Interval Routing

J. VAN LEEUWEN AND R. B. TAN†
Department of Computer Science, University of Utrecht, P.O. Box 80.012, 3508 TA Utrecht, the Netherlands

An interval routing scheme is a general method of routing messages in a distributive network using compact routing tables. In this paper, concepts related to optimal interval routing schemes are introduced and explored. Several problems concerning the insertion of nodes and joining of separate networks by a new link to form larger ones are considered. Various applications to distributed computing are given. In particular, leader-finding and generation of spanning trees in arbitrary networks are shown to require at most $O(N+E)$ messages when a suitable interval routing scheme is available.

Received November 1985

1. INTRODUCTION

In a computer network, a routing method is required in order that the nodes can communicate messages to each other. Normally this is provided by a routing table of size $O(N)$ at each node, where N is the number of nodes in the network. The table shows the link(s) to be traversed for each destination node. Santoro & Khatib[3] have shown that routing can be achieved without the need for any routing tables at all, provided the nodes of the network are suitably labelled and the routing is restricted to a spanning tree. The technique has subsequently been extended by van Leeuwen and Tan[5] to obtain a method that utilizes every link of the network but requires tables of size $O(d)$, where d is the degree of a node.

In this paper we consider the intricate question of generating optimum or near-optimum routing schemes of this kind, and the impact of utilizing any such scheme on the message complexity of common distributed network problems.

Basically the idea presented in Ref. 5 is to label the nodes and the edges of the graph (network) by labels from a linearly ordered set, say $\{i_0, i_1, \ldots, i_{N-1}\}$, in a suitable manner. The labels i_0 to i_{N-1} are cyclically ordered.

An interval labelling scheme (ILS) for a connected N-node network G is a scheme for labelling the nodes and links such that (i) all nodes get different labels and (ii) at every node each link receives a distinct label. The labels assigned to the links at node i are stored in a table at node i. To send a message m from node i to node j we use a 'recursive' routine *SEND* (i, j, m) where at each intermediate node k, starting with node i, node k will look up its table and find link α_s such that the interval $[\alpha_s, \alpha_{s+1})$ contains j. Node k then sends message m down the link labelled α_s, and the whole process is repeated until message m does arrive at node j, if ever. The routine can be described simply as follows:

procedure *SEND* (i, j, m);
begin
  if $i = j$ **then** *process m*
    else
    begin
    find label α_s *in the labelling at node i such that* $\alpha_s \leqslant j < \alpha_{s+1}$; i: = *the neighbour of i reached over link* α_s; *SEND* (i, j, m)
    end
end.

One cannot just pick an arbitrary scheme and hope that it will route a message correctly, as the message may never reach its destination due to a cycle in the route. An ILS is valid if all messages sent from any source node arrive at their destinations. Most ILS are in fact not valid. In Ref. 5 it was shown that there is an $O(N^2)$ algorithm to determine whether an ILS is valid or not. The main result of Ref. 5 is the following.

Theorem 1.1

For every network G there exists a valid interval labelling scheme.

In this paper we study various techniques for generating valid ILS that are optimum, or otherwise sufficiently flexible to allow for, for example, the addition of nodes or the joining of networks in an easy manner. We also study the effect of having a valid ILS available in a network on the design and the complexity of distributed control problems.

We briefly digress and describe the particular ILS that was used in Ref. 5 to prove Theorem 1.1. The scheme is generated by an algorithm that traverses G and assigns labels $\alpha(u)$ to the nodes u that are visited. The algorithm is based on the technique of depth-first search,[4] and works as follows.

Start at an arbitrary node and number it 0, pick an outgoing link and label it 1 (by which we mean that the corresponding exit at node 0 is labelled 1), follow the link to the next node and number it 1. Continue numbering nodes and links consecutively. If a link is encountered that reaches back to a node w that has been numbered previously (a link of this type is called a frond), it is labelled by $\alpha(w)$ instead and another link is selected. If a node is reached that admits no forward link any more (a node of this type is either a leaf or otherwise 'fully' explored) and i is the largest node-number assigned until this moment, we backtrack and label every link over which we backtrack by $(i+1)$ mod N until we can proceed forward on another link again. There is a slight twist to the labelling of links in this phase in case one backtracks from a node v that has a frond that reaches back to 0. The frond will have label 0 at v, and just when i happens to be $N-1$ the same label would be

* This work was carried out while the second author visited the University of Utrecht, supported by a grant of the Netherlands Organization for the Advancement of Pure Research (ZWO).

