ELSEVIER Science of Computer Programming 22 (1994) 3–19

Science of Computer Programming

Calculating path algorithms

Roland C. Backhouse*, J.P.H.W. van den Eijnde, A.J.M. van Gasteren

Department of Mathematics and Computing Science, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, Netherlands

Communicated by C. Morgan; revised October 1993

Abstract

A calculational derivation is given of two abstract path algorithms. The first is an all-pairs algorithm, two well-known instances of which are Warshall's (reachability) algorithm and Floyd's shortest-path algorithm; instances of the second are Dijkstra's shortest-path algorithm and breadth-first/depth-first search of a directed graph. The basis for the derivations is the algebra of regular languages.

1. Introduction

This paper presents a calculational derivation of two algorithms that are abstractions of path problems on directed labelled graphs, where the labels satisfy the properties of a regular algebra.[1] The fact that the elementary operators involved in several path-finding algorithms obey the axioms of regular algebra is widely known. Specific instances of the abstract algorithms are also well-known, such as Warshall's (all-pairs) reachability algorithm and Floyd's shortest-path algorithm for the first abstract program to be presented, and Dijkstra's shortest-path algorithm and depth-first/breadth-first search for the second.

In the present paper, however, the emphasis is on the *derivation* rather than on the algorithms themselves. The main distinguishing feature of the derivation is that, by exploiting the correspondence between the graphs involved and matrices, the algorithms are developed by calculating with *matrices* rather than *matrix elements*. Only as a final step of the derivation is the algorithm operating on matrices transformed into a program expressed in terms of matrix elements. In our view this leads to a more compact and more disentangled derivation.

* Corresponding author. E-mail:wsinrbc@win.tue.nl.

[1] Sometimes known as the algebra of regular languages.

0167-6423/94/$07.00 © 1994 Elsevier Science B.V. All rights reserved
SSDI 0167-6423(93)E0019-V

The first problem dealt with in this paper is the simpler of the two. As the most elementary illustration of the use of matrix calculation it is included mainly to show the calculus in action before it is used in a more complicated problem.

2. Problem statement

Given is a (non-empty) set N and an $|N| \times |N|$ matrix A, the rows and columns of which are indexed by elements of N. It is assumed that the matrix elements are drawn from a regular algebra $(S, +, \cdot, *, 0, 1)$. The first problem is to derive an algorithm computing matrix A^+, i.e. $A \cdot A*$, whereby the primitive terms in the algorithm do not involve application of the $*$ operator to matrices.

In the second problem, the regular algebra satisfies two additional properties:

S1. The ordering $\leqslant$ induced[2] by $+$ is a total ordering on the elements of S.

S2. 1 is the largest element in the ordering.

In addition to N and A, one is given a $1 \times |N|$ matrix b. Hereafter, $1 \times |N|$ matrices will be called "vectors", 1×1 matrices will be called "elements" and $|N| \times |N|$ matrices will be called "matrices". The second problem is to derive an algorithm to compute the vector $b \cdot A*$ whereby the primitive terms in the algorithm do not involve any use whatsoever of the $*$ operator.

3. Interpretations

The relevance and interest of the stated problems is that they abstract from several path problems on labelled, directed graphs. Let $G = (N, E)$ be a directed graph with node set N and labelled-edge set E, the labels being drawn from some regular algebra $(S, +, \cdot, *, 0, 1)$. Then (as we deem known, see, for instance, [5,10]), there is a correspondence between edge sets E and $|N| \times |N|$ matrices A whereby the (i, j)th element of A equals the label of the edge from node i to node j in the graph, if present, and 0 otherwise. (For graphs with multiple edges, the correspondence will be adjusted later; note that absent edges and edges with label 0 are not distinguished.)

Since S forms a regular algebra $(S, +, \cdot, *, 0, 1)$, matrix multiplication can be defined in the usual way with the usual properties. An important theorem is that the set of square matrices of a fixed size with entries drawn from S itself forms a regular algebra, with the usual definitions of the zero and identity matrices, matrix addition and matrix multiplication. The derivation to be presented makes extensive use of this theorem. (For further discussion see [1,5,9].)

[2] $x \leqslant y \equiv x + y = y$.

If the label of a path is defined to be the product of its constituent edge labels (taken in the order defined by the path), the (i, j)th entry of A^k is the sum of the labels of the paths of length k from node i to node j. Note that this still holds true if we redefine the (i, j)th entry of A to be the *sum* of the labels of all edges from i to j, thus admitting multiple edges. Similarly, the (i, j)th entry of $A*$ (or A^+) is the sum of the labels of all (all non-empty) paths from node i to node j. Moreover, if b is a vector that differs from the zero vector only in its ith entry, for some node i, then the jth entry of $b \cdot A*$ is the sum of all labels of paths beginning at node i and ending at node j.

Three interpretations of a regular algebra satisfying properties S1 and S2 are given in Table 1. (Note that property S2 implies that $x* = 1$ for all $x \in S$. For this reason the interpretation of $*$ has been omitted.)

The first interpretation is that appropriate to finding shortest paths; the length (i.e. label) of a path is the (arithmetic) sum of its constituent edge lengths (labels) and for each node x the minimum (denoted in Table 1 by $\downarrow$) such length is sought to node x from a given, fixed start node. In this case, the first algorithm we derive finds the length of all shortest paths through a graph, and is known as Floyd's algorithm. The second algorithm we derive is known as Dijkstra's shortest-path algorithm [7] (the carrier set, S, being restricted to *non-negative* reals in order to comply with requirement S2).

