A Note On Computing Set Overlap Classes

Pierre Charbit[1]   Michel Habib[1]   Vincent Limouzy[1]
Fabien de Montgolfier[1]   Mathieu Raffinot[*1]   Michaël Rao[2]

[1] LIAFA, Univ. Paris Diderot - Paris 7, 75205 Paris Cedex 13, France.
[2] LIRMM, 161 rue Ada, 34392 Montpellier, France.

Abstract. Let $\mathcal{V}$ be a finite set of n elements and $\mathcal{F} = \{X_1, X_2, \ldots, X_m\}$ a family of m subsets of $\mathcal{V}$. Two sets X_i and X_j of $\mathcal{F}$ overlap if $X_i \cap X_j \neq \emptyset$, $X_j \setminus X_i \neq \emptyset$, and $X_i \setminus X_j \neq \emptyset$. Two sets $X, Y \in \mathcal{F}$ are in the same overlap class if there is a series $X = X_1, X_2, \ldots, X_k = Y$ of sets of $\mathcal{F}$ in which each $X_i X_{i+1}$ overlaps. In this note, we focus on efficiently identifying all overlap classes in $O(n + \sum_{i=1}^{m} |X_i|)$ time. We thus revisit the clever algorithm of Dahlhaus [2] of which we give a clear presentation and that we simplify to make it practical and implementable in its real worst case complexity. An useful variant of Dahlhaus's approach is also explained.

1   Introduction

Let $\mathcal{V}$ be a finite set of $n = |\mathcal{V}|$ elements and $\mathcal{F} = \{X_1, X_2, \ldots, X_m\}$ a family of m subsets of $\mathcal{V}$. Two sets X_i and X_j of $\mathcal{F}$ overlap if $X_i \cap X_j \neq \emptyset$, $X_i \setminus X_j \neq \emptyset$, and $X_j \setminus X_i \neq \emptyset$. We denote $|\mathcal{F}|$ as the sum of the sizes of all $X_i \in \mathcal{F}$. We define the overlap graph $OG(\mathcal{F}, E)$ as the graph with all X_i as vertices and $E = \{(i,j) \mid X_i \text{ overlaps } X_j\}, \forall\, 1 \leq i, j \leq m$. A connected component of this graph is called an *overlap class*.

In this note we focus on efficiently identifying all overlap classes of $OG(\mathcal{F}, E)$. This problem is a classical one in graph clustering related topics but it also appears frequently in many graph problems related to graph decomposition [2] or PQ-tree manipulation [3].

An efficient $O(n+|\mathcal{F}|)$ time algorithm has already been presented by Dahlhaus in [2]. The algorithm is very clever but uses an off-line Lowest Common Ancestor algorithm (LCA) as subroutine. From a theoretical point of view, off-line LCA queries have been proved to be solvable in constant time (after a linear time preprocessing) in a RAM model (accepting an additional constant time specific register operation) but also recently in a pointer machine model [1]. However, in practice, it is very difficult to implement these LCA algorithms in their real linear complexity. Another difficulty with Dahlhaus's algorithm comes from that its original presentation is difficult to follow. These two points motivated this note. Dahlhaus's algorithm is really clever and deserves a clear presentation, all the more so we show how to replace LCA queries by set partitioning, which

---

[*] Corresponding author. E-mail: `raffinot@liafa.jussieu.fr`

makes Dahlhaus's algorithm easily implementable in practice in its real complexity. We also provide a source code freely available in [4]. We eventually explain how to simply modify Dahlhaus's approach to efficiently compute a spanning tree of each connected component of the overlap graph. This simplifies a graph construction in [3].

2 Dahlhaus's algorithm

The overlap graph $OG(\mathcal{F}, E)$ might have $\Theta(m^2)$ edges, which can be quadratic in $O(|\mathcal{F}|)$. For instance, if $\mathcal{F} = \{\{x_1, x_2\}, \{x_1, x_3\}, \ldots, \{x_1, x_m\}\}$, $|E| = m(m - 1)/2 = \Theta(m^2)$.

The approach of Dahlhaus is quite surprising since that, instead of computing a subgraph of the overlap graph, Dahlhaus considers a second graph $D(\mathcal{F}, L)$ on the same vertex set but with different edges. This graph has however a strong property: its connected components are the same than that of $OG(\mathcal{F}, E)$, although that in the general case $D(\mathcal{F}, L)$ is not a subgraph of $OG(\mathcal{F}, E)$.

