g)
  (if (endp pt) t
     (and (if (not (cdar pt)) t
                (path-from-to (cdar pt) s (caar pt) g))
          (paths-from-s-table s (cdr pt) g))))
```

Function details

- (prop-ts-node a ts fs pt g)

```
(defun prop-ts-node (a ts fs pt g)
    (if (endp ts) t
      (and (shorter-all-inter-path a (car ts)
                                   (path (car ts) pt) fs g)
           (all-but-last-node (path (car ts) pt) fs)
           (prop-ts-node a (cdr ts) fs pt g))))
```

- (all-but-last-node p fs)

```
(defun all-but-last-node (p fs)
   (if (endp p) t
     (if (endp (cdr p)) t
       (and (mem (car p) fs)
            (all-but-last-node (cdr p) fs)))))
```

Function details

- (shorter-all-inter-path a b p fs g)

```
(defun-sk shorter-all-inter-path (a b p fs g)
   (forall path (implies (and (path-from-to path a b g)
                              (all-but-last-node path fs))
                         (shorter p path g))))
```

- (prop-fs-node a fs s pt g)

```
(defun prop-fs-node (a fs s pt g)
  (if (endp fs) t
    (and (shortest-path a (car fs) (path (car fs) pt) g)
         (all-but-last-node (path (car fs) pt) s)
         (prop-fs-node a (cdr fs) s pt g))))
```

Proof sketch

- initially the invariant is correct

```
(defthm inv-0
  (implies (nodep a g)
           (inv (all-nodes g) (list (cons a (list a))) g a)))
```

- the invariant is maintained by the iteration

```
(defthm inv-choose-next
    (implies (and (inv ts pt g a)
                  (my-subsetp ts (all-nodes g))
                  (graphp g)
                  (consp ts)
                  (setp ts)
                  (nodep a g)
                  (equal (path a pt) (list a)))
             (let ((u (choose-next ts pt g)))
                (inv (del u ts)
                     (reassign u (neighbors u g) pt g) g a))))
```

Proof sketch

- the final form of the invariant is correct

```
(defthm inv-last
    (implies (and (nodep a g)
                  (graphp g))
             (inv nil
                  (dsp (all-nodes g)
                       (list (cons a (list a)))
                       g)
                  g a)))
```

- main lemma

```
(defthm main-lemma
  (implies (and (inv nil pt g a)
                (nodep b g))
           (shortest-path a b (path b pt) g)))
```

Prove inv-0

- sub-goal 1

```
(implies (mem a (all-nodes g))
         (prop-fs-node a
                       (comp-set (all-nodes g) (all-nodes g))
                       (comp-set (all-nodes g) (all-nodes g))
                       (list (list a a)) g))
```

- lemma 1

```
(defthm comp-set-id
  (not (comp-set s s)))
```

Prove inv-0

- sub-goal 2

```
(implies (mem a (all-nodes g))
         (prop-ts-node a (all-nodes g) nil (list (list a a)) g))
```

- lemma 2

```
(defthm prop-path-nil
   (prop-ts-node a s nil (list (cons a (list a))) g))
```

Prove inv-choose-next

- lemma 1

```
(defthm paths-from-s-table-reassign
   (implies (and (paths-from-s-table a pt g)
                 (graphp g)
                 (my-subsetp v-lst (all-nodes g)))
            (paths-from-s-table a (reassign u v-lst pt g) g)))
```

- not hard to prove this lemma

Prove inv-choose-next

- lemma 2

```
(defthm prop-fs-node-choose
  (implies (and (inv ts pt g a)
                (my-subsetp ts (all-nodes g))
                (graphp g)
                (consp ts)
                (setp ts))
           (let ((u (choose-next ts pt g)))
              (prop-fs-node a
                            (comp-set (del u ts) (all-nodes g))
                            (comp-set (del u ts) (all-nodes g))
                            (reassign u (neighbors u g) pt g)
                            g))))
```

Prove inv-choose-next

- lemma 3

```
(defthm prop-ts-node-choose-next
  (implies (and (inv ts pt g a)
                (my-subsetp ts (all-nodes g))
                (setp ts)
                (consp ts)
                (graphp g)
                (nodep a g)
                (equal (path a pt) (list a)))
           (let ((u (choose-next ts pt g)))
                (prop-ts-node a (del u ts)
                              (comp-set (del u ts)
                                        (all-nodes g))
                              (reassign u (neighbors u g) pt g)
                              g))))
```

Prove prop-fs-node-choose-next

- the form of (prop-fs-node a ss ss pt g), has to be generalized
- (comp-set (del u ts) s) VS (cons u (comp-set ts s))
- u is the chosen vertex, which should have the shortest path
- General lemma

