Proceedings of the First Southern Symposium on Computing
The University of Southern Mississippi, December 4-5, 1998

ALGORITHMIC ISSUES IN NODE MANAGEMENT POLICIES FOR DISTRIBUTED R$^+$-TREES

ANSELMO COCCHI* AND ENRICO NARDELLI†

Abstract. In this paper we address algorithmic issues arising in considering the extension to a distributed framework of data structures for spatial data.

1. Framework. The newer applications that are being developed require more and more an efficient representation and access to multi-dimensional objects. One approach to gain efficiency is to use the computing power that is collectively available in organizations over the network.

The technological framework we make reference to is the so-called *Network Computing* [9], characterized by fast communication networks, i.e. fiber-optic networks delivering 100Mbits per second that are now cheaper and cheaper, and powerwul and cheap workstations (consider that for 10K dollars you can now have very fast machines).

With these ingredients it is easily possible to accumulate a very large computing power. Communication between machines takes the form, at a logical point of view, of point-to-point messages, which is the standard assumption.

The objective of our paper is to define a distributed data structure able to manage efficiently k-dimensional points. The approach we take to his aim is to extend to the considered distributed framework efficient data structures already developed for the case of a single machine.

In this kind of distributed environment the key requirements to obtain efficiency, introduced by Litwin *et al.* [8], are:

- *no centralized control*, otherwise bottlenecks may derive with the increase of the size of the data set,
- *scalability*, that is capability of the structure to adapt itself to a growing number of points, so that advantage can be obtained from a distributed context, where one can find additional computing power.

For what regard queries, the basic requirements for every multidimensional data structure are:

- *exact match*, where the query is looking for a point whose all coordinates are given,
- *range*, looking for all points lying in a given k-dimensional interval.

We assume the distributed data structure is used by a variable number of machines, called *clients*, which query the machines managing the k-dimensional space and storing the k-dimensional points, called *servers*. Clients have different and variable behaviour, hence it cannot be anticipated if and when they are connected to the network to be kept up-to-date with the evolution of the structure. This means that, in general, some adaptable indexing mechanism have to be set up to avoid that, for each query, a client is disturbing all servers [10].

The performance measure we consider are hence geared at evaluating how well the structure is behaving from a distributed point of view. Hence our measure is the overall number of messages traveling over the network for a given query. For more details on issues regarding distributed data structure in the described framework see [9].

*Dipartimento di Matematica Pura ed Applicata, Univ. of L'Aquila, Via Vetoio, Coppito, I-67010 L'Aquila, Italia.
†Dipartimento di Matematica Pura ed Applicata, Univ. of L'Aquila, Via Vetoio, Coppito, I-67010 L'Aquila, Italia, (nardelli@merlino.iasi.rm.cnr.it). Istituto di Analisi dei Sistemi ed Informatica, Consiglio Nazionale delle Ricerche, Viale Manzoni 30, I-00185 Roma, Italia.

2. The data structure. The distributed data structure we have defined in this work is an extension to the distributed framework of the *R^+-tree* [1], a well known and widely used data structure to manage spatial data of the class of R-trees [3].

More precisely, our distributed data structure can be considered, from a a conceptual point view, as a unique R^+-tree. From a physical point of view, this unique R^+-tree is cut in various pieces, each one managed by a different machine. To be more precise, each leaf of the R^+-tree stores one bucket of data, and each bucket is managed by its own server. Each server also manages one of the internal nodes of the R^+-tree, that are used to guide the search process. This schema was introduced, with reference to binary (but not balanced) search trees by Kröll and Widmayer [7].

The behaviour of the structure is determined by the clients, that add k-d points. A new k-d point is added to the pertinent bucket, that is to the bucket covering the part of the k-d space where the object lies. When a bucket overflows, due to the insertion of a new element, we split it following the usual rules for R^+-tree.

Before proceeding further we now briefly recall the definition and the behaviour of R-trees, for the case of a single machine. R-trees can be considered as an extension of a B-tree to the multidimensional case. For more details see [3].

