ETH zürich

ETH Library

Singularities make spatial join scheduling hard

Report

Author(s):
Neyer, Gabriele; Widmayer, Peter

Publication date:
1997

Permanent link:
https://doi.org/10.3929/ethz-a-006652221

Rights / license:
In Copyright - Non-Commercial Use Permitted

Originally published in:
Technische Berichte / ETH Zürich, Departement Informatik 280

This page was generated automatically upon download from the ETH Zurich Research Collection.
For more information, please consult the Terms of use.

Singularities Make Spatial Join Scheduling Hard*

Gabriele Neyer and Peter Widmayer
{neyer, widmayer}@inf.ethz.ch
Institute for Theoretical Computer Science
8092 ETH Zurich, Switzerland

Abstract

It is long known that scheduling relational joins, where relations reside on disk, is $\mathcal{NP}$-hard in general. This result motivated a number of heuristics for scheduling spatial joins, where spatial data are distributed on disk blocks in spatial clusters. Since spatial clustering makes use of the rich structure of Euclidean space, the $\mathcal{NP}$-hardness of relational join scheduling does not imply $\mathcal{NP}$-hardness for spatial join scheduling. We show that indeed for a class of popular spatial clustering techniques used for spatial data structures, an optimum page fetch schedule can be computed in linear time. In full generality, we prove spatial join scheduling to be $\mathcal{NP}$-hard. Our proof makes extensive use of a particular singularity; this leaves room for the development of further efficient scheduling algorithms for interesting spatial data structures.

1 Introduction

In databases in general and spatial databases in particular, join processing is one of the most expensive operations. One of the reasons is that for large databases, main memory capacity is a bottleneck: Pages may need to be fetched from disk more than once in order to compute a join. Since disk access time usually is the dominant part of the join computation time, it pays to schedule disk accesses carefully. This is not always easy: For two relations on disk, where each page contains a set of tuples, and an equijoin over some attribute, it is an $\mathcal{NP}$-hard problem to find an optimal disk access schedule [MKY81].

*We acknowledge the support of the ESPRIT IV LTR Project No. 21957 (CGAL). An extended abstract is to appear in Proceedings of the Eighth Annual International Symposium on Algorithms and Computation, ISAAC'97, Singapore, December 17-19, 1997.

This $\mathcal{NP}$-hardness result has motivated a flurry of heuristics for scheduling spatial joins in spatial databases over the past decade; for a recent account, see e.g. [DP96]. In spatial databases, however, the data are not spread over disk pages arbitrarily: Virtually all spatial data structures partition the data space geometrically into cells, and they store those geometric objects on a page that lie within a cell. Therefore, the $\mathcal{NP}$-hardness of the relational join scheduling problem does not imply $\mathcal{NP}$-hardness for spatial join scheduling.

In this paper, we study the complexity of the spatial join scheduling problem. The spatial join is based on two (different) partitions of a rectangular data space into isothetic rectangles. Each rectangle is a cell that represents a page, where the geometric data stored in the page lie geometrically within the cell. The join operation computes some predicate based on spatial locality. For concreteness, let two sets of points from the same rectangular universe be maintained in the two cell partitions, and let the join predicate be the equality for points. To compute the join, any two pages whose cells overlap need to be in main memory at the same time. Let us now assume that memory capacity is severely limited: Only two pages can be kept in main memory at any given time. The spatial join scheduling problem asks for the smallest number of disk accesses and the corresponding disk access schedule such that any two pages whose cells intersect meet in main memory at some point in time.

We show that for an important class of cell partitions, the spatial join scheduling problem is not hard at all: It can be solved in linear time whenever no two rectangles, one from each partition, share some part of their boundary. This is likely to be true for data structures that compute the cell partition according to the data that are stored, such as k-d-trees in their disk variant (k-d-B-trees) or hB-trees. The scheduling algorithm is simple enough to be useful in practice. Therefore, in these cases a heuristic for a spatial join is a loss of both, efficiency and quality of the solution, as compared with the exact solution.

For the general case, where cell boundaries of both partitions are allowed to coincide, we show that the scheduling problem is indeed $\mathcal{NP}$-hard. This comes as a late (but first) justification for the search for heuristics over the past decade. We prove $\mathcal{NP}$-hardness by reducing the 3-SAT problem to our scheduling problem. The reduction is based on the proof in [GJT76], but has the extra complication of requesting that the "gadgets" be embedded into two rectangular space partitions.

Our complexity results give a first answer to the question of when one might use a heuristic to schedule a spatial join. But even data structures that produce partition lines according to some regular scheme that is fairly independent of the data, so that cell boundaries may equal each other, do exhibit a high degree of regularity, and therefore also in this case good join scheduling algorithms might exist.

More precisely, let R be a two-dimensional rectangle that represents the data space, and let a *rectangular partition* of R be a set of isothetic rectangles that partition

R. For two rectangular partitions A, B of a rectangle R, we call a sequence α of pairs $\alpha_i = (a_i, b_i)$, $a_i \in A$, $b_i \in B$, $i = 1, \ldots, n$, where each pair (a, b) with $a \in A$, $b \in B$, $a \cap b \neq \emptyset$ appears in the sequence, a *page fetch schedule* for A and B. Here and throughout the paper, the intersection of two rectangles is the closure of the intersection of their topological interiors. The *number of page fetches* in a schedule $\alpha = (\alpha_i)_{i=1,\ldots,n}$ is the number of changes in consecutive pairs $\alpha_i, \alpha_{i+1} + 2$; more precisely, it is defined as $\left|\{i, 1 \leq i \leq n-1 |\ a_i \neq a_{i+1}\}\right| + \left|\{i, 1 \leq i \leq n-1 |\ b_i \neq b_{i+1}\}\right| + 2$.

Let $G = (V, E)$ be an undirected graph. V is a set of *vertices*, E is a multiset of *edges*. A *path of length* $k \geq 0$ from vertex v to vertex w in G is a sequence of vertices $[v = x_0, x_1, \ldots, x_k = w]$ with $\{x_i, x_{i+1}\} \in E$ for $0 \leq i \leq k-1$. A *cycle* is a path from a vertex to itself, i.e., a path $[v, w]$ with $v = w$. A *Hamiltonian path* of G is a path of length $|V| - 1$ in which every vertex of V appears exactly once. A *Hamiltonian cycle* is a cycle in which every vertex appears exactly once, except the first vertex which has to be equal to the last vertex.

We will study the following problem:

Problem 1.1 (rectangular join scheduling)

Instance: *Two rectangular partitions A, B of a rectangle R.*

Problem: *Find a page fetch schedule realizing the minimum number of page fetches.*

In order to attack this problem, let $A \cap B$ denote the *join* of two rectangular partitions A, B, defined as $A \cap B = \{a \cap b | a \in A, b \in B, a \cap b \neq \emptyset\}$.

Proposition 1.1 *Let A, B be two rectangular partitions of R. Then $A \cap B$ is also a partition of R into rectangles.*

Proof: A and B consist of rectangles. Thus, every element in $A \cap B$ is the cut of a rectangle in A with a rectangle in B. Since the cut of two rectangles is a rectangle, it follows that $A \cap B$ is a partition of R into rectangles. □

We identify $A \cap B$ with a graph, the *rectangular overlay graph (ROG)* $G_{A \cap B}$, defined as follows:

Definition 1.1 (ROG) *Let R be a two dimensional rectangle, A, B two partitions of R. Let $A \cap B$ be the join of the two rectangular partitions. A ROG $G_{A \cap B}$ is a graph where each rectangle of $A \cap B$ is associated with a vertex; there is an edge between two different vertices (a, b) and (c, d) if $a = c$ or $b = d$, $a, c \in A$, $b, d \in B$.*

Figure 1: Rectangular partitions.

