Distance Routing on Series Parallel Networks *

Paola Flocchini[†] Flaminia L. Luccio[‡]

Abstract

In this paper we consider the problem of routing messages on Series Parallel Graphs (SPGs), and we introduce a new technique called Distance Routing. This technique is based on the idea of encoding in the label of each node x some information about a shortest path from the source of the SPG to x, and from x to the terminal node of the SPG. We first compare shortest path Distance Routing and 1-Interval Routing Schemes on directed SPGs. We then show that Distance Routing can be used to route on bidirectional SPGs, where no general shortest path 1-Interval Routing Scheme can be applied. We also show the relevance of the study of the time complexity in the choice of a Compact Routing method.

1 Introduction

In a non anonymous network (i.e., in a network where to each node it is associated a different identity) routing can be easily accomplished if each node has available a routing table. This table has $n-1$ entries (where n is the number of nodes), and shows the edge through which a message has to be forwarded in order to reach each destination on a shortest path. Clearly, this technique is often not applicable, since each node has to store too much information. An interesting idea is to try to reduce as much as possible the size of the routing table, for example by grouping the nodes for which the same edge has to be traversed, i.e., by implicitly coding some information. The *Compact Routing techniques* either try to minimize the space to store the routing information and search for shortest paths, or try to minimize the trade-off between the length of a path and the space required. Time has also to be considered.

*Research partially supported by an international CNR Scholarship and a NATO Advanced Research Fellowship. A preliminary version of this paper will appear on the Proceedings of the *16th International Conference on Distributed Computing Systems.*

[†]School of Computer Science, Carleton University, K1S5B6 Ottawa, Canada. Email: flocchin@scs.carleton.ca.

[‡]Università di Milano, Dipartimento di Scienze dell'Informazione, via Comelico 39, 20135 Milano, Italy. Email: luccio@dsi.unimi.it

We now present some Compact Routing techniques: the Prefix , Interval, and Boolean Routing Schemes.

In the *Prefix Routing Schemes* (PRS), each node is labeled with a unique address consisting of a string of symbols. When a node receives a message that needs to be routed, it checks for an edge labeled with a prefix of the destination address. If the routing technique is correctly defined, at least one edge with such a property exists. The edge whose label has the longest prefix is chosen, the message is routed through this edge [1] and the prefix is cut from the destination address.

Interval Routing Schemes (IRS) have been widely studied in literature (see [1, 6, 4, 7, 11, 16, 13]). They are based on the definition of Interval Labeling Schemes (ILS). A $k - ILS$ assigns at most k intervals to each edge of the graph. Each node is labeled with an integer, and each interval on an edge contains the subset of nodes that are on a shortest path through this edge.

In *Boolean Routing Schemes* (BRS) a particular boolean function is assigned to each edge. The destination address of a message is given at run-time as input to the function. The output of the function is true if and only if the corresponding edge is on a shortest path from the source to the destination (see [3, 12]).

In this paper we consider Series Parallel Graphs (SPGs), a particular topology widely studied in literature for different kind of problems (see [2, 9, 8, 10, 15]). We show how to apply Compact Routing techniques to SPGs, and we introduce a new technique, the *Distance Routing* (DR), specifically defined for SPGs and can be extended, with slight modifications, to other topologies [5].

This technique is similar to Boolean Routing, and Prefix Routing, however, the PRS with their classical definition cannot be applied to SPGs, and the BRS would require a high complexity cost because of the asymmetry of the topology.

DR is based on the idea of coding in the label of a node x, the information about the shortest path from a source s to x and from x to a terminal node t of the graph. If node x wants to send a message to a node y, it computes (run-time) a particular function (as in BRS), and it encodes in the message the path to be followed. Every other node that receives the message, forwards it through the edge previously chosen, and eventually cuts off part of the address stored in the message (as in PRS). Notice that DR on SPGs is a very powerful technique which might be applied to High Speed Networks even in the case of topologies that change dynamically over the time [5].

We first consider directed SPGs ($DSPG$s) and we show how to apply shortest path DR. We then extend some results to bidirectional SPGs ($BSPG$s). We study the complexity of the algorithms, and we show the relationship between the shortest path DR and the shortest path 1-IRS on $DSPG$s. We also prove that no general shortest path 1-IRS exists in the case of $BSPG$s. Referring to [12] we study the time complexity, and we show how it can be a discriminating point on the choice of a Compact Routing method.

2 Definitions and Basic Properties

A distributed system of n processors can be viewed as a graph $G = (V, E)$ of n nodes, where nodes correspond to entities and edges correspond to communication links. We assign a unique label $\beta(x)$ to a node x, and a unique label $\lambda(x, z)$ to an edge (x, z). We call *out-degree* (*in-degree* respectively) of a node x the number of edges out-going from (in-going into) x. If the graph has bidirectional edges, the out-degree and in-degree coincide. We denote with Δ the diameter of the graph G, i.e., the maximal distance between two distinct nodes. We denote with $p_{x \to y}$ a path from x to y, and with $P_{x \to y}$ one of the possible shortest paths from x to y.
Given a graph $G = (V, E)$, we define the Series and Parallel Composition (see Figure 1) as follows:

Definition 1 Series and Parallel Composition
(i) A Series composition consists of replacing an edge $e = (u, v) \in E$ with two edges $e_1 = (u, z)$ and $e_2 = (z, v)$, and with z a new node.
(ii) A Parallel composition consists of replacing an edge $e = (u, v) \in E$ by three edges $e_1 = (u, v)$, $e_2 = (u, z)$, $e_3 = (z, v)$, and with z a new node.