The second interpretation is that appropriate to solving reachability problems; the (i, j)th entry of matrix A is **true** if and only if there is an edge in E from node i to j, and the "length" of a path is always **true**. The first algorithm we derive determines for each pair of nodes i and j whether i is reachable from j (by a path of non-zero length) and is known as Warshall's algorithm. The second algorithm forms the basis of the so-called depth-first and breadth-first search methods for determining for each node in the graph whether or not there is a path in the graph to that node from the given start node. (The choice of 'breadth-first" or "depth-first" search depends on further refinement steps that we do not discuss in detail.)

The third interpretation is appropriate to bottleneck problems; the edge labels may be construed as, say, bridge-widths and the "length" of a path is the minimum width of bridge on that path. Sought is, for each node in the graph, the minimum bridge-width on a route to the node from – in the case of the first algorithm – all other nodes in the graph, or – in the case of the second algorithm – a given start node that maximises that minimum. In this case neither algorithm appears to have any specific name.

For more discussion of these interpretations see [5].

Table 1
Regular algebras

	S	$+$	$\cdot$	0	1	$\leqslant$
Shortest paths	Non-negative reals	$\downarrow$	$+$	∞	0	$\geqslant$
Reachability	Booleans	$\vee$	$\wedge$	false	true	$\Rightarrow$
Bottlenecks	Reals	$\uparrow$	$\downarrow$	$-\infty$	∞	$\leqslant$

4. Regular algebra

The framework for the current derivation is regular algebra. The axioms of regular algebra – the algebra of regular languages – are now widely known and publicised. (See e.g. [3,5,6].) The specific details of the framework are that $(S, +, \cdot, *, 0, 1)$ is a regular algebra. That is, S is a set on which are defined two binary operators $+$ and $\cdot$ and one unary operator $*$ (written as a postfix of its argument). Addition $(+)$ is associative, commutative and idempotent. Multiplication $(\cdot)$ distributes over addition and is associative but is not necessarily commutative. The basic properties of $*$ that we use here are, for all $a, b \in S$:

$$a* = 1 + a \cdot a* \;\wedge\; a* = 1 + a* \cdot a\,, \tag{1}$$

$$a \cdot (b \cdot a)* = (a \cdot b)* \cdot a\,, \tag{2}$$

$$(a + b)* = a* \cdot (b \cdot a*)* \;\wedge\; (a + b)* = (a* \cdot b) \cdot a*\,, \tag{3}$$

$$1* = 1\,. \tag{4}$$

Rule (2) will be referred to as the "leap-frog rule" whilst rule (3) will be called the "star-decomposition rule". The main contribution made by Backhouse and Carré [3] was to show that these four rules are at the heart of several elimination techniques for solving shortest-path and other path-finding problems.

A well-known rule of regular algebra identifies $a* \cdot b$ as a least fixed point:

$$a* \cdot b \leqslant x \;\Leftarrow\; a \cdot x + b \leqslant x\,.$$

We do *not* use this rule in our derivations. Use of this rule is nonetheless implicit in our claim that the problems we consider are indeed abstractions of the specific path-finding problems discussed in Section 3.

Two additional rules (which we *do* use) are the following consequences of the idempotence of $*$ and $^+$, respectively.

$$(a^+)* = a*\,, \tag{5}$$

$$a* \cdot a* = a*\,. \tag{6}$$

5. Selectors

Although the matrices we consider form a regular algebra, they will obviously never satisfy the requirements S1 and S2, even if their elements do. Given that we wish to appeal to these properties from time to time, it is important to keep track of which terms in our calculations denote elements, which denote vectors and which denote matrices. The rules for doing so are simple and (hopefully) familiar: the product of an $m \times n$ and an $n \times p$ matrix is an $m \times p$ matrix, and addition and $*$ preserve the dimension of their arguments. It remains, therefore, to adopt a systematic naming convention for the variables that we use. This, and a primitive mechanism for forming vectors, is the topic of this section.

During the course of the development the following naming conventions will be used.

$\mathbf{0}$	$1 \times \lvert N \rvert$ vector that is everywhere 0,
k, j	nodes of the graph (i.e. elements of N),
L, M, P	subsets of N,
V, W, X, Y, Z	$\lvert N \rvert \times \lvert N \rvert$ matrices,
u, v, w, x	$1 \times \lvert N \rvert$ vectors.

For each node k we denote by $k\blacklozenge$ the $1 \times |N|$ vector that differs from $\mathbf{0}$ only in its kth component which is 1. Such a vector is called a *primitive selector vector*. The transpose of $k\blacklozenge$ (thus, an $|N| \times 1$ vector) is denoted by $\blacklozenge k$. We define the *primitive selector matrix* $\underline{k}$ by the equation

$$\underline{k} = \blacklozenge k \cdot k\blacklozenge . \tag{7}$$

Any sum (including the empty sum, of course) of primitive selector vectors (respectively, matrices) is called a *selector vector* (respectively, *matrix*). In fact, in most cases, instead of using one of these four terms we shall just use the term *selector*, it being clear from the context whether the designated selector is primitive or not, and a vector or a matrix.