† Address: Department of Computer Science, University of Sciences and Arts of Oklahoma, Chickasha, OK 73018, U.S.A.

backtracks. The conflict is resolved by assigning the label $\alpha(u)$ to the link instead. In order to do this right, the algorithm marks a node as soon as it finds that it has a frond to 0. (It should be intuitive now that the ILS so constructed does its routing over the depth-first search spanning tree, with additional shortcuts over the fronds.) The following procedure makes the algorithm precise. Comments contain additional explanation.

```
{N is the number of nodes, and i a global variable
ranging over 0..N-1 which denotes the next node-
number or edge-label that is to be assigned. The variable
i is initially set to 0. For convenience we use a boolean
array MARK to keep track of the node(s) that have a frond
back to 0. MARK is initially set to false. The procedure
starts out at an arbitrary node x of the network, and is
called as LABEL (x, x).}
procedure LABEL (u, v);
{u and v are nodes, u is the father of v in the depth-first
search tree being constructed, and v is being visited.)
  begin
  {assign number i to v}
  α(v): = i;
  i: = (i+1) mod N;
  for each node w on the adjacency list of v do
    begin
      if w is not numbered then
        begin
          {proceed forward and add the link v, w to the
          depth-first search spanning tree}
          label link v, w at v by i;
          LABEL (v, w)
        end
      else
        begin
          {link v, w is a frond unless w = u. Mark v if
          the frond reaches back to 0}
          if w ≠ u then
      begin
        label link v, w at v by α(w);
          if α(w) = 0 then MARK[v]: = true
      end
    end
  end;
{backtrack over the link v, u unless v = x}
if v ≠ x then
    begin
      if i = 0 mod N & MARK[v] = true then
        label link v, u at v by α(u)
      else
        label link v, u at v by i
    end
{end of the procedure}
end;
```

We shall refer to the ILS obtained by applying the procedure *LABEL* as the DFS scheme, because it is generated during a depth-first search. Recall that depth-first search visits the entire network, that the links over which the algorithm moves 'forward' (and backtracks again at a later stage) together form a rooted tree spanning the network, and that fronds always point from a node to an ancestor of the node in this tree.[4] In Ref. 5 it was proved that the DFS scheme is indeed a valid scheme for the purposes of routing.

We note that the DFS scheme is in fact valid when the $i_{N-1}\}$. Any general ILS will have an equivalent form using labels from $\{0, \ldots, N-1\}$ by the natural correspondence between $\{i_0, \ldots, i_{N-1}\}$ and $\{0, \ldots, N-1\}$. An ILS is called *normal* if the set of labels is indeed the set $\{0, \ldots, N-1\}$.

In this paper we further explore the theory of general interval labelling schemes and its various implications for network problems, and apply it to solve some common distributed problems. In Section 2 we look at optimum schemes for some common networks such as rings and grids. Various concepts of 'near optimality' are introduced. In Section 3 several insertion and joining techniques are given to form a larger network that still preserve some desirable properties of a given ILS. Section 4 contains applications of ILS to solve for, for example, the leader-finding problem and the spanning-tree problem in substantially fewer message-exchanges than are required in general networks such as rings without the effect of a valid ILS. The results are intriguing from the point of view of distributed algorithms, as they show that implicit information can severely affect (lower) the message complexity of distributed problems. Finally some open problems are stated in Section 5.

2. OPTIMUM SCHEMES AND RELATED CONCEPTS

Ideally we would like an ILS not only to be valid but also able to deliver messages over the shortest possible routes. We call such a scheme *optimum*. The DFS scheme as discussed in Section 1 is a valid scheme, but it is far from optimum. For instance, a DFS scheme will not label a ring with more than four nodes optimally.

In the following, we list some common types of network and present optimum schemes for them. For the sake of simplicity we assume all ILS to be normal throughout this section.

(i) *Trees.* A DFS scheme gives an optimum scheme here. Santoro & Khatib[3] use a similar depth-first search of the tree, but with a different ordering of labels.

(ii) *Complete graphs.* A DFS scheme again suffices here, since all links are either direct links or fronds and will deliver messages in one hop.