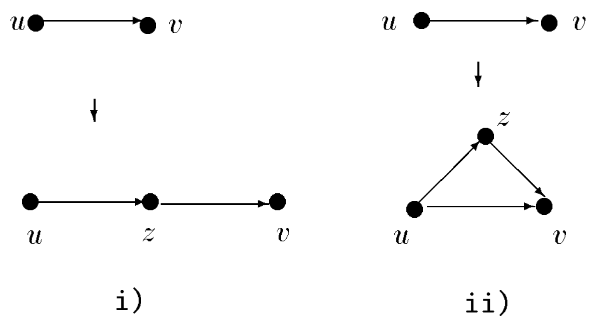

Figure 1: Series and Parallel Composition

Let s and t be two distinct nodes in G. G is called a *two terminal series parallel graph* (*TTSPG*) with respect to s and t if it can be obtained by a sequence of series and parallel compositions from a single edge (s, t).

Definition 2 *G is called* series parallel graph *(SPG) if there exist two vertices s and t so that G is a $TTSPG$ with respect to s and t.*

Definition 3 *A* Directed Series Parallel Graph *(DSPG), is a SPG whose edges are oriented from left to right. We assume that s is the source, and t the terminal node. We also add the edge (t,s), so that it is possible to cycle back.*
A Bidirectional Series Parallel Graph *(BSPG) is a SPG whose edges are bidirectional.*

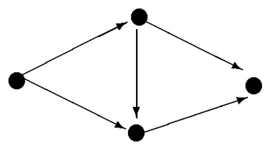

Figure 2: The forbidden graph for directed *TTSPG*s

Theorem 1 *Let G be a directed acyclic graph with one source and one sink. G is a directed TTSPG iff G does not contain a subgraph homeomorphic to the one in Figure 2.*

In a *SPG*, a node is an *opening node* iff it has out-degree greater than one; a node is a *closing node* iff it has in-degree greater than one. An edge (u,v) is an *opening edge* iff u is an opening node; it is a *closing edge* iff u is a closing node; it is a *serial edge* otherwise.

In a *DSPG* we say that node x is *before* y (or y is *after* x), if y can be reached from x without using the edge (t,s), in this case we write $x << y$. x and y are *separated* ($x <> y$) if y can be reached from x only using the edge (t,s).

For a clearer presentation of the algorithms in the next sections, we define $[[a_1], [a_2], \ldots, [a_n]]$ as a list containing the n elements $a_1, a_2, \ldots, a_n$.

3 Routing on *DSPG*s

In this section we present two different routing techniques on *DSPG*s. We first introduce a new technique called Distance Routing, we then present a 1-Interval Routing Scheme (1-*IRS*). With both the techniques we show how to perform shortest path routing, we analyze the complexity of the algorithms, and we study the relationship between the two techniques.

3.1 Distance Routing on *DSPG*s

In order to define the routing technique on *DSPG*s, we first observe few nice properties of these graphs.

Using the definition of *DSPG* we trivially obtain that to each opening (closing) edge corresponds at least one closing (opening) edge. Notice that it is possible to have that an opening (closing) edge corresponds to more than one closing (opening) edge. For example, in Figure 3 the opening edge (A, B) corresponds only to the closing edge (B', A'), while the opening edge (A, C) corresponds to the two closing edges (C', A') and (D, E). Observe that an edge could be simultaneously an opening and a closing node.

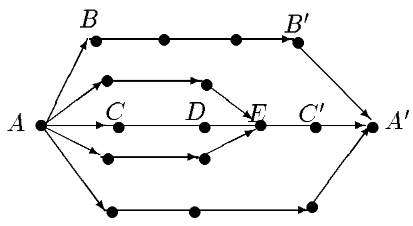

Figure 3: Opening and closing edges.

It is well known that:

Property 1 *A DSPG is a partial order of dimension two.*

Proof. It directly follows from the fact that a *DSPG* is a lattice and a lattice is a partial order of dimension 2 [14]. □

From the property it follows that it is possible to associate a pair of labels (x_0, x_1) to each node x in such a way that $x << y$ iff $x_0 \leq y_0$ and $x_1 \leq y_1$.

We now define the Distance Routing technique by giving the edge labeling, the node labeling and the routing function.

- **Edge labeling**

  Given a node x, we assign distinct labels to all the closing edges of x, and to all the opening edges of x respectively.

  A **serial** edge (u, v) is labeled with $\lambda(u, v) = R$ (right).
  The labeling of an **opening** edge (u, v) is given by a pair (l, m_1^+), where: $1 \leq l \leq$

d_1, d_1 is the number of edges out-going from u, m_1 is the number of closing edges which correspond to the opening edge (u, v), and $+$ is an additional bit indicating the fact that (u, v) is an opening edge.
The labeling of a **closing** edge (u, v) is given by a pair (l, m_2^-), where: $1 \leq l \leq d_2$, d_2 is the number of edges in-going into v, m_2 is the number of opening edges which correspond to the closing edge (u, v), and $-$ is an additional bit indicating the fact that (u, v) is a closing edge.
If an edge (u, v) is **opening and closing**, distinct labels are assigned to all closing edges ending in v and all opening edges starting from u. Moreover the label of (u, v) is given by a triple (l, m_1^-, m_2^+), where: $1 \leq l \leq d$, $d = d_1 + d_2$, d is the number of edges in-going into v, plus the number of edges out-going from u, m_1 is the number of closing edges which correspond to (u, v), m_2 is the number of opening edges which correspond to (u, v), $+$ and $-$ indicates that (u, v) is both an opening and a closing edge (see Figure 4, where only some labels are shown).