This terminology is motivated by the interpretation of the product of a matrix and a selector. Specifically, $j\blacklozenge \cdot Y$ is a vector consisting of a copy of the jth row of matrix Y, and $j\blacklozenge \cdot Y \cdot \blacklozenge k$ is the (j,k)th element of Y. Furthermore, there is a (1–1) correspondence between subsets of N and selector matrices given by the function mapping M to $\underline{M}$ where, by definition,

$$\underline{M} = \Sigma(k : k \in M : \underline{k}) . \tag{8}$$

Note that

$$\underline{\{k\}} = \underline{k} . \tag{9}$$

This silent "lifting" of a function from elements to sets is not uncommon and very convenient; it should not be a cause for confusion since we do not mix the two forms, preferring always to use the shorter form.

For all vectors x and all matrices Y, $x \cdot \underline{M}$ is a copy of x except that all elements of x with index outside M are zero, and $Y \cdot \underline{M}$ (respectively, $\underline{M} \cdot Y$) is a copy of matrix Y except that all columns (respectively, rows) of Y with index outside M are zero.

The derivation that follows is not dependent on knowing these interpretations of the selectors; rather, we make use of a small number of characteristic algebraic properties. The first is that matrix product is associative. This is the most important property and its exploitation is the reason for introducing the primitive selectors. However, we shall nowhere explicitly mention the use of associativity, in line with the doctrine that the application of the most important properties should be invisible. The second property is that $\underline{\phi}$ (where ϕ denotes the empty set of nodes) is the zero element in the algebra of $|N| \times |N|$ matrices. In particular, for all vectors u,

$$u \cdot \underline{\phi} = \mathbf{0} \tag{10}$$

and, for all matrices X,

$$X \cdot \underline{\phi} = \underline{\phi} \cdot X = \underline{\phi} \tag{11}$$

and

$$X + \underline{\phi} = X\,. \tag{12}$$

(These properties are immediate consequences of the definition of the "underlining" function.) Two other properties are that, for all nodes k,

$$k_{\blacklozenge} \cdot {}_{\blacklozenge}k = 1 \tag{13}$$

and, for all distinct nodes j and k,

$$j_{\blacklozenge} \cdot {}_{\blacklozenge}k = 0\,. \tag{14}$$

It follows, by straightforward calculation, that, for all sets of nodes M and all nodes k,

$$k \in M \;\Rightarrow\; (\underline{M} \cdot {}_{\blacklozenge}k = {}_{\blacklozenge}k \;\wedge\; \underline{M} \cdot \underline{k} = \underline{k}) \tag{15}$$

and, for all sets of nodes L and M,

$$\underline{L} \cdot \underline{M} = \underline{L \cap M}\,. \tag{16}$$

In particular,

$$\underline{M} \cdot \underline{M} = \underline{M} \tag{17}$$

and

$$L \cap M = \phi \;\Rightarrow\; \underline{L} \cdot \underline{M} = \underline{\phi}\,. \tag{18}$$

The final property is that all matrices and all vectors are indexed by the given node set N. This we render by the equations:

$$X \cdot \underline{N} = X = \underline{N} \cdot X \tag{19}$$

and

$$x \cdot \underline{N} = x \tag{20}$$

for all matrices X and all vectors x.

6. The Warshall–Floyd algorithm

In this section we deal with the simpler of the two problems, viz. the computation of A^{+}. The goal is to provide a gentle introduction to the calculational techniques, in particular the use of star decomposition to derive an initial algorithm at matrix level, followed by the use of the leap-frog rule to convert that algorithm to one in terms of matrix elements.

6.1. The algorithm

The heuristic underlying the computation is to use star decomposition in order to build up the given matrix A under the star operator from the null matrix to its final form. More precisely, we introduce selector matrix $\underline{L}$ with loop invariant

$$X = A\cdot(\underline{L}\cdot A)* \tag{21}$$

and postcondition

$$X = A\cdot A*. \tag{22}$$

The invariant is established by $X, L := A, \phi$, since by (1) we have $\mathbf{0}* = \mathbf{1}$. For $\underline{L} = \mathbf{1}$, i.e. $L = N$, (21) implies the postcondition, and our aim is to extend L with one node in each step of the repetition. This results in the following skeleton algorithm.

```
$X, L := A, \phi$
;do $L \neq N \rightarrow$
        $k :\in N - L$
        $; X := A\cdot((\underline{L} + \underline{k})\cdot A)*$
        $; L := L \cup \{k\}$
od $\{X = A^+\}$.
```

In this algorithm only the assignment to X in the body of the repetition needs to be simplified. Proceeding with the evaluation of its right-hand side, we calculate

$$A\cdot((\underline{L} + \underline{k})\cdot A)*$$
$$= \quad \{\text{distributivity}\}$$
$$A\cdot(\underline{L}\cdot A + \underline{k}\cdot A)*$$
$$= \quad \{\text{star decomposition (3)}\}$$
$$A\cdot(\underline{L}\cdot A)*\cdot(\underline{k}\cdot A\cdot(\underline{L}\cdot A)*)*$$
$$= \quad \{(21)\}$$
$$X\cdot(\underline{k}\cdot X)*.$$