(iii) *Rings.* An optimum scheme is given in van Leeuwen & Tan.[5] Basically the idea there is to orient the ring in one direction and label the nodes consecutively from 0 to $N-1$. Then for each node i, label the left link by $(i+1)$ mode N and the right link by $([N/2]+i)$ mod N.

(iv) *Complete bipartite graphs.* Let the set of nodes of the graph be separated into two parts, A and B. Label the nodes consecutively from 0 to $N-1$ in any order. Label the links by the node numbers they are connected to. For instance, if there is a link connecting node i and node j, label the link at node i by j and that at node j by i. Thus, by construction, there exists a direct hop from each node in one partition to all the nodes in the other partition. If nodes i and j are in the same partition and need to communicate then, by the circular nature of the interval order, there must be a link that carries the message to the opposite partition. From there it only takes one more hop to reach node j. So only one hop is needed to go across the partitions and two hops within the same partition, and this is optimum.

(v) *Grids.* we consider several grid configurations.

(va) *Grids with no wrap-around.* Let G be a rectangular grid of M rows and N columns. Label the nodes consecutively by rows from left to right, so that the first row will be labelled 0 to $N-1$, the second row N to $2N-1$, and so forth. Informally each link in the four directions (if there is any) will be labelled as follows. The up link is labelled 0 and the down link is labelled with the node number of the leftmost element of the next row. The left link is labelled by the node number of the leftmost element on the current row and the right link by the next consecutive element on the right. More precisely, for each node i, if there exist the appropriate links, label the up link by 0, the down link by $N+N.\lfloor i/N \rfloor$, the left link by $N.\lfloor i/N \rfloor$ and the right link by $i+1$. Note that the up link for all nodes is 0, all the down links for each row are identical, and so are the left links for each row. Thus we have the interval structure as shown in Fig. 1, where r_0 is $N.\lfloor i/N \rfloor$ and $(r+1)_0 = N.\lfloor i/N \rfloor + N$.

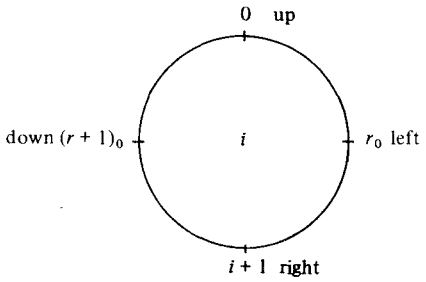

Figure 1.

We now show that the scheme is optimum by referring to the interval structure of Fig. 1. Suppose node i needs to send a message to node j. Assume first that i and j are on the same row of the grid, i.e. $r_0 \leqslant j < (r+1)_0$. If $i < j$ then $j \in [i+1, (r+1)_0)$ so the message is passed to the right and kept on passing to the right until node j is reached because j must belong to one of the successive intervals $[i+2, (r+1)_0), \ldots, [(r+1)_0 - 1, (r+1)_0)$. Similarly, if $i > j$ then j belongs to interval $[r_0, i+1)$ and the message is passed to the left until it arrives at j. If i and j are not on the same row then $j \in [0, r_0]$ or $j \in [(r+1)_0, 0)$ so the message is passed up or down to the next row respectively. After the next row is reached and if j is on that row, the previous process is applied and j is reached. If j is not on that row the message must be passed on to the next row in the same direction, i.e. once the message is passed up the link it cannot at any point be passed downward again and vice versa. This is because each r_0 keeps on decreasing for each row and $(r+1)_0$ keeps on increasing and the intervals $[0, r_0)$ and $[(r+1)_0, 0)$ are disjoint. Thus eventually the message will arrive on the row on which j is located by vertical travels, and from then on by horizontal hops to node j. The route the message travelled is not the only shortest one possible, but it is optimum.