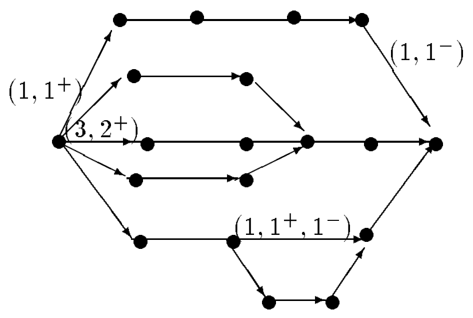

Figure 4: Edge Labeling.

- **Node labeling**

  First of all we construct the partial order of the $DSPG$ using the canonical assignment of pairs [14] (see Figure 5). This is possible by Property 1. A label $\beta(x)$ is assigned to a node x; $\beta(x)$ contains two informations: the shortest path $P^x_{s \to t} = P_{s \to x} \cup P_{x \to t}$, and the canonical coordinates (x_0, x_1) of the partial order. Note that for the correctness of the following theorems it is important to choose an order among all the different shortest paths from s to x, and from x to t, so that all the nodes after (respectively before) x contain $P_{s \to x}$ ($P_{x \to t}$) when possible. This choice is made at the very beginning starting from node s, and we call it the *canonical assignment* of the shortest paths.

Each path is encoded in a list containing an edge name, and the number of times adjacent edges with the same name have to be crossed. I.e., if the sequence of edges to traverse is $(1,1^+), R, R, R, (2,1^-), R, R, R, R,$ then $\beta(x) = [[1,(1,1^+)],[3,R],[1,(2,1^-)],[4,R]]$.

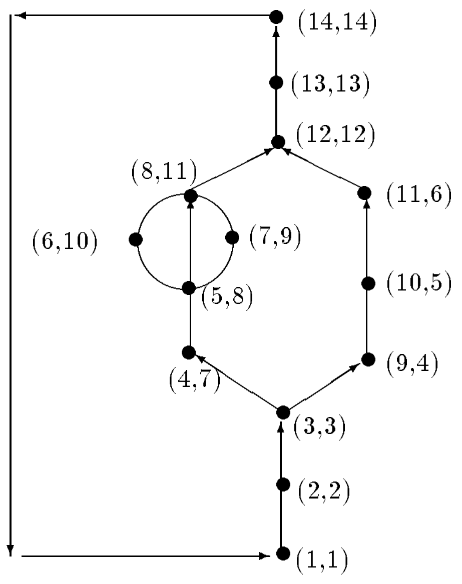

Figure 5: Partial order of a *DSPG*.

- **Routing function**

  Let us assume that a node x wants to send a message to another node y along a shortest path. Node x has to compute a particular function (below defined), and encodes the path to be followed in the message. All the other nodes receiving the message, forward it on the first edge of the path. If an edge with the same name has to be crossed more than once, the node decreases the multiplicity by one, otherwise it cuts off part of the message.

  The function which must be computed by x is the following:

  A) If $x >> y$, or $x <> y$, y cannot be reached from x without passing through (t,s) and thus we have $P_{x\to y} = P_{x\to t} \bigcup \{(t,s)\} \bigcup P_{s\to y}$.

  B) If $x << y$, y can be reached from x without passing through (t,s). We define I to be an *intersection string*, i.e., a string shared both by $P^x_{s\to t}$ and by $P^y_{s \to t}$, and placed between x and y (see the proof below of the uniqueness of this string in $P_{x \to y}$). Chosen x_g and y_f to be two points inside I, with $x_g \leq y_f$, (the choice of them is explained below), we have that $P_{x \to y} = P_{x \to x_g} \bigcup P_{x_g \to y_f} \bigcup P_{y_f \to y}$.

Let $P^x_{s' \to t'} = [x_0, \ldots, x_h]$ and $P^y_{s' \to t'} = [y_0, \ldots, y_k]$ be the resulting paths after removing the common prefix and suffix of $P^x_{s \to t}$ and $P^y_{s \to t}$.

Consider the following lemmas (Only for the case $s' < x << y < t'$, and $P_{s \to x} \not\subset P_{s \to y}$ and $P_{y \to t} \not\subset P_{x \to t}$.):

Lemma 1 *The paths $P^x_{s' \to t'}$ and $P^y_{s' \to t'}$ share a unique intersection string.*