Let LF be the list of all $X \in \mathcal{F}$ sorted in decreasing size order. The ordering of sets of equal size is arbitrarily fixed. Given $X \in \mathcal{F}$, we denote $\mathrm{Max}(X)$ as the largest $Y \in \mathcal{F}$ taken in LF order such that $|Y| \geq |X|$ and Y overlaps X. Note that $\mathrm{Max}(X)$ might be undefined for some sets of $\mathcal{F}$. In this latter case, in order to simplify the presentation of some technical points, we write $\mathrm{Max}(X) = \emptyset$. Dahlhaus's algorithm is based on the following observation:

Lemma 1 ([2]). *Let $X \in \mathcal{F}$ such that $Max(X) \neq \emptyset$. Then for all $Y \in \mathcal{F}$ such that $Y \cap X \neq \emptyset$ and $|X| \leq |Y| \leq |Max(X)|$, Y overlaps X or $Max(X)$.*

Proof. If Y does not overlap X, as $|X| \leq |Y|$ and $Y \cap X \neq \emptyset$, $X \subseteq Y$. Thus $Y \cap \mathrm{Max}(X) \neq \emptyset$. Then, if Y does not overlap $\mathrm{Max}(X)$, then $\mathrm{Max}(X) \subseteq Y$. But in this case, as $|Y| \leq |\mathrm{Max}(X)|$, $Y = \mathrm{Max}(X)$ and overlaps X. Therefore Y overlaps X or $\mathrm{Max}(X)$. □

Let us assume that we already computed all $\mathrm{Max}(X)$. For each $v \in \mathcal{V}$ we compute the list $SL(v)$ of all sets $X \in \mathcal{F}$ to which v belongs. This list is sorted in increasing order of the sizes of the sets. Computing and sorting all lists for all $v \in \mathcal{V}$ can be done in $O(|\mathcal{F}|)$ time using a global bucket sort.

Dahlhaus's graph $D(\mathcal{F}, L)$ is built on those lists. Let X be a set containing v such that $\mathrm{Max}(X) \neq \emptyset$. Then for all consecutive pairs YW after X in $SL(v)$ (X included, i.e. Y can be instanced by X) and such that $|W| \leq |\mathrm{Max}(X)|$, create an edge (Y, W) in the graph D.

Lemma 2 ([2]). *The two graphs $D(\mathcal{F}, L)$ and $OG(\mathcal{F}, E)$ have the same connected components.*

Proof. ($\Rightarrow$) Let $Y, W \in \mathcal{F}$ such that $(Y, W) \in L$. By construction there exists v such that Y and W are consecutive on $SL(v)$ and there exists X that appears before YW on $SL(v)$ such that $\mathrm{Max}(X) \neq \emptyset$ and such that $|X| \leq |Y| \leq |W| \leq |\mathrm{Max}(X)|$. By lemma 1, Y and W overlap either X or $\mathrm{Max}(X)$. As X and

$\text{Max}(X)$ overlap, the sets X, Y, W, and $\text{Max}(X)$ belong to the same overlap class of $OG(\mathcal{F}, E)$. By extension, the vertices of any connected path in $D(\mathcal{F}, L)$ belong to the same overlap class of $OG(\mathcal{F}, E)$.

($\Leftarrow$) Let $A, B \in \mathcal{F}$ be two overlapping sets, *i.e.* $(A, B) \in E$. Let $v \in A \cap B$. Assume w.l.o.g. that $|A| \leq |B|$. Then $\text{Max}(A) \neq \emptyset$ and $|\text{Max}(A)| \geq |B|$. Therefore, in $SL(v)$, there exits a serie of consecutive pairs YW from A to B that are linked in $D(\mathcal{F}, L)$. In consequence, A and B are connected in $D(\mathcal{F}, L)$. $\square$

Notice that the order of equally sized sets in SL lists has no importance for the construction of a Dahlhaus's graph. Figure 1 shows an example of an overlap graph and a Dahlhaus's graph.
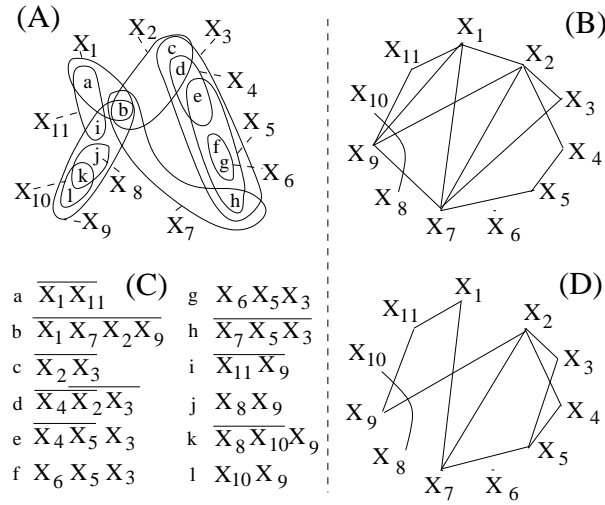