Figure 1 shows a rectangular partition, the join of two rectangular partitions and the corresponding ROG. Let n denote the number of rectangles in $A \cap B$, i.e. the number of pairs of rectangles from A and B that intersect. Our goal is to order the n pairs in such a way that the number of changes in consecutive pairs is minimum. Since two consecutive pairs change in at least one element, $n-1$ is the minimum possible number of changes. Furthermore, in a sequence with $n-1$ changes, any two consecutive vertices have a common edge. Since a change corresponds to a page fetch, we get:

Proposition 1.2 *There is a page fetch schedule with $n+1$ page fetches if and only if $G_{A\cap B}$ has a Hamiltonian path.*

Proof: "$\Longrightarrow$": Let α be a sequence of pairs (a_i, b_i) that solves the Problem 1.1 and has $n-1$ consecutive changes. Each pair in the sequence corresponds to a rectangle of $A \cap B$ and each rectangle of $A \cap B$ corresponds to a vertex in $G_{A\cap B}$. Since two consecutive pairs have one element in common they correspond to two pairs of rectangles $((a_i, b_i)$ and $(a_{i+1}, b_{i+1}))$ in $A \cap B$ that fulfill $a_i = a_{i+1}$ or $b_i = b_{i+1}$. With Definition 1.1, the corresponding vertices of $G_{A\cap B}$ are incident. Since every rectangle of $A \cap B$ corresponds to exactly one pair in α, it follows that the sequence defines a Hamiltonian path in $G_{A\cap B}$.

"$\Longleftarrow$": Let η be the sequence of vertices in a Hamiltonian path of a graph $G_{A\cap B}$. Each vertex corresponds to a rectangle of $A \cap B$ and therefore to a pair (a, b) of rectangles with $a \in A$ and $b \in B$. Two vertices u and v of $G_{A\cap B}$ are incident if the corresponding rectangles ($u = a_u \cap b_u$ and $v = a_v \cap b_v$) fulfill $a_u = a_v$ or $b_u = b_v$, $a_u, a_v \in A$, $b_u, b_v \in B$. Therefore, it follows that in the corresponding sequence of pairs of rectangles consecutive pairs differ in exactly one element. Thus, the number of consecutive changes is $n-1$. □

This leaves us with the Hamiltonian path problem for $G_{A\cap B}$:

Problem 1.2 (ROG Hamiltonian path)

Instance: *A ROG $G_{A\cap B}$.*

Problem: *Does $G_{A\cap B}$ contain a Hamiltonian path?*

In the next section, we argue that whenever the rectangles of A and B are in general position, the scheduling problem is easy. Section 3 shows that for unrestricted rectangle position, the problem is hard. An approximate solution for the scheduling problem can be obtained as follows: We assign each edge of $G_{A\cap B}$ a distance one and make the graph complete by adding edges of distance two. Now, it is easy to

see that the scheduling problem corresponds to finding a traveling salesman tour of minimum length in the complete graph. Papadimitriou and Yannakakis have given a polynomial time approximation algorithm with worst-case ratio $\frac{7}{6}$ for the special case of the traveling salesman problem for which all distances are either one or two [PY93].

2 Spatial Join Scheduling without Singularities is Easy

Let R be a rectangle. Let A, B be two rectangular partitions of R. We request that the rectangles *lie in general position* in the sense that no two rectangles $a \in A$ and $b \in B$ share a (part of a) common boundary (apart from the common boundary sides of R). More precisely:

Definition 2.1 ("common boundary")
Let $[x_1, x_2]$ be an interval on the x-axis. Let $[y_1, y_2]$ be an interval on the y-axis. Then $[x_1, x_2] \times [y_1, y_2]$ defines an axis parallel rectangle containing all points (x, y) with $x_1 \leq x \leq x_2$ and $y_1 \leq y \leq y_2$.
Let $a = [x_1^a, x_2^a] \times [y_1^a, y_2^a] \in A$ and $b = [x_1^b, x_2^b] \times [y_1^b, y_2^b] \in B$ be two rectangles. Two rectangles a and b share a common boundary if

$$(x_1^a, x_2^a) \cap (x_1^b, x_2^b) \neq \emptyset \text{ and } y_i^a = y_j^b, \text{ or } (y_1^a, y_2^a) \cap (y_1^b, y_2^b) \neq \emptyset \text{ and } x_i^a = x_j^b$$

for at least one pair, $i, j \in \{1, 2\}$, $i \neq j$.

Definition 2.2 (dual graph) *Let R be a partition of a rectangle into rectangles. The* dual graph *G_d of R is a graph where each rectangle of R is associated with a vertex. Two different vertices u and v are joined by an edge if the corresponding rectangles touch (in more than a single point).*

Kranakis [Kra97] pointed out to us that Czyzowicz et. al. [CRCS+94] showed the following theorem:

Theorem 2.1 ([CRCS+94]) *If a rectangle R is partitioned into n rectangles, then the dual graph of R has a Hamiltonian path.*

In order to apply Theorem 2.1 to a ROG $G_{A \cap B}$ we identify the dual graph of R with the ROG $G_{A \cap B}$:

Lemma 2.1 *Let A, B be two rectangular partitions that lie in general position of a rectangle R. Let $A \cap B$ be the join of A and B. Let the ROG be $G_{A \cap B} = (V, E)$ and the dual graph be $(A \cap B)_d = (V', E')$. Then $V = V'$ and $E \supseteq E'$.*

Proof: Clearly, the definition of the vertices in $G_{A\cap B}$ corresponds to the definition of the vertices in the dual graph $(A \cap B)_d$. Let $\{u, v\}$ be an edge in $(A \cap B)_d$. By the definition of the dual of a rectangular partition it follows that u and v correspond to two rectangles $u = a_u \cap b_u$ and $v = a_v \cap b_v$ of $A \cap B$ that share a common boundary side. Since the rectangular partitions A and B lie in general position, the corresponding rectangles u and v fulfill: $a_u = a_v$ or $b_u = b_v$, $a_u, a_v \in A$, $b_u, b_v \in B$. Thus the corresponding vertices of u and v in $G_{A\cap B}$ are connected by an edge. □

It has been shown [CRCS+94] that the rectangular dual graphs are internally 4-connected, hence they are Hamiltonian, by a theorem of Tutte [Tut56]. We can then use the algorithm in [CN89] to find a Hamiltonian path in $G_{A\cap B}$ in linear time.

3 Spatial Join Scheduling is $\mathcal{NP}$–Hard

We now consider the unrestricted case, i.e. two rectangles of different partitions may have a common boundary.

Problem 3.1 (ROG Hamiltonian circuit)

Instance: *A ROG $G_{A\cap B}$ for two rectangular partitions A, B of a universe R.*
Problem: *Does $G_{A\cap B}$ contain a Hamiltonian circuit?*

The remainder of this paper is devoted to a proof of the main result:

Theorem 3.1 *The ROG Hamiltonian circuit problem is $\mathcal{NP}$-complete.*

Clearly, the ROG Hamiltonian circuit problem is in $\mathcal{NP}$. Our construction of the $\mathcal{NP}$-hardness proof for the ROG Hamiltonian circuit problem is based on the reduction from 3-SAT in the $\mathcal{NP}$-hardness proof of the planar Hamiltonian circuit problem in [GJT76].

