View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Erasmus University Digital Repository

A general framework for shortest path algorithms

Wim Pijls,
Erasmus University Rotterdam, P.O.Box 1738, 3000 DR Rotterdam,
The Netherlands, *wimp@cs.few.eur.nl*
Antoon Kolen,
University of Limburg, P.O.Box 616, 6200 MD Maastricht,
The Netherlands, *A.Kolen@KE.RULimburg.NL*

Abstract

In this paper we present a general framework for shortest path algorithms, including amongst others Dijkstra's algorithm and the A* algorithm. By showing that all algorithms are special cases of one algorithm in which some of the nondeterministic choices are made deterministic, termination and correctness can be proved by proving termination and correctness of the root algorithm. Furthermore, several invariants of the algorithms are derived which improve the insight with respect to the operations of the algorithms.

1 Introduction

In the context of this paper, the shortest path problem is defined as the problem of finding in a directed graph the shortest path from a source vertex to a set of target vertices. For variations of the problem we refer to the taxonomy of Deo and Pang [Deo-Pang].

The shortest path is a classic topic both in the field of Combinatorial Optimization and in the field of of Artificial Intelligence. In Combinatorial Optimization, the Dijkstra algorithm [Dijkstra] in case of nonnegative arc distances, and the Modified Dijkstra algorithm in case of arbitrary arc distances, are well known. In Artificial Intelligence, the shortest path problem arises in the context of heuristic search. In heuristic search a heuristic estimate function is given, which returns for each vertex v an estimate of the length of the shortest path from v to the single target vertex. The best known algorithm is A*, originating from Hart [Hart].

In this paper we present a general framework for shortest path algorithms; see Figure 1. By showing that all algorithms are special cases of one algorithm in which some of the non-deterministic choices are made deterministic, termination and correctness can be proved by proving termination and correctness of the root algorithm. In this way we avoid some of the ad hoc proofs found in the literature. Furthermore, we shall provide invariants of the algorithms which improve

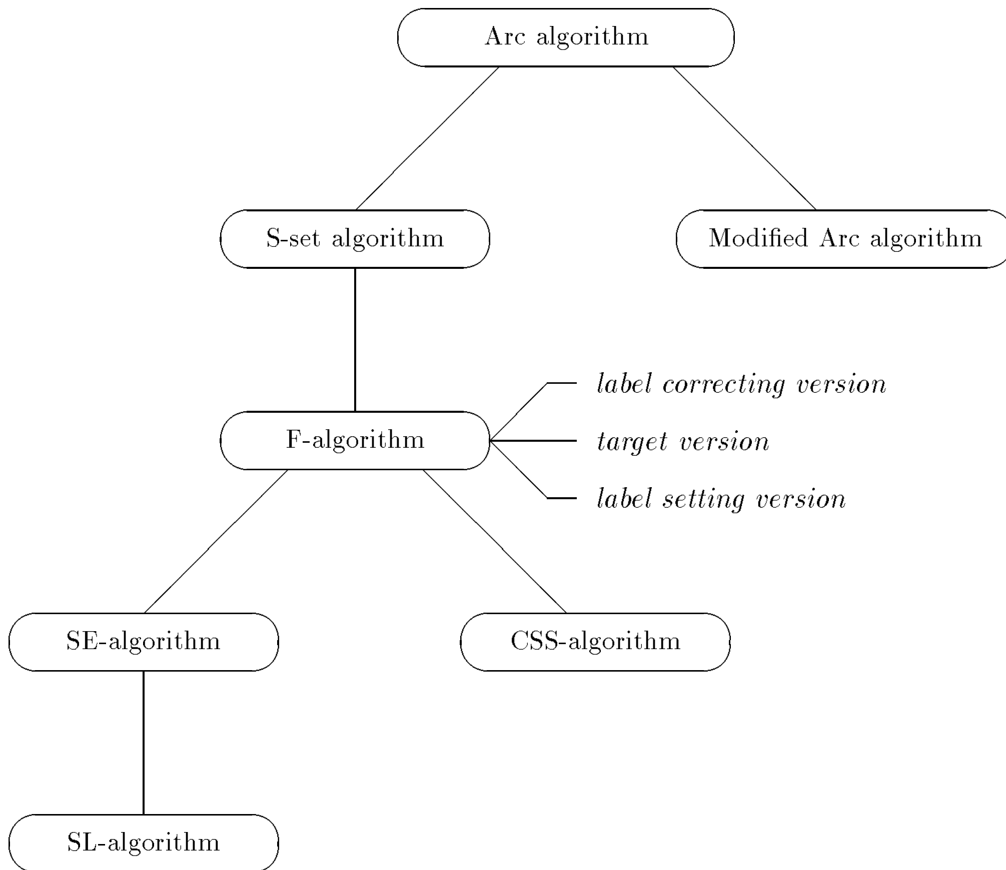

Figure 1: The framework

our understanding of these algorithms. In [Pijls-Kolen] this approach has led to improved dominance results with respect to space and time complexity of the algorithms.

An instance of the shortest path problem is defined by a directed graph $G = (V, A)$, where V is the set of vertices and A is the set of arcs (an arc is an ordered pair of vertices), a source vertex s and a subset $T \subseteq V$ of target vertices, a distance function $d : A \rightarrow I\!R$, and heuristic estimate function $h : V \rightarrow I\!R$.
Following standard graph terminology we define a *walk* W in G from start vertex v_1 to end vertex v_n to be a sequence $(v_1, v_2, \ldots, v_n)$ such that $(v_i, v_{i+1}) \in A$, $i = 1, 2, \ldots n \Leftrightarrow 1$; the length (denoted by $length(W)$) is defined by $\sum_{i=1}^{n-1} d(v_i, v_{i+1})$.
A cycle is a walk in which the start and the end vertex are identical and no other vertex occurs more than once in the walk. A path is a walk in which no vertex occurs more than once. The shortest path problem is to find a path of minimal length from s to every target vertex. In most of our algorithms $T = V$ or T consists of one single vertex; in the latter case the target vertex is denoted by t. The length of a path with minimal length from s to v is denoted by $\hat{g}(v)$.

Our framework consists of the tree in Figure 1. The root algorithm is the so-called Arc algorithm. This is shown in Figure 2. In this algorithm every vertex has a label, denoted by g, which is regularly updated. Finding the shortest path

```
g(s):= 0;
for all v ∈ V, v ≠ s, do g(v) := ∞;
while any arc (v,w) ∈ A satisfies g(w) > g(v) + d(v,w)
   [ select any arc (v,w) with g(w) > g(v) + d(v,w);
     g(w) := g(v) + d(v,w);
   ]
```

Figure 2: The Arc algorithm.

in a graph containing cycles of negative length is NP-hard [Garey]. As is shown below, the Arc algorithm does not terminate in this case.

