Heuristic estimates in shortest path algorithms

Wim Pijls[*]

Econometric Institute Report EI 2006-40

Abstract

Shortest path problems occupy an important position in Operations Research as well as in Artificial Intelligence. In this paper we study shortest path algorithms that exploit heuristic estimates. The well-known algorithms are put into one framework. Besides we present an interesting application of binary numbers in the shortest path theory.

Keywords: Network flows, Graph theory, Operations Research, Search problems.

[*]Econometric Institute, Erasmus University Rotterdam, P.O.Box 1738, 3000 DR Rotterdam, The Netherlands, e-mail: pijls@few.eur.nl

1 Introduction

Finding a shortest path in a network is a major topic in Operations Research (OR) and Combinatorial Optimization. However, shortest path algorithms exploiting heuristic estimates are hardly studied in the literature on those areas. On the other hand, the Artificial Intelligence literature (AI) addresses search problems involving heuristics estimates. The related algorithms can be viewed as shortest paths algorithms. Only in the past few years, a connection between the two communities has been established in the study of real road networks. In this paper, algorithms from both fields are combined into one framework. As a bonus, a nice and surprising relationship between binary numbers and shortest path algorithms is presented. Furthermore a general termination proof for shortest path algorithms is given. A notable role is played by so-called invariants, a common concept in Computer Science. An invariant is a property holding after each iteration during the execution of an algorithm. Invariants are proved by induction on the number of iterations. We only discuss the induction step, while proving invariants.

Preliminaries. A graph or network G is defined by a pair (V, E) with V the set of nodes and E the set of edges. A path is a sequence of nodes without duplicate elements. In the graph one node s is designated as the start or the source. The shortest path and its length from s to any other node v is looked for. The weight or length of an edge (u, v) is denoted by $d(u, v)$, whereas $\hat{d}(u, v)$ denotes the length of a shortest path from u to v.

Overview. Section 2 recalls some basics of the shortest path theory. This section contains a new termination proof. Section 3 puts Dijkstra and related algorithms, well-known in OR, on the one hand and A* and the search algorithms, mostly treated in the AI literature, on the other hand into one framework. Although the algorithms are not new, the idea of an underlying framework is. Further, a new property of so-called proper estimates is given. Section 4 presents a nice relationship between binary numbers and a shortest path instance. The study of paths in real road networks is briefly discussed in Section 5. Section 6 discusses the search space visited by shortest path algorithms.

2 The shortest path problem

The problem of finding the shortest path length between a source s and a target t can be described as an LP-problem (see Papadimitriou and Steiglitz (1982)). The dual of this LP-problem is:

$$\begin{aligned} &\max g(t), \\ \text{s.t.}\quad &g(v) - g(u) \leq d(u, v), \quad \forall u, v \in V \\ &g(s) = 0. \end{aligned}$$

We assume that the graph does not contain a cycle of negative length, since finding a shortest path in a graph with negative cycles is NP-hard. This is proved in Garey and Johnson (1979).

2.1 The Generic algorithm

The foregoing LP-description leads us in a natural way to the Generic algorithm, see Algorithm 1. The values $g(v)$ and *pred*(v) are called respectively the label and the predecessor of v. The operation in line 3 is called *edge relaxation* in some studies, see e.g. GOODRICH and TAMASSIA (2002). The edge was stretched out by the difference of $g(v)$ and $g(u)$ and by the new value $g(v)$ it is relaxed and comes back into its resting state. Given the Generic algorithm, the following question arises: does this algorithm terminate for any instance? In most of the literature an affirmative answer is assumed. However, we did not find clear evidence for this answer. Here, we give a termination proof. The key to this proof is provided by the following invariant.

Invariant 1 *For any $w \in V$, $g(w)$ is associated with a path $P = (s = p_0, p_1, \ldots, p_{n-1}, p_n = w)$ such that*

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

(so $g(w)$ equals the length of P) and

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

for $1 \leq k < n$.

Proof. Suppose an edge (u, v) is selected. If $v \neq p_k$, $1 \leq k \leq n$, the invariant is maintained trivially for w. If $v = p_k$, $1 \leq k < n$, $g(w)$ and its associated path are unaffected, but $g(p_k)$ takes a smaller value. The relation

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

and hence the invariant are maintained. If $v = p_n = w$, a new path will be associated with w: the path associated with u is enhanced with edge (u, v). Also in this case the invariant is preserved for w and for the other nodes.
Suppose an edge (w, v) is selected. We show that $v \neq p_k$ for any p_k, $0 \leq k \leq n$, in the path associated to w. If $v = p_k$ was the case, we would have before the update of $v = p_k$:

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

Algorithm 1 The Generic algorithm

```
1: $g(s) = 0$ and $g(u) = \infty$ for $u \neq s$;
2: while any edge $(u, v) \in E$ satisfies $g(v) - g(u) > d(u, v)$ do
3:     $g(v) = g(u) + d(u, v)$;
4:     pred$(v) = u$;
5: end while
```

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

The latest inequality is due to the invariant itself. The above relation implies $(p_k, p_{k+1}, \ldots, w = p_n, v = p_k)$ would be a negative cycle. However, we have excluded negative cycles from our graphs under investigation. We conclude that a path associated to a node w cannot convert into a cycle. $\Box$.