```
(defthm prop-fs-node-choose-lemma2
  (implies (and (prop-fs-node a fs s pt g)
                (my-subsetp fs (all-nodes g))
                (all-but-last-node (path u pt) s)
                (paths-from-s-table a pt g)
                (nodep u g)
                (graphp g)
                (shortest-path a u (path u pt) g))
           (prop-fs-node a (cons u fs) s
                         (reassign u (neighbors u g) pt g) g)))
```

Prove prop-fs-node-choose-next

- consider (comp-set (del u ts) s) as a subset of (cons u (comp-set ts s))

```
(defthm prop-fs-node-choose-lemma3
  (implies (and (my-subsetp s fs)
                (my-subsetp fs (all-nodes g))
                (paths-from-s-table a pt g)
                (prop-fs-node a fs ss pt g))
           (prop-fs-node a s ss pt g)))
```

- compare (comp-set ts s) with (comp-set (del u ts) s)

```
(defthm prop-fs-node-choose-lemma4
  (implies (and (my-subsetp s ss)
                (prop-fs-node a fs s pt g))
           (prop-fs-node a fs ss pt g)))
```

Prove prop-fs-node-choose-next

- has to establish (shortest-path a u (path u pt) g)

```
(defthm choose-next-shortest
  (implies (and (graphp g)
                (consp ts)
                (my-subsetp ts (all-nodes g))
                (inv ts pt g a))
           (shortest-path a (choose-next ts pt g)
                          (path (choose-next ts pt g) pt) g)))
```

- traditional proof: for the chosen vertex u and any path p from a to u in g, find the leftmost vertex v, which is in ts, in the path p, then the path associated with v in pt is shorter than the partial path from a to v in p, and the partial path is shorter than p, while u is chosen before v, which means the path associated with u in pt is shorter than the one associated with v

Prove choose-next-shortest

- auxiliary function (find-partial-path p s)

```
(defun find-partial-path (p s)
  (if (endp p) nil
    (if (mem (car p) s)
        (cons (car p) (find-partial-path (cdr p) s))
      (list (car p)))))
```

- the partial path is shorter than the original one

```
(defthm partial-path-shorter
  (implies (graphp g)
           (shorter (find-partial-path p s) p g)))
```

Prove choose-next-shortest

- (find-partial-path p s) returns a path, whose internal vertices are all in s

```
(defthm pathp-partial-path
  (implies (pathp p g)
           (and (path-from-to (find-partial-path p s)
                              (car p)
                              (car (last (find-partial-path p
                                                            s)))
                              g)
                (all-but-last-node (find-partial-path p s) s))))
```

Prove choose-next-shortest

- the last vertex of (find-partial-path p (comp-set ts (all-nodes g)) is in ts

```
(defthm find-partial-path-last-mem
  (implies (and (mem (car (last p)) ts)
                (pathp p g)
                (my-subsetp ts (all-nodes g)))
           (mem (car
                (last
                  (find-partial-path p
                                     (comp-set ts
                                               (all-nodes g)))))
                ts)))
```

Prove choose-next-shortest

- for any vertex v in ts, the path associated with the chosen vertex is shorter than the one associated with v

```
(defthm choose-next-shorter-other
  (implies (mem v ts)
           (shorter (path (choose-next ts pt g) pt)
                    (path v pt) g)))
```

- the transitivity of shorter relation

```
(defthm shorter-trans
  (implies (and (shorter p1 p2 g)
                (shorter p2 p3 g))
           (shorter p1 p3 g)))
```

Prove prop-ts-node-choose-next

```
(defthm prop-ts-node-choose-next
  (implies (and (inv ts pt g a)
                (my-subsetp ts (all-nodes g))
                (setp ts)
                (consp ts)
                (graphp g)
                (nodep a g)
                (equal (path a pt) (list a)))
           (let ((u (choose-next ts pt g)))
                (prop-ts-node a (del u ts)
                              (comp-set (del u ts)
                                        (all-nodes g))
                              (reassign u (neighbors u g) pt g)
                              g))))
```

Prove prop-ts-node-choose-next

- similarily consider (comp-set (del u ts) s) as (cons u (comp-set ts s))