Theorem 1.1 *If $(v_1, v_2, \ldots, v_n)$ is a walk such that $g(v_{i+1}) \leq g(v_i) + d(v_i, v_{i+1})$ for $i = 1, \ldots, n \Leftrightarrow 1$, then $g(v_n) \leq g(v_1) + \sum_{i=1}^{n-1} d(v_i, v_{i+1})$.*

Proof
Add the inequalities $g(v_{i+1}) \leq g(v_i) + d(v_i, v_{i+1})$, $i = 1, \ldots, n \Leftrightarrow 1$. □

Corollary 1.1 *If the Arc algorithm terminates on an instance, then every cycle has nonnegative length.*

Proof
Let $(v_1, v_2, \ldots, v_n, v_1)$ be a cycle. On termination of the Arc algorithm we have $g(v_{i+1}) \leq g(v_i) + d(v_i, v_{i+1})$ and $g(v_1) \leq g(v_n) + d(v_n, v_1)$. By Theorem 1.1 $g(v_n) \leq g(v_1) + \sum_{i=1}^{n-1} d(v_i, v_{i+1})$. Combining this with $g(v_1) \leq g(v_n) + d(v_n, v_1)$ gives $\sum_{i=1}^{n-1} d(v_i, v_{i+1}) + d(v_n, v_1) \geq 0$. □

In Section 2 we prove the correctness of the Arc algorithm for graphs without negative cycles, i.e., we prove that the Arc algorithm terminates on any graph without negative cycles and that, on termination, $g(v)$ equals the shortest path distance from s to v, for any $v \in V$. The shortest path itself from s to v can be traced on termination by a method, explained in Section 3.
Although we will prove termination, there are problem instances, even for a deterministic algorithm as the SE-algorithm, which take an exponential number of updates.

The Arc algorithm originates from Ford [Ford]. A termination proof for the case of integer arc distances can be found in [Ahuja]. Our termination proof, which does not assume that the arc distances are integer, has not been explicitly stated in literature.

In the Arc algorithm the choice of the arc (v, w) with $g(w) > g(v) + d(v, w)$

is completely non-deterministic. In the Modified Arc algorithm (see Figure 3), we loop through all the arcs to see if there is a possible label update. The order,

```
while any arc (v,w) ∈ A satisfies g(w)>g(v)+d(v,w) do
      for each arc (v,w) ∈ A do
          if g(w) > g(v)+d(v,w) then g(w) := g(v) + d(v,w);
```

Figure 3: The Modified Arc algorithm.

in which the arcs are considered by the for loop, is arbitrary. Hence the Modified Arc algorithm is still non-deterministic. As we will see in Section 4, $|V| - 1$ repetitions of such a loop suffice.

In the S-set algorithm (see Figure 4), we have yet another way of grouping arcs together, to see if any update is possible. In the S-set algorithm, we maintain a set S of vertices, for which we can guarantee a special property: if an arc has its start vertex in S, then no label updates are possible, using this arc. In each

```
g(s) := 0;
for v∈V, v≠s, g(v):=∞;
S:=∅;
while S≠V do
   [ select any vertex k ∉ S with g(k)< ∞;
     S:= S +[k];
     for all (k,v) ∈ A do
         if g(v) > g(k) + d(k,v) then
            [ g(v) := g(k) + d(k,v);
              if v ∈ S then S := S - [v];
            ]
   ]
```

Figure 4: The S-set algorithm.

iteration, we take a vertex outside S and we consider all arcs starting at that vertex. Using these arcs, we perform all possible updates and the start vertex of these arcs is added to S. If an end vertex of an arc is updated, this end vertex is removed from S, because the special property can no longer be guaranteed for this vertex.

In the F-algorithm, to be discussed in Section 6, we encounter the heuristic estimate for the first time. The f-label of a vertex v is defined as $g(v) + h(v)$. In the algorithm a global variable F appears, which is initialized to $-\infty$. When we replace the selection statement in the S-set algorithm by:

```
(a) if F < minimum{f(v)| v ∉ S}
```

```
        then F := minimum{f(v) | v ∉ S};
(b) select any vertex k with k ∉ S and f(k)≤F
```

then the F-algorithm is obtained.
The F-algorithm has several instances. The Smallest Estimate algorithm (SE-algorithm) is defined by specifying the non-deterministic choice in the selection statement of the F-algorithm as:

```
(b) select a vertex k, k∉S and f(k)=minimum{f(v)| v ∉ S}.
```

Notice that for the vertex selected, the condition $f(k) \leq F$ is satisfied. The SE-algorithm can also be defined as the instance of the S-set algorithm in which selection is prescribed as: *select* k *with* $f(k) = mimimum\{f(v) \mid v \notin S\}$.
When the non-deterministic choice in the F-algorithm is specified as:

```
(b) select a vertex k, k∉S and
          g(k)=minimum{g(v)| v ∉ S and f(v) ≤ F}
```

the Combined Selection Strategy algorithm (CSS-algorithm), is defined, which considers both the f-label and the g-label.
The Smallest Label algorithm (SL-algorithm) is a special version of the SE-algorithm with $h = 0$. The SL-algorithm can also be defined as the instance of the S-set algorithm that has the following selection criterion: *select* k *with* $g(k) = mimimum\{g(v) \mid v \notin S\}$.

For the instances of the F-algorithm several so called *versions* will be distinguished. We will refer to the complete version of the F-algorithm as the *label-correcting* version. A contrasting version is the *label-setting* version, in which no vertex ever leaves the S-set. A third version is the *target version* which may be applied in case $T = [t]$. In this version $g(t) = \hat{g}(t)$ holds, as soon as t is in S, and therefore the execution is stopped at that time.

2 Correctness of the Arc algorithm

In this section we first prove termination of the Arc algorithm, and we next prove that this algorithm determines the length of the shortest path from s to any vertex v. In order to prove termination, we introduce a new variable and we insert into the algorithm two statements dealing with this variable. The new variable is called *ancestors* and its type is *array of sequence*. A quantity of the sequence type is an ordered sequence of vertices. So the quantity *ancestors(v)* contains for any vertex v an ordered sequence of vertices. We add the following statement to the initializing statements before the main loop:

ancestors(s)*:=*(s)*;*

Together with updating $g(w)$, the quantity *ancestors*(w) is updated. For that purpose, we insert the statement:

ancestors(w)*:=ancestors*$(v) + w$*;*

The operator '+' achieves concatenation of a sequence of vertices and a single vertex. The result of the operation $s + v$ with s of the sequence type and v of the vertex type consists of a new sequence containing s with v added at the end. In Lemma 2.1 we prove, that, in the absence of negative cycles, each finite g-label $g(v)$ equals the length of a path from s to v represented by *ancestors*(v). This statement is used in Theorem 2.1 to prove that the number of label updates and hence the number of iterations in the algorithm is finite.
Moreover, Theorem 2.1 shows the correctness of the Arc algorithm, i.e., the Arc algorithm provides the shortest path distance from s to v for any vertex v.