Problem 3.2 (3-SAT)

Instance: *A boolean formula in conjunctive normal form with exactly three literals per clause.*
Problem: *Is there a satisfying truth assignment for the formula?*

Garey, Johnson and Tarjan express the formula by combining *logic elements* like *"exclusive-or"* and *"three-input-or"* to a *logic graph*. In a second step, they design graph components that have the functionality of the logic elements, and then embed these components into a graph according to the logic graph.

3.1 Construction of the *logic graph*

For the sake of making the presentation self-contained, let us recall the construction of the *logic graph* as defined in [GJT76]. This section closely follows the presentation in [GJT76], and it is even taken literally in some places.
First, we introduce the logic *"exclusive-or"* graph.

Definition 3.1 (Logic "exclusive-or") *The "exclusive-or" graph is a subgraph that acts like two separate edges, one connecting vertices u and u' and the other connecting vertices v and v', with the constraint that for each graph G that contains this graph as a vertex induced subgraph, exactly one of these two edges must occur in any Hamiltonian circuit of G. In this case, we say that the edges $\{u, u'\}$ and $\{v, v'\}$ have been "connected" by an "exclusive-or".*

Definition 3.2 (Logic "three-input-or") *The "three-input-or" graph is a subgraph that acts like three separate edges, one connecting vertices u and u', one connecting vertices v and v', and the other one connecting vertices w and w', with the constraint that for each graph G that contains this graph as a vertex induced subgraph, at least one of these three edges must occur in any Hamiltonian circuit of G. In this case, we say that the edges $\{u, u'\}$, $\{v, v'\}$ and $\{w, w'\}$ have been "connected" by a "three-input-or".*

The logic graph consists of edges as usual and the logic components "exclusive-or" and "three-input-or", as follows:

Definition 3.3 (Logic Graph) *Let an instance of 3-SAT be given by a set V of n variables and a set C of m clauses over V. For each of the variables x_i, $1 \leq i \leq n$, we construct four vertices v_{i1}, v_{i2}, v_{i3} to v_{i4} and for each clause C_j, $1 \leq j \leq m$ we construct six vertices w_{j1}, w_{j2}, w_{j3}, w_{j4}, w_{j5} and w_{j6}. These vertices are connected by the following edges:*

1. *two copies each of $\{v_{i1}, v_{i2}\}$ and $\{v_{i3}, v_{i4}\}$, $1 \leq i \leq n$;*
2. *$\{v_{i2}, v_{i3}\}, 1 \leq i \leq n$;*
3. *$\{v_{i4}, v_{i+1,1}\}$, $1 \leq i \leq n-1$;*
4. *$\{v_{n4}, w_{m6}\}$;*
5. *$\{v_{11}, w_{11}\}$;*
6. *two copies of $\{w_{j1}, w_{j2}\}$, $\{w_{j3}, w_{j4}\}$ and $\{w_{j5}, w_{j6}\}$, $1 \leq j \leq m$;*

7. $\{w_{j2}, w_{j3}\}, \{w_{j4}, w_{j5}\}, 1 \leq j \leq m$;

8. $\{w_{j6}, w_{j+1,1}\}, 1 \leq j \leq m-1$;

For each i, we connect one copy of $\{v_{i1}, v_{i2}\}$ to one copy of $\{v_{i3}, v_{i4}\}$ with an "exclusive-or". For each j, we connect one copy each of $\{w_{j1}, w_{j2}\}$, $\{w_{j3}, w_{j4}\}$ and $\{w_{j5}, w_{j6}\}$ with a "three-input-or".

Now let us consider each literal p_{jk} in F. If $p_{jk} = x_i$, we use an "exclusive-or" to connect the copy of $\{w_{j,2k-1}, w_{j,2k}\}$ not connected to a "three-input-or" with the copy of $\{v_{i1}, v_{i2}\}$ which is not connected to $\{v_{i3}, v_{i4}\}$ with an "exclusive-or". If $p_{jk} = \bar{x}_i$, we use an "exclusive or" to connect that copy of $\{w_{j,2k-1}, w_{j,2k}\}$ with a copy of $\{v_{i3}, v_{i4}\}$ which is not connected to $\{v_{i1}, v_{i2}\}$ with an "exclusive-or".

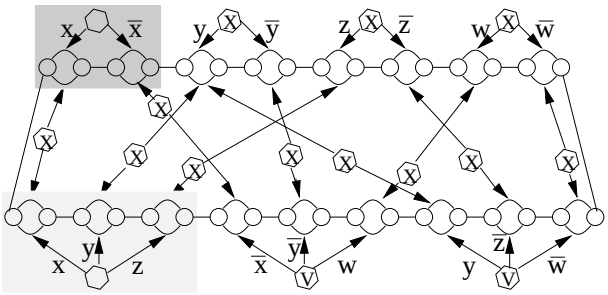

Figure 2: A logic graph.

See Figure 2 for a schematic of this construction for $F = (x \vee y \vee z) \wedge (\bar{x} \vee \bar{y} \vee w) \wedge (y \vee \bar{z} \vee \bar{w})$. A (X) symbolizes an "exclusive-or" relation and a (V) symbolizes a "three-input-or" relation. The dark shaded part is the logic graph for a variable and its negation, and the light shaded part is the logic graph for a clause. Garey, Johnson and Tarjan prove:

Theorem 3.2 ([GJT76]) *The* logic graph *constructed for a 3-SAT formula F as defined in Definition 3.3 has a Hamiltonian circuit if and only if F is satisfiable.*

Thus, the Hamiltonian circuit problem for these logic graphs is $\mathcal{NP}$-complete. This completes our presentation of the logic graph from [GJT76].

The idea of the proof condensed in 4 sentences is the following: We use the same transformation of a 3-SAT formula into a logic graph. In a second step, we transform the logic elements into ROG components which have the same functionality as the logic elements. The ROG components are then combined according to the logic graph. In order to embed crossing logic *"exclusive-or"* elements into a ROG, we solve a *channel routing problem in knock knee mode* and then transform the solution into a ROG which has a Hamiltonian circuit if and only if the corresponding 3-SAT formula is satisfiable.

3.2 Transformation into a ROG

In order to be able to combine the ROG components, we design each component to have a *standard length (SL)* or a multiple of SL. Furthermore, we unify the interfaces

of the components: For the left (right, top, bottom) front of a component we define interfaces LFI (RFI, TFI, BFI).

Figure 3 shows a TFI. Its mirror image on the horizontal axis is a BFI. A 90° (270°) rotation and swapping the dashed with the dotted lines gives a LFI (RFI). A dotted and dashed border line is defined as LFB (RFB, TFB, BFB). All interfaces have standard length (SL) or a multiple of SL and can be "plugged together".

Figure 3: TFI.

Each logic component consists of logic edges with a relation (e.g. *"exclusive-or"*). For each logic edge e of a logic component there exist two vertices u, v in the corresponding ROG component such that for any Hamiltonian circuit in the logic graph that contains edge e, there is a Hamiltonian circuit in the ROG that enters the component through u and leaves it through v. We say u, v *act like an edge* (build a logic edge). Any two vertices p, q of the ROG component that are connected to other components and do not build a logic edge have the following property: For any ROG, there is no Hamiltonian circuit that enters the component at p and leaves it at q.