(vb) *Grids with column-wrap-around.* G is a rectangular grid of M rows and N columns, but each column is extended so as to be a ring also. The nodes are labelled consecutively as in case (va). The left and right links for each node remain identical as in (va). Label the first column using the optimum scheme for a ring, then copy the up and down links of the first column to remainder columns. Precisely, the left link is $N.\lfloor i/N \rfloor$, the right link is $i+1$, the down link is $(N+N.\lfloor i/N \rfloor) \bmod (M.N)$ and the up link is $(N.\lfloor M/2 \rfloor + N.\lfloor i/N \rfloor) \bmod (M.N)$. The forbidding appearances of the vertical links are harmless. They are just straightforward translations of the ring with M elements. Recall the formulae there are $(i+1) \bmod M$ and $(\lfloor M/2 \rfloor + i) \bmod M$. Now $\lfloor i/N \rfloor$ plays the role of i, and since there are N columns, multiplying both equations by N yields the desired links. For a message to reach node j from node i, assuming they are on the same row, the message will travel by horizontal hops to its destination as before. If i and j are not on the same row the message must go round the ring until the correct row containing j is reached. This can be seen by applying the proof for optimum ring networks, where i now stands for the ith row, so that the row containing j is found in the most optimum way. Then the message is delivered by horizontal hops.

Unfortunately the same technique does not work for grids with row and column wrap-around. This is because we lose the circular ordered effect of the ring interval on a row. So the question of an optimum scheme for row and column wrap-around remains open.

One way to salvage the above situation is to introduce multiple labels on a link.

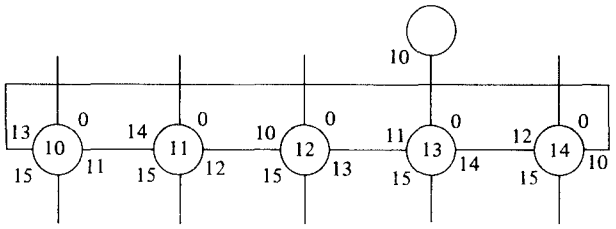

Figure 2.

Definition

A *k-labelled* ILS is an ILS where (i) each link may receive up to k distinct labels and (ii) at every node all the link-labels must be distinct.

Thus the usual ILS is simply a 1-labelled ILS. We now show how this concept can be applied.

(vc) *Grids with row and column wrap-around.* Let G be a grid of M rows and N columns with each row and column extended to be a ring. The idea is to label the nodes as before, then label each column as a ring as in case (vb), following the first column, and finally to label each row also as a ring. However, this naïve approach does not give us even a valid scheme. For instance on a 5×5 wrap-around grid, the optimum ring on the third row is as shown in Fig. 2.

It is not possible for node 13 to send a message to node 10 via the usual circular route. The message has to go up and come down again, forming a cycle. This is solved by labelling link (13, 14) by both 14 and 10. The labelling scheme is then as follows. Label the nodes consecutively by rows. Label the up link by $(\lfloor M/2 \rfloor . N + \lfloor i/N \rfloor . N) \bmod (M.N)$, the down link by $(N+N.\lfloor i/N \rfloor) \bmod (M.N)$, the left link by $(\lfloor N/2 \rfloor + i) \bmod N + N.\lfloor i/N \rfloor$ and the right link by $(i+1) \bmod N + N\lfloor i/N \rfloor$. Now, for each row r, $r = 0, \ldots, M-1$, check each element i, $i \neq rN$. If the horizontal links do not contain rN as a label, pick the horizontal link with the highest label and add label rN to it. We thus have a two-labelled scheme. Note that the previous two grids of cases (va) and (vb) all have $r.N = \lfloor i/N \rfloor . N$ as their left links, so that we have no such problem. By

every node i on that row r, node i has a link-label rN and also $(r+1)N \bmod (MN)$. Furthermore, all the horizontal link-labels are within this interval. Thus when a message travels from node i to node j, it first reaches the correct row r, such that j is on that row. Then it reaches j by horizontal hops. The scheme is optimum.

We have only given optimum schemes for a few types of common graphs. In general it is not clear how one would construct an optimum scheme for an arbitrary graph, if such a scheme is possible. However, it can be quite easy to do this for multiple-labelled schemes.

Proposition 2.1

For any graph with N nodes, there exists an $(N-1)$-labelled ILS that is optimum.

Proof

Label the nodes somehow from 0 to $N-1$. For each node i, pick a node $j \in [0, N-1]$. Find a path to j that is optimum, say via link (i, p). Then label link (i, p) by j. Do this for all $j, j \neq i$. A maximum of $N-1$ labels suffices. □

Note that the above $(N-1)$-labelled ILS is nothing but the traditional routing table in disguise, with one label for each node. Thus the multiple-labelled ILS is just a generalization and simplification of the traditional routing table. We are trying to achieve the same goal with fewer labels!