Lemma 2.1 *(invariant) The Arc algorithm has the following invariants*

a) for every v with $g(v) < \infty$, ancestors(v) is a walk $(p_0, p_1, p_2, \ldots, p_n)$, $n \geq 0$, with $p_0 = s$ and $p_n = v$ and $g(v) =$length(ancestors(v));

b) in ancestors(v) the following inequality holds:

$$g(p_k) + \sum_{i=k}^{n-1} d(p_i, p_{i+1}) \leq g(v),\ 0 \leq k \leq n \Leftrightarrow 1,$$

where strict inequality holds in case $p_k = v$, $0 \leq k \leq n \Leftrightarrow 1$;

c) if ancestors(v) contains a cycle for a vertex v, then this cycle has negative length.

Proof

Proof of a)

It is easily seen that this invariant holds after the initialisations and is maintained in each iteration of the main loop.

Proof of b)

This invariant holds after the initialisations. We shall prove that the invariant is maintained during the while loop. Assume that the arc (v, w) is selected and $g(w)$ and *ancestors*(w) are updated. Let *ancestors(v)* be given by $(p_0, p_1, p_2, \ldots, p_n = v)$.

Before the update, we have

$$g(p_k) + \sum_{i=k}^{n-1} d(p_i, p_{i+1}) \leq g(v),\quad 0 \leq k \leq n \Leftrightarrow 1,$$

This is equivalent to:

$$g(p_k) + \sum_{i=k}^{n-1} d(p_i, p_{i+1}) + d(v, w) \leq g(v) + d(v, w),\quad 0 \leq k \leq n \Leftrightarrow 1, \tag{2.1}$$

Assume that w does not occur in *ancestors*(v). Then the left-hand side of (2.1) is unchanged by the update and the right-hand side is equal to $g(w)$ after the update.

Assume that $w = p_k$ for some index k, $1 \leq k \leq n \Leftrightarrow 1$. Before the update, the right-hand side of (2.1) is smaller than $g(w)$ and hence

$$g(p_k) + \sum_{i=k}^{n-1} d(p_i, p_{i+1}) + d(v, w) < g(w).$$

By the update, both sides are decreased by the same amount and thus strict inequality is preserved.
Since also $g(w) = g(v) + d(v, w)$, invariant b) is maintained.

Proof of c)
Assume that by the update of $g(v)$, v a given vertex, *ancestors*(v) takes the form $(p_0, p_1, p_2, \ldots, p_n)$ with $p_k = p_n = v$ for some k with $1 \leq k < n$. By invariant b), $g(p_k) + \sum_{i=k}^{n-1} d(p_i, p_{i+1}) < g(p_n)$ and hence

$$\sum_{i=k}^{n-1} d(p_i, p_{i+1}) < 0$$

□

Theorem 2.1 *The Arc algorithm terminates for any graph without negative cycles and, on termination, $g(v) = \hat{g}(v)$ for any v.*

Proof
Due to part a) and c) of Lemma 2.1 *ancestors*(v) is a path from s to v for any v with $g(v) < \infty$. Therefore the number of label updates is is bounded by the number of paths. Since this number is finite, the algorithm terminates.
Let a shortest path from s to v be given by $(s = p_0, p_1, \ldots p_n = v)$. Since $g(p_{k+1}) \leq g(p_k) + d(p_k, p_{k+1})$, $1 \leq k < n$, on termination, we obtain, using Theorem 1.1, $g(v) \leq g(s) + \sum_{k=1}^{n-1} d(p_k, p_{k+1}) = g(s) + \hat{g}(v)$. Since $g(v)$ is equal to the length of a path from s to v, $g(v) \geq \hat{g}(v)$. Using $g(s) \leq 0$, we conclude $g(v) = \hat{g}(v)$. □

3 Tracing the shortest path

On termination of the Arc algorithm, we have $g(v) =$*length*(*ancestors*(v)) and $g(v) = \hat{g}(v)$. Since *ancestors*(v) is a path, this quantity provides a path of minimal length. However, the use of the array variable *ancestors* is not practical, because it takes a lot of memory to associate to each vertex a sequence of vertices. This variable has been introduced only for the termination proof. For finding the shortest path itself another variable, called *backpointer*, is more appropriate. Henceforth, instead of updating *ancestors*(w), we update, together with $g(w)$, a variable *backpointer*(w) by the statement:

backpointer$(w) := v$

The line of reasoning is the following. In Theorem 3.1 we prove that, in the absence of negative cycles, for every vertex v with $g(v) < \infty$ a path from s to v exists, such that each vertex is backpointer of its successor in the path and $g(v)$ is greater than or equal to the length of this path. It follows that, as soon the equality $\hat{g}(v) = g(v)$ is achieved during execution, this path is the shortest path from s to v, as is stated in Theorem 3.2. Since $\hat{g}(v) = g(v)$ on termination, the shortest path can be traced on termination.

Definition 3.1 *A walk* $(p_1,\ p_2,\ \ldots,\ p_n = v)$ *with* $p_k = \mathit{backpointer}(p_{k+1})$ *for* $1 \leq k \leq n \Leftrightarrow 1$ *is called a backpointer walk of* v*.*

Lemma 3.1 *(invariant) The Arc algorithm has the following invariants:*
a) if v*=backpointer(*w*) then* $g(w) \geq g(v) + d(v, w)$*;*
b) if $(p_1,\ p_2,\ \ldots,\ p_n)$ *is a backpointer walk of* p_n*, then*

$$g(p_1) + \sum_{i=1}^{n-1} d(p_i, p_{i+1}) \leq g(p_n);$$

c) if a backpointer walk contains a cycle, then this cycle has negative length;
d) $g(s) = 0 \Leftrightarrow s$ *has no backpointer.*

Proof
a) This property holds before the first iteration. It is obvious that the invariant holds after an iteration, in which arc (v, w) is chosen and $g(w)$ is updated by $g(w) := g(v) + d(v, w)$. As long as $v =$*backpointer*(w), $g(w)$ remains unchanged. Since $g(v)$ is non-increasing, the property remains valid.

b) We can prove that

$$g(p_1) + \sum_{i=1}^{m-1} d(p_i, p_{i+1}) \leq g(p_m),$$

for $m = 2, \ldots, n$, by induction on m, using part a).

c) Suppose that an arc (v, w) is selected. If a cycle occurs, then the cycle has the form $(v, w = p_1, \ldots, p_n = v)$. Since $(w = p_1, \ldots, p_n = v)$ is a backpointer walk before the update, it follows from b) that

$$g(w) + \sum_{i=1}^{n-1} d(p_i, p_{i+1}) \leq g(v),$$

where the g-labels refer to g-labels before the last update. Since (v, w) is selected, we have $g(w) > g(v) + d(v, w)$. Combining the inequalities, we obtain

$$\sum_{i=1}^{n-1} d(p_i, p_{i+1}) + d(v, w) < 0.$$

d) As long as $g(s)$ is not updated, it has no backpointer and it keeps the initial label $g(s) = 0$. □

Theorem 3.1 *If* $g(w) < \infty$ *for a given* $w \neq s$ *during the Arc algorithm on an instance without negative cycles, then a backpointer path* $P = (p_0, p_1, \ldots, p_n)$ *with* $s = p_0$ *and* $w = p_n$ *exists; moreover* $g(w) \geq \mathit{length}(P)$*.*