Lemma 3.1 ("exclusive-or" ROG) *Figure 4 shows a ROG component that has the functionality of the logic "exclusive-or" graph.*

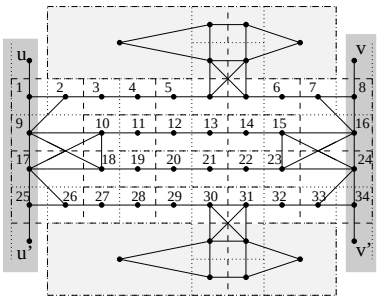

Figure 4: **"exclusive-or"**.

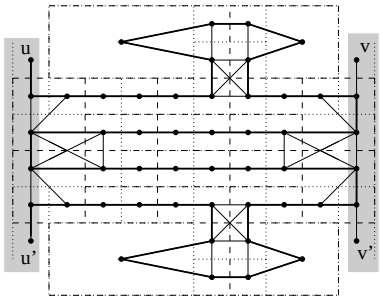

Figure 5: Local state.

More precisely, Figure 4 shows a horizontal "exclusive-or" ROG component tied to edge $\{u, u'\}, \{v, v'\}$. The dark gray shaded part shows the connection of the component with its surroundings. A 90° rotation and swapping the dashed with the dotted lines results in a vertical "exclusive-or". The proof of this theorem as well as the proofs of the following theorems concerning ROG components is based on the following observation: Consider two edges $(x, y), (y, z)$, where y has degree 2. Thus, any Hamiltonian circuit has to visit these edges in consecution. A connected vertex induced subgraph that is connected to other components by exactly two edges is called a *chain*. Any Hamiltonian circuit through a graph containing such a component as a vertex induced subgraph enters the component and visits all vertices before the component is left. E.g. in Figure 4 there are only four vertices u, u', v, v' where

a path can enter or leave the component, and four horizontal chains. The theorem is proven by observing that "any path from u to v' (v to u', u to v, u' to v') can only visit 1 or 3 chains" and "there is no Hamiltonian circuit that enters and leaves the component twice" combined with showing that there is a Hamiltonian path that enters and leaves the component through u, u' (or v, v'). For the sake of completeness we will spell out the complete proofs.

Proof: We show that the graph fulfills the ROG properties and has the functionality of the logic "exclusive-or" as defined in Definition 3.1.

1. The dotted lines divide the area of the component into rectangles; the dashed lines do the same. In the join of the dotted and dashed rectangular partition, each rectangle is identified with a vertex. Two vertices u and v are joined by an edge if the corresponding rectangles $u = a_u \cap b_u$ and $v = a_v \cap b_v$ from the original partitions satisfy: $a_u = a_v$ or $b_u = b_v$, $a_u, a_v \in A$, $b_u, b_v \in B$. Furthermore, the left (right, bottom, top) interface corresponds to LFI (RFI, BFB, TFB).

2. First, we observe that the vertices in each of the sets $\{3,4,5\}$, $\{11,12,13\}$, $\{19,20,21\}$, $\{27,28,29\}$ are chains. The component can be visited through 4 edges: $\{u,1\}, \{v,8\}, \{u',25\}$ and $\{v',34\}$. Assume there exists a Hamiltonian path that enters this component through u, visits all vertices of the component, and leaves it through v. Any Hamiltonian path has to visit chain $\{3,4,5\}$. It follows that the path starting with $\{u,1\}$ has to visit chain $\{3,4,5\}$ before any other chain is visited, since the only possibilities to visit this chain are given by path $(1,2,3,4,5,...)$ and path $(9,2,3,4,5,...)$, where vertex 1 only is adjacent to u, 2 and 9. One possibility to leave the chain is $(3,4,5,,...,6,7,8)$. Since 8 is the only vertex adjacent to v, there is no path visiting the other chains before leaving the component through v'. The other possibility to leave the chain is given by $(1,2,3,...,7,16)$. Since v, 7 and 16 are the only vertices that are adjacent to vertex 8, it also follows that there is no path visiting the other chains before leaving the component through v'. Thus, there exists no Hamiltonian path that enters this component through u and leaves it through v.

   Since the number of chains is even and any path from u to v' visits an odd number of chains, it follows that there is no Hamiltonian path that enters the component through u and leaves it through v'.

   From the symmetry of this component and the above reasoning it follows that there is no Hamiltonian path that uses all 4 edges $\{u,1\}, \{v,8\}, \{u',25\}$ and $\{v',34\}$.

   Figure 5 shows a Hamiltonian path that enters the component through u and leaves it through u'; we call this a *local state* of the component. A Hamiltonian

path that enters the component through v and leaves it through v' is symmetric to this path. Any Hamiltonian path through a graph containing this component as a vertex induced subgraph has only these two possibilities to enter and leave the component.

Thus it follows that the component acts like two separate edges, one connecting u and u' and the other connecting v and v', with the constraint that exactly one of these two logic edges must occur in any Hamiltonian circuit of G. Thus, the edges $\{u, u'\}$ and $\{v, v'\}$ have been "connected" by an "exclusive-or".

□

This ROG with the functionality of the logic "exclusive-or" graph can be modified geometrically for various purposes. Figure 6 shows a ROG component for a **variable and its negation**. Without the gray shaded part (the "double-edges"), the configuration corresponds to a "exclusive-or" where the pairs of vertices $\{u, u'\}$ and $\{v, v'\}$ are laid out along one straight boundary.

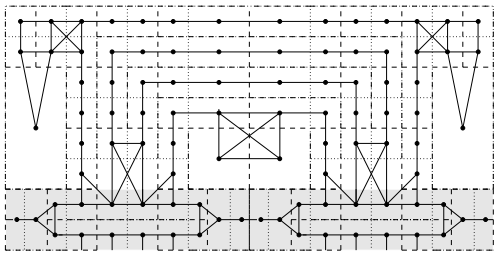

Figure 6: **Variable and its negation**.

Figure 7 shows the layout of the pairs of vertices $\{u, u'\}$ and $\{v, v'\}$ around the corner, which is called a "turn". More precisely, it is a **"left-top-turn"** ROG. A rotation of this configuration of 180° (90°+swapping the dashed lines with the dotted lines, 270°+ swapping) is a **"right-bottom-turn"**, (**"left-bottom-turn"**, **"right-top-turn"**).

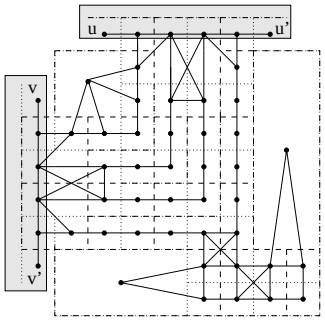

Figure 7: **"turn"**.

A logic *"double-edge"* consists of 2 different vertices that are combined by two edges. Any Hamiltonian path visits a vertex exactly once and from the construction of the logical graph it follows that exactly one edge of each "double-edge" occurs in a Hamiltonian path.

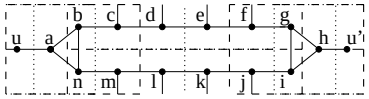

Figure 8: **"double-edge"**.

Lemma 3.2 ("double-edge" ROG) *In combination with an "exclusive-or" and "three-input-or" ROG component the ROG for a "double-edge" as shown in Figure 8 acts like the logic "double-edge".*

We say the *upper* (*lower*) logic edge of the logic "double-edge" occurs in a path, if in the "double-edge" ROG edges $\{b, c\}$ and $\{f, g\}$ ($\{i, j\}$ and $\{m, n\}$) occur in a path.

