A REDUNDANCE AWARE ALGORITHM FOR THE RING PERCEPTION PROBLEM

GIORGIO MANCINI

Dipartimento di Matematica e Fisica,
Universitá di Camerino, Madonna delle Carceri, 62032 Camerino, Italia
mancini@task.gda.pl

Abstract: Following the guidelines proposed by R. Balducci and R. Pearlman [1] for an efficient exact solution of the Ring Perception Problem, a new approach based on 'pre-filtering' technique is introduced to perceive rings in structures represented by 2-connected graphs. The resulting algorithm has proved to reduce both resources allocation and redundant information processing when dealing with chemical cases. Actual computing times have constantly shown a conspicuous reduction with respect to methods using hash-tables [2,3] to treat redundant information. Furthermore no user intervention to 'tune' effectiveness is required (e.g. hash-table dimensioning).

1. Introduction

The detection and analysis of cyclic structures is of interest in various fields. Obviously the most effective approach is to get the least possible amount of data needed to describe the structures under study; the identification of a minimal basis of the ring space (a Smallest Set of Smallest Rings) of a structure meets the requirement.

No doubt the work by Balducci and Pearlman has represented a major achievement in exact determination of SSSR's. Following their guidelines to implement a ring perception algorithm for chemical cases, the flow of operation has been restated and a new approach (dubbed pre-filtering) to information propagation has resulted in an original way to handle and reduce redundant information processing obtaining an efficient algorithm using less resources and demanding no user intervention to 'tune' parameters in order to balance resources and performance.

Given a graph of N nodes and E edges, Balducci and Pearlman's algorithm (BPA) performs two basic tasks:

(1) identification of a superset of a minimal basis of rings;

(2) selection of a set of $R = E - N + 1$ rings to form the required SSSR.

The original idea in BPA is the simulation of a synchronous communication network in which each node represents a transceiver and each edge a communication channel. All nodes simultaneously receive and then simultaneously send 'path-messages' through communication channels from/to adjacent nodes in such a way that any message received by a node n from a node m is forwarded to all nodes adjacent to *n. but m*. The union of the paths traversed by two different path-messages originating from a common source and reaching a common node (colliding on that node) represents a ring in the structure.

BPA is designed such that all 'path-messages' have the same length, this length being increased each time a message is transmitted, so that rings are identified and processed in increasing length order to build up an SSSR.

Basically the whole process consists of four phases:

(a) network initialisation;

(b) messages sending;

(c) messages receiving;

(d) rings selection;

and develops cycling over b through d until a SSSR is obtained.

On the base BPA a model has been developed and implemented for chemical cases (structures represented by 2-connected graphs for which N << E).

2. The underlying model

Adjacency matrix. The upper diagonal part of the adjacency matrix of the structure is represented by an array of variable-length integer arrays such that the lesser the degree of a node the shorter the representing array. The adjacency matrix is used for nodes-to-edge and conversion together with connection checking throughout the process.

Path-messages. Each path message is represented by a structure made up of four fields:

(1) an integer recording the node that originated the message (*nfirst*);
(2) an integer recording the first edge traversed by the message (*efirst*);
(3) an integer recording the last node traversed by the message (*nlast*);
(4) a dynamically allocated integer array (*DIA*) keeping track of all nodes traversed, save the first and the last.

The length of a path-message is the number of edges traversed by the message; each path is reconstructed by means of a function **indices_to_edge()** operating on the couples:

(*nfirst*; *DIA*[1]), (*DIA*[1]; *DIA*[2]), . . . , (*DIA*[n]; *nlast*), (*nlast*; *current_node*).

Such a representation of path-messages (see [1] for comparison) has proved very effective in saving memory resources.