Namely, an R-tree is a m-ary tree to index 2-dimensional points or 2-dimensional extended objects. For ease of description in the following we describe it in the case it manages 2-dimensional points. An R-tree has the following characteristics (see also figure 2.1):

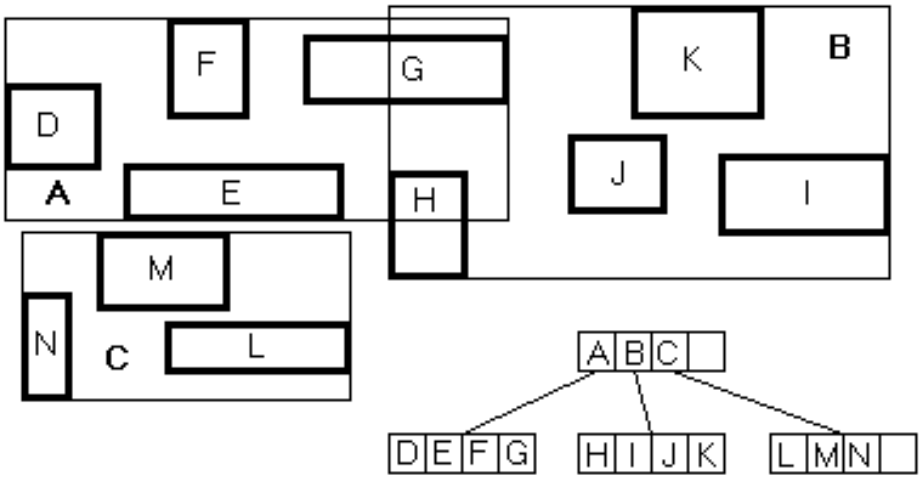

FIG. 2.1. *Example of an R-tree*

- Each leaf nodes stores a set of records of type (*B, id*), where B is the smallest rectangle containing the set of points referred to by pointer *id*;
- Each internal nodes stores a sets of records of type (*B, id*), where *id* points to a child and B is the smallest rectangle containing all rectangles associated the child node;
- Each node has thus associated a *minimum bounding rectangle* (MBR) that is the smallest rectangle containing all rectangles B stored in the node;
- Each node but the root has a degree less than or equal m and greater than or equal $m/2$. The root has degree ≥ 2 and $\leq m$;
- All paths from the root to a leaf have the same lenght. This means that all leaves are at the same level, hence all the search paths have the same upper bound on their length.

The last two items above are similar to the analogous constraints for B-tree.

Search in an R-tree is executed by a top-down traversal, by looking for the desired element in all nodes such that their associated MBRs intersect the query element.

Example. In the R-tree shown in figure 2.1 there are three leaf nodes, A, B, and C. Nodes A and B overlap. Rectangle G is present only in node A, but has to be searched also in node B, since node

B is overlapping part of the space where G is, and, before searching, one cannot say if G is in A or in B. □

Insertion is done by carrying out a search and then inserting the point in a leaf node, chosen by means of some suitable heuristics.

The MBR of the leaf node which takes care of the just inserted element may be enlarged in consequence of the insertion. Possibly, also parents of such a leaf node may be enlarged.

If it is needed to maintain the constraint on node degree, nodes are split during the insertion process, and this is managed like in B-trees, by letting the R-tree grows towards the root.

Many variant of R-trees have been defined. The one we are interested to is called R^+-tree [1]. Its distinctive characteristic is that to avoid multiple search paths, in R^+-trees MBRs are split during the insertion process so that, at each level of the tree, MBRs associated to nodes never intersect (see also figure 2.2).