Proof. Let us assume by contradiction that this is not true. This implies that there are only two possible situations: there is no intersection string between x and y, but this is not possible by the definition of $x << y$, and by the fact that the *SPG* is directed from s to t, or there are at least two intersection strings. Let us denote these two strings with $I_1 = [x_i \ldots x_{i+m}] = [y_j \ldots y_{j+m}]$, with $i, j > 0$, $m \geq 0$, $i + m \leq h$, $j + m \leq k$; $I_2 = [x_{i+m+a} \ldots x_{i+m+a+d}] = [y_{j+m+c} \ldots y_{j+m+c+d}]$, with $a, c > 0$, $d \geq 0$, $i + m + a + d < h$, $j + m + c + d < k$. Since $x << y$ and $i, j > 0$, $i + m + a + d < h$, and $j + m + c + d < k$. We can trivially deduce that x lies in one of the at least two possible paths from s' to x_i, and y in one of the at least two possible paths from $x_{i+m+a+d}$ to t'. Therefore I_1 and I_2 both lie between x and y. Since $x_{i+m+1} \neq x_{i+m+a}$, we have that the length of $[x_{i+m+1} \ldots x_{i+m+a}]$ and of $[y_{j+m} \ldots y_{j+m+c}]$ are different, and this is not possible since we have assumed a canonical assignment of the intervals, i.e., we have a contradiction. This holds even for more than two intervals.

□

Let $N(x_i)$ be the "level " of node x_i with respect to x_0, where the level of a node is recursively computed as follows:
$N(x_0) = 0$. If $\lambda(x_i, x_{i+1}) = (l, n^+)$, then $N(x_{i+1}) = N(x_i) + n$; if $\lambda(x_i, x_{i+1}) = (l, m^-)$, then $N(x_{i+1}) = N(x_i) - m$; if $\lambda(x_i, x_{i+1}) = (l, n^+, m^-)$, then $N(x_{i+1}) = N(x_i) + n - m$; if $\lambda(x_i, x_{i+1}) = R$, then $N(x_{i+1}) = N(x_i)$. Let $\mathcal{N} = max\{N(s'), N(t')\}$.

Let y'_f be the first opening node before y such that $N(y'_f) = \mathcal{N}$, and y_f be the first node before y'_f that belongs to I, and such that $N(y_f) = \mathcal{N}$. Let also x'_g be the first opening node after x such that $N(x'_g) = \mathcal{N}$, and x_g be the first node after x'_g that belongs to I, and such that $N(x_g) = \mathcal{N}$. In the following we will prove the existance of $(x_g,\ x'_g,\ y_f,\ y'_f)$.

Lemma 2 *Such x_g, x'_g, y_f, and y'_f exist. Moreover $x_g \leq y_f$.*

Proof. First observe that $N(s') = 0$, so $\mathcal{N} = max\{0, N(t')\}$. There are three different cases.

1. $N(t') = 0$, i.e., there is no edge opening outside $\tilde{P} = P^x_{s' \to t'}$ and closing inside $\tilde{P}$, or opening inside $\tilde{P}$ and closing outside $\tilde{P}$. Observe that $x << y$. Moreover the only possible situation is the one with x contained in a path created by an edge opening in s' (and closing in a node x' such that $N(x') = N(s') = 0$), and y in a path that ends in an edge closing in t' (and opening in an edge y' such that $N(y') = N(t') = 0$). None of these paths are shared by x and y respectively since $P_{s \to x} \not\subset P_{s \to y}$ and $P_{y \to t} \not\subset P_{x \to t}$, and since we cut off the common prefix and suffix of $P^x_{s \to t}$ and $P^y_{s \to t}$. We can choose $x'_g = x'$, and $y'_f = y'$ since these nodes are both at the same level 0. If x'_g and y'_f belong to I we are done, otherwise there is at least a node where $P^x_{s \to t}$ and $P^y_{s \to t}$ intersect since $x << y$ and we can trivially choose the new x_g and y_f at level 0. Note that I must be contained into $P_{x_g \to y_f}$. Observe that $x'_g \leq y'_f$ since the $DSPG$ cannot be homeomorphic to the graph in Figure 2, and therefore we can choose $x_g \leq y_f$.

2. $N(s') = 0$ and $N(t') < 0$, i.e., there are edges opening outside $\tilde{P}$ and closing inside. This implies $N = 0$. The situation is similar to the first case and x'_g and x_g are chosen in the same way. The only difference is that we require $N(y') = N(s') = 0$.

3. $N(s') = 0$ and $N = N(t') > 0$. Choosing $N = N(t') > 0$ is the same as excluding the edges that close outside and subtracting this value from N to obtain 0. x_g and y_f are chosen similary as in the above cases.

No other case (i.e., a combination of case 2 and 3) can exist since $DSPG$ cannot be homeomorphic to the graph in Figure 2.

□

We can now prove the correctness of the routing function.

Theorem 2 *The Distance Routing algorithm correctly routes the messages along a shortest path in DSPGs.*

Proof.
Case A) is trivial. Let us consider Case B).
We have the following situations:
1) $P_{s \to x} \subset P_{s \to y}$ or $P_{y \to t} \subset P_{x \to t}$.
In this case $P_{x \to y}$ can be trivially derived by choosing two common points x_g and y_f such that are shared by both paths, and $x_g \leq y_f$.
2) $s \leq s' < x << y < t' \leq t$.
Use Lemmas 1 and 2.

It follows that $P_{x\to y} = P_{x\to x_g} \bigcup P_{x_g \to y_f} \bigcup P_{y_f \to y}$. □

Example.
Consider, for example, Figure 6. Let $P_{s\to x} = (s, A, x)$, $P_{x\to t} = (x, C, D, E, F, t)$, $P_{s\to y} = (s, B, C, D, G, y)$ $P_{y\to t} = (y, F, t)$. Removing common prefix and suffix to $P^x_{s\to t}$ and $P^y_{s\to t}$, it is easy to see that $s' = A$, $t' = F$. Then we have that $N(s') = 0$, $N(t') = 1$, thus, $\mathcal{N} = 1$. The first node before y of level 1 is clearly D, while the first node after x of level 1 is C, thus the shortest path $P_{x\to y} = (x, C, D, G, y)$.