Proof
If $g(w) < \infty$, $w \neq s$, then at least one backpointer walk exists, namely (v, w), where (v, w) is the arc which has generated the last update of $g(w)$. By Lemma 3.1c) every backpointer walk is path, and has a finite number of vertices therefore. Consider the backpointer path $(p_0, p_1, \ldots, p_n = w)$ of w such that the number of vertices in the backpointer path is maximal. Hence p_0 has no backpointer.
In general, every vertex $p \neq s$ that is a backpointer, satisfies $g(p) < \infty$, and consequently $g(p)$ has been updated once and *backpointer*(p) exists.
It follows that, in the above backpointer path, $p_0 = s$.
Since $p_0 = s$ has no backpointer, by Lemma 3.1d) $g(s) = 0$. It follows from Lemma 3.1b) that $g(w) \geq$ *length*(P). □

Corollary 3.1 *During the Arc algorithm applied to an instance without negative cycles,* $g(s) = 0$.

Proof
This property holds after initialisations. In the proof of Theorem 3.1 we have argued that, whenever a vertex $w \neq s$ with $g(w) < \infty$ exists, then $g(s) = 0$. □

The path, which is referred to in Theorem 3.1, is called the *backpointer path* of v, denoted by *Backp*(v).

Theorem 3.2 *(invariant) During execution of the Arc algorithm on a graph without negative cycles,* $g(v) = \hat{g}(v)$ *implies that Backp*(v) *is a path from* s *to* v *with minimal length.*

Proof
Since *Backp*(v) is a path from s to v we have $\hat{g}(v) \leq$ *length*(*Backp*(v)). By Theorem 3.1 $g(v) \geq$ *length*(*Backp*(v)). Combining these inequalities with $g(v) = \hat{g}(v)$, we conclude $\hat{g}(v) =$*length*(*Backp*(v)). □

Due to Theorem 3.1, the difference between *ancestors(v)* and *Backp*(v) is clear. In case of only nonnegative cycles, the length of *Backp*(v) is a lower bound for $g(v)$, whereas the length of the path *ancestors*(v) is equal to $g(v)$. Since the inequality *length*(*Backp*(v)) $< g(v)$ may happen, the notion backpointer path cannot be used to prove termination of the Arc algorithm. In the following theorem, we characterise the conditions under which the backpointer path and the *ancestors* sequence are identical.

Theorem 3.3 *(invariant) During execution of the Arc algorithm on an instance without negative cycles, for every* $v \neq s$ *with* $g(v) < \infty$, *ancestors*(v) =*Backp*(v) *if and only if* $g(v)$ =*length*(*Backp*(v)).

Proof
The invariant holds after the initialisations. Suppose an arc (u, w) is selected. The *only-if* part still holds after the update, due to Lemma 2.1a.
Suppose *ancestors*(w) $\neq$*Backp*(w) after the update. Then, before the update, *ancestors*(u) $\neq$*Backp*(u) and, by the invariant, $g(u)$ >*length*(*Backp*(u)). It follows that after the update, $g(w)$ >*length*(*Backp*(w)).

Suppose *ancestors*$(v) \neq$ *Backp*(v) for some v, $v \neq w$, after the update, whereas equality held before the update. Then w occurs in *ancestors*(v). Let *ancestors*(v) be given by $(s = p_0, p_1, \ldots, p_n = v)$ and assume $w = p_k$ with $1 \leq k < n$. The relation $p_{i-1} =$ *backpointer*(p_i) holds before the update for $1 \leq i \leq n$, and after the update, amongst others, for $k < i \leq n$. Furthermore, before the update $g(p_k) = \sum_{i=0}^{k-1} d(p_i, p_{i+1})$. After the update, by Theorem 3.1, *length*(*Backp*$(p_k)) \leq g(p_k)$ and, since $g(p_k)$ has been updated, $g(p_k) < \sum_{i=0}^{k-1} d(p_i, p_{i+1})$. Therefore, *length*(*Backp*$(p_k)) < \sum_{i=0}^{k-1} d(p_i, p_{i+1})$ and after adding $\sum_{i=k}^{n-1} d(p_i, p_{i+1})$ to both sides of this inequality, we obtain *length*(*Backp*$(v)) < \sum_{i=0}^{n-1} d(p_i, p_{i+1}) = g(v)$. □

4 The Modified Arc algorithm

The Modified Arc algorithm has been presented in Figure 3. This algorithm is also discussed in [Ahuja]. It can be regarded as a non-deterministic version of the Bellman-Ford method with a modification from Yen, cf. [Lawler]. In our paper it is shown that the algorithm can also be used to trace cycles of negative length. Theorem 4.1 tells us that the algorithm terminates after at most $|V| \Leftrightarrow 1$ iterations of the while loop or, otherwise, a cycle with negative length can be found. The proof of this theorem shows that such a cycle, similar to a shortest path, can be traced by constructing a backpointer walk.

Lemma 4.1 *(invariant) After n iterations of the outer loop during the Modified Arc algorithm, it holds for any path P, consisting of at most n arcs, from s to a given vertex v that $g(v) \leq$ length(P).*

Proof
We give a proof by induction. The invariant holds after 0 iterations. Assume the invariant holds after n iterations. Since the g-labels are decreasing, the lemma also holds after $n+1$ iterations for paths of at most n arcs. It remains to prove that the lemma holds for a path with $n+1$ arcs after $n+1$ iterations.
Let a path consisting of $n+1$ arcs be given by $(s = p_0, p_1, \ldots, p_n, p_{n+1})$. Since the invariant holds after n iterations and g-labels are non-increasing, we have, directly before arc (p_n, p_{n+1}) is considered in outer iteration $n+1$, $g(p_n) \leq \sum_{j=0}^{n-1} d(p_j, p_{j+1})$. After arc (p_n, p_{n+1}) is considered, $g(p_{n+1}) \leq g(p_n) + d(p_n, p_{n+1})$. Hence $g(p_{n+1}) \leq \sum_{j=0}^{n} d(p_j, p_{j+1})$ and this relation also holds after the outer iteration has completed, because g-labels are non-increasing. □

Theorem 4.1 *After $|V| \Leftrightarrow 1$ iterations of the outer loop of the Modified Arc algorithm on an instance, there exists an arc (v, w) such that $g(w) > g(v) + d(v, w)$ if and only if the instance contains a negative cycle.*