Thus, the assignment to X in the body can be replaced by

$$X := X\cdot(\underline{k}\cdot X)*.$$

In order to arrive at an algorithm in which the star operator is applied to matrix elements only, or, more precisely, to singleton matrices we continue the calculation:

$$
\begin{aligned}
& X\cdot(\underline{k}\cdot X)* \\
= \;& \{\text{unfolding (1), distributivity}\} \\
& X + X\cdot(\underline{k}\cdot X)*\cdot\underline{k}\cdot X \\
= \;& \{\underline{k} = \blacklozenge k\cdot k_\blacklozenge \text{ (7)}\} \\
& X + X\cdot(\blacklozenge k\cdot k_\blacklozenge\cdot X)*\cdot\blacklozenge k\cdot k_\blacklozenge\cdot X \\
= \;& \{\text{leap-frog rule (2) for } \blacklozenge k\} \\
& X + X\cdot\blacklozenge k\cdot(k_\blacklozenge\cdot X\cdot\blacklozenge k)*\cdot k_\blacklozenge\cdot X.
\end{aligned}
$$

Note that in the above calculation the star decomposition rule is used so as to arrive at the intermediate expression $X\cdot(\underline{k}\cdot X)*$, while the leap-frog rule is exploited so as to transform this expression into the required form: only in the final step of the calculation do we arrive at an expression in which the star operator is applied to an element, or more precisely to the singleton matrix $k_\blacklozenge\cdot X\cdot\blacklozenge k$. Rewriting the assignment to X in the body according to the above and replacing set variable L, using representation invariant $L = [0\,.\,.k) \wedge 0 \leqslant k \leqslant n$, we arrive at the following abstract algorithm:

```
$X, k := A, \phi$
;do $k \neq n \rightarrow X := X + X\cdot\blacklozenge k\cdot(k_\blacklozenge\cdot X\cdot\blacklozenge k)*\cdot k_\blacklozenge\cdot X$
          $;k := k + 1$
od $\{X = A^+\}$.
```

6.2. *Implementation freedom*

The algorithm we have obtained is not quite Warshall's algorithm or Floyd's algorithm (even after suitable interpretation of the operators). The reason is that at element level the assignment in the body of the loop is a *simultaneous* assignment to all matrix elements. Spelling this out in detail, premultiplying by $i_\blacklozenge$ and postmultiplying by $\blacklozenge j$, the matrix assignment

$$X := X + X\cdot\blacklozenge k\cdot(k_\blacklozenge\cdot X\cdot\blacklozenge k)*\cdot k_\blacklozenge\cdot X$$

is directly implemented as the simultaneous assignment

```
sim_for $i := 0$ to $n-1$ and $j := 0$ to $n-1$ do
        $i_\blacklozenge\cdot X\cdot\blacklozenge j :=$
            $(i_\blacklozenge\cdot X\cdot\blacklozenge j) + (i_\blacklozenge\cdot X\cdot\blacklozenge k)\cdot(k_\blacklozenge\cdot X\cdot\blacklozenge k)*\cdot(k_\blacklozenge\cdot X\cdot\blacklozenge j)$.
```

(Writing $i_\bullet \cdot X \cdot {}_\bullet j$ conventionally as x_{ij} this takes on the more familiar appearance:

```
sim_for $i := 0$ to $n-1$ and $j := 0$ to $n-1$ do
        $x_{ij} := x_{ij} + x_{ik} \cdot (x_{kk})* \cdot x_{kj}$.
```

But, of course, the problem of the simultaneous assignment remains.)

Exploitation of the, as yet unused, idempotency of addition and star, however, gives unlimited freedom in the order in which the matrix elements are assigned. They may be assigned sequentially as in Warshall's and Floyd's algorithms, or completely in parallel! To explain why this is so we take a closer look at the assignments at element level. The expression assigned to element x_{ik}, for instance, evaluates to

$$x_{ik} + x_{ik} \cdot x_{kk}* \cdot x_{kk}$$

$$= \quad \{\text{distributivity}\}$$

$$x_{ik} \cdot (1 + x_{kk}* \cdot x_{kk})$$

$$= \quad \{\text{folding: (1)}\}$$

$$x_{ik} \cdot x_{kk}* .$$

The expression assigned to x_{kj} by symmetry equals $x_{kk}* \cdot x_{kj}$, and the expression assigned to x_{kk}, instantiating the formula above, equals $x_{kk}{}^+$. Since, according to (6) and (5), respectively, we have $x_{kk}* \cdot x_{kk}* = x_{kk}*$ and $(x_{kk}{}^+)* = x_{kk}*$, it follows that for the assignment

$$x_{ij} := x_{ij} + x_{ik} \cdot (x_{kk})* \cdot x_{kj}$$

it is irrelevant whether x_{ik}, x_{kk}, and x_{kj} are changed *before* or *after* the assignment to x_{ij}. Hence the order of the assignments in the sim_for statement can be chosen freely.

An alternative proof of this fact, phrased completely in terms of matrices rather than matrix elements, can be found in [2]. The proof shows that the function

$$X \mapsto X + X \cdot {}_\bullet k \cdot (k_\bullet \cdot X \cdot {}_\bullet k)* \cdot k_\bullet \cdot X, \tag{23}$$

which, as we know, is equal to the function

$$X \mapsto X \cdot (\underline{k} \cdot X)*, \tag{24}$$

is a closure operator, and this is sufficient to permit the conversion of the simultaneous assignment to a parallel assignment, i.e. to a statement in which the individual assignments can be performed in any order. A proof of the latter is also included in [2].