In the following we introduce a few concepts that are related to optimum schemes, though they are strictly weaker than optimality. Observe that in a DFS scheme a node may not necessarily send a message addressed to its neighbour directly in one hop.

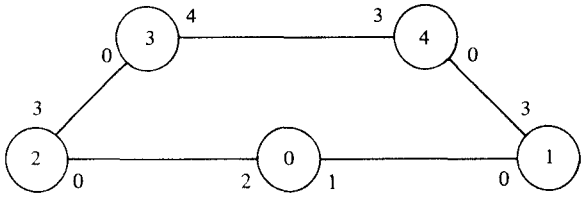

Figure 3.

Definition

An ILS is a *neighbourly scheme* if it is valid and all messages for a neighbour are delivered directly in one hop.

An optimum scheme of course is a neighbourly scheme. The converse is false, as shown by the following example in Fig. 3. The scheme is neighbourly, but not optimum, since *SEND* (0, 4, m) traverses the path $0 \rightarrow 2 \rightarrow 3 \rightarrow 4$ instead of the shorter route $0 \rightarrow 1 \rightarrow 4$.

Lemma 2.2

The only nodes in a DFS scheme that do not necessarily deliver messages to neighbours in one hop are those nodes k that have fronds to nodes i with $i \neq 0$, $i < k$.

If j and k are neighbours and link (j, k) is a frond in the spanning tree of DFS, then by construction link (j, k) is labelled k and link (k, j) is labelled j, so messages are delivered in one hop. So assume j and k are neighbours but link (j, k) is not a frond. If $j < k$ then the label of link (j, k) must be k (by the labelling procedure of DFS), so messages get there in one hop from j to k. Thus we only have to consider *SEND* (k, j, m). If there are no fronds coming down from k to i where $i < j$ link (k, j) labelled b is a backtrack edge. Also there must be a link labelled $k+1$ emanating from k (by the labelling procedure of DFS). Thus $j \in [b, k+1)$, and the message gets routed to j via link (k, j) labelled b. If there is a frond coming down from k to i, but $i = 0$, then by the labelling procedure of DFS, link (k, j) must be labelled by j, so that message gets to j in one hop. The remaining case is when k has a frond coming down to i and no $i = 0$. Let the backtrack link be (k, j) with label b. Then $j \notin [b, i)$ since $i < j < b$ or $b = 0$, so $j \notin [0, i)$. Thus the message cannot come down from k to j via link (k, j). It has to be routed via one of the fronds, link (k, i). □

We use a multiple-label scheme to salvage the above situation.

Theorem 2.3

There exists a two-labelled neighbourly scheme for any arbitrary graph.

Proof

We first do a DFS scheme on the graph G. By Lemma 2.2, the only concern are those nodes k that have fronds going down to i with $i < k$ but no $i = 0$. For each such k and its neighbour j via the backtrack link labelled b, we double-label link (k, j) by b and j. We thus have a two-labelled scheme that is neighbourly. To show that the resulting scheme is valid, we only have to be concerned with those special k-nodes. Let i be the maximum frond node in the above situation. Then normally messages to any node $t \in [i, k+1)$ will travel via link i. With the introduction of the new label j, with $i < j < k$, those messages to $t \in [j, k+1)$ get transferred to node j first. So we only have to make sure that any message from k to $t \in [j, k+1)$ is routed correctly. Now $j \leqslant t < k$, so it is not possible for the message to return to node k again via link (j, k) labelled k. Furthermore, since $t \geqslant j$, the message will be routed to the subtree of the DFS spanning tree rooted at j. The message will never encounter the situation of Lemma 2.2 again on the way as it will be an upward climb. As the DFS scheme is valid, the message will eventually reach t. □

Another way to salvage the situation in Lemma 2.2 is to restrict the way the DFS labelling algorithm proceeds in generating the spanning tree. We would like the depth-first search to proceed in an orderly manner, exploring all the sub-branches as much as possible before encountering a 'backward' frond.

Definition

A DFS scheme is *orderly* if, whenever there is a 'backward' frond from node k to node i and $x > k$, either

x must belong to the subtree of the DFS tree with k as a root or x does not belong to the subtree with i as a root in the DFS tree. This means that if x is explored after k, x must be further 'up' the tree from k or further 'down' the tree from i.