As the number of paths is finite, the number of updates is finite for any node w. Consequently, the algorithm has a finite number of iterations.
The proof of Invariant 1 also shows that $g(s)$ is never updated, which implies that $g(s) = 0$ is an invariant. Now, we show that $g(w)$ provides us with the shortest path length on termination. As $g(w)$ equals the length of a path by Invariant 1, $g(w) \geq \hat{d}(s, w)$. Every edge (u, v) on any path satisfies $d(u, v) \geq g(v) - g(u)$ and hence, every path P from s to w has the property: $\mathit{length}(P) \geq g(w) - g(s) = g(w)$. Conclusion $g(w) = \hat{d}(s, w)$.
Apart from the path in Invariant 1 we can identify a second path, whose definition is based on the following invariant.

Invariant 2 *For any edge $(u, v) \in E$ with $u =$pred(v), $g(v) \geq g(u) + d(u, v)$.*

The proof of this invariant is almost trivial. Using this invariant we can prove, analogously to Invariant 1, that any node w with finite label is associated with a path $(s = p_0, p_1, \ldots, p_k = w)$ such that p_{k-1}=*pred*(p_k). This path is called the predecessor path of w. The predecessor path P of a node w satisfies: $g(w) \geq$*length*(P), an immediate result of Invariant 2 and the invariant $g(s) = 0$. If the strict inequality *length*$(P) > g(w)$ holds, the predecessor path P differs from the path mentioned in Invariant 1. The discussion in the rest of this paper prefers the use of the predecessor path. On termination of the algorithm both types of paths provide us with the shortest path.

2.2 Refining the Generic algorithm

Algorithm 2, a refined version of the Generic algorithm, is called the S-set algorithm because of the crucial role of the set S. The edges adjacent to one node are selected jointly for relaxation. The variable c is an iteration counter, to be used in Section 4. There is an important invariant, the proof of which is left to the reader.

Invariant 3 *For any $(u, v) \in E$ with $u \in S$, $g(v) \leq g(u) + d(u, v)$.*

This invariant makes sure that the stop criterion of the S-set algorithm implies the stop criterion of the Generic algorithm. The S-set algorithm is non-deterministic, as the choice of u_0 is not prescribed. It can be viewed as a family of algorithms, comprising almost all shortest path algorithms. However, one well-known algorithm viz. Bellman-Ford does not belong to this family, but in the case that the data structure for the S-set is a queue, Algorithm 2 is a Bellman-Ford instance.
A great deal of papers, e.g. Cherkassky, Goldberg and Radzik (1996) and Gallo and Pallottino (1986), have been devoted to the question as to which data structure of the

Algorithm 2 The S-set algorithm

```
1: $g(s) = 0$ and $g(u) = \infty$ for $u \neq s$;
2: $S = \emptyset$;
3: $c = 0$;
4: **while** any node $u_0 \notin S$ has finite $g$-label **do**
5:    c=c+1;
6:    $S = S + \{u_0\}$;
7:    **for all** edges $(u_0, v)$ with $v \notin S$ **do**
8:       **if** $g(v) > g(u_0) + d(u_0, v)$ **then**
9:          $g(v) = g(u_0) + d(u_0, v)$;
10:         pred$(v) = u_0$;
11:         **if** $v \in S$ **then** $S = S - \{v\}$ **endif**
12:      **end if**
13:   **end for**
14: **end while**
```

S-set is suitable for which type of input graph. In the current paper this issue is left out of consideration.

2.3 Using a heuristic estimate

In some situations one may assume the availability of a so-called heuristic estimate. A heuristic estimate h is defined as a function from V into $\mathbb{R}$. Many search problems, e.g. the Traveling Salesman Problem (TSP) and the 13-14-15 puzzle, can be modelled as a shortest path problem with a heuristic estimate. The value $h(v)$ for any $v \in V$ is chosen such that this value estimates the distance from v to a target node. If no estimation method is available, $h \equiv 0$ is applied.
The h-value is a second label for each node next to the g-label. The value $g(v) + h(v)$ is referred to as $f(v)$. Using the heuristic estimate, we can define a more specific selection rule for the S-set algorithm. To that end, in front of line 5 or 6 we insert:

choose u_0 such that $f(u_0)$ is minimal outside S.