Collisions. The collisions of two path-messages p_i and p_j of the same length *l* received by the same node are classified as follows:

(1) no collision: $nfirst_i \neq nfirst_j$ *and* $efirst_i \neq efirst_j$

[two path-messages originating from distinct nodes get to a common receiver; the union of the two paths does not form a ring];

(2) node collision: $nfirst_i = nfirst_j$ *and* $efirst_i \neq efirst_j$

[two path-messages originating from the same transmitter get to a common receiver. If the paths are entirely separate, their union gives a ring of size $2l$, otherwise their union contains at least two shorter rings];

(3) inverse-edge collision: $nfirst_i \neq nfirst_j$ *and* $efirst_i = efirst_j$

[two path-messages originating from distinct transmitters traverse a common edge in opposite directions and get to a common receiver. If the paths are entirely separate save the first edge, their union gives a ring of size $2l - 1$, otherwise their union contains at least two shorter rings];

(4) direct-edge collision: $nfirst_i \neq nfirst_j$ *and* $efirst_i = efirst_j$

[two path-messages originating from the same transmitter and traversing the same edge as the first one get to a common receiver; the union of the two paths contains a ring of size at most $2(l-1)$].

3. Pre-filtering

Perceived rings must undergo a linear independence test against the previously identified elements of the SSSR in order to enter it or be discarded. Since the same ring of size *l* may close on all of its nodes, a new technique - pre-filtering - has been developed to exploit such behaviour (referred to as *full formation* in the following): having previously processed the rings formed on nodes $1, \ldots, n-1$, while operating on the n-th, we can skip testing (pre-filter) all rings closing on it after passing through a node $m < n$ - since they have been already handled while processing node *m*. Obviously full formation requires that all path-messages colliding on any node be propagated. As a matter of fact the pre-filtering approach only requires that any ring of length *l* passing through m and closing on *n* be a linear combination of a ring of length *l* closing on *m* and some of the identified elements of the SSSR. It then suffices to propagate just one of all the paths-messages originating from a common *nfirst* involved in a node-collision [4].

Such a propagated path-message (a path-message in general), in its ramification process, may eventually originate one or more collapsing path-messages (i.e. re-aching again a node previously traversed); this event is prevented during messages sending phase.

4. Path-message functions

Path messages are operated on by a set of functions:

- **indices_to-edge()**: see above;
- **push()**: appends *nlast* to *DIA* and updates *nlast* with the sending node;
- **merge()**: selects a path-message to be propagated out of a set of mutually direct-edge colliding messages;
- **join()**: given a pair of node/inverse-edge collided paths of length *l*, checks paths for passage through one of the nodes already processed and for paths intersection:if the check fails an ordered sequence of edges is returned identifying a ring of length 2*l*/2*l*-1.

5. Transceivers

Each transceiver node (*Tnode*) is represented by a structure made of a receive and a send buffer consisting in two variable length array of path-messages. Path-messages in the receive buffer are processed by function **receive()** that identifies rings, submits them to a ring selection process, moves messages to be propagated into the send buffer, resets the receive buffer to the empty state. Path-messages in the send buffer are processed by function **send()** that forwards an updated copy of each to the receive buffers of adjacent nodes in such a way as not to collapse them.

6. Network Initialisation

The receive buffer of each *Tnode n* is set to the empty state and the send buffer contains d_n (being d_n the degree of the node) distinct path-messages of length *1*. Each path-message having *nfirst* and *nlast* set to the same *Tnode* n_i adjacent to *n*, *efirst* set to the edge connecting *n* to n_i and an empty *DIA*.