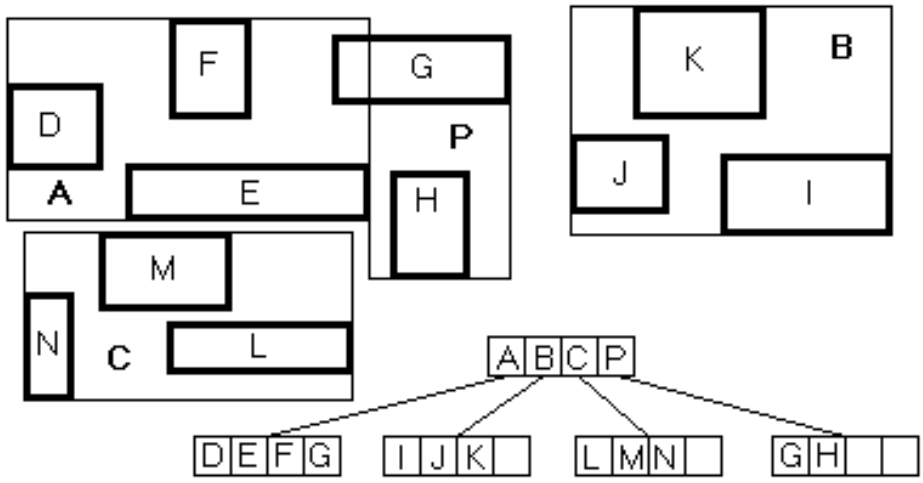

FIG. 2.2. *Example of an R^+-tree*

Example. In the R^+-tree shown in figure 2.2 the requirement of having non overlapping nodes is satisfied by the insertion of a new node, named P, which stores H and part of G. In fact since G is anyway overlapping both nodes A and P, rectangle G is split and stored twice, once in A and once in P. □

The drawback is that the maintenance of the non-overlapping constraints between MBR may cause the rearrangement of the assignment to nodes of the already existing points.

A further problem is that when, during insertion, a node is forced to split, the maintenance of the non-overlapping constraints may cause some node to have less than $m/2$ children. In this case a restructuring of the tree (or of some subset of it) is required.

3. The deadlock problem. When a new point is inserted it might happen that no server can enlarge its MBR to take care of the new point withour intersecting other servers.

Example. A deadlock is shown in figure 3.1. There are four servers, namely A , B, C, and D. The MBR of the four servers is the rectangle named N, that is the MBR of the parent in the R^+-tree of the four nodes managed by the four servers. The shaded area in the figure is a deadlock zone: if a point is inserted in it, any of the servers that enlarges its MBR is going to insersect another server. □

In the distributed version a reorganization of the tree is too costly in terms of messages, since such a reorganization can involve, in the worst case, all node of the tree, hence can require to send messages to all servers.

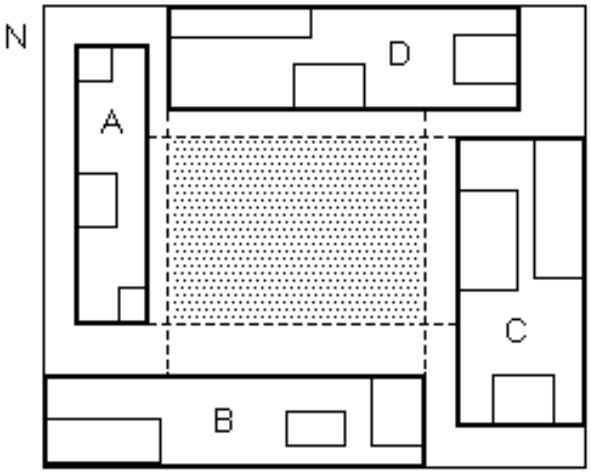

FIG. 3.1. *A deadlock configuration*

The only technique to avoid deadlock is to prevent it by suitably choosing, at each insertion, the node to be expanded. We now show how this is modeled from an algorithmic point of view.

To avoid deadlock we introduce the *Coverage Problem*, whose definition is the following:

PROBLEM 3.1 (The Coverage Problem). *Given a set S of disjoint rectangles, which are called* seeds*, does it exist or not a set C, called* coverage*, of rectangles such that:*

- *each rectangle in S is contained in exactly one different rectangle of C,*
- *rectangles in C are pairwise disjoint,*
- *the union of rectangles is C is the MBR of rectangles in S.*

Example. In figure 3.2 there are four seeds numbered from 1 to 4. For this case the Coverage problem admits a solution, shown by the 4 rectangles named A to D which satify the above constraints. □

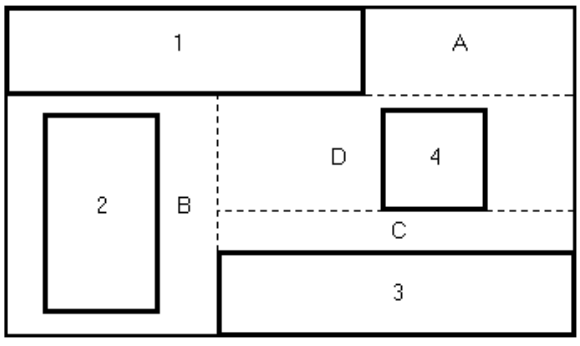

FIG. 3.2. *The Coverage problem*

To avoid the deadlock, the approach is to solve the Coverage problem for the node of the distributed R^+-tree such that the point has to be inserted into one of its children.