Proof
By Corollary 1.1, the termination criterion of the Arc algorithm cannot be achieved, if the instance contains a negative cycle.
If there exists an arc (v, w) such that $g(w) > g(v) + d(v, w)$ after $|V| \Leftrightarrow 1$ iterations of the main loop, then update $g(w)$. Next, construct in reverse order a backpointer walk P, starting at w by repeatedly taking the backpointer of the

vertex under consideration. We distinguish two cases. Either the backpointer walk is a path ending at vertex s and s has no backpointer, or the backpointer walk can be continued infinitely and hence contains a cycle, which has negative length by Lemma 3.1.
First assume that the backpointer walk is a path ending at s and s has no backpointer. Then, by Lemma 3.1d), $g(s) = 0$. Let the backpointer walk from s to w be denoted by P and the part between s and v be denoted by P'. Hence $\mathit{length}(P) = \mathit{length}(P') + d(v, w)$. Since P is a path, P contains at most $|V| \Leftrightarrow 1$ arcs. Before the update we have by Lemma 4.1 $g(w) \leq \mathit{length}(P)$, and, by Lemma 3.1, $g(v) \geq g(s) + \mathit{length}(P') = \mathit{length}(P')$. It follows that before the update:

$$g(w) \leq \mathit{length}(P) = \mathit{length}(P') + d(v, w) \leq g(v) + d(v, w).$$

This contradicts $g(w) > g(v) + d(v, w)$. Therefore the first case cannot occur. We conclude that, tracing back the backpointer walk starting at w after the update of $g(w)$, we encounter a cycle. □

5 The S-set algorithm

The S-set algorithm consists of statements corresponding to label updates, interchanged with statements corresponding to updates of the S-set. After a label update, we can have at most $|V| \Leftrightarrow 1$ consecutive additions to the S-set, because the algorithms terminates when $S = V$. Therefore the algorithm terminates if the number of label updates is finite, i.e., if the Arc algorithm incorporated is finite. Conversely, when the S-set algorithm terminates, the stop criterion of the Arc-algorithm is satisfied, as is proved in the next theorem by taking $S = V$. We conclude that the S-set algorithm is an instance of the Arc algorithm.

Theorem 5.1 *(invariant) If $v \in S$ during the S-set algorithm, then $g(w) \leq g(v) + d(v, w)$, for all $(v, w) \in A$.*

Proof
The invariant holds after the initialisations. Assume that the invariant holds before k is selected. After insertion of k into S, $g(w) \leq g(k) + d(k, w)$ for all $(k, w) \in A$. For any vertex $v \in S$ which remains in the S-set, $g(v)$ is unchanged. Since $g(w)$ is non-increasing, $g(w) \leq g(v) + d(v, w)$ remains true. □

Now, we present some other properties, related to the S-set. Lemma 5.1 and 5.3 will be used in Section 6.

Definition 5.1 *A path $(s = p_0, p_1, \ldots, p_n = v)$, is called an S-path during the S-set algorithm, if all intermediate vertices p_i, $0 \leq i \leq n \Leftrightarrow 1$ are in S.*

Lemma 5.1 *For any S-path P from s to v during the S-set algorithm $g(v) \leq \mathit{length}(P)$.*

Proof
By Theorem 5.1 and Theorem 1.1 $g(v) \leq g(s) + \mathit{length}(P)$. Since $g(s) \leq 0$, the result follows. □

Lemma 5.2 *(invariant) If v=backpointer(w) during the S-set algorithm, then* $g(w) = g(v) + d(v, w)$ *if and only if* $v \in S$.

Proof

It follows from Lemma 3.1 that, if v =backpointer(w) then $g(v)+d(v, w) \leq g(w)$. By Theorem 5.1, $g(w) \leq g(v) + d(v, w)$ if $v \in S$. Hence the *if* part is proved.
The *only-if* part is proved as a loop invariant for any two given vertices v and w. Assume that the relations v =*backpointer*(w) and $g(w) = g(v) + d(v, w)$ are achieved in an iteration. Then $v \in S$ after this iteration. In the subsequent iterations, $g(v)$ decreases if and only if v is deleted from S and $g(w)$ decreases if and only if *backpointer*(w) changes. □

Lemma 5.3 *For any vertex v with* $g(v) < \infty$ *during the S-set algorithm applied to an instance without negative cycles, Backp*(v) *is an S-path if and only if* $g(v) =$ *length*(*Backp*(v)).

Proof

Let *Backp*(v) be given by $(s = p_0, p_1, \ldots, p_n = v)$. By Corollary 3.1 $g(s) = 0$ and by Lemma 3.1 $g(p_{i+1}) \geq g(p_i) + d(p_i, p_{i+1})$. Therefore, $g(v)$=*length*(*Backp*(v)) if and only if $g(p_{i+1}) = g(p_i) + d(p_i, p_{i+1})$ for $0 \leq i < n$. The lemma follows from Lemma 5.2. □

It follows immediately from Theorem 3.3 and Lemma 5.3 that *Backp*(v) = *ancestors*(v) if and only if *Backp*(v)) is an S-path. In [Pijls] it is proved that, in the SE-algorithm, *Backp*(k) is always an S-path, when k is selected. This property does not hold in general for the CSS-algorithm; see the counterexample in [Pijls]. However, it is proved in [Pijls-Kolen], that $g(k) =$ *length*(*Backp*(k)), when k is selected in the CSS-algorithm, applied to an instance with nonnegative distances.

In contrast with the Modified Arc algorithm, the S-set algorithm does not have polynomial run time. We give an instance illustrating that the run time may be exponential.

Definition 5.2 *The instance* $K(n)$ *for the shortest path problem is defined by:*

$$\begin{aligned} V &= \{n, n \Leftrightarrow 1, \ldots, 2, 1\} \\ A &= \{(i,j) \mid i,j = 1, \ldots, n, \;\; i > j\} \\ d(i,j) &= 2^{i-1} \Leftrightarrow 2^j \\ s &= n \\ T &= V \end{aligned}$$

Theorem 5.2 *Suppose that the S-set algorithm runs on* $K(n)$*, where in each iteration the smallest vertex with finite g-label is selected. Let the binary representation of a given number p,* $0 \leq p < 2^{n-1}$*, be given by* $a_{n-1}a_{n-2}\ldots a_2 a_1$. *Define* $a_n = 1$. *Then after* $p + 1$ *iterations:*

a) backpointer(w) *is the smallest integer v greater than w such that* $a_v = 1$.

b) $S = \{j \mid a_j = 1\}$

Proof
Since vertex $s = n$ is inserted in the first iteration, the invariant holds for $p = 0$. Suppose that the invariant holds, after $p+1$ iterations. Let k be the smallest index such that $a_k = 0$. Then the binary representation of p is: $a_{n-1} \ldots a_{k+1} 01 \ldots 1$. Due to b), k is the smallest vertex outside S and hence, k is selected in iteration $p+1$. We will show that each label $g(q)$ with $(k, q) \in A$, i.e., $1 \leq q \leq k - 1$, is updated, and hence q is deleted from S in this iteration.
We conclude from a) that *backpointer*$(j) = j + 1$ for $1 \leq j \leq k - 2$, and *backpointer*$(k - 1) =$ *backpointer*$(k) = \ell$ where ℓ is the smallest integer such that $\ell > k$ and $a_\ell = 1$. It follows from a) and b) that every backpointer is in S and every backpointer path is an S-path. We conclude from Lemma 5.2 that $g(k) = g(\ell) + d(\ell, k)$ and $g(q) = g(\ell) + d(\ell, k - 1) + \sum_{i=k-2}^{q} d(i+1, i) = g(\ell) + d(\ell, k - 1)$, $q = 1, \ldots, k - 1$. The following scheme shows that $g(q)$, $1 \leq q \leq k - 1$ is updated.