7. Send() algorithm

The **send()** function operates on the path-messages stored in the send buffers as follows:

```
For each Tnode n (n =1, . . . , N):
{
  for each path-message $p_{n_i}$ (i =1, . . . , M) of length l in the send buffer of n:
    {
     for each $Tnode\ m$ adjacent to $n$:
       {
        if in $p_{n_i}$ results $m \neq nfirst, DIA[1], \ldots, DIA[l-1], nlast$:
          {
           make a copy $p'_{n_i}$ of $p_{n_i}$
           push() $n$ on $p'_{n_i}$
           put $p'_{n_i}$ in the receive buffer of $m$
           delete $p'_{n_i}$
          }
       }
    delete $p_{n_i}$ }
}
```

8. Receive() algorithm

The **receive()** function operates on the path-messages stored in the send buffers as follows (see [1] for comparison and for an efficient collision detector algorithm):

(1) [odd-sized rings processing]

For each *Tnode* n ($n = 1, \ldots, N-1$):

(a) **merge()** into a single message any group of path-messages involved in a direct-edge collision with each other
(b) **join()** any pair of path-messages [in the set identified by **merge()**] involved in an inverse-edge collision and forward the resulting ring, if any, to the ring selector

(2) [even-lengthed rings processing]

For each *Tnode* n ($n = 1, \ldots, N-1$)

(a') **join()** any pair of path-messages [in the set identified by **merge()**] originating from *Tnodes* $m > n$ involved in a node collision and forward the resulting ring, if any, to the ring selector
(b') move any non collided path-message considered in (a) to the send buffer
(c) **merge()** into a single message any group of two or more path-messages involved in a node collision and move the resulting messages to the send buffer
(d) reset the receive buffer to the empty state.

9. Ring selector

Supposing $r_c < E - N + 1$ rings of the SSSR have been previously selected, the ring selector operates as follows on a forwarded ring:

for $i = 1, \ldots r_c$ if the ring is not a linear combination of the r_c rings previously selected

(a) the ring enters the SSSR
(b) r_c is increased by 1
(c) if $r_c = E - N + 1$ then SSSR determination process is stopped.

Linear independence of rings is tested using a conceptual Gaussian elimination procedure: noting that a linear combination of two set of edges amounts to their symmetric difference, the Gaussian matrix (maintained in upper echelon form) is represented as a variable-length array of variable-length integer arrays and the elimination process is carried out in a 'quasi-formal' manner as follows:

```
(a) according to the permutations array associated to the Gaussian matrix, build
    the resulting set $\{E_0\}$ of edges out of the edges of the ring to be tested (e.g.
    if the ring is individuated by the set of edges $\{2,4,6\}$ and the permutation
    arrays contains $\{2,1,4,5,3,6,7,8,\ldots\}$ then $E_0 = \{1,3,6\}$)
(b) append $\{E_0\}$ to the Gaussian matrix
(c) for $i = 1,\ldots, r_c$:
    {
    carry out any new formal permutation required and perform linear combi-
    nation of rows $i$ and $c + 1$ of the Gaussian matrix
    if an empty set is obtained then: discard the incoming ring, remove row
    $c+1$, break cycle.
    }
(d) if an empty set is not obtained then add the ring to SSSR.
```

10. Process driving

Finally, the overall process of SSSR perception is performed as follows:

(1) initialise the network
(2) set the iteration counter c to one
(3) increment c by one
(4) call **send()** for each node
(5) call **receive()** for each node
(6) cycle over steps 3 - 5 until $E - N + 1$ linearly independent minimal rings are identified.

11. Conclusions

The algorithm described in the present paper has been implemented and tested over a large number of chemical structures on different platforms /machines (VMS, Linux, Irix, AIX. . . / Digital alpha, Pentium pc, SGI, IBM...) and to my best knowledge it has been the first implementation of an algorithm following BAP guidelines.

References

[1] R. Balducci and R. S. Pearlman. Efficient Exact Solution of the Ring Perception Problem, *J. Chem Inf. Comput. Sci.* 1994, **34**, 822-831

[2] Knut, D. E. Sorting and Searching. The Art of Computer Programming, Addison-Wesley: Reading, MA, 1973; Vol. 3.

[3] Sedgewick, R. Algorithms in C. Addison-Wesley: Reading, MA,1990

[4] Horton, J. D. A Polynomial-Time Algorithm to Find the Shortest Cycle Basis of a Graph. SIAM J. Comput. 1987, 16, 358-366