Infact, given the solution to the Coverage problem, the point will be assigned for the insertion to the child node that is contained in the rectangle of the coverage that includes the point itself. This guarantees that such a child node can include the point without causing any deadlock in the future.

4. Solving the deadlock problem. To tackle the Coverage problem we first consider a simpler version of it, namely the *Coverage Problem for Iso-Polygonal Zones*. In such a simpler version:

- the set of seeds defines a polygonal zone whose sides are parallel to the orthogonal axes,
- no seed is present internally to the defined zone,
- all seeds have one of their sides aligned along the perimeter of the defined zone.

An example of an iso-polygonal zone is shown in figure 4.1.

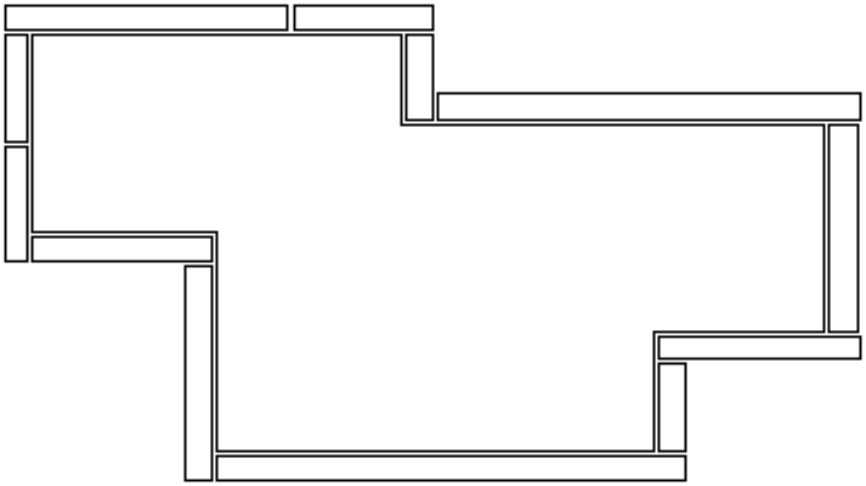

FIG. 4.1. *An example of iso-polygonal zone*

A basic concept for an iso-polygonal zone is the *cut*. A *cut* is a isothetic line cutting in two the polygonal zone without cutting any seed. Note that a cut can go along a side of a seed, but cannot traverse the internal of a seed. In figure 4.1 the iso-polygonal zone admits no cut, since all isothetic lines that do not cut seeds are not able to partition in two the iso-polygonal zone.

A particular case of iso-polygonal zone is when the zone is simply a rectangle. In such a case we have the following result.

THEOREM 4.1. *A rectangular zone admits a coverage if and only if it admits at least one cut.*

Proof. We first prove the condition is sufficient, namely if a cut exists then the zone admits a coverage.

In figure 4.2 a horizontal cut is shown for the rectangular zone. The case of a vertical cut is analogous. The cut, see the leftmost drawing, evidentiates two zones, named zone 1 and zone 2.

FIG. 4.2. *A rectangular zone and its cut*

Let us consider zone 1. The configuration of seeds along the borders of zone 1 can only be one of the two shown in the center or to the right (or the symmetric version of the configuration to the right). If we have the configuration in the center, then we cover zone 1 with expansion of rectangles coming from above. If we have the configuration to the right, then we cover zone 1 with expansion of rectangles coming from the left.

To show that the condition is necessary we have to introduce the concept of *visible partition*. A *visible partition* for an iso-polygonal zone is a partition of it into rectangles so that each rectangle share at least one of its sides with the perimeter of the zone. In figure 4.3 on the left is shown a

visible partition and on the right a non-visible partition, since the shaded rectangle in the middle has no side in common with the perimeter of the zone.