This concludes the derivation of the Warshall–Floyd algorithm. Note that the total calculation (including the discussion of implementation freedom) takes roughly ten elementary steps which is about what it should be for such a compact algorithm.

7. The second algorithm

In this section we derive an algorithm computing the vector $b \cdot A*$, given properties S1 and S2. Since $A* = 1 + A^{+}$, a possible solution is to take the Warshall–Floyd algorithm for A^{+}, compute $A*$ and subsequently $b \cdot A*$. This would lead to an algorithm that, like the Warshall–Floyd algorithm, is cubic in the size of the matrix. We shall not do so, however, because properties S1 and S2 are the key to a solution that is more efficient than the one just proposed: as will be shown next, a consequence of properties S1 and S2 is that, for any vector x and any matrix Y, at least one element of $x \cdot Y*$ is easy to compute, which suggests that a gain in efficiency of at least one order of magnitude might be feasible. (See the end of this section for a more detailed discussion.)

7.1. The key theorem

A key insight in deriving an algorithm is that, because of properties S1 and S2, for any vector x and any matrix Y, at least one element of $x \cdot Y*$ is easy to compute. Specifically, choose k such that

$$\forall(j : j \in N : x \cdot \blacklozenge j \leqslant x \cdot \blacklozenge k). \tag{25}$$

Thus the kth element of x is the largest. (Such a choice can always be made because the ordering on elements is total.) Then we claim that

$$x \cdot Y* \cdot \blacklozenge k = x \cdot \blacklozenge k. \tag{26}$$

In other words, no computation whatsoever is required to compute this one element.

We formulate and prove a slightly more general theorem: first, we abstract from the irrelevant properties of $Y*$, using only the fact that $Y* \geqslant \mathbf{1}$, which follows from (1); second, in order to comply with conditions on k ($k \in M$, say) we formulate the theorem for vector $u \cdot \underline{M}$ rather than vector u.

Theorem 7.1. *For all $M \subseteq N$, $k \in N$, matrices Z and vectors u,*

$$k \in M \ \wedge\ \forall(j : j \in M : u \cdot \blacklozenge j \leqslant u \cdot \blacklozenge k) \ \wedge\ Z \geqslant \underline{N}$$
$$\Rightarrow\ u \cdot \underline{M} \cdot Z \cdot \underline{k} = u \cdot \underline{k}.$$

Proof. The proof is by mutual inclusion. Assume the antecedent of the implication. Then,

$$u \cdot \underline{M} \cdot Z \cdot \underline{k}$$
$$\geqslant \quad \{\text{assumption: } Z \geqslant \mathbf{1}\}$$
$$u \cdot \underline{M} \cdot \underline{k}$$
$$= \quad \{\text{assumption: } k \in M,\ (15)\}$$
$$u \cdot \underline{k}$$

and

$u \cdot \underline{M} \cdot Z \cdot \underline{k}$

$=$ {definition of $\underline{M}$}

$u \cdot \Sigma(i : i \in M : \underline{i}) \cdot Z \cdot \underline{k}$

$=$ {definition of $\underline{i}$ and distributivity}

$\Sigma(i : i \in M : u \cdot {\blacklozenge}i \cdot i{\blacklozenge} \cdot Z \cdot {\blacklozenge}k) \cdot k{\blacklozenge}$

$\leqslant$ {$i{\blacklozenge} \cdot Z \cdot {\blacklozenge}k$ is an element, hence by property S2 $i{\blacklozenge} \cdot Z \cdot {\blacklozenge}k \leqslant 1$}

$\Sigma(i : i \in M : u \cdot {\blacklozenge}i) \cdot k{\blacklozenge}$

$\leqslant$ {assumption on k}

$u \cdot {\blacklozenge}k \cdot k{\blacklozenge}$

$=$ {(7)}

$u \cdot \underline{k}$. □

Note that the penultimate step of the above calculation shows how the condition on k could have been "invented": it strongly suggests itself, given the one inclusion, $u \cdot \underline{M} \cdot Z \cdot \underline{k} \geqslant u \cdot \underline{k}$, and the wish to have an equality.

7.2. A skeleton algorithm

The algorithm we develop is based on an iterative process in which at each iteration Theorem 7.1 is used to "process' one node. At some intermediate stage of the computation some set L of nodes has been dealt with. These heuristics are captured formally by postulating as loop invariant

$$w = b \cdot A* \cdot \underline{L} \ \wedge \ L \subseteq N \tag{27}$$

and as postcondition

$$w = b \cdot A*, \tag{28}$$

where w is a vector. By property (10), the invariant is established by the assignment

$$L, w := \phi, \mathbf{0}.$$

For the guard of the repetition we choose for the moment the simplest possible condition, namely $L \neq N$. Since N is a finite set, progress towards $L = N$, and thus towards termination, is achieved by choosing statement

$$L := L \cup \{k\}, \quad \text{for } k \notin L$$

for the loop body. In order to maintain the loop invariant, the assignment to L is accompanied by the assignment $w := b \cdot A* \cdot (\underline{L} + \underline{k})$, i.e. by

$$w := w + b \cdot A* \cdot \underline{k}. \tag{29}$$

Thus we have arrived at the following skeleton algorithm

```
$L, w := \phi, \mathbf{0}$
; do $L \neq N \rightarrow k :\in N - L$
            ; $w := w + b \cdot A* \cdot \underline{k}$
            ; $L := L \cup \{k\}$
od $\{w = b \cdot A*\}$.
```

Our next concern is to rewrite $b \cdot A* \cdot \underline{k}$ in order to get rid of the star operator applied to a matrix. Our tactic is to use star decomposition so as to get into a position in which the Key Theorem can be invoked.