```
(defthm prop-ts-node-lemma3
  (implies (and (paths-from-s-table a pt g)
                (graphp g)
                (nodep a g)
                (equal (path a pt) (list a))
                (prop-fs-node a fs fs pt g)
                (prop-ts-node a ts fs pt g)
                (mem u ts)
                (shortest-path a u (path u pt) g))
           (prop-ts-node a (del u ts) (cons u fs)
                         (reassign u (neighbors u g) pt g) g)))
(defthm prop-ts-node-lemma1
  (implies (and (my-subsetp s fs)
                (my-subsetp fs s)
                (prop-ts-node a ts fs pt g))
            (prop-ts-node a ts s pt g)))
```

Prove prop-ts-node-lemma3

- 2 sub-goals to prove:
  - for any vertex v in (del u ts), the path associated with v in the reassigned path table is shorter than any path from a to v with internal vertices in (cons u fs), stated by prop-ts-node-lemma2
  - internal vertices of all paths in the reassigned path table are in the set (cons u fs), stated by prop-ts-node-lemma3-3

```
(defthm prop-ts-node-lemma3-3
  (implies (and (paths-from-s-table a pt g)
                (all-but-last-node (path v pt) fs)
                (all-but-last-node (path u pt) fs))
           (all-but-last-node (path v (reassign u v-lst
                                                  pt g))
                              (cons u fs))))
```

Prove prop-ts-node-lemma2

- prop-ts-node-lemma2

```
(defthm prop-ts-node-lemma2
  (implies (and (shorter-all-inter-path a v (path v pt) fs g)
                (graphp g)
                (nodep a g)
                (equal (path a pt) (list a))
                (prop-fs-node a fs fs pt g)
                (shortest-path a u (path u pt) g)
                (paths-from-s-table a pt g))
     (shorter-all-inter-path a v
                             (path v (reassign u
                                               (neighbors u g)
                                               pt g))
                             (cons u fs) g)))
```

Prove prop-ts-node-lemma2

- prop-ts-node-lemma2-3

```
(defthm prop-ts-node-lemma2-3
  (implies (and (shorter-all-inter-path a v (path v pt) fs g)
                (graphp g)
                (prop-fs-node a fs fs pt g)
                (nodep a g)
                (path-from-to p a v g)
                (all-but-last-node p (cons u fs))
                (shortest-path a u (path u pt) g)
                (paths-from-s-table a pt g)
                (equal (path a pt) (list a)))
           (shorter (path v (reassign u (neighbors u g) pt g))
                    p g)))
```

- two cases to prove
  - a and v are identical, easy to prove
  - a and v are not equal, by prop-ts-node-lemma2-2

Prove prop-ts-node-lemma2-2

- prop-ts-node-lemma2-2

```
(defthm prop-ts-node-lemma2-2
  (implies (and (shorter-all-inter-path a v (path v pt) fs g)
                (graphp g)
                (prop-fs-node a fs fs pt g)
                (path-from-to p a v g)
                (not (equal a v))
                (shortest-path a u (path u pt) g)
                (all-but-last-node p (cons u fs))
                (paths-from-s-table a pt g))
           (shorter (path v (reassign u (neighbors u g) pt g))
                    p g)))
```

Prove prop-ts-node-lemma2-2

- two cases to prove
  - (path u pt) is NIL, (not (all-but-last-node p fs)) happens in the hypotheses. We know (shortest-path a u (path u pt) g) holds and (path u pt) is NIL, therefore there is no path from a to u, then u won't happen in any path, especailly in the path p; and we know (all-but-last-node p (cons u fs)) holds, therefore (all-but-last-node p fs) holds.

    ```
    (defthm not-path-implies-path-in-fs
      (implies (and (shortest-path a u (path u pt) g)
                    (not (path u pt))
                    (graphp g)
                    (path-from-to p a v g)
                    (all-but-last-node p (cons u fs)))
               (all-but-last-node p fs))
    ```

  - (path u pt) is not NIL, by prop-ts-node-lemma2-1

Prove prop-ts-node-lemma2-1

- prop-ts-node-lemma2-1