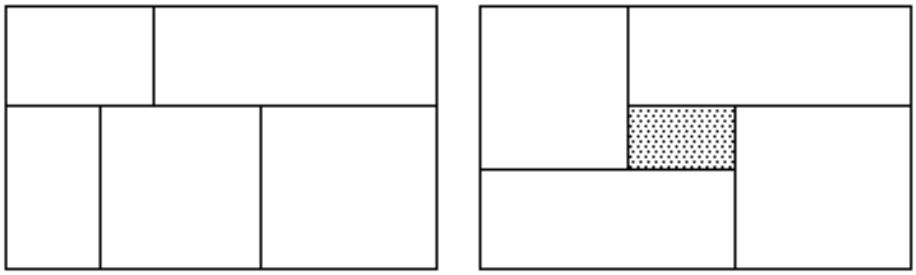

FIG. 4.3. *A visible partition (left) and a non-visible partition (right)*

A consequence of the existence of a visible partition is the following result.

LEMMA 4.2. *If a rectangular zone admits a visible partition then it admits a cut.*

Proof. (Sketch) If we start from one side of the rectangular zone along one side of a rectangle, due to the fact that cannot exist rectangles with no side in common with the perimeter, sooner or later we encounter an alignment of sides of rectangles, that is a cut. □ Now, it is esay to prove the necessity of the condition. Infact, if a rectangular zone admits a coverage then such a coverage is a visible partition, hence, by the previous lemma, it admits a cut. □

In general we have the following result.

THEOREM 4.3. *The Coverage problem is solvable for an iso-polygonal zone if and only it admits a cut dividing it into two zones such that for each of them the Coverage problem is solvable.*

Proof. (Sketch) In figure 4.4 you can see that it is possible to expand each of the seed with a little arrow, each until it reaches the continuous lines. Such an expansion does not respect the

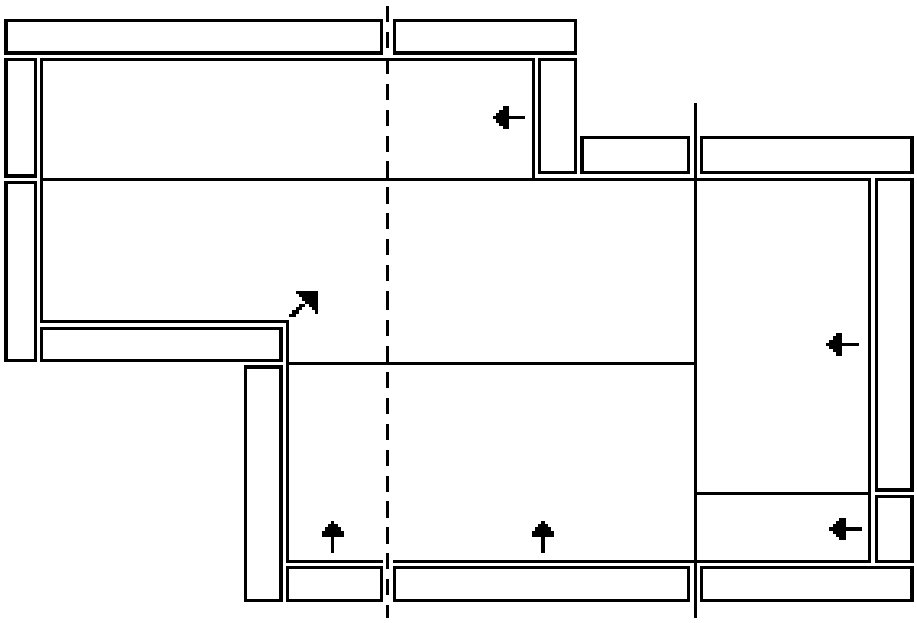

FIG. 4.4. *An example for the Coverage problem*

cut indicated by the dashed line, that is some seed are traversed by the dashed cut. Instead, such an expansion respects the cut indicated by the continuous line on the right. But we can modify expansions of seed so that the dashed cut is respected. For example we can expand the two seeds at the right not just until the continuous line but until the dashed line, expand the bottom small seed

just on the left of the dashed line all the way through to the upper border of the rectangular zone, and then complete the coverage with the expansion of the seed with the oblique arrow on the left and the expansion of the seed with the horizontal arrow on the top. □

The meaning of this theorem is that the Coverage problem for iso-polygonal zones can be recursively solved, by first finding a cut and then solving in the problem in each of the two obtained parts. The importance of the theorem is that *any* cut that one finds will work, that is, it is not required to find a *particular* cut.