The S-set algorithm with this enhancement is called the Heuristic Estimate (HE) algorithm. The goal of the heuristic is to narrow the search space, which is defined as the set of nodes with finite g-label. Notice that adding a constant C to the function h does not affect the way a run of the HE-algorithm proceeds.

3 Heuristic estimates

In this section, we address several types of estimates. In subsection 3.1 a framework is discussed comprising amongst others Dijkstra, A* and the Modified label correcting version as mentioned in Ahuja, Magnanti and Orlin (1989). Subsection 3.3 treats the so-called non-misleading estimates.

3.1 Admissible and consistent estimates

The heuristic estimate h is called *admissible* with respect to a set $A \subseteq V$, if $h(v) - h(a) \leq \hat{d}(v,a)$ for any $v \in V$ and $a \in A$. We call a heuristic estimate h *consistent*, if h is admissible with respect to the full node set V, or put another way $h(u) - h(v) \leq \hat{d}(u,v)$ for any two nodes $u, v \in V$. In some literature a different but equivalent definition of *consistent* is found: $h(u) - h(v) \leq d(u,v)$ for any edge $u, v \in V$. For admissible estimates, we have the following important invariant.

Invariant 4 *If the heuristic estimate is admissible with respect to a set A, then*

a) $\forall u \in S,\ \forall v \in \bar{S} \cap A,\ f(u) \leq \hat{d}(s,v) + h(v) \leq f(v)$;

b) $\forall u \in S \cap A,\ g(u) = \hat{d}(s,u)$.

Proof. a) The right-hand inequality is obvious, because every g-label is associated with a path by Invariant 1. Therefore, we focus on the left-hand inequality.
Suppose a node u_0 is selected for insertion into S. Let v be any node in $\bar{S} \cap A$ and let P be the shortest path from s to v. The first node outside S on this path is denoted by p. The two parts of P between s and p and between p and v respectively are called P_1 and P_2. As a result of Invariant 3 we have $\mathit{length}(P_1) \geq g(p) - g(s) = g(p)$. Because $v \in A$, $\mathit{length}(P_2) = \hat{d}(p,v) \geq h(p) - h(v)$. Combining these inequalities gives $\hat{d}(s,v) = \mathit{length}(P) \geq g(p) + h(p) - h(v)$ or equivalently $\hat{d}(s,v) + h(v) \geq g(p) + h(p)$. The latest value is $\geq f(u_0)$ by the selection criterion.
Any vertex $v \in S \cap A$ is not removed from S, because $g(v) = \hat{d}(s,v)$ by invariant b) and hence, $g(v)$ has a permanent label. So $\bar{S} \cap A$ is not enhanced.
We conclude that the invariant is preserved in every iteration.
b) When a node $v \in \bar{S} \cap A$ is selected, then $v = u_0$ coincide with node p as defined in part a). This is a result of the inequalities in part a) which also imply $\hat{d}(s,v) = g(p)$. □

If the heuristic estimate is consistent, every node is inserted into S exactly once. In that case the algorithm is called a label-setting instance, as every node has a permanent label, when inserted into S. The run-time is $O(n^2)$ in that case. If the estimate is not consistent, a node in S may take a new label and leave the S-set. Then the algorithm is called label-correcting.
Invariant 4 implies that, when the heuristic estimate algorithm runs on a graph with an *admissible* estimate, each f-value in $\bar{S} \cap A$ is greater or equal than any f-value in $S \cap A$. Consequently, the f-values of the nodes of A consecutively selected make up a non-decreasing series. Notice that the heuristic estimate $h \equiv 0$ is consistent, provided that the weights are non-negative. The HE algorithm running on graph with non-negative weights and $h \equiv 0$ is exactly Dijkstra's algorithm.

The A* algorithm. Most of the textbooks on Artificial Intelligence (AI) discuss A*, which is defined in our terminology as the HE-algorithm using an estimate admissible with respect to a set A consisting of one single node. Curiously, this algorithm is not well-known among researchers of the OR community, which is focused on Dijkstra and Bellman-Ford. Conversely, the AI literature pays a lot of attention toA*, but ignores widespread algorithms such as Dijkstra and Bellman-Ford. Only recently, some OR researchers working on real