$$\begin{array}{ll} & g(k) + d(k, q) \\ = & g(\ell) + d(\ell, k) + d(k, q) \\ = & g(\ell) + 2^{\ell-1} - 2^k + 2^{k-1} - 2^q \\ = & g(\ell) + 2^{\ell-1} - 2^{k-1} - 2^q \\ < & g(\ell) + 2^{\ell-1} - 2^{k-1} \\ = & g(\ell) + d(\ell, k - 1) \\ = & g(q). \end{array}$$

The binary representation of $p + 1$ is given by $a_{n-1} \ldots a_{k+1} 10 \ldots 0$. Therefore, the invariant also holds after $p + 2$ iterations. □

Corollary 5.1 *The S-set algorithm on $K(n)$, selecting the smallest vertex in each iteration, has 2^{n-1} iterations.*

Proof
Substituting $p = 2^{n-1} - 1$ in Lemma 5.2, we conclude that $S = \{n, n - 1, n - 2, \ldots, 1\}$ and hence $S = V$, after 2^{n-1} iterations. □

The following numbers are inserted successively after vertex n: 1, 2, 1, 3, 1, 2, 1, 4, 1, 2, 1, 3, 1, 2, 1, ... This series is also known from the problem 'Towers of Hanoi', a famous problem, often used in text books on programmming to illustrate recursion.
It can be proved that $i < j, i, j \notin S, \Rightarrow g(i) - g(j) \leq 2^j - 2^i$ (cf. [Pijls]). Therefore, by the choice $h(v) = 2^v + v$ we achieve: $i < j, i, j \notin S \Rightarrow g(i) + h(i) < g(j) + h(j)$. It appears that a run on $K(n)$ of the S-set algorithm selecting the smallest vertex can be viewed as as a run of the SE-algorithm.

6 The F-algorithm and its versions

We will refer to the complete version of the F-algorithm as the *label-correcting* version. A contrasting version is the *label-setting* version, in which no vertex

ever leaves the S-set. This version applies, whenever the S-set algorithm has the following invariant:

$$\forall w \in S,\ \forall v \notin S:\ g(w) \leq g(v) + d(v,w) \tag{6.1}$$

Due to this invariant the labels of S-set elements are never updated and no vertex is ever removed from S. Consequently, it holds that $g(v) = \hat{g}(v)$ for any vertex v which is inserted into S.
A third, intermediate, version is the so called *target version* which may be applied in case $T = [t]$. If for an instance the condition $g(t) = \hat{g}(t)$ holds, as soon as t is in S, then the execution can be stopped. This code of a *target version* differs from the *label-correcting* by one condition, namely, the halting condition in the while-clause of the S-set algorithm:

while $S \neq V$ *do*

is replaced by

while $t \notin S$ *do*

So, every instance of the F-algorithm has three versions. We will see under which conditions the target version or label setting version applies.

First of all, we will consider some special types of heuristic estimates, for which the label-setting or the target version of the F-algorithm applies. If $T = \{t\}$ (i.e., there exists one single target vertex), then $\hat{h}(v)$ denotes the shortest path length from a given vertex v to t.

Definition 6.1 *A heuristic estimate h is called consistent if $h(v) \Leftrightarrow h(w) \leq d(v,w)$ for each arc (v,w).*

Definition 6.2 *An heuristic estimate h is called admissible, if $T = \{t\}$ and $h(v) \Leftrightarrow h(t) \leq \hat{h}(v)$ for any vertex v.*

Definition 6.3 *A heuristic estimate h is called non-misleading if $T = \{t\}$ and if for any two arbitrary paths P and Q from s to m and to n respectively the following assertion holds:*

$$\begin{aligned} &\mathit{length}(P) + h(m) < \mathit{length}(Q) + h(n) \quad \Rightarrow \\ &\mathit{length}(P) + \hat{h}(m) \leq \mathit{length}(Q) + \hat{h}(n) \end{aligned} \tag{6.2}$$

A heuristic estimate h is called proper *if the assertion in* (6.2) *holds for any two paths, such that P and Q are not contained in each other (i.e., P is not a subpath of Q and Q is not a subpath of P).*

In [Pearl] also the notion monotone is considered and is proved to be equivalent to consistent. A heurictic estimate is called *monotone*, if $h(v) \Leftrightarrow h(w)$ is smaller than or equal to the shortest pathlength from v to w for any pair v, w.
The definition of *admissible* is a generalisation of that in literature hitherto. It

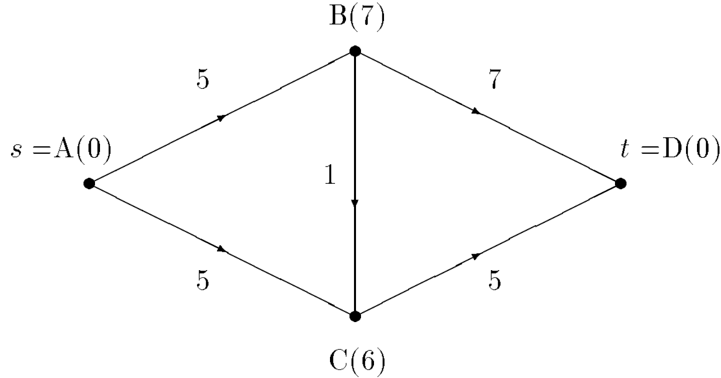

Figure 5: A non-misleading estimate.

can be proved easily that a monotone and thus a consistent estimate is always admissible.
The concept *non-misleading* has been introduced by Ibaraki [Ibaraki] in the context of the branch and bound method. The definition in [Ibaraki] differs from the current one, which is taken from [Bagchi 83]; both definitions are equivalent in case of an acyclic graph. In [Ibaraki] it is noted that non-misleading estimates are rare. The notion *proper* has been introduced in [Bagchi 83]. It is clear that a *non misleading* estimate is always *proper*. The converse is not true; see again [Bagchi 83]. An example of a non-misleading estimate is shown in Figure 5. (The arc distances are given along the arcs and $h(v)$ is given between parenthesis at vertex v.)

Lemma 6.1 *(invariant) During execution of the F-algorithm on a graph without negative cycles the following properties hold, if the heuristic estimate is consistent:*

a) $\forall w \in S, \forall v \notin S: \; f(w) \leq F \leq f(v)$;
b) $\forall w \in S, \forall v \notin S: \; g(w) \leq g(v) + d(v,w)$.