On the basis of the previous theorem we can define a polynomial algorithm to solve the Coverage problem for an iso-polygonal zone. Infact, it is sufficient to start from an arbitrary cut and proceed recursively into the obtained zones. If one arrives at rectangular zones containing each one seed the Coverage problem is solvable for the iso-polygonal zone, otherwise is not solvable.

5. The Coverage problem for arbitrary polygonal zones. For an arbitrary polygonal zone we have no definitive result. We have instead the following conjecture.

CONJECTURE 5.1. *The Coverage problem is NP-complete for an arbitrary iso-polygonal zone.*

To support the conjecture we show two formulations of the problem leading to NP-complete problems. But we have not yet been able to find a reduction.

We first have to introduce the concept of *base grid* of a set of seeds. A *base grid* of a set S of seeds is built by stretching the sides of the seeds until they encounter the side of the MBR of S. The result is a partition in rectangles of the MBR itself.

In figure 5.1 you can see four rectangles, shown in bold, and the base grid resulting from the stretching of their sides up to the perimeter of the MBR of the set of seeds.

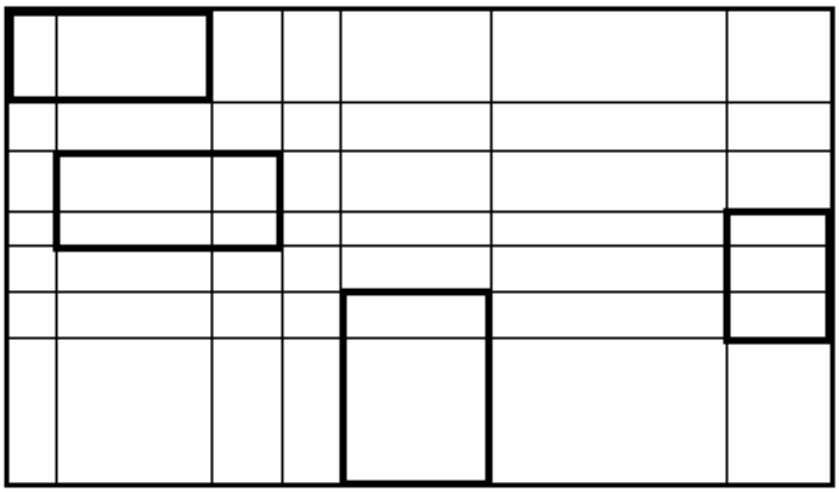

FIG. 5.1. *An example of base grid*

Then we introduce the concept of *feasible expansion of a seed* (with respect to the MBR of the set of seeds). A *feasible expasion* of a seed s of a set S, is any rectangle which contains s and is made up by the elements of the partition of the base grid of S.

The first formulation transform the Coverage problem in a *Max Weight Independent Set* problem. The idea is to derive from the Coverage problem a graph, with weights associated to nodes. Looking for an independent set of node with maximum weight and cheching that it is equal to some value depending on the base grid allows us to solve the problem.

More formally, given an instance of the Coverage problem we obtain an instance of the Max Weight Independent Set problem by applying the following steps:

1. From the set S of seeds derive the set $E_{S\sharp}$ of all feasible expansions of all seeds in S with respect to their base grid.

2. Build a node-weighted graph $G = (V, E, w)$ where $V = E_{S\sharp}$ and edge (x, y) exists if and only if the rectangle represented by x intersects the rectangle represented by y. The weigth $w(x)$ of node x is the number of rectangles of the partition of the base grid of S that are contained in x.