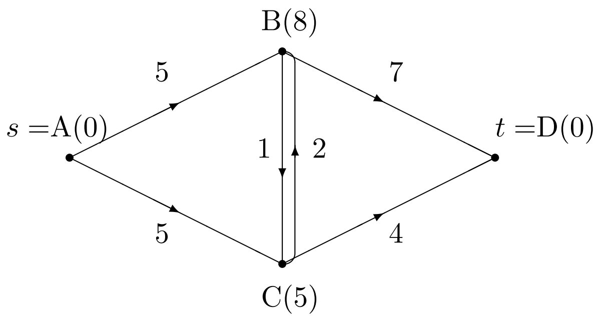

Figure 1: A non-misleading estimate, the h-values are in brackets.

road networks include A* in their investigations, e.g. GOLDBERG and HARRELSON (2005). The A* algorithm is presented in most of the AI textbooks, e.g. PEARL (1984) or RUSSELL (2003), as a suitable method to solve search problems, such as the 13-14-15 puzzle. The best-first branch-and-bound solution for TSP using a heuristic lower bound is actually an A* instance.

3.2 Running with or without heuristic function

Given a graph with weight function d and a heuristic estimate h, we introduce new weights:

$$d'(u,v) = d(u,v) + h(v) - h(u) \tag{1}$$

When running the HE-algorithm, we define an alternate label $g'(u) = f(u) - h(s)$ for any u. The initial labels $g(s)$ and $g'(s)$ are $= 0$. Whenever $g(v)$ is given a new value $g(v) = g(u) + d(u,v)$, this update can also be viewed as setting the alternate label $g'(v)$ to $g'(u) + d'(u,v)$. It follows that the HE algorithm using an h-function is equivalent to HE on a graph with transformed weights and a heuristic $\equiv 0$. If the heuristic is consistent, then $d' \geq 0$ and the algorithm running on the transformed graph is an instance of Dijkstra's algorithm. A graph with an admissible heuristic corresponds to one with possibly negative weights, but where every distance $\hat{d}(v,a)$ with $v \in V$ and $a \in A$ is non-negative.

3.3 Non-misleading and proper estimates

Suppose we focus on one target node t. A heuristic estimate h is called *non-misleading* if any two nodes m and n have the property:

$$\mathit{length}(P) + h(m) < \mathit{length}(Q) + h(n) \Rightarrow \mathit{length}(P) + \hat{d}(m,t) \leq \mathit{length}(Q) + \hat{d}(n,t) \tag{2}$$

for any two paths P and Q from s to m and to n respectively. An example of a non-misleading estimate is shown in Figure 1. A weaker notion is *proper*. A heuristic estimate h is called *proper* if (2) only holds for any two paths P and Q, that are not included in each other.

Invariant 5 *Suppose the HE algorithm runs on a graph with non-negative weights and with a proper estimate, where ties in the selection criterion are solved in favor of the smallest g-value.*

a) $\forall u \in S,\ \forall v \notin S,\ g(u) + \hat{d}(u,t) \leq g(v) + \hat{d}(v,t) \vee g(u) \leq g(v)$;

b) $\forall u \in S,\ g(u) = \hat{d}(s,u)$.

Proof. a) When a node u_0 is selected, the selection criterion says:

$$\forall v \notin S,\ g(u_0) + h(u_0) < g(v) + h(v) \quad \vee \quad g(u_0) \leq g(v).$$

Because of invariant b), no node has been deleted from S and hence, all predecessors are in S. Consequently the predecessor paths of u_0 and v cannot be subpaths of each other. As a result of the Invariants 2 and 3, the lengths of both predecessor paths are equal to the corresponding g-labels. As the estimate function h is proper, we state for all $v \notin S$:

$$g(u_0) + h(u_0) < g(v) + h(v) \Rightarrow g(u_0) + \hat{d}(u_0,t) \leq g(v) + \hat{d}(v,t)$$

If $g(v)$ is not changed, invariant a) holds for v and any $u \in S$. If v is given a new label $g(v) = g(u_0) + d(u_0,v)$, then for any $u \in S$ either $g(u) \leq g(u_0) \leq g(v)$ or:

$$g(u) + \hat{d}(u,t) \leq g(u_0) + \hat{d}(u_0,t) = g(v) - d(u_0,v) + \hat{d}(u_0,t) \leq g(v) + \hat{d}(v,t),$$

where the rightmost inequality is a consequence of a general inequality for the $\hat{d}$-function.
b) Let P be the shortest path from s to u_0. Similarly to the proof of Invariant 4 we derive: *length*$(P_1) \geq g(p)$. If $g(u_0) + \hat{d}(u_0,t) \leq g(p) + \hat{d}(p,t)$ in a) holds, then using *length*$(P_2) = \hat{d}(p,u_0) \geq \hat{d}(p,t) - \hat{d}(u_0,t)$ we obtain: *length*$(P) \geq g(p) + \hat{d}(p,t) - \hat{d}(u_0,t) \geq g(u_0)$. If $g(u_0) \leq g(p)$ in a) holds, then *length*$(P) \geq$*length*$(P_1) \geq g(p) \geq g(u_0)$. □