Proof: First, it is easy to see, that the dotted (resp. dashed) lines form a rectangular partition. Furthermore, the component has left (right) interface LFI (RFI). Since the graph fulfills the properties of Definition 1.1 it follows that this component is a ROG.

A "double-edge" exists in the logic graph only in combination of two "exclusive-ors" or one "exclusive-or" and one "three-input-or". In each local state of an "exclusive-or" ROG component (and "three-input-or" ROG component, as we will see later) in a Hamiltonian path either both edges $\{b, c\}$ and $\{f, g\}$ ($\{i, j\}$ and $\{m, n\}$ resp.) occur or none of them. Since u has degree 2, edge $\{u, a\}$ has to appear in any Hamiltonian circuit, thus exactly one of the edges $\{b, c\}$ and $\{m, n\}$ has to appear in any Hamiltonian circuit. The "double-edge" configuration is symmetric, such that this condition also holds for the other direction and other side. □

In the logical graph, the configurations for the variables are simply combined with edges. This functionality is provided by placing the ROG components of Figure 6 next to each other.

Lemma 3.3 ("three-input-or" ROG) *Figure 9 shows the "three-input-or" component tied to the (logic) "double-edge" $\{u, u'\}$, $\{v, v'\}$ and $\{w, w'\}$. Without the light gray shaded part, the component corresponds to the pure "three-input-or" ROG component.*

The entire Figure is a ROG configuration for a clause.

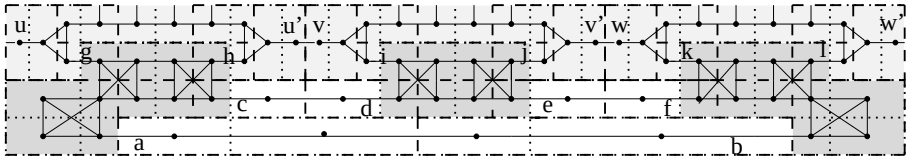

Figure 9: **"clause"**.

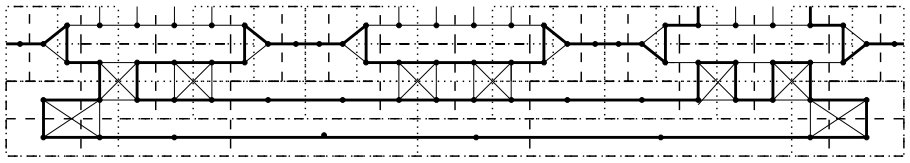

Figure 10: Local state.

Proof: First, we observe that the component fulfills the ROG properties and has right (left) interface RFI (LFI) three times in consecution. We subdivide the figure into three components (dark shaded in Figure 9). Each component is linked to other components by 4 edges. E.g. the leftmost dark shaded component is combined to the edges a, c, g and h. Observe that any Hamiltonian circuit in a graph G which contains this graph as a vertex induced subgraph must visit the edges a through f.

Observe that any dark shaded component has 4 edges connecting it to the outside, at least 2 of them have to be taken, and every Hamiltonian circuit visits a component an even number of times. Thus, the following condition holds: If edge g is visited in a Hamiltonian circuit, then edge h is also visited. Similarly, if edge i is visited, then also edge j is visited; if edge k is visited then also edge l is visited. Furthermore, any Hamiltonian circuit has to visit at least one of these pairs of edges, since these edges are the only edges that enter the component. Figure 10 shows a possible local state with the lower edge of "double-edge" $\{u, u'\}$ and $\{v, v'\}$ taken. Thus, this subgraph acts like three "double-edges", one connecting u and u', one connecting v and v' and the other connecting w and w', with the constraint that at least one of the three lower edges must occur in any Hamiltonian circuit of G. □

Since the ROG for the logical graph of a clause shown in Figure 2 consists of a "three-input-or" combined with three "double-edges", the entire Figure 9 builds a ROG for a clause. In the logical graph, the configurations for the clauses are combined with single edges. This functionality is provided by placing these components next to each other.

3.3 Embedding of "exclusive-or"s between literals and variables

We now have constructed the ROG components for the variables and the clauses. To conclude the construction we have to embed the "exclusive-or" connections between the literals of the clauses and the variables. Each literal of each clause is connected to exactly one variable. But a variable can be connected to more than one clause. We call this configuration a *"multiple-exclusive-or"* graph.

We combine a configuration of a variable x and its negation $\bar{x}$ to $k+l$ "exclusive-ors" that combine the variable with the corresponding literals, where k (l) is the number of occurrences of variable x ($\bar{x}$). We divide each ROG component for a variable and its negation (see Figure 6) at its vertical central line. Then, we stretch each part such that it has width k (l) times SL. The gray shaded part of Figure 11 shows the "multiple-double-edge"
ROG which acts like a "double-edge" and builds the connection between the stretched configuration of a variable and its negation and $2+1$ "exclusive-ors".

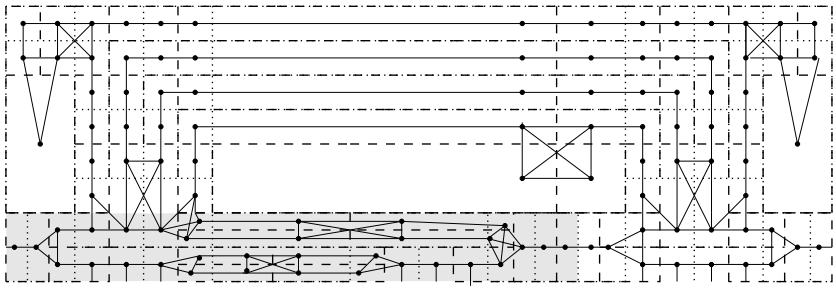

Figure 11: Variable with two adjacent "exclusive-ors".

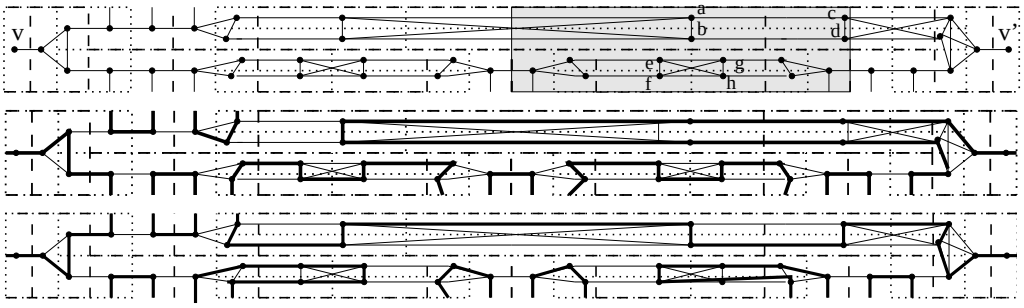

Figure 12: **"multiple-double-edge"**, local state with upper edge taken and lower edges taken.

Lemma 3.4 ("multiple-double-edge") *The ROG shown in Figure 12 acts like a "double-edge" where the gray shaded part can be copied ($k-2$ times) and juxtaposed horizontally. The ROG has to be combined with k "exclusive-or"s on its lower interface and one stretched "exclusive-or" on its upper interface.*

Proof: For the logic lower (upper) edge it is easy to see that the vertices a, b, c and d (f, g, h and i) can be visited independently from the fact whether the lower (upper) edge is taken or not. Since v and v' have degree 2 and there is no path from v to v' visiting both edges the component acts like a "double-edge". □

Observe that the horizontal length of the configuration for all variables is equal to the horizontal length of the configuration for all clauses (which is $3m$ times SL, where m is the number of clauses).