```
(defthm prop-ts-node-lemma2-1
  (implies (and (shorter-all-inter-path a v
                                           (path v pt) fs g)
                (graphp g)
                (prop-fs-node a fs fs pt g)
                (path-from-to p a v g)
                (not (equal a v))
                (path u pt)
                (shortest-path a u (path u pt) g)
                (all-but-last-node p (cons u fs))
                (paths-from-s-table a pt g))
           (shorter (path v (reassign u (neighbors u g) pt g))
                    p g)))
```

Prove prop-ts-node-lemma2-1

- two cases to prove
  - for the path p from a to v, the vertex neighbored to v in p is u
    - (path u pt) is the shortest path from a to u, so (append (path u pt) (list v)) is shorter than p
    - (path v pt) is shorter than (append (path u pt) (list v))
    - (path v (reassign u (neighbors u g) pt g)) is shorter than (path v pt)
  - for the path p from a to v, the vertex neighbored to v in p isn't u, we have to define two auxiliary functions

```
(defun find-last-next-path (p)
  (if (or (endp p) (endp (cdr p))) nil
       (cons (car p) (find-last-next-path (cdr p)))))
(defun last-node (p)
  (car (last (find-last-next-path p))))
```

Prove prop-ts-node-lemma2-1

- (append (path (last-node p) pt) (list v)) is shorter than (append (find-last-next-path p) (list v)), by last-node-lemma1
- (append (find-last-next-path p) (list v)) is actually the path p, by last-node-lemma2
- (path v pt) is shorter than (append (path (last-node p) pt) (list v)), by shorter-than-append-fs

Prove prop-ts-node-lemma2-1

- last-node-lemma2

```
(defthm last-node-lemma2
  (implies (and (equal (car (last p)) v)
                (pathp p g))
           (equal (append (find-last-next-path p) (list v)) p)))
```

- shorter-than-append-fs

```
(defthm shorter-than-append-fs
  (implies (and (shorter-all-inter-path a v (path v pt) s g)
                (prop-fs-node a fs s pt g)
                (my-subsetp fs s)
                (path w pt)
                (paths-from-s-table a pt g)
                (mem w fs))
           (shorter (path v pt)
                    (append (path w pt) (list v)) g)))
```

Prove last-node-lemma1

- (last-node p) is not equal to u, but still in (cons u fs)
- (path (last-node p) pt) is the shortest path from a to (last-node p), so shorter than (find-last-next-path p)
- shorter-implies-append-shorter

```
(defthm shorter-implies-append-shorter
  (implies (and (shorter p1 p2 g)
                (graphp g)
                (true-listp p1)
                (equal (car (last p1)) (car (last p2)))
                (pathp p2 g))
           (shorter (append p1 (list v))
                    (append p2 (list v)) g)))
```

- to apply shorter-implies-append-shorter, establish (pathp (find-last-next-path p)), by path-from-to-implies-all-path-lemma

Prove last-node-lemma1

- path-from-to-implies-all-path-lemma

```
(defthm path-from-to-implies-all-path-lemma
  (implies (and (path-from-to p a v g)
                (not (equal a v)))
           (and (pathp (find-last-next-path p) g)
                (mem v
                     (neighbors
                        (car (last (find-last-next-path p)))
                        g))))))
```

- path p is from a to v, where a isn't equal to v, the length of p is at least 2, by path-length
- the length of path p is at least 2, then the conclusion of the lemma holds, by pathp-find-last-next

Prove last-node-lemma1

- path-length

```
(defthm path-length
  (implies (and (pathp p g)
                (not (equal (car p) (car (last p)))))
           (<= 2 (len p))))
```

- pathp-find-last-next

```
(defthm pathp-find-last-next
  (implies (and (pathp p g)
                (<= 2 (len p)))
           (and (pathp (find-last-next-path p) g)
                (mem (car (last p))
                     (neighbors
                        (car (last (find-last-next-path p)))
                        g)))))
```

Prove inv-last

- maintain some hypotheses

```
(defthm del-subsetp
  (implies (my-subsetp ts s)
           (my-subsetp (del u ts) s)))
(defthm del-true-listp
  (implies (true-listp ts)
           (true-listp (del u ts))))
(defthm del-noduplicates
     (implies (setp ts)
              (setp (del u ts))))
(defthm path-a-pt-reassign
  (implies (and (paths-from-s-table a pt g)
                (graphp g)
                (nodep a g)
                (equal (path a pt) (list a)))
           (equal (path a (reassign u v-lst pt g)) (list a))))
```

Conclusion

- Dijkstra's shortest path algorithm
- 122 lemmas and 48 goals proved by hints, within which 27 hints are only the hint of in-theory kind, 6 hints are given on sub-goal level, 19 hints are explicit instantiation of lemmas, 2 hints are explicit induction scheme, and 2 hints are explict expansion of functions
- follow the traditional proof scheme
- trying to find some common schemes and propose a ACL2 book for further proof in graph algorithms