This invariant shows that the HE-algorithm running with a proper estimate is a label-setting instance. In contrast with a consistent estimate, the f-values successively selected do not generate an increasing series.
The notion of *non-misleading* has been introduced in Ibaraki (1977) in the context of the branch-and-bound method. The definition in that paper differs from the current one, which is taken from Bagchi (1983). They are equivalent in case of an acyclic graph. The notion *proper* has been introduced in Bagchi (1983). In that paper, also a weaker version of Invariant 5 was presented. The result that the label-setting version holds for a proper heuristic estimate, is new.

4 An application of binary numbers

In this section, we present a pathological graph, where the HE-algorithm has an exponential run time of $2^{|V|-1}$. Consider the following graph K where the nodes are denoted by integer numbers.

$$\begin{aligned} V &= \{1, 2, \ldots, n, n+1\} \\ E &= \{(u,v) \mid u, v \in V \ \textit{and}\ u > v\} \\ d(u,v) &= 2^{u-1} - 2^{v} \\ s &= n+1 \end{aligned}$$

Suppose we insert the following statement into the S-set algorithm in front of line 5 or 6:

$$\textit{choose } u_0 \textit{ such that } u_0 \textit{ is the smallest integer outside } S. \tag{3}$$

Later on in this section, we show that the S-set algorithm with this selection rule is an instance of HE. Running the S-set algorithm on graph K with this selection rule provides an interesting application of the binary number system, expressed by Invariant 7. This invariant deals with the variable c, the iteration counter (see line 5 in the S-set algorithm). The following definition $I(u)$ is useful for Invariant 6: $I(v) = \{p \mid v < p \leq n \textit{ and } p \textit{ integer}\}$.

Invariant 6 *Let $I(v) \cap S$ for any v be given by $\{p_1, \ldots, p_k\}$ with $p_1 > p_2 \ldots > p_k$. Then the predecessor path of v is $(n+1 = p_0, p_1, p_2 \ldots, p_k, v)$ and $g(v)$ equals the length of this path.*

Proof. Suppose u_0 is selected to be inserted into S. Let the set $I(u_0) \cap S$ be given by $\{p_1, \ldots, p_h\}$. By the invariant itself the predecessor path of u_0 is $(s = p_0, p_1, p_2 \ldots, p_h, u_0)$ and this path has length $= g(u_0)$. Since u_0 is the smallest integer outside S according to the selection rule, the set $\{u_0 - 1, u_0 - 2, \ldots, 2, 1\}$ is also included in S.
When u_0 is inserted into S, any node v with $v > u_0$ is not affected, because every edge goes from a higher to a lower number. So we consider only nodes v with $u_0 > v \geq 1$. For such a node v the predecessor path has the form $(n+1 = p_0, p_1, p_2 \ldots, p_h, u_0 - 1, u_0 - 2, \ldots, v)$. Notice that every edge in the sequence $(u_0 - 1, u_0 - 2, \ldots, v)$ has weight $= 0$. As the length of this path equals $g(v)$ by the invariant, we have:

$$\begin{array}{rcl} g(v) & = & \sum_{i=0}^{i=h-1} d(p_i, p_{i+1}) + d(p_h, u_0 - 1) \\ & = & \sum_{i=0}^{i=h-1} d(p_i, p_{i+1}) + 2^{p_h - 1} - 2^{u_0 - 1}. \end{array}$$

The edge (u_0, v) is relaxed, because

$$\begin{array}{rcl} g(u_0) + d(u_0, v) & = & \sum_{i=0}^{i=h-1} d(p_i, p_{i+1}) + d(p_h, u_0) + d(u_0, v) \\ & = & \sum_{i=0}^{i=h-1} d(p_i, p_{i+1}) + 2^{p_h - 1} - 2^{u_0} + 2^{u_0 - 1} - 2^v \\ & = & \sum_{i=0}^{i=h-1} d(p_i, p_{i+1}) + 2^{p_h - 1} - 2^{u_0 - 1} - 2^v \\ & = & g(v) - 2^v. \end{array}$$

Hence, when u_0 is inserted into S, every v with $u_0 > v \geq 1$ takes a new g-label and leaves the S-set. For each v in this range the new set $I(v) \cap S$ is equal to $(p_1, \ldots, p_h, u_0)$. Since (u_0, v) is relaxed, the new predecessor path of v consists of the predecessor path of u_0 enhanced with (u_0, v). So this path conforms to the new value of $I(v) \cap S$. As a result of Invariant 2 and 3 the length of the predecessor path equals $g(v)$. $\square$