Now, we first find a set W of independent nodes in G with maximum weight. Then we check if this weigth is equal to the area of MBR of S measured in terms of the rectangles of the partition of the base grid of S.

If the weight is equal then W identifies a solution to the Coverage problem. If it is not the Coverage problem has no solution.

But unfortunately the Max Weight Independent Set is in general an NP-complete problem [2], and is also NP-complete when restricted to this kind of graphs [4, 5, 6], called *Boxicity-2* graphs since they derive from the intersection of rectangles [11].

Example. In figure 5.2 an example of such a formulation is shown. There are two seeds, named

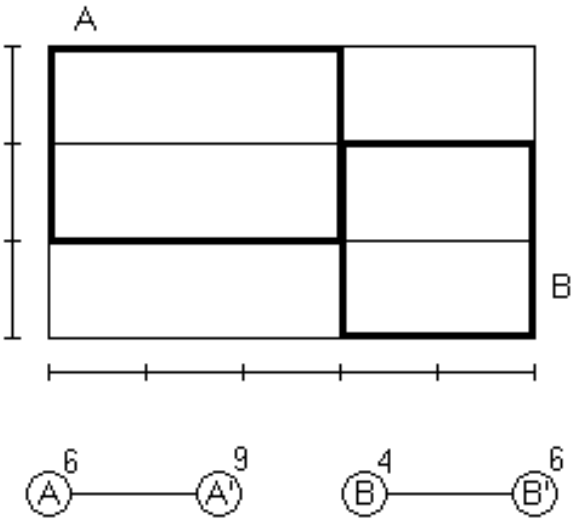

FIG. 5.2. *An example of the transformation to Max Weight Independent Set*

A and B and shown in bold. The base grid is shown with finer lines. Rectangle A has a weight of 6. For rectangle A only one expansion, using the lines defined by the base grid, is possible. It is indicated by A'. Expansion A' has a weight of 9. Rectangle B has a weight of 4. Also for rectangle B only one expansion is possible using the base grid. Such an expansion, named B', has a weight of 6. The resulting graph is shown below. The independent set in such a graph that has maximum weight is formed by A' and B'. Its total weight is 15 that is equal to the area of the base grid. Therefore the Coverage problem has solution. □

In the second formulation we transform the Coverage problem in *Exact Cover* on a bipartite graph. The idea is to cover nodes in the lower layer of the bipartite with nodes in the upper layer.

More formally, given an instance of the Coverage problem we obtain an instance of the Exact Cover problem by applying the following steps:

1. From the set S of seeds derive the set $E_{S\sharp}$ of all feasible expansions of all seeds in S with respect to their base grid.
2. Build a bipartite graph $G = (N, V, E)$ where $N = E_{S\sharp}$ and V is the set of elements of the partition of the base grid of S, and an edge (x, y) exists if and only if the feasible expansion represented by x contains the element of the partition represented by y.

Now we find a set W of nodes of N such that

$$(\forall v, w \in W \nexists u \in V \mid (v,u),(w,u) \in E) \bigwedge (\forall u \in V \exists v \in W \mid (u,v) \in E)$$

If such a set exists then it identifies a solution to the Coverage problem. If it does not exist then the Coverage problem has no solution.

Unfortunately, the Exact Cover is an NP-complete problem [2].

Example. In figure 5.3 an example of the formulation as an Exact Cover problem on bipartite graph is shown. The instance of the Coverage problem is the same as before with two seeds, A and