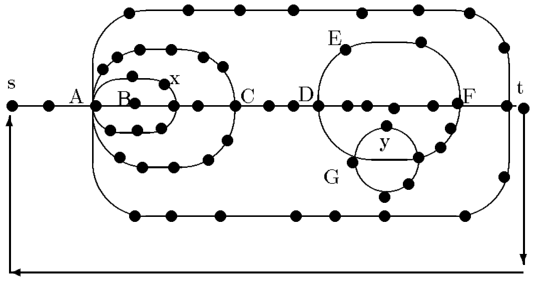

Figure 6: Example of routing.

We now study the complexity of DR, and in order to compare the different routing methods we will be considering different complexity costs, such as the number of bits used for nodes and edges labels, and the time to compute the routing function. In [12] it was observed how time can become crucial in the use of Compact Routing techniques, since we could define techniques that require a small amount of memory, but the time needed to retrieve the routing information could be high. In the following, we define the time complexity as the total number of steps necessary for all nodes on the shortest path chosen, to decide, run-time, the routing.

We will assume that arithmetical operations, comparisons, etc., require constant time when applied to integers of $O(\log n)$ bits.

The following property states that the number of nodes n of a series parallel graph lies between $O(d + \Delta)$ and $O(d^{\Delta})$.

Property 2 *There exist two positive integer constants c' and c'' such that $c'\,(d + \Delta) \leq n \leq c''\, d^{\Delta}$*

Proof. It is easy to see that $c'\,(d+\Delta) \leq n < c''\,d^{\Delta}$.
We now show that, for any d and Δ there exists at least one series parallel graph with $n = O(d^{\Delta})$ nodes. Such a graph can be contructed as follows:
If Δ is even, consider two complete trees of order d and depth $\frac{\Delta}{2}$, connect the two trees by their leaves (see, for example Figure 7 *a*)); if Δ is odd, consider two complete trees of order d and depth $\lfloor \frac{\Delta}{2} \rfloor$ and connect their leaves by new edges (see, for example Figure 7 *b*)). It is easy to show that, in both cases, we have obtained series parallel graphs where the two roots are the source and the terminal nodes. Moreover, it is possible to compute their exact number of nodes.
In fact, when Δ is odd, the total number of nodes is $2\sum_{i=0}^{\frac{\Delta}{2}-1} d^i = 2\left(\frac{d^{\frac{\Delta}{2}}-1}{d-1}+1\right) = O(d^{\Delta})$
When Δ is even, the total number of nodes is $d^{\frac{\Delta}{2}} + 2\sum_{i=0}^{\frac{\Delta}{2}-2} d^i = d^{\frac{\Delta}{2}} + 2\left(\frac{d^{\frac{\Delta}{2}-1}-1}{d-1}+1\right) = O(d^{\Delta})$. □

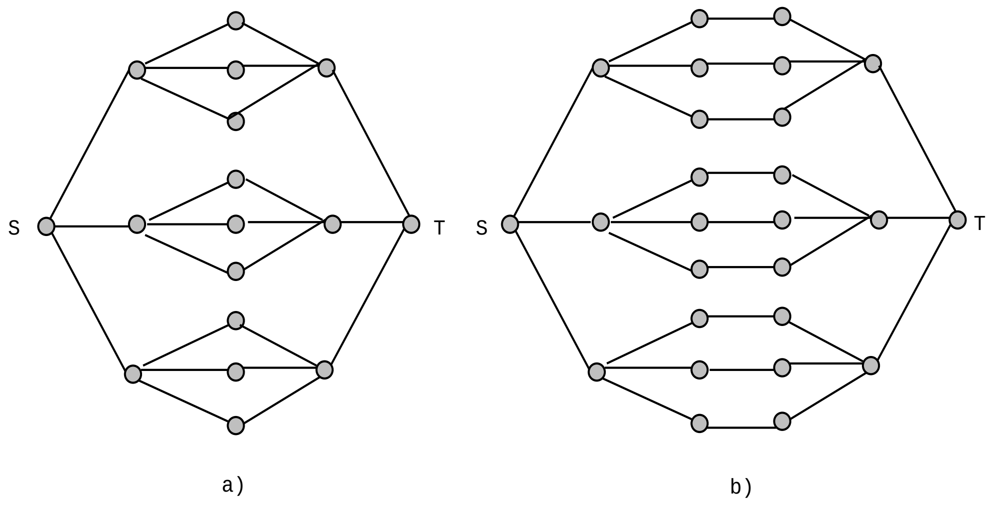

Figure 7: Series Parallel graphs with $O(d^{\Delta})$ nodes.