Invariant 7 *Let the binary notation of $c - 1$ be given by $a_n a_{n-1} \ldots a_2 a_1$. This notation and the set S are related by the equality: $S \backslash \{s\} = \{p \mid a_p = 1\}$*

Proof. Notice that the invariant holds when $c = 1$ (only source $n + 1$ has been inserted). Assume that the invariant holds at the time u_0 is selected, so $c - 1$ has the form: $a_n a_{n-1} \ldots a_{u_0+1} 01 \ldots 1$. In the next iteration u_0 enters S causing the nodes v with $1 \leq v < u_0$ to leave the S-set, as we have seen in the proof of Invariant 6. Adding 1 to the old $(c-1)$-value gives $a_n a_{n-1} \dots a_{u_0+1} 10 \dots 0$, which is in accordance with the new S-set. □

Now it is obvious that the number of iterations equals exactly 2^n, since $c-1 = 2^n - 1$, when $S = V$ is achieved ($c-1$ consists of n digits 1).
Invariant 6 implies that $g(v)$ is a multiple of 2^v at any time, since the length of the predecessor path is composed of numbers 2^p with $p \geq v$. Every subset of $I(v)$ is equal to $I(v) \cap S$ at some time. Consequently, $g(v)$ takes $2^{|I(v)|} = 2^{n-v}$ different values, ranging from $0 * 2^v$ up to $(2^{n-v} - 1) * 2^v = 2^n - 2^v$. The proof of Invariant 6 also shows that, whenever v is given a new label, the label is decreased by 2^v.
The following nodes are inserted successively after node s: 1, 2, 1, 3, 1, 2, 1, 4, 1, 2, 1, 3, 1, 2, 1, ... This series is also known from 'Towers of Hanoi', a famous topic out of recreational mathematics, which is often treated in textbooks on programming to illustrate recursion.

The definition of graph K has been presented in Pijls (1991) and in a slightly different form in Ahuja, Magnanti and Orlin (1989). It is based upon an older instance, introduced in Martelli (1977).

By adding target node 0 and adding an edge $(1, 0)$ with $d(1, 0) = 2^n$ to the graph K, we obtain an instance K' of the A*-algorithm. It follows that also A* may have exponential run time.
The following lemma implies that the above instance K with $2^{|V|-1}$ iterations achieves a maximal number of iterations.

Lemma 1 *Consider two arbitrary iterations of the S-set algorithm. Then there is a node u with $u \notin S$ after the earlier iteration and $u \in S$ after the later iteration.*

Proof. Let g_1 and g_2 denote the g-labels after the two iterations under consideration. Let p be a node such that $g_2(p) < g_1(p)$. Let the predecessor path after the second iteration be given by $(s = p_0, p_1, \dots, p_n = p)$. Let p_{k-1} and p_k be two consecutive nodes in this path, such that $g_1(p_{k-1}) = g_2(p_{k-1})$ and $g_1(p_k) > g_2(p_k)$. Such nodes must exist, due to the fact that $g_1(p_0) = g_2(p_0)$ and $g_1(p_n) > g_2(p_n)$. We conclude that the latest update of $g(p_k)$ has been executed more recently than the latest update of $g(p_{k-1})$. Hence, p_{k-1} has not been updated after it became p_k's predecessor, so $u = p_{k-1}$ is in S after the second iteration. The invariants 2 and 3 imply that $g_2(p_k) = g_2(p_{k-1}) + d(p_{k-1}, p_k)$. Since $g_1(p_k) > g_2(p_k) = g_2(p_{k-1}) + d(p_{k-1}, p_k) = g_1(p_{k-1}) + d(p_{k-1}, p_k)$, we conclude from Invariant 3 that $u = p_{k-1}$ was not in S after the first iteration. □

By Lemma 1 the S-set after each iteration is unique. In the very first iteration of the algorithm s is inserted into S and this node remains in S throughout the algorithm. Hence, the number of iterations is at most the number of subsets (the empty set included) of $V \backslash \{s\}$, which number equals $2^{|V|-1}$. We see that graph K having $2^n = 2^{|V|-1}$ iterations attains the maximal number of iterations.

We conclude this section with a personal note. When I communicated my discovery on the role of binary numbers to Antoon, he responded: "Wat is wiskunde toch mooi!". (How beautiful mathematics is!). No doubt, we agree with this statement.