Proof
a) We will prove that this property holds, whenever a new F-value is determined and is maintained in all other cases.
Suppose that the statement, establishing a new F-value, is executed. As a consequence of the fact that a) holds before the execution of this statement, a) holds afterwards.
When k is selected, we have by the selection criterion that $f(k) \leq F$ and by a) that $f(k) \geq F$. We conclude $f(k) = F$. Due to a) we have before the update: $\forall w \in S, \forall v \notin S: \; f(w) \leq F = f(k) \leq f(v)$.
If a vertex v is updated, then $g(v) = g(k) + d(k,v) \geq g(k) + h(k) \Leftrightarrow h(v)$. It follows that $f(k) \leq f(v)$. Hence after the update we still have for each $v \notin S$: $f(k) = F \leq f(v)$. By b) $g(w)$ with $w \in S$ is not updated and hence $f(w) \leq F$

remains true.
b) The inequality $f(w) \leq f(v)$ can be rewritten as $g(w) \leq g(v) + h(v) \Leftrightarrow h(w)$, which implies by the consistency of h that $g(w) \leq g(v) + d(v, w)$. □

Lemma 6.2 *(invariant) During execution of the F-algorithm on a graph without negative cycles the following properties hold if the heuristic estimate is admissible:*

a) if $t \notin S$, then $f(k) \leq F \leq \hat{g}(t) + h(t)$.

b) when t is inserted into S, then $\hat{g}(t) = g(t)$ and $f(t) = F$.

Proof
a) We will prove that this property holds, whenever a new F-value is determined and is maintained in all other cases.
In an iteration, which establishes a new F-value, it holds that $f(k) = F$. Consider a path from s to t with minimal length. Let p be the first vertex in this path, which is not in S. Since $t \notin S$, the vertex p exists; (p may be equal to k or t). We have the following (in)equalities:

$$\begin{aligned}
F &\leq f(p) && \textit{(since } F \textit{ is the minimum of the } f\textit{-labels outside } S\textit{)} \\
&= g(p) + h(p) && \textit{(by definition)} \\
&\leq \hat{g}(p) + h(p) && \textit{(by Lemma 5.1)} \\
&\leq \hat{g}(p) + \hat{h}(p) + h(t) && \textit{(due to the admissibility)} \\
&= \hat{g}(t) + h(t) && \textit{(since } p \textit{ lies on a shortest path)}
\end{aligned}$$

In an iteration in which no new F-value is established, we have $f(k) \leq F$ and a) is maintained trivially.

b) As shown in the proof of a) we have, when t is selected, that $F \leq \hat{g}(t) + h(t)$. Moreover we have in general: $g(t) + h(t) = f(t) \leq F$. Since, as a consequence of Lemma 2.1 $g(v) \geq \hat{g}(v)$ for all v, we conclude that these three inequalities are equalities. □

Theorem 6.1 *For the F-algorithm the following holds on a graph without negative cycles:*

a) if the heuristic estimate function is consistent, then the label-setting version is correct and is equivalent to the label-setting version of the SE-algorithm;

b) if the heuristic estimate function is admissible, then the target version is correct and the greatest F-value during execution is equal to $\hat{g}(t)$+$h(t)$.

Proof
a) The label-setting version is correct, because, by b) of Lemma 6.1, the invariant (6.1) holds. Due to a) of Lemma 6.1, the vertex which is selected has minimal f-label outside S.
b) This follows from b) of Lemma 6.2 □

Theorem 6.2 *For the SL-algorithm,*

a) the label-setting version is correct if all distances are nonnegative;

b) the target version is correct if $\hat{h}(v) \geq 0$ for each $v \in V$.

Proof
If $h = 0$ and all distances are nonnegative, then h is consistent. If $h = 0$ and $\hat{h}(v) \geq 0$ for any v, then h is admissible. Since the SL-algorithm is the instance of the SE-algorithm with $h = 0$, the results follows from Theorem 6.1. □

Note
In the proof of Lemma 6.2, we only use the fact that in at least one path P with minimal length $h(p) \Leftrightarrow \hat{h}(p) \leq h(t)$ for all $p \in P$. Therefore the related conditions in Theorems 6.1b and Theorem 6.2 can be relaxed similarly.

There is another way to connect the SE- and SL-algorithm.

Theorem 6.3 *Let graph $G = (V, E)$ be given with distance functions d_1 and d_2, such that*

$$d_1(i,j) = d_2(i,j) \Leftrightarrow h(i) + h(j), \ \forall (i,j) \in A, \tag{6.3}$$

where h denotes a heuristic estimate function on V. If the SE-algorithm runs with distance function d_2 and the SL-algorithm runs with distance function d_1, then in each iteration of each algorithm the same vertex is selected for insertion into S and the same vertices are updated. It is assumed that ties are broken in favour of the same vertex in both algorithms.

Proof
Let g_1 and g_2 denote the g-labels in case distance function d_1 or d_2 respectively is used, and let f_2 denote $g_2 + h$. We prove by induction that in each next iteration, the same vertex is selected and the same labels are updated.
Assume that up to a certain iteration, for both algorithms the same vertex has been selected and the same vertices have been updated in each iteration. By Lemma 2.1 $g_1(v)$ and $g_2(v)$ are equal to the length of a walk. Since in all previous iterations the same vertex has been selected and the same vertices have been updated for each algorithm, these walks are the same. Determining the length of this walk in each algorithm, we come to the following equality, using (6.3):

$$g_1(v) = g_2(v) + h(v) \Leftrightarrow h(s) = f_2(v) \Leftrightarrow h(s), \ \forall v \in V \tag{6.4}$$

Since both algorithms have selected the same vertex and have updated the same labels in each iteration, the S-sets are equal. Due to (6.4), in the next iteration the SL-algorithm with distance function d_1 selects the same vertex as the SE-algorithm does with d_2.
Using (6.3) and (6.4), the following equality can be derived by straightforward substitution:

$$g_1(v) \Leftrightarrow g_1(w) \Leftrightarrow d_1(v,w) = g_2(v) \Leftrightarrow g_2(w) \Leftrightarrow d_2(v,w), \ \forall v, w \in V \tag{6.5}$$

Hence the same labels are updated in both algorithms. □

Note that Theorem 6.3 matches Theorems 6.1 and 6.2. If h is consistent, then d_1 is nonnegative. If h is admissible, then $\hat{h}(v) \geq 0$, where $\hat{h}$ is measured with respect to d_1.

Definition 6.4 *The selection criterion of an instance of the F-algorithm is called proper, if for all $v \notin S$, $k \neq v$, the relation*

$$f(k) < f(v) \bigvee g(k) \leq g(v)$$

is satisfied, when k is selected.

Notice that CSS has a proper selection criterion. The SE-algorithm has a proper selection criterion, when an additional rule is applied: *ties are resolved in favour of the smallest g-label.*
The next lemma gives some invariants related to the F-algorithm with a proper heuristic estimate. These invariants will be used in Theorem 6.4 to prove that the label-setting versions is correct in case of a proper heuristic estimate.