In our embedding of the "exclusive-or" lines we have to handle "crossing-exclusive-or" lines. The property which permits this is that "exclusive-or" lines can be connected in series, to cross over an edge of G, when that edge is required to occur in any Hamiltonian circuit.

Lemma 3.5 ("crossing-exclusive-or" ROG) *Figure 13 shows the ROG component of two "crossing-exclusive-or" lines in connection with the edges $\{u,u'\}$, $\{v,v'\}$,*

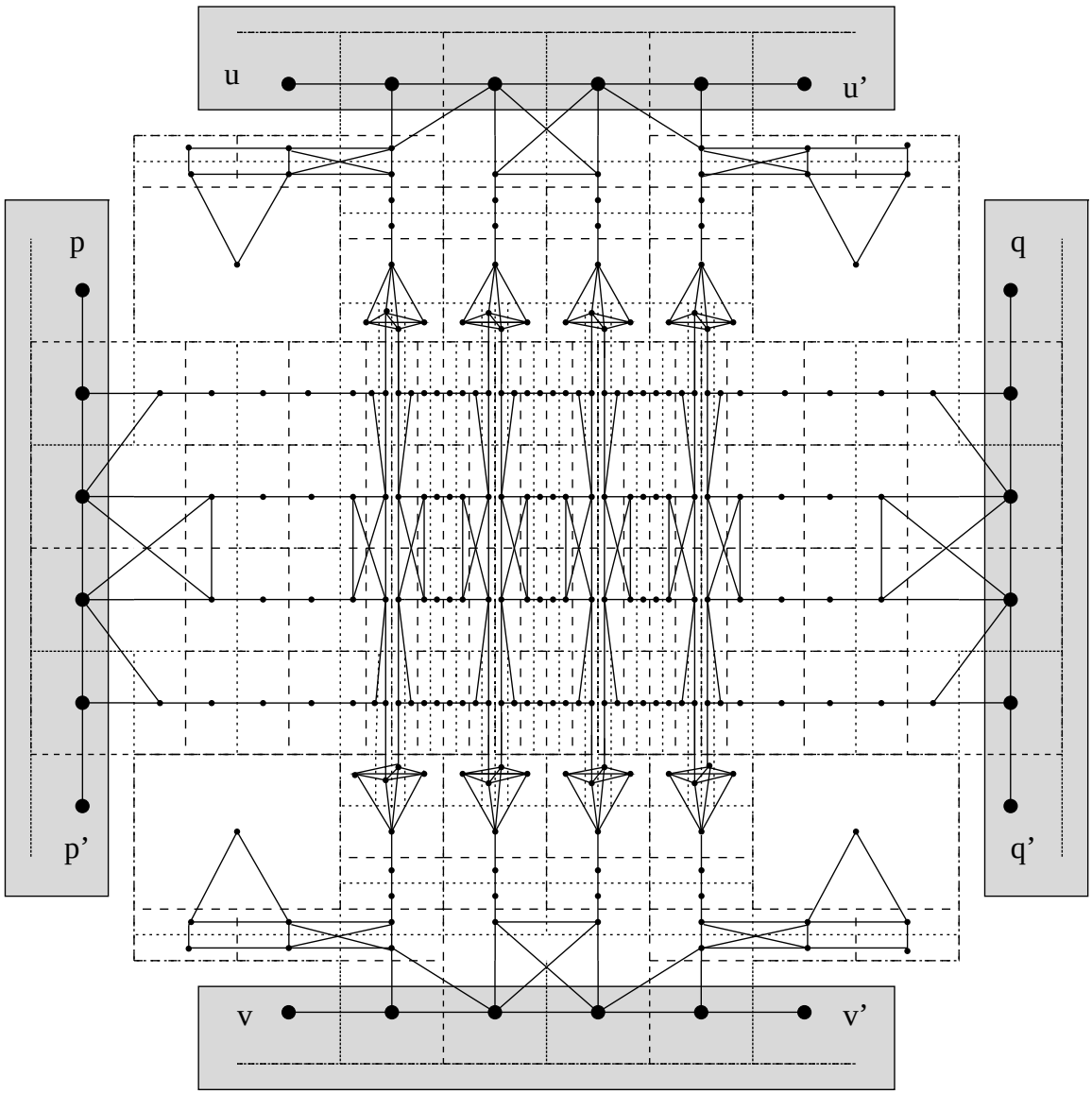

Figure 13: **"crossing-exclusive-or"**.

$\{p, p'\}$ and $\{q, q'\}$. This component acts like four separate edges with the property that in any Hamiltonian circuit in a graph G which contains this graph as a vertex induced subgraph, either $\{u, u'\}$ or $\{v, v'\}$ are connected by an edge and either $\{p, p'\}$ or $\{q, q'\}$ are connected by an edge.

Proof: First, observe that the ROG in Figure 13 (without the gray shaded parts) fulfills the ROG properties.

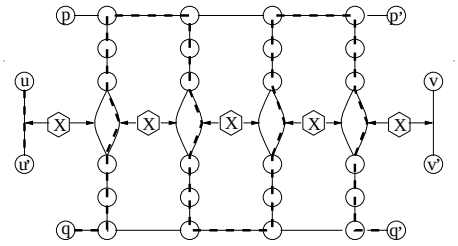

Figure 14: Logic "crossing-exclusive-or".

Figure 14 shows the logic components the configuration is composed of; the bold dashed lines show a possible local state. Between $\{p, p'\}$ and $\{q, q'\}$ we have 4 chains, as in the "exclusive-or" configuration, except that these chains are discontinued in the middle by "double-edge" configurations. These "double-edges" are combined with "exclusive-or" configurations. Similarly to all other "double-edges" in this construction, exactly one edge of these "double-edges" has to be in any Hamiltonian circuit. Thus, this graph acts like an "exclusive-or" for edge $\{p, p'\}$ and $\{q, q'\}$. Which one of these "double-edges" is in the Hamiltonian circuit is determined by the edge $\{u, u'\}$. Edge $\{u, u'\}$ is connected to the rest of the component by an "exclusive-or". Thus, if $\{u, u'\}$ is in a Hamiltonian circuit, the left edge of a "double-edge" can not be in the Hamiltonian circuit and all right edges have to. Therefore, it follows that edge $\{v, v'\}$ is not in the Hamiltonian circuit. Thus, it follows that edge $\{u, u'\}$ and $\{v, v'\}$ are joined by an "exclusive-or", independently from the edges $\{p, p'\}$ and $\{q, q'\}$. □

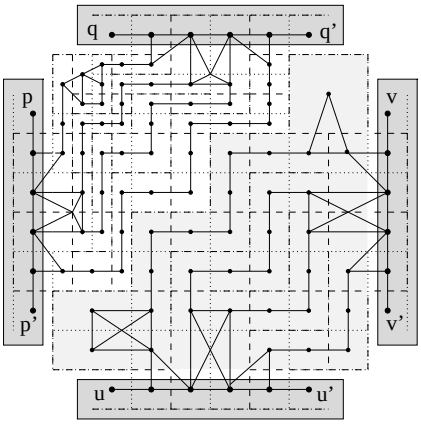

Figure 15: "knock-knee".