7.3. *Using star decomposition and Key Theorem*

For convenience we introduce additional invariant

$$M = N - L; \tag{30}$$

in other words, M is the complement of L in N. Thus we have, by invariant (27) and the fact that $\underline{M} + \underline{L} = \mathbf{1}$,

$$b \cdot A* = b \cdot A* \cdot \underline{M} + w. \tag{31}$$

We now calculate, bearing in mind that $b \cdot A* \cdot \underline{k} = b \cdot A* \cdot \underline{M} \cdot \underline{k}$, for $k \in M$,

$$
\begin{aligned}
& b \cdot A* \cdot \underline{M} \\
= \; & \{\text{star decomposition using } \bullet \; A = V + W \quad \text{for } V = \underline{L} \cdot A \text{ and } W = \underline{M} \cdot A\} \\
& b \cdot (W* \cdot V)* \cdot W* \cdot \underline{M} \\
= \; & \{\text{unfolding } (W* \cdot V)* : (1)\} \\
& b \cdot (\mathbf{1} + (W* \cdot V)* \cdot W* \cdot V) \cdot W* \cdot \underline{M} \\
= \; & \{\text{star decomposition}\} \\
& b \cdot (\mathbf{1} + (V + W)* \cdot V) \cdot W* \cdot \underline{M} \\
= \; & \{\text{distribution, definition of } V \text{ and } W\} \\
& (b + b \cdot A* \cdot \underline{L} \cdot A) \cdot (\underline{M} \cdot A)* \cdot \underline{M} \\
= \; & \{\text{invariant and leap-frog}\} \\
& (b + w \cdot A) \cdot \underline{M} \cdot (A \cdot \underline{M})* \\
= \; & \{\bullet \text{ introduce invariant (32): } u = b + w \cdot A\} \\
& u \cdot \underline{M} \cdot (A \cdot \underline{M})*.
\end{aligned}
$$

The driving force in the first four steps of the above calculation was to expose a selector matrix $\underline{L}$ immediately after $b \cdot A*$ in order to use the loop invariant. The decomposition $A = \underline{L} \cdot A + \underline{M} \cdot A$ appears to be the only option for achieving this goal.

So now, with additional invariant

$$u = b + w \cdot A, \tag{32}$$

we are in a position to invoke the Key Theorem: since for $k \in M$ the above calculation implies

$$b \cdot A* \cdot \underline{k} = u \cdot \underline{M} \cdot (A \cdot \underline{M})* \cdot \underline{k},$$

application of the Key Theorem yields

$$k \in M \ \wedge\ \forall(j : j \in M : u \cdot \bullet j \leqslant u \cdot \bullet k) \ \Rightarrow\ b \cdot A* \cdot \underline{k} = u \cdot \underline{k}. \tag{33}$$

As for the invariance of (30) and (32), initializing statement $L, w := \phi, \mathbf{0}$ is augmented with $M, u := N, b$; assignment $L := L \cup \{k\}$ is extended with $M := M - \{k\}$; and statement $w := w + u \cdot \underline{k}$ is extended with $u := u + u \cdot \underline{k} \cdot A$, since, by (32),

$$(b + w \cdot A)(w := w + u \cdot \underline{k}) = u + u \cdot \underline{k} \cdot A.$$

Thus, we have arrived at an algorithm in which the star operator no longer occurs.

```
$L, w, M, u := \phi, \mathbf{0}, N, b$
; do $L \neq N \rightarrow k : k \in M \ \wedge\ \forall(j : j \in M : u \cdot \bullet j \leqslant u \cdot \bullet k)$
            ; $w, u := w + u \cdot \underline{k},\ u + u \cdot \underline{k} \cdot A$
            ; $L, M := L \cup \{k\},\ M - \{k\}$
od $\{w = b \cdot A*\}$
```

maintaining

$$w = b \cdot A* \cdot \underline{L} \ \wedge\ L \subseteq N,$$

$$M = N - L,$$

$$u = b + w \cdot A.$$

Note. With regard to the guard $L \neq N$, or $M \neq \phi$, the introduction of variable u enables us to strengthen it, if so desired, without affecting the postcondition: by (31) and the derived equality $b \cdot A* \cdot \underline{M} = u \cdot \underline{M} \cdot (A \cdot \underline{M})*$, we have, using the invariants

$$u \cdot \underline{M} = \mathbf{0} \ \Rightarrow\ w = b \cdot A*. \tag{34}$$

As a result, the guard can be strengthened to $u \cdot \underline{M} \neq \mathbf{0}$.