Theorem 3 *The Distance Routing technique requires $O(\log d)$ bits for the edge labels (if d is the maximum between the out-going and the in-going degree of all nodes), and $O(\Delta \log d)$ bits for the node labels in DSPGs.*

Proof. The label of an edge (u,v) has to be different from the labels of the other edges out-going from u (at most $d-1$), or out-going from nodes having an edge in-going into v (at most $d-1$, having at most other $d-1$ out-going edges), i.e., there are at most d^2 different edges whose labels have to be assigned. This implies that $2\log d$ bits are sufficient to distinguish all these edges. Some edges also contain at most other two integers (of at most $\log d$ bits) that indicate the number of corresponding opening or closing edges, and other two extra bits, i.e., totally at most other $2\log d + 2$ bits. In total at most $4 \log d + 2$ bits are sufficient to represent each edge label.

The node label of a node x contains $P^x_{s\rightarrow t}$, and the two labels of the partial order. In the worst case an upper bound on the length of the path is given by the diameter Δ of the graph (i.e., the maximum distance between two nodes), whereas, the name of the edge to be taken is bounded by $4\log d + 2$ bits, as we have stated above. Therefore $O(\Delta \log d)$ bits are sufficient. Then every pair of labels of the partial order is of at most $2\lceil\log n\rceil$ bits. To bound this quantity we use property 2. This trivially implies that the $O(\log n)$ bits required for the pair labels are included in the $O(\Delta \log d)$ bits of the path.
Finally at run-time in the case $x << y$, the space required to compute the levels of the intersection node and therefore the new path, is included in the above $O(\Delta \log d)$ bits, since only a constant number of levels (represented with at most $2\log n$ bits) and pointers to the node labels (i.e., at most $\log(\Delta \log d)$ bits) has to be stored to distinguish for instance node s', t', x_g, and y_f. Also during the computation the addition or subtraction required to compute the level can be sequentially done using at most $2\log n$ bits, completing the proof of Theorem 3. $\square$

Theorem 4 *The time complexity for Distance Routing is* $O(\Delta)$ *steps.*

Proof. In the first step the sender has to compare the pair of the partial order, i.e., two numbers of $O(\log n)$ bits. This can be done in constant time. If $x >> y$ or $x <> y$ the path is already known. If $x << y$ then other steps are required. First a path of length at most Δ has to be compared, i.e., Δ steps are sufficient. If one of the path is included in the other we are done, otherwise part of the path is cut off, and the levels of nodes s', t', x_g, and y_f have to be computed. This can be done in at most $O(\Delta)$ steps, adding or subtracting integers of at most $2\log n$ bits. At this point the new path to be followed is trivially given. All the other nodes (at most Δ) in the shortest path to the destination need to check only the first element of the list contained in the message, in order to choose the right out-going edge. Then they have to decrease a counter, or cut off a piece of the message. All this requires constant time on each node. $\square$

3.2 1-Interval Routing on *DSPG*s

The 1-*IRS* is based on the definition of a 1-Interval Labeling Scheme (1-*ILS*) given as follows:

- **Node labeling**

  We assign a different label to each node with this rule: we first choose $\beta(s) = 1$, and then give increasing integer values to the nodes that are in series. As soon as we reach an opening node, we choose a branch and we assign increasing labels on each branch, up to the closing node (closing with respect to the biggest branch we opened) excluded. The last labels are given to the nodes on the branch containing a shortest path to the above defined closing node. The labeling is applied recursively on the branches contained inside. In this way every message whose destination is outside the biggest branch, will follow a shortest path, and the same will happen inside the smaller branches (see Figure 8).

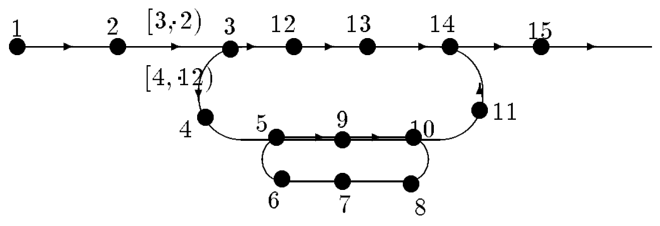

Figure 8: 1-*IRS* on a *DSPG*

- **Edge labeling** If a node u has out-degree one, we associate to a corresponding edge (u, v) the interval $[v, u-1]$, i.e., all the nodes are on a shortest path through (u, v). If u has out-degree greater than one, we associate a different interval to each out-going edge that opens a new branch. Each interval contains the nodes up to the closing node of the related branch excluded. If the branch contains the shortest path to the closing node, the interval contains all the node in the branch and all the others up to $u - 1$ (see Figure 8).

- **Routing function**

  When a node x wants to send a message to a node y, it searches for the name y in the intervals associated to the edges out-going from x, and it forwards the message through that edge. If x receives a message it first checks whether the destination is x itself, or otherwise it forwards it.

Theorem 5 *The* $1-IRS$ *correctly routes the messages along a shortest path in DSPGs.*

Proof. If a node u is in series with another node v, the interval associated to (u, v) is $[v, u-1]$ (or $[2, n]$ if $u = 1$), since the graph G is directed, and all the shortest path to any possible destination have to cross the edge (u, v). By definition of the edge labeling, if u is an opening node, there are two different kinds of intervals: if (u, v) is not on a shortest path from u to the furthest closing node z of a branch we opened, the interval contains only $[v, v + s - 1]$, where s is the number of nodes up to z excluded; if (u, v) is on a shortest path, the interval is $[v, u - 1]$. If a message has to be routed to a node inside an opened branch, the only possible path has to traverse the branch itself. If, on the other hand, the destination node is passed the furthest closing branch, the edge chosen is the one the shortest path. This is possible since the interval on this edge contains all nodes but u. This edge has trivially to be chosen even to reach all the nodes from s to u. The above rules are applied in the same way inside each node of the different branches. $\square$