Another important ROG configuration for the embedding of the "exclusive-or" lines connecting literals with variables is the one shown in Figure 15. This component is called *"knock-knee"*, reminiscent of a certain wiring mode in VLSI design. The ROG component is tied to the edges $\{u, u'\}$, $\{v, v'\}$, $\{p, p'\}$ and $\{q, q'\}$. Without the dark shaded part the component is a **top-left-knock-knee**; its mirror image with respect to the horizontal axis is a **top-right-knock-knee**. An inspection of the figure reveals that this configuration consists of two independent "exclusive-or" configurations ($\{u, u'\}$ "exclusive-or" $\{v, v'\}$, $\{p, p'\}$ "exclusive-or" $\{q, q'\}$) providing interfaces LFI, RFI, TFI and BFI.

3.4 Routing problem

We now transform the problem of embedding the "exclusive-or" lines connecting literals with variables into a channel routing problem in knock-knee mode.

Let p_{ij} be the i-th literal in the j-th clause, $1 \leq i \leq 3$, $1 \leq j \leq m$. Let $x_1, \ldots, x_s$ be the variables; let k_r be the number of occurrences of literal x_r, and let $\bar{k}_l$ be the number of occurrences of literal $\bar{x}_r$. In VLSI design terminology, we create a channel with $3m$ bottom terminals p_{ij} in the order of their appearance in the clauses. Then, we create $3m$ top terminals $x_{1,1}, \ldots, x_{1,k_1}$, $\bar{x}_{1,1}, \ldots, \bar{x}_{1,\bar{k}_1}$, $\ldots$, $x_{n,1}, \ldots, x_{n,k_n}$, $\bar{x}_{n,1}, \ldots, \bar{x}_{n,\bar{k}_n}$, such that each literal occurs exactly as many times as needed in the clauses. Then, the following *top to bottom nets* are created: Terminal p_{ij} builds a top to bottom net with $x_{p,q}$ ($\bar{x}_{p,q}$), if p_{ij} is the q-th occurrence of x_p ($\bar{x}_p$) in the ordered set of clauses.

Now, the problem is to find a routing that connects the terminals of each net, where knock-knees are allowed. This can be done in $\mathcal{O}(m)$ time with an algorithm due to [MPS86].

For a net consisting of 2 terminals t_i and t_j, we say t_i is the *starting terminal*, if the column of t_i lies left of the column of t_j. In this case t_j is the terminating terminal. The width of the channel is $\mathcal{O}(m)$ and the height is bounded by the density, which in turn is $\mathcal{O}(m)$.

Algorithm 1 [MPS86] CHANNEL ROUTING $(columns, nets)$

```
forall columns from left to right do
  if there are only starting terminals in the current column
    if both starting terminals belong to one net
      combine the terminals by a straight line;
    else
      if there are two tracks occupied by one net
        close the net and use its tracks for the starting nets;
      else
        use a free track for every starting net;
  else
    close a terminating net and use its track for the other net (if any) which
      has a terminal in the current column;
```

Figure 16 shows the solved routing problem according to formula $F = (x \vee y \vee z) \wedge (\bar{x} \vee \bar{y} \vee w) \wedge (y \vee \bar{z} \vee \bar{w})$.

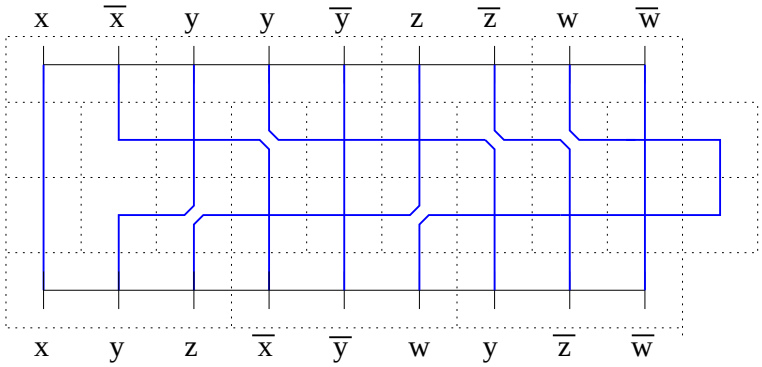

Figure 16: Solved routing problem.

3.5 Combination of all ROG components

From a solution of the knock-knee routing problem we get a ROG layout as follows: We draw a *dotted square* around each internal grid point, such that the square is axis parallel, the corners are equidistant from the grid points and the side length corresponds to the minimum distance of two grid-points. Figure 17 shows all possible dotted squares that can occur in the routing.

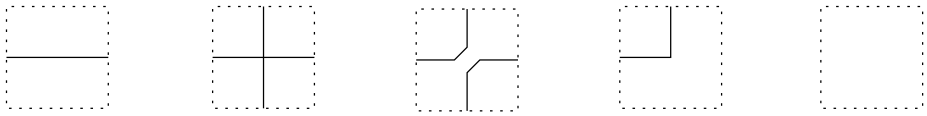

Figure 17: Routing configurations (from left to right): edge, cross, knock-knee, turn, empty.

In order to transform the solution into a ROG, the configurations for the variables are placed above the configuration of the clauses, with distance SL times the number of needed tracks in the solution of the routing problem. Thus, the space between the clause and variable ROG configurations can be subdivided into squares of size SL by SL. These squares are then filled with the corresponding ROGs.

Observe that an empty dotted square of a column only occurs in connection with a horizontal "exclusive-or", a "turn", or another empty dotted square. In this empty dotted square we fill in ROG component "empty" which is shown in Figure 18, stretch it to fill all adjacent empty dotted squares of the column, and combine it with the adjacent "exclusive-or" or "turn".

In Algorithm 2, the construction of the ROG is described more formally.

Up to now we left open how to embed the leftmost logic edge $\{v_{1,1}, w_{1,1}\}$ and the rightmost logic edge $\{v_{n,4}, w_{m,4}\}$ (see Figure 2). Observe from the logic graph that both edges must be used in any Hamiltonian circuit. Thus, we can embed the leftmost edge with the configuration in Figure 19, with the S part stretched according to the size for the embedding of the "exclusive-or" lines. Similarly, the rightmost edge can be embedded (see Figure 19). This edge can be used to fill the empty space behind the variables and the clauses according to the number of additionally used channels for the routing (stretch $S1$ and $S2$). The graph thus constructed fulfills the ROG properties and has a Hamiltonian circuit iff the initial 3-SAT formula has a truth assignment.

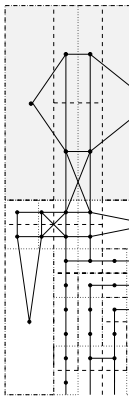

Figure 18: **"empty"** (shaded).

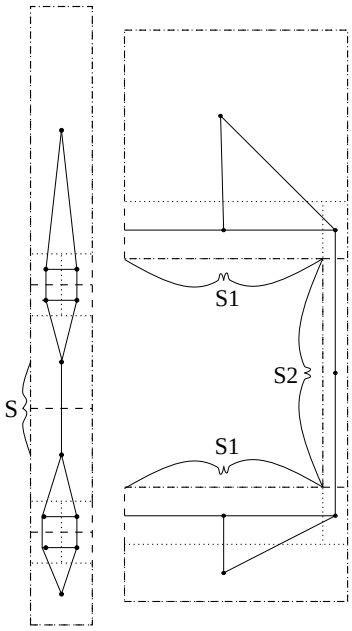

Figure 19: **"leftmost-"** and **"rightmost edge"**.