7.4. Optimizations

Before dealing with the final task of reexpressing the vector assignments in terms of elementwise operations, we first investigate possible simplifications of the algorithm developed so far. First note that guard $L \neq N$ can be replaced by the equivalent $M \neq \phi$; as a result, variable L can be removed from the program. Secondly, from the invariant (32) (and (1)) it is immediate that

$$w = b \cdot A* \;\Rightarrow\; u = b \cdot A*, \tag{35}$$

as a result of which w can be removed from the program provided the postcondition is replaced by $u = b \cdot A*$. For later use note that, similarly, using $w \leqslant b \cdot A*$ (see (27)), we can conclude from the invariant (32)

$$u \leqslant b \cdot A*. \tag{36}$$

A final optimization results from a closer investigation of the values of $u \cdot \underline{L}$ and $u \cdot \underline{k}$. More precisely we shall show that the assignment to u leaves the values of $u \cdot \underline{L}$ and $u \cdot \underline{k}$ unchanged, viz. from the invariants we shall deduce

$$(u + u \cdot \underline{k} \cdot A) \cdot \underline{k} = u \cdot \underline{k}, \tag{37}$$

$$(u + u \cdot \underline{k} \cdot A) \cdot \underline{L} = u \cdot \underline{L}. \tag{38}$$

With regard to property (37), by distributivity and the definition of $\leqslant$, it is equivalent to

$$u \cdot \underline{k} \cdot A \cdot \underline{k} \leqslant u \cdot \underline{k},$$

which is valid: using the definition of $\underline{k}$ and the fact that element $k\bullet \cdot A \cdot \bullet k \leqslant 1$ by property S2 we derive $\underline{k} \cdot A \cdot \underline{k} \leqslant \underline{k}$. For property (38) we calculate

$$(u + u \cdot \underline{k} \cdot A) \cdot \underline{L} = u \cdot \underline{L}$$

$\equiv$ {calculus}

$$u \cdot \underline{k} \cdot A \cdot \underline{L} \leqslant u \cdot \underline{L}$$

$\Leftarrow$ {$u \leqslant b \cdot A*$: (36), and $\underline{k} \leqslant 1$}

$$b \cdot A* \cdot A \cdot \underline{L} \leqslant u \cdot \underline{L}$$

$\Leftarrow$ {$b \cdot A* \cdot A \cdot \underline{L} \leqslant b \cdot A* \cdot \underline{L} = w$}

$$w \leqslant u \cdot \underline{L}$$

$\equiv$ {$\bullet$ introduce new invariant $w \leqslant u \cdot \underline{L}$}

true.

Invariant $w \leqslant u \cdot \underline{L}$ is established by assignment $L, w := \phi, \mathbf{0}$ and maintained by the body of the repetition since

$$\begin{aligned}
& (w \leqslant u \cdot \underline{L})(w, u, L := w + u \cdot \underline{k},\ u + u \cdot \underline{k} \cdot A,\ L \cup \{k\}) \\
\equiv \quad & \{\text{substitution}\} \\
& w + u \cdot \underline{k} \leqslant (u + u \cdot \underline{k} \cdot A) \cdot (\underline{L} + \underline{k}) \\
\Leftarrow \quad & \{\text{regular algebra}\} \\
& w + u \cdot \underline{k} \leqslant u \cdot \underline{L} + u \cdot \underline{k} \\
\Leftarrow \quad & \{\text{monotonicity}\} \\
& w \leqslant u \cdot \underline{L}.
\end{aligned}$$

7.5. Elementwise implementation

In this final section we implement the assignment $u := u + u \cdot \underline{k} \cdot A$ in terms of elementwise operations. Some remarks on the relationship between the algorithm presented here and conventional descriptions of Dijkstra's shortest-path algorithm and of traversal algorithms are also included.

What the assignment $u := u + u \cdot \underline{k} \cdot A$ entails at element level can be ascertained by postmultiplying by ${\scriptstyle\blacklozenge} j$ for each node j; thus we get

$$u \cdot {\scriptstyle\blacklozenge} j := (u + u \cdot \underline{k} \cdot A) \cdot {\scriptstyle\blacklozenge} j.$$

By properties (37) and (38) derived earlier, this assignment boils down to a skip for $j = k$, since ${\scriptstyle\blacklozenge} k = \underline{k} \cdot {\scriptstyle\blacklozenge} k$ and $(u + u \cdot \underline{k} \cdot A) \cdot \underline{k} = u \cdot \underline{k}$, and similarly for $j \in L$, since ${\scriptstyle\blacklozenge} j = \underline{L} \cdot {\scriptstyle\blacklozenge} j$. Thus, the element assignments can be restricted to $j \in M - \{k\}$. In terms of elements, the value assigned to $u \cdot {\scriptstyle\blacklozenge} j$ is $u \cdot {\scriptstyle\blacklozenge} j + u \cdot \underline{k} \cdot A \cdot {\scriptstyle\blacklozenge} j$, i.e.

$$(u \cdot {\scriptstyle\blacklozenge} j) + (u \cdot {\scriptstyle\blacklozenge} k) \cdot (k {\scriptstyle\blacklozenge} \cdot A \cdot {\scriptstyle\blacklozenge} j),$$

where the parentheses indicate which subexpressions are elements. Assembling the results of the previous section with the above, we have arrived at our final algorithm.

```
$M, u := N, b$
; do $M \neq \phi \rightarrow$
        $k : k \in M \ \wedge\ \forall(j : j \in M : u \cdot {\scriptstyle\blacklozenge} j \leqslant u \cdot {\scriptstyle\blacklozenge} k)$
        ; simfor $(j : j \in M - \{k\} : u \cdot {\scriptstyle\blacklozenge} j := (u \cdot {\scriptstyle\blacklozenge} j) + (u \cdot {\scriptstyle\blacklozenge} k) \cdot (k {\scriptstyle\blacklozenge} \cdot A \cdot {\scriptstyle\blacklozenge} j))$
        ; $M := M - \{k\}$
od $\{u = b \cdot A*\}$.
```

Note that since $k \notin M - \{k\}$, the simultaneous assignment can be replaced by a sequential for-statement.