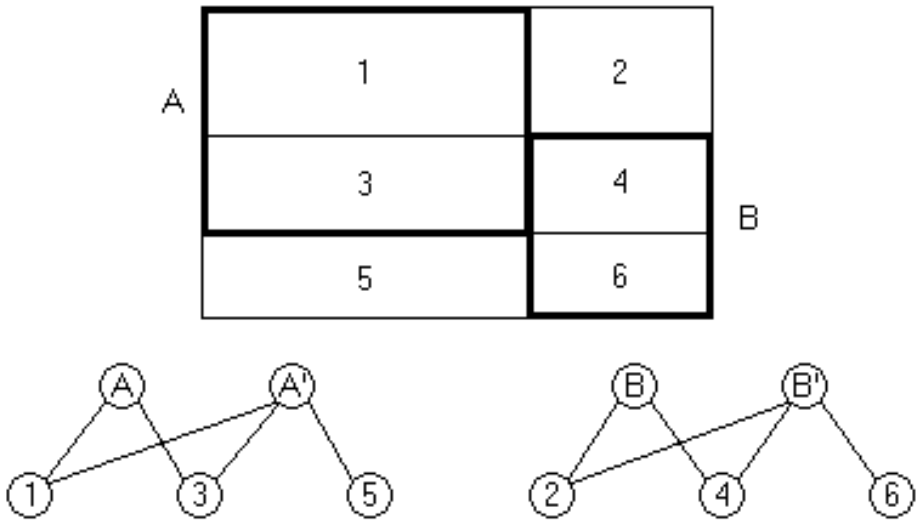

FIG. 5.3. *An example of the transformation to Exact Cover*

B. But now both the seeds and their expansions with respect to the base grid are represented as nodes in the upper layer of a bipartite, labeled A, A' B and B', while on the lower layer we represent all the elementary cells defined in the base grid. In the figure each cell is numbered from 1 to 6. By finding a set of nodes in the upper layer such that (1) no two nodes in the found set have a common adjacent in the lower layer and (2) all nodes in the lower layer are adjacent to some node in the found set, we solve the Exact Cover problem. In the example, by choosing nodes A and B' we have that node 5 is not adjacent to a node in the set, if we take node A and A' we violate the constraint of not having common adjacent nodes, while taking nodes A' and B' we solve the Exact Cover problem and hence the Coverage problem. □

We remark that while instances for both formulations can be built in polynomial time, once given an instance of the Coverage problem, we have *not* been able to find a general reduction from their instances to the Coverage problem. This means that we are not able, given an instance I of Max Weight Independent Set or of Exact Cover, to build a set I' of rectangles such that from the solution to the Coverage problem for I' we can derive a solution for I.

Acknowledgments. This research was partially supported by the European Union TMR project "Chorochronos".

REFERENCES

[1] C. Faloutsos, T. Sellis, N. Roussopoulus, Analysis of object oriented spatial access methods, *Proc. ACM SIGMOD Int. Conf. on the Management of Data*, (1987), 426–439.

[2] M. R. Garey, D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-completeness, *Freeman, San Francisco* (1979).

[3] A. Guttman, R-Trees: A dynamic index structure for spatial searching, *Proc. ACM SIGMOD Int. Conf. on the Management of Data*, Boston (1984), 45–57.

[4] J. Kratochvíl, String graphs II. Recognizing string graphs is NP-hard, *J. Combin. Theory Ser. B*, 52 (1991), 67–78.

[5] J. Kratochvíl, A Special planar satisfiability problem and consequence of its NP-completeness, *Discrete Appl. Math.*, 52 (1994), 233–252.

[6] J. Kratochvíl and J. Nešetřil, INDEPENDENT SET and CLIQUE problems in intersection defined classes of graphs, *Comment. Math. Univ. Carolin.*, 31 (1990), 85–93.

[7] B. Kröll, P. Widmayer, Distributing a search tree among a growing number of processor, in *ACM SIGMOD Int. Conf. on the Management of Data*, Minneapolis, MN, (1994), 265–276.

[8] W. Litwin, M.A. Neimat, D.A. Schneider, LH* – Linear hashing for distributed files, *ACM SIGMOD Int. Conf. on the Management of Data*, Washington, D.C., (1993).

[9] E. Nardelli, F.Barillari, M. Pepe, Distributed Searching of Multi-Dimensional Data: a Performance Evaluation Study, *Journal of Parallel and Distributed Computation*, 1998.

[10] E. Nardelli, F. Barillari, M. Pepe. Design issues in distributed searching of multi-dimensional data. In *3rd International Symposium on Programming and Systems (ISPS'97)*, San Jose, California, USA, April 1997. Lecture Notes in Artificial Intelligence, Springer-Verlag.

[11] F. S. Roberts, On the boxicity and cubicity of a graph, *Recent Progress in Combinatorics*, (ed. W. T. Tutte); Academic Press, New York (1969), 301–310.