Lemma 2.4

In an orderly DFS scheme, if there is a backward frond from node k to node i and the backtrack link at k is labelled b, the backtrack link at i is also labelled b.

Proof

Since b is the label for the backtrack link, b does not belong to the subtree with k as a root, which implies that b does not belong to the subtree with i as a root also. Thus every backtrack link from i to k must be labelled b. □

Theorem 2.5

There exists a neighbourly interval-labelling scheme for every graph that has an orderly DFS scheme.

Proof

We first relabel the given orderly DFS scheme. For each node k that has a backward frond to some nodes i, we relabel two links. First the label on the backtrack link is changed from b to j, the father of k in the spanning tree. Secondly, we find the smallest frond link i and relabel it from i to b at k.

Claim (*i*). The scheme is neighbourly.

By Lemma 2.2, we only have to examine nodes k that have a backward frond. Let i, j, k and b be defined as above. *SEND* (k, j, m) now delivers message m to j in one hop. *SEND* (k, i, m) used to deliver messages to i via the frond link in one hop also, but now we have changed the link to b. Suppose there are frond links to $i_1, i_2, \ldots, i_s$ with $i = i_1 < i_2 < \ldots < i_s$. Now $i \in [b, i_2]$ or $i \in [b, j]$ depending on how many fronds there are. In either case, the message to i gets there in one hop. The rest of the links are unchanged, so the scheme remains neighbourly.

Claim (*ii*). The scheme is valid.

Any message sent to x will arrive properly if it does not pass through a node k with a backward frond, since the DFS scheme is valid. Therefore we only need to consider *SEND* (k, x, m). Only two links have been relabelled at k. Messages for most nodes x still follow the same link and lie in the same interval, with the exception of those in the intervals $[j, k)$ and $[b, i_1)$. Messages for those nodes in $[j, k)$ used to follow the frond link i_s (or i, if there is only one frond) down to node i_s (i_1) and then 'up' the tree to their destinations. Now they only have to take one hop to j and go from there. Thus the new scheme bypasses the intermediary, and cuts down on the actual distance. Messages for the other nodes in interval $[b, i_1)$ used to climb 'down' the tree from node k to node i_1 first and then go to their destinations. Now they take one hop to i_1 and go to their destination from there. So the actual distance gets smaller once again. Note also that after a message traverses down the two links it cannot go back up the link in the next hop. Thus after reaching node k, the message still follows the path of the DFS scheme, and in some cases it even shortens the path. □

Corollary 2.6

There exists a neighbourly scheme for any Hamiltonian graph.

Proof

Apply the DFS labelling algorithm to the Hamiltonian graph G following a 'hamiltonian traversal'. The resulting DFS scheme is orderly. The result now follows from Theorem 2.5. □

Finally, we introduce another concept that measures the effectiveness of an ILS. Ideally, if a node blindly sends out a message to itself it should receive the message back in minimum time. The number of hops the message takes is the *index* of the node. The index of an ILS is the maximum of indices of all nodes. Clearly, the smallest possible index is 2. Both optimum schemes and neighbourly schemes necessarily satisfy the 'index 2' condition. The converse is not true.

Proposition 2.7

A DFS scheme is of index 2.

Proof

Suppose node i wants to send a message to itself. If the link that it traverses is a frond link to node j, then by the construction of DFS link (j, i) is labelled by i, so the message immediately returns to i. Suppose the link is not a frond. Then it cannot be a forward link in the spanning tree generated by the depth-first search algorithm, since all forward links have labels $j > i$. Thus the link must be a backward link to node k. This means that k has been numbered before i, so $i > k$ and thus link (k, i) must be labelled by i, and the message returns to i again. □

A DFS scheme also has the property that each node i has a link labelled $(i+1)$ mod N. Such an ILS is called *sequential*. All the optimum schemes presented earlier are sequential, with the exception of the ILS for the complete bipartite graph. Thus an optimum scheme need not be sequential.

3. INSERTION AND CONNECTION OF SCHEMES

Consider the practical situation in which a network expands and grows by incremental insertion of nodes or by connection to other networks. In this section we study how a network with a given ILS can 'grow' by incremental insertion of a node or by connection to another network with a given ILS so that the combined network still has an ILS of some desired form.

Central to the insertion and connection problem is the concept of cyclically shifting a node number until it reaches a desired value. We again assume all ILS to be normal.

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.