5 Real road networks and Bidirectional search

The past few years witnessed a revival of shortest paths algorithms, in particular algorithms focused on real road networks. The instance that is solved by the present-day car navigators, is the single-source single-target version of the shortest path problem. Bidirectional search is a powerful method for finding the shortest path in a road network from one point to another and is applied therefore in almost any navigator. The method originates from Pohl (1972). Substantial improvements were proposed in amongst others Ikeda et al. (1994), Goldberg and Harrelson (2005) or Yamaguchi and Masuda (2006). Given a source and a target, Bidirectional search works as follows. The HE algorithm runs simultaneously on two sides, starting from the source and the target respectively, so we have forward and backward search. The backward search uses different edges: just edges (v, u) such that (u, v) is in the original set E, are used. The cost of an edge (v, u) equals the value of the original edge (u, v). Each search process has its own heuristic. The algorithm stops when the S-sets of the two searches have some overlap. The exact termination criterion depends on the properties of the heuristics. An detailed treatment of Bidirectional search can be found in Pijls and Post (2006).

Suitable estimates. An obvious consistent heuristic in roads networks is: $h(v)$ equal to the astronomical distance from v to t. HE running with this heuristic estimate visits considerably less nodes than Dijkstra's algorithm ($h \equiv 0$) as is illustrated by some pictures in Roos (2004) or in Klunder and Post (forthcoming).
In Goldberg and Harrelson (2005) a new estimate was introduced, incorporated in the so-called ALT algorithm, which stands for A*, Landmarks and Triangle inequality. The algorithm needs some preprocessing of the road network. A number of nodes are designated as so-called *landmarks*. In advance, $\hat{d}(v, m)$ is computed for any node v and every landmark m. Using the triangle inequality we can show that the choice $h(v) = \hat{d}(v, m) - \hat{d}(t, m)$, m a fixed landmark, gives a consistent heuristic. If we choose m close to t, a tight lower bound to $\hat{d}(v, t)$ is obtained. An extensive discussion is given in Goldberg and Harrelson (2005).

New methods. Apart from the landmarks several other new heuristics have been invented in the past few years. All of them need an elaborate preprocessing phase to compute the heuristic. Examples of new heuristics can be found in: Möhring et al. (2005), Wagner, Willhalm and Zaroliagis (2005), or Sanders and Schultes (2005).

6 The search space

In this section we want to find which set of nodes is visited during the HE algorithm. For an admissible h the set $R_h(H)$ with H a real number is defined as the set of nodes v for which a path $P = (s = p_0, p_1, \ldots, p_n = v)$ exists such that for $0 \le k \le n$,

$$\sum_{j=0}^{k-1} d(p_j, p_{j+1}) + h(p_k) \le H, \tag{4}$$

If the heuristic h is *consistent*, the definition can be simplified. Then the inequality (4) holds automatically for $1 \le k < n$ if it holds for $k = n$.

The following proposition follows immediately from the definition: $h_1 \leq h_2 \wedge H_1 \geq H_2 \Rightarrow R_{h_2}(H_2) \subseteq R_{h_1}(H_1)$.
During execution of the S-set algorithm the variable F is defined at any time as:

$$F = \min\{f(v) \mid v \in S\}$$

Invariant 4 shows that, in case of a consistent estimate, $F \leq f(v)$ for any $v \notin S$. If the estimate is not consistent, for some $v \notin S$ the inequality $f(v) < F$ may happen. We introduce the following selection criterion into the S-set algorithm:

(1) *select, if any, an arbitrary node $u_0 \notin S$ with $f(u_0) \leq F$*
(2) *if $f(u) > F$ for any $u \notin S$, select u_0 such that $f(u_0)$ is minimal outside S*
(and after inserting u_0 into S, $f(u_0)$ is the new F-value)

We call the S-set algorithm with this additional selection rule the *F-algorithm*. Notice that the selection criterion in section 2.3 is a special case of this one. If the estimate is consistent, the two criteria are equivalent. As soon as t is inserted into S, the algorithm can stop, since $g(t) = \hat{d}(s,t)$ due to Invariant 4.
The following property holds during execution: $S \subseteq R_h(F)$ and on termination: $S \subseteq R_h(\hat{d}(s,t) + h(t))$. At the moment that every node $v \notin S$ satisfies $f(v) > F$, then even $S = R_h(F)$.
Although H may be any real number, there is finite number of distinct sets $R_h(H)$ for a given estimate h on a graph G. To be more precise, there is a finite series of values $H_0, H_1, H_2, \ldots H_n$ with the property that any set $R_h(H)$, H a real number, is equal to $R_h(H_{k_0})$ with k_0 in the range $0 \ldots n$. Consequently, the series of values assigned to the variable F during execution of the F-algorithm is equal to the series $H_0, = 0, H_1, H_2, \ldots, H_n$. So the F-algorithm may be viewed as a method to trace the sets $R_h(H_k)$, $0 \leq k \leq n$. The IDA algorithm (see Pearl (1984) or Russell (2003)) is based upon this principle. In the IDA* algorithm, the sets $R_h(H_k)$, $0 \leq k \leq n$ are traced successively using backtracking. This makes the algorithm appropriate to be used in restricted memory.