Lemma 6.3 *(invariant) If an instance of the F-algorithm using a proper selection criterion runs on a graph with non-negative distances and with a proper heuristic estimate, then for any $v \notin S$ and any $w \in S$:*
a) $g(w) \leq g(v) \vee g(w) + \hat{h}(w) \leq g(v) + \hat{h}(v)$
b) $g(w) \leq g(v) + d(v,w)$

Proof
This invariant holds before the first iteration. We will show that it holds after each next iteration.
Suppose the vertex k is selected and inserted into S. Since the selection criterion is proper, we have for all $v \notin S$:

$$g(k) \leq g(v) \bigvee g(k) + h(k) < g(v) + h(v). \tag{6.6}$$

Because of invariant b), no vertex has been deleted from S and therefore, each backpointer path is an S-path. Consequently the backpointer paths of k and v cannot be subpaths of each other. Because of Lemma 5.3 the lengths of both these backpointer paths are equal to the corresponding g-labels. Since the estimate function h is proper, we state:

$$g(k) + h(k) < g(v) + h(v) \Rightarrow g(k) + \hat{h}(k) \leq g(v) + \hat{h}(v) \tag{6.7}$$

It follows from (6.6) and (6.7) that a) is preserved, when k is inserted into S.
We now show that a) is preserved, when a given vertex v is updated. After the update, $g(v) = g(k) + d(k,v)$. If $g(w) \leq g(k)$ for a vertex $w \in S$, then a) is preserved trivially for this vertex w. If $g(w) + \hat{h}(w) \leq g(k) + \hat{g}(k)$ for a vertex $w \in S$, then we state:

$$g(w) + \hat{h}(w) \leq g(k) + \hat{h}(k) = g(v) \Leftrightarrow d(k,v) + \hat{h}(k) \leq g(v) + \hat{h}(v) \tag{6.8}$$

The right inequality in (6.8) is a consequence of a general inequality for the $\hat{h}$-function:

$$\forall (u,v) \in A : \hat{h}(u) \leq d(u,v) + \hat{h}(v) \tag{6.9}$$

By (6.8), a) is also preserved for this vertex w.
Now we show that b) is preserved. We have the implication:

$$g(w) + \hat{h}(w) \leq g(v) + \hat{h}(v) \leq g(v) + d(v,w) + \hat{h}(w) \Rightarrow g(w) \leq g(v) + d(v,w) \tag{6.10}$$

(Again the inequality in (6.9) is used.) Using the nonnegativity of the distances, we conclude that each inequality in a) implies b). □

Theorem 6.4 *For an instance of the F-algorithm with a proper selection criterion, the label-setting version is correct on a graph with non-negative distances and a proper or non-misleading heuristic estimate function*

Proof
Notice that a non-misleading heuristic estimate is always proper, as mentioned earlier. Due to part b) of Lemma 6.3, the label-setting version is correct. □

In contrast with a consistent estimate, the f-values, successively selected, do not generate an increasing series. This is illustrated by the instance in Figure 5.

We conclude with remarks on the existing literature. The target version of SE-algorithm is known as the A* algorithm [Hart, Pearl]. The target version of CSS is called the C-algorithm [Bagchi 83, Bagchi 85]. Several other algorithms have been designed, which are essentially specializations of F. In literature the following instances can be found (see also [Mahanti 88]): B [Martelli], B' [Mero], PropA [Bagchi 85], PropC [Bagchi 85]. The label-setting and the label-correcting version of SL are known in literature as Dijkstra' algorithm [Dijkstra] and Modified Dijkstra's algorithm [Johnson] respectively. In [Martelli] a correspondance between A* and Modified Dijkstra is pointed out, which is similar to Theorem 6.3.
By adding target vertex 0 and adding an arc $(1, 0)$ with $d(1, 0) = 2^n$ to the graph $K(n)$, (see Section 5), we obtain an instance of the A*-algorithm. (This instance resembles instances in [Martelli] and [Mahanti 88].) We conclude that also A* may have exponential run time.
A weaker version of Theorem 6.4 has been derived in [Bagchi 83], which states that a non-misleading heuristic estimate with nonnegative weights results in a linear number of insertions. The result that the label-setting version is also correct for a proper heuristic estimate, has not been stated before in literature.

References

[Ahuja] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin, *Network Flows,* in: Optimization (Chapter IV), volume 1 of Handbooks in Operations Research and Management Science; 1989, North-Holland.

[Bagchi 83] A. Bagchi and A. Mahanti, *Search Algorithms Under Different Kinds of Heuristics - A Comparative Study,* JACM, vol 30, nr. 1, Jan 1983, pp 1-21.

[Bagchi 85] A. Bagchi and A. Mahanti, *Three Approaches to Heuristic Search in Networks,* JACM vol 32, nr. 1, Jan 1985, pp 1-27.

[Deo-Pang] N. Deo and C. Pang, Shortest path algorithms: Taxonomy and annotation, Networks 145, pp 275-323.

[Dijkstra] E. W. Dijkstra, *A note on two problems in connexion with graphs,* Numer. Math. 1 (1959), pp 269-271.

[Ford] L.R. Ford jr., *Network Flow Theory,* Report P923, Rand Corp., Santa Monica, CA.

[Gallo] G.Gallo and S.Pallottino, *Shortest path methods: a unifying approach,* Mathematical Programming Study 26 (1986), pp 38-64.

[Garey] M.R. Garey, D.S, Johnson, *Computers and intractability, A guide to the theory of NP-Completeness,* New York 1979.

[Hart] P.E. Hart, N.J.Nilsson and B. Raphael, *A formal basis for the heuristic determination of minimum cost paths,* IEEE Trans. Syst. Sci. and Cybernetics 4 (2) (1968), pp 100-107.

[Ibaraki] T.Ibaraki, *The power of dominance relations in branch and bound algorithms.* JACM vol 24, nr 2, April 1977, pp 264-279.

[Johnson] D. B. Johnson, *A Note on Dijkstra's Shortest path Algorithm* JACM vol 20, nr.3, July 1973, pp 385-388.

[Lawler] E.G.Lawler, *Combinatorial Optimization: Networks and Matroids,* New York, 1976.

[Martelli] A. Martelli, *On the Complexity of Admissible Search Algorithms,* Artificial Intelligence 8 (1977), pp 1-13.

[Mahanti 87] A. Mahanti and K.Ray, *Heuristic search in Network Under Modifiable Estimate,* ACM CSC87 Proceedings, pp 166-174.

[Mahanti 88] A. Mahanti and K.Ray, *Network search algorithms with modifiable heuristics,* in: Kanal and Kumar, Search in Artificial Intelligence, Springer, 1988.

[Mero] L. Mero, *A heuristic Search algorithm with modifiable estimate,* Artificial Intelligence, 23 (1984), pp 13-27.

[Pearl] J. Pearl, *Heuristics, Intelligent search strategies for Computer Problem Solving,* Addison Wesley, 1984.

[Pijls] W. Pijls, *Shortest paths and Game trees*, Ph.D.thesis, Erasmus University Rotterdam, 1992.

[Pijls-Kolen] W. Pijls and A.Kolen, *Space and Time complexity in heuristic search,* to appear.