We note that the set L, the complement of M, would conventionally be called the "black" nodes; see, for example, Dijkstra and Feijen's account [8] of Dijkstra's shortest-path algorithm. The latter also distinguishes "white" and "grey" nodes. The "grey" nodes are just nodes $j \in M$ for which $u \cdot {\bullet}j \neq 0$, and the "white" nodes are the remaining nodes in M. As was shown in (34), another suitable choice for the termination condition would be $u \cdot \underline{M} = \mathbf{0}$, or, in conventional terminology: "the set of grey nodes is empty"; conventionally, this is, indeed, the choice made.

Such a termination condition $u \cdot \underline{M} = \mathbf{0}$ requires the algorithm to keep track of the nodes $j \in M$ for which $u \cdot {\bullet}j \neq 0$. We will not go into the precise details of such an addition to the algorithm; suffice it to note that, in the interpretation pertaining to reachability problems (see Table 1), keeping track of the "grey" nodes in a queue leads to a breadth-first traversal algorithm; using a stack instead of a queue leads to depth-first traversal.

We finish with a few remarks on the efficiency of the above algorithm. Note, first of all, that if the choice of k in the body of the repetition takes time proportional to the size of M, the computation time of the algorithm is quadratic in the size of matrix A. The regular algebra of matrix elements may, however, be such that choosing k takes constant time only (see, for instance, the regular algebra for reachability). In this case the time complexity depends on the **for** statement in the body: note that, in total, each element of matrix A, is referred to at most once, and that the assignment to $u \cdot {\bullet}j$ amounts to skip if matrix element $(k{\bullet} \cdot A \cdot {\bullet}j)$ equals 0, in the interpretation: if there is no edge from node k to node j. Then an implementation taking time linear in the number of edges is feasible.

8. Commentary and credits

The goal of this paper has been to show how two classes of standard path algorithms can be derived by algebraic calculation. This is, of course, not the first and nor (we hope) will it be the last such derivation. (For an earlier derivation of the second algorithm see [4]; the present derivation was inspired by the second author's involvement with similar problems [11]). The algebraic basis for the calculation given here was laid in [3], and some of its details were influenced by Carré's derivation of the same algorithm [5]. A great many other authors have described and applied related algebraic systems to a variety of programming problems; Tarjan's paper [10] includes many references. The main distinguishing feature of the development presented here, however, is its reliance on calculations with matrices rather than with matrix elements.

Acknowledgements

Thanks go to Wim Feijen and Lambert Meertens for their critical comments and suggestions for improvement regarding an earlier presentation of the Warshall–Floyd algorithm.

Preparation of this paper was expedited by the use of the MATHPAD proof editor developed by Richard Verhoeven and Olaf Weber.

References

[1] R.C. Backhouse, Closure algorithms and the star-height problem of regular languages, Ph.D. Thesis, University of London (1975).

[2] R.C. Backhouse, Calculating the Warshall/Floyd path algorithm, Computer Science Note No. 92/09, Eindhoven University of Technology, Department of Computing Science, Eindhoven, Netherlands (1992).

[3] R.C. Backhouse and B.A. Carré, Regular algebra applied to path-finding problems, *J. Inst. Math. Appl.* **15** (1975) 161–186.

[4] R.C. Backhouse and A.J.M. van Gasteren, Calculating a path algorithm, in: R.S. Bird, C.C. Morgan and J.C.P. Woodcock, eds., *Mathematics of Program Construction*, Lecture Notes in Computer Science **669** (Springer, Berlin, 1993) 32–44.

[5] B.A. Carré, *Graphs and Networks* (Oxford University Press, Oxford, 1979).

[6] J.H. Conway, *Regular Algebra and Finite Machines* (Chapman and Hall, London, 1971).

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

[8] E.W. Dijkstra and W.H.J. Feijen, *Een Methode van Programmeren* (Academic Service, the Hague, 1984); also: *A method of Programming* (Addison–Wesley, Reading, MA, 1988).

[9] D. Kozen, A completeness theorem for Kleene algebras and the algebra of regular events, in: *Proceedings 6th Annual IEEE Symposium on logic in Computer Science* (1991) 214–225.

[10] R.E. Tarjan, A unified approach to path problems, *J. ACM* **28** (1981) 577–593.

[11] J.P.H.W. van den Eijnde, Conservative fixpoint functions on a graph, in R.S. Bird, C.C. Morgan and J.C.P. Woodcock, eds., *Mathematics of Program Construction*, Lecture Notes in Computer Science **669** (Springer, Berlin, 1993) 80–100.