Theorem 6 *The 1-IRS requires $O(\log n)$ bits for each edge label, and $O(\log n)$ bits for each node label in $DSPG$s.*

Proof. Each node has a different label, chosen among the first n integers. Each edge has a different interval containing the labels of two nodes. $\square$

Theorem 7 *The worst-case time complexity for the 1-IRS is $O(\Delta \log d)$ steps.*

Proof. For the time complexity, each node on the shortest path can use simple variation of the binary search to find the destination node among at most d different intervals. In the worst case the shortest path has length $O(\Delta)$. $\square$

3.3 Complexity comparison between *DR* and 1-*IRS*

In this section we compare the complexities of DR and 1-IRS on $DSPG$s.

The following table compares the costs of the two techniques. Notice that from Theorem 2 we get that $O(\log n) \leq \Delta \log d$. The cost is defined in terms of space (in terms of bits) required in worst case on each edge and node, and of time (i.e., the total number of steps required).
We have that:

Theorem 8
(i) The edge cost of DR is never worse then the one of 1-IRS.
(ii) The node cost of $1-IRS$ *is never worse then the one of DR; they are equal when $n = O(d^{\Delta})$.*
(iii) The time complexity of DR is always *better then the one of 1-IRS.*

costs	*DR*	1-*IRS*
Edge	$O(\log d)$	$O(\log n)$
Node	$O(\Delta \log d)$	$O(\log n)$
Time	$O(\Delta)$	$O(\Delta \log d)$

Table 1: Comparison between *DR* and 1-*IRS* in *DSPG*s.

Proof.
Cases (i) and (iii) follow directly from Table 1.
Case (ii). We can trivially see that the only situation for which *DR* and 1-*IRS* are equivalent is when $n = O(d^{\Delta})$. □

From Table 1 we can compare the complexities of the two techniques in particular cases. For example if we consider the graph of Figure 7, we get that the cost on each edge and the time are less for *DR*, and the cost on each node is the same as *IRS*.

4 Routing on *BSPG*s

In this section we consider Bidirectional Series Parallel Graphs (*BSPG*s), and we show how to extend the definition of *DR* to this case. We also consider 1-*IRS* and show that, in the case of *BSPG*, there are graphs for which no shortest path 1-*IRS* exists.

4.1 Distance Routing on *BSPG*s

We define the Distance Routing technique by giving the edge labeling, the node labeling and the routing function.

- **Edge labeling**

  We first label the edges by considering the induced directed subgraph of the *BSPG* that has the edges oriented left to right. An edge (u, v) is labeled with $\lambda(u, v) = R$ (right) iff (u, v) is the only edge out-going from u and in-going into v. Otherwise a distinct label is chosen among at most $2 \log d$ different ones (where d is the maximum between the out-going and in-going degree of all nodes). Given a bidirectional edge (u, v), we give the labels to the edges oriented in the opposite direction as follows: if $\lambda(u, v) = R$, then $\lambda(v, u) = L$, otherwise if $\lambda(u, v) = a_i$, then $\lambda(v, u) = \bar{a}_i$.

- **Node labeling**

  We assign to a node x a label $\beta(x)$ containing both $P_{s \to x}$, and $P_{x \to t}$. Notice that since the edges are bidirectional the path can follow different directions, i.e., messages can move left to right or right to left in respect to s and t.

- **Routing function**

  If x wants to route a message to y, it computes a particular function defined below, and it encodes the path to be followed in the message. All the other nodes that receive the message forward it in the same way as in the directed case.

  Notice that given a path $p_{x \to y}$, the reverse path $p_{y \to x}$ can be easily computed, since every R is changed into a L (and vice versa), and every a_i into an $\bar{a}_i$.

  There are five different possible cases. At every step if a case does not hold, we try the next one.

  1. If $P_{s \to x} \subseteq P_{s \to y}$ then the path $P_{x \to y}$ is computed by cutting off the common prefix of $P_{s \to x}$ and $P_{s \to y}$.
  2. If $P_{y \to t} \subseteq P_{x \to t}$, then $P_{x \to y}$ is computed by cutting off the common suffix of $P_{y \to t}$ and $P_{x \to t}$.
  3. If $P_{s \to y} \subseteq P_{s \to x}$, then take $P_{y \to x}$ and reverse it.
  4. If $P_{x \to t} \subseteq P_{y \to t}$, then take $P_{y \to x}$ and reverse it.
  5. If conditions of point 1, 2, 3, and 4 do not hold, then consider again $P_{s' \to x}$ and $P_{s' \to y}$ ($P_{x \to t'}$ and $P_{y \to t'}$) to be obtained from $P_{s \to x}$ and $P_{s \to y}$ by removing the common prefix (suffix). $p_{x \to y}$ is then chosen among the shortest of two different paths: $p_{x \to s'} \bigcup p_{s' \to y}$, and $p_{x \to t'} \bigcup p_{t' \to y}$

We now prove the correctness of the algorithm:

Theorem 9 *The Distance Routing algorithm correctly routes the messages in BSPGs.*

Proof. For the first four cases it trivially uses the definition of a shortest path. For case five it is easy to see that the path found is one of the possible paths between x and y. $\Box$

With an argument similar to the directed case we can prove the following:

Theorem 10 *The DR technique requires $O(\log d)$ bits for the edge labels and $O(\Delta \log d)$ bits for the node labels in BSPGs.*

Theorem 11 *The time complexity of DR on BSPGs is $O(\Delta)$ steps.*

4.2 Interval Routing on *BSPG*s

In this section we study the *IRS* on *BSPG*s and we show that there are *BSPG*s for which no 1-*IRS* exists. Let us consider the class of *regular globes.*

A regular globe graph is a particular *SPG*; it is a *SPG* consisting of an opening node with out-degree s, a closing node with in-degree s and in which every branch has the same number of serial nodes (see Figure 9).