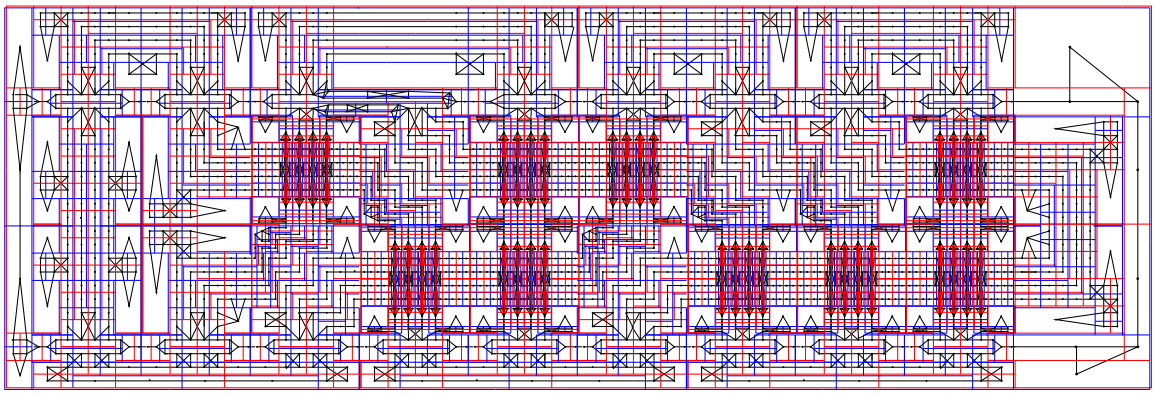

Figure 20: Complete ROG.

Figure 20 shows the complete ROG which has a Hamiltonian circuit if and only if the corresponding 3-SAT formula $F = (x \vee y \vee z) \wedge (\bar{x} \vee \bar{y} \vee w) \wedge (y \vee \bar{z} \vee \bar{w})$ is satisfiable.

Algorithm 2 TRANSFORMATION $(F, G_{A\cap B})$

Input: *An instance F of 3-SAT.*
Output: *A ROG $G_{A\cap B}$ which has a Hamiltonian circuit if and only if F is satisfiable.*
(1)*Solve the Routing Problem described in Section 3.4 for F.*
(2)*Arrange the ROG component of the logical graph for a* **clause** *m times in consecution (see Figure 9).*
(3)*For the i-th variable and its negation we take ROG component* **"exclusive-or"** *with logic edges in a horizontal line (see Figure 4). Let k (l) be the number of occurrences of literal x ($\bar{x}$). We divide this component at its vertical central line. Then, we stretch the left (right) part in the horizontal direction, such that it has width k (l) times SL (see Figure 11). We then combine the left (right) part with a* **"double-edge"** *(see Figure 8) in case $k = 1$ ($l = 1$) and a* **"multiple-double-edge"** *(see Figure 12), otherwise. After that, we put it next the ROG component of the $i-1$-th variable and its negation.*
(4)*Arrange the row of ROG components for the variables above the row of ROG components for the clauses, with distance SL times the number of used tracks in step 1.*
(5)*We divide the space between the row of the ROG components for the variables and the row of the ROG components for the clauses into squares SL×SL, which correspond to the dotted squares from step (1).*
(6)*for each dotted square do*
(7) *insert the corresponding ROG component at the corresponding place in the ROG:*
case edge: **"exclusive-or"** *(see Figure 4).*
case cross: **"crossing-exclusive-or"** *(see Figure 13).*
case knock-knee: **"knock-knee"** *(see Figure 15).*
case turn: **"turn"** *(see Figure 7).*
case empty: **"empty"**, *connect it to a "turn" or "horizontal-edge" ROG as described below (see Figure 18).*
(8)*Insert ROG component* **left edge** *at the left side of the ROG (see Figure 19). For this, stretch side S to the size of the number of used tracks times SL.*
(9)*Insert ROG component* **right edge** *at the right side of the ROG (see Figure 19). For this, stretch side $S1$ to the size of the number of additionally used channels times SL and stretch side $S2$ to the size of the number of used tracks times SL.*

Lemma 3.6 (polynomial reduction) *The reduction from 3-SAT to ROG Hamiltonian Circuit takes polynomial time.*

Proof: The ROG component for the clauses can be constructed in linear time, since the ROG component for each clause is equal to Figure 9 which has fixed size.

The ROG component for the variables can also be constructed in linear time, since each component is either equal to Figure 6 or equal to a stretched version of that figure, and the stretching factor is bounded by the number of clauses times SL.

The routing problem is solved in linear time, using at most $O(m^2)$ space, where m is the number of clauses. Thus, the transformation from the routing problem to the construction of the ROG for the whole embedding of the "exclusive-or" lines, takes at most quadratic time and needs $O(m^2)$ space, since the ROG components for the routing configurations (see Figure 17) have fixed size.

The ROG configurations of edge $\{v_{n4}, w_{m4}\}$ and edge $\{v_{11}, w_{11}\}$ have a fixed size and can be embedded in $O(1)$ time and space. Thus, it follows that the reduction is polynomial.

Theorem 3.3 *The ROG Hamiltonian Path Problem 1.2 is $\mathcal{NP}$-complete.*

Proof: Since the leftmost logic edge $\{v_{11}, w_{11}\}$ has to be in any Hamiltonian circuit, by deletion of this edge and closing the left side with a LFB, the $\mathcal{NP}$-completeness proof of the Hamiltonian path problem carries over. □

As a consequence, we get:

Theorem 3.4 (rectangular join scheduling) *Rectangular join scheduling is $\mathcal{NP}$-complete.*

Proof: An algorithm solving the rectangular join scheduling problem also solves the ROG Hamiltonian path problem.

Acknowledgement

We thank Evangelos Kranakis and Wolfram Schlickenrieder for stimulating discussions.

References

[CN89] N. Chiba and T. Nishizeki. The Hamiltonian Cycle Problem is Linear-Time Solvable for 4-Connected Planar Graphs. *Journal of Algorithms*, 10(2):187–211, June 1989.

[CRCS+94] J. Czyzowicz, E. Rivera-Campo, N. Santoro, J. Urrutia, and J. Zaks. Guarding Rectangular Art Galleries. *DAMATH: Discrete Applied Mathematics and Combinatorial Operations Research and Computer Science*, 50:149–157, 1994.

[DP96] D. J. DeWitt and J. M. Patel. Partition Based Spatial-Merge Join. In *Proceedings of the 1996 ACM SIGMOD, Montreal*, pages 259–270, June 1996.

[GJT76] M.R. Garey, D.S. Johnson, and R.E. Tarjan. The Planar Hamiltonian Circuit Problem is NP Complete. *SIAM Journal on Computing*, 5:704–714, 1976.

[Kra97] E. Kranakis. *Personal communication*, 1997.

[MKY81] T. H. Merrett, Y. Kambayashi, and H. Yasura. Scheduling of Page-Fetches in Join Operations. In *Proceedings of the 7th Conference on Very Large Databases, Morgan Kaufman*, September 1981.

[MPS86] K. Mehlhorn, F.P. Preparata, and M. Sarrafzadeh. Channel Routing in Knock-Knee Mode: Simplified Algorithms and Proofs. *Algorithmica 1*, pages 213–221, 1986.

[PY93] C. Papadimitriou and M. Yannakakis. The Traveling Salesman Problem with distances one and two. *Mathematics of Operations Research*, 18(1):1–11, 1993.

[Tut56] W. Tutte. A Theorem on Planar Graphs. *Trans. American Math. Soc.*, 82:99–116, 1956.