7 Concluding remarks

According to Antoon there is no need to sum up and to repeat the results in a conclusion at the end of a paper. Our conclusion is different. We remember Antoon as our supervisor and master with a strong commitment to his students. In memory to him and in great gratitude we have written this article.

References

Ahuja R.K., T.L. Magnanti and J.B. Orlin (1989), *Network Flows, Theory, Algorithms and Applications,* Prentice Hall.

Bagchi A. and A. Mahanti (1983), Search Algorithms under Different Kinds of Heuristics - A Comparative Study, *Journal American Computing Machinery* **30**, 1-21.

CHERKASSKY B.V., A.V. GOLDBERG and T. RADZIK (1996), Shortest path algorithms: Theory and Experimental Evaluation, in: *Mathematical Programming* **73:22**, 129-174.

GALLO G. and S.PALLOTTINO (1986), Shortest path methods: a unifying approach, *Mathematical Programming Study* **26**, 38-64.

GAREY M. R. AND D.S, JOHNSON (1979), *Computers and intractability, A guide to the theory of NP-Completeness,* Freeman, New York.

GOLDBERG A.V. and C.HARRELSON (2005), Computing the Shortest Path: A* Search Meets Graph Theory, in: 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'05), extended version in: Technical Report MSR-TR-2004-24, (2004) Microsoft Research.

GOODRICH M.T. and R. TAMASIA 2002, *Algorithm Design*, John Wiley and Sons, 2002.

IBARAKI T. 1977, The Power of Dominance Relations in Branch-and-Bound Algorithms, *Journal American Computing Machinery* **24**, 264-279.

IKEDA T.K., M.-Y. HSU, H. INAI, S. NISHIMURA, H. SHIMOURA, T. HASHIMOTO, K. TENMOKU, and K. MITOH (1994), A Fast Algorithm for Finding Better Routes by AI Search Techniques, in: Proceedings Vehicle Navigation and Information Systems Conference, Yokohama, IEEE, p. 291-296.

KLUNDER G.A. and H.N. POST (forthcoming), The Shortest Path Problem on Large Scale Real Road Networks, Networks, 48:4, 2006, p. 182-194.

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

MŐHRING ROLF H., HEIKO SCHILLING, BIRK SCHŰTZ, DOROTHEA WAGNER and THOMAS WILLHALM (2005), Partitioning Graphs to Speed Up Dijkstra's Algorithm, in: Sotiris E. Nikoletseas (editor), *Proceedings Experimental and Efficient Algorithms: 4th International Workshop*, Lecture Notes in Computer Science vol. 3503, 198-202, Springer.

PAPADIMITRIOU CHR. and K. STEIGLITZ (1982), *Combinatorial Optimization: Algorithms and Complexity*, Prentice Hall.

PEARL J. (1984), *Heuristics, Intelligent search strategies for Computer Problem Solving,* Addison Wesley.

POHL I. (1971), Bi-Directional Search, *Machine Intelligence* **6**, 124-140.

PIJLS W. (1991), *Shortest paths and Game trees*, Ph.D.thesis, Erasmus University Rotterdam.

PIJLS W. and H.N. POST (2006), Bidirectional A*: Comparing balanced and symmetric heuristic methods, Econometric Institute Report EI 2006-41, Erasmus University Rotterdam.

ROOS C. (2004), De tijd zal het leren, inaugural address (in Dutch), in: *Nieuw Archief voor Wiskunde*, vijfde serie **5** nr 3, september 2004.

RUSSELL S.J. and P.NORVIG (2003), *Artificial Intelligence, A modern Approach*, Prentice Hall.

SANDERS P. and D. SCHULTES (2005), Highway Hierarchies hasten Exact Shortest path Queries, in: G.S. Brodal and S. Leonardi (editors), *Proceedings 17th European Symposium on Algorithms (ESA)*, Lecture Notes in Computer Science vol. 3669, 568-579, Springer.

WAGNER D., T. WILLHALM and CHR. ZAROLIAGIS (2005), Geometric Containers for Efficient Shortest-Path Computation, in: ACM Journal of Experimental Algorithms **10** article no. 1.3, 1-30.

YAMAGUCHI K. AND S. MASUDA (2006), An A* Algorithm with a New Heuristic Distance Function for the 2-Terminal Shortest Path Problem, *IEICE Transactions on Fundamentals of Electronics* **E89-A**, 544-550.