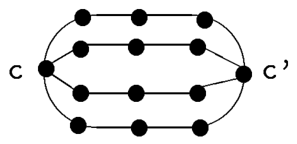

Figure 9: A regular globe graph

More formally,

Definition 4 *[11]*
A regular globe graph G^n_s consists of s line segments $L_1, \ldots, L_s$ each containing n nodes $l_{i1}, \ldots, l_{in}$, $i = 1, \ldots, s$. The line segments are joined by two isolated nodes c, c'. Formally:

$$G^n_s = (V, E),$$

$$V = \{l_{ij},\ \imath = 1, \ldots, s,\ j = 1, \ldots, n\} \bigcup \{c, c'\},$$

$$E = \{\{l_{i,j}, l_{i,j+1}\}\, i = 1, \ldots, s, j = 1, \ldots, n-1\}$$

$$\bigcup \{\{c, l_{i,1}\}, \{c', l_{i,n}\},\ i = 1, \ldots, s\}.$$

We now show that there exist graphs that require $k > 1$ intervals, which an easy *DR* can be defined. We will need the following theorems:

Theorem 12 *[11]*
If s is odd, and $n \geq s^2$, then the number of intervals required by a $k - LIRS$ is $k = \lfloor \frac{s}{2} \rfloor + 1$.

From Theorem 12, we can easily derive the following:

Theorem 13 *There exist regular globe graphs that require a number of intervals that is a function of the number of nodes in the graph.*

Theorem 13 computes the number of intervals required by a *LIRS*. It is trivial to extend this result to *IRS*. The existence of a *DR* algorithm for these graphs follows from the fact that the globe graph is a *BSPG*. Thus, we have:

Theorem 14 *There exists an infinite family of graphs for which no 1-IRS exists, but a DR is defined.*

Acknowledgments

The authors would like to thank Danny Krizanc, Umberto Nanni, Nicola Santoro, and Richard Tan for helpful discussions.

References

[1] E.M. Bakker, J. van Leeuwen, and R.B. Tan. Linear interval routing. *Algorithms Review*, 2:45–61, 1991.

[2] R.J. Duffin. Topology of series-parallel networks. *J. Math. Anal. Appl.*, 10:303–318, 1965.

[3] M. Flammini, G. Gambosi, and S. Salomone. Boolean routing. In *Proc. of the 7th Int. Workshop on Distributed Algorithms*, Springer Verlag LNCS 725, pages 219–233, 1994.

[4] M. Flammini, G. Gambosi, and S. Salomone. Interval labeling schemes for chordal rings. In *Proc. of 1st Colloqium on Structural Information and Communication Complexity*, pages 111–124, 1994.

[5] P. Flocchini, D. Krizanc, and F.L. Luccio. Dynamic distance routing on series parallel high-speed networks. in preparation.

[6] P. Fraigniaud and C. Gavoille. A characterization of networks supporting linear interval routing. In *Proc. of ACM Conference on Principles of Distributed Computing*, pages 216–224, 1994.

[7] G.N. Frederikson and R. Janardan. Designing networks with compact routing tables. *Algorithmica*, 3:171–190, 1988.

[8] X. He. Efficient algorithms for optimization and selection in series-parallel graphs. *SIAM Journal of Discrete Mathematics*, 7:379–389, 1986.

[9] X. He. Efficient parallel algorithms for series parallel graphs. *Journal of Algorithms*, 12:409–430, 1991.

[10] E. Korach and A. Tal. General vertex disjoint paths in series-parallel graphs. *Discrete Applied Mathematics*, 41:147–164, 1993.

[11] E. Kranakis, D. Krizanc, and S.S.Ravi. On multi-label linear interval routing schemes. In *Proc. of the 19th Int. Workshop on Graph-Theoretic Concepts in Computer Science*, Springer Verlag LNCS 790, pages 338–349, 1993.

[12] D. Krizanc and F.L. Luccio. Boolean routing on chordal rings. In *Proc. of the 2nd Colloqium on Structural Information and Communication Complexity*, pages 89–100, 1995.

[13] N. Santoro and R. Khatib. Labeling and implicit routing in networks. *The Computer Journal*, 28:5–8, 1985.

[14] K.A. Vaker, P.C. Fishburn, and F.S. Roberts. Partial orders of dimention 2. *Networks*, 2:11–28, 1971.

[15] J. Valdes, R.E. Tarjan, and E.L. Lawler. The recognition of series-parallel digraphs. *SIAM J. Computing*, 11(2):298–313, 1982.

[16] J. van Leeuwen and R.B. Tan. Interval routing. *The Computer Journal*, 30(4):298–307, 1987.