Fig. 1. Global example: (A) input family of 11 sets; (B) Overlap graph; (C) SL lists; (D) Dahlhaus's graph. On (C) intervals defined by $\text{Max}(X)$ are overlined. Notice that Dahlhaus's graph is not a subgraph of the Overlap graph.

Lemma 3 ([2]). *Given all $\text{Max}(X), X \in \mathcal{F}$, the graph $D(\mathcal{F}, L)$ can be built in $O(|\mathcal{F}|)$ time and its number of edges is less than or equal to $|\mathcal{F}|$.*

Proof. To build the graph $D(\mathcal{F}, L)$ from the SL lists, it suffices to go through each SL list from the smallest set to the largest and remenber at each step the largest $\text{Max}(X)$ already seen. If the size of the current set is smaller than or equal to this value, an edge is created between the last two sets considered.

Let us now consider the number of edges of $D(\mathcal{F}, L)$. As at most one edge is created for each set in a list SL, at most $|\mathcal{F}|$ edges are created after processing all lists. $\square$

Identifying the overlap classes of $OG(\mathcal{F}, E)$ can therefore be done by a simple Depth First Search on $D(\mathcal{F}, L)$ in $O(n+|\mathcal{F}|)$ time. It remains however to explain how to efficiently compute all $\text{Max}(X)$.

3 Computing all Max(X)

Let LF be the list of all $X \in \mathcal{F}$ sorted in decreasing size order. The order of sets of equal size is not important. We consider a boolean matrix BM of size $|\mathcal{F}| \times |V|$ such that each row represents a set $X \in \mathcal{F}$ in the order of LF, and each column an element $v \in V$. The value $\text{BM}[i, j]$ is 1 if and only if $v_j \in X_i$.

The first step of Dahlhaus's algorithm is to sort the columns of BM in lexicographical order, although that there is no detail in [2] on how to do it efficiently in $O(|\mathcal{F}|)$ time. We postpone all explanations concerning this step to section 3.2 and we consider below that all columns of BM are lexicographically sorted. Figure 2 shows the BM matrix for the set family of Figure 1.

	1 a	2 i	3 l	4 j	5 k	6 b	7 c	8 d	9 h	10 f	11 g	12 e	left	right
X_3	0	0	0	0	0	0	1	1	1	1	1	1	7	12
X_9		1	1	1	1	1	0	0	0	0	0	0	2	6
X_5		0	0	0	0	0	0	0	1	1	1	1	9	12
X_2	0					1	1	1	0	0	0	0	6	8
X_1	1					1	0	0				0	1	6
X_4	0					0		1		0	0	1	8	12
X_6						0		0	0	1	1	0	10	11
X_7				0	0	1			1	0	0		6	9
X_8			0	1	1	0			0				4	5
X_{10}	0	0	1	0	1	0							3	5
X_{11}	1	1	0	0	0	0	0	0	0	0	0	0	1	2

Fig. 2. Example continued: BM matrix which lines are sorted by decreasing sizes of $X \in \mathcal{F}$ and which columns are sorted in lexicographic order.

For each $X \in \mathcal{F}$ we denote $\text{left}(X)$ (resp. $\text{right}(X)$) the number of the column of BM containing the leftmost (resp. rightmost) 1 in the row of X.

Lemma 4. *Let $X, Y \in \mathcal{F}$ such that Y overlaps X and let r_Y be the row of Y in BM. Then there exists a row t higher than or equal to r_Y such that $BM[t, \text{left}(X)] = 0$ and $BM[t, \text{right}(X)] = 1$.*

Proof. As Y overlaps X, $|X| \geq 2$. Let r_X be the row corresponding to X in BM. Since Y overlaps X, there exist two indices $1 \leq i < j \leq |V|$ and a row r such that $\text{BM}[r_X, i] = \text{BM}[r_X, j] = 1$, such that one of the value of $\text{BM}[r, i]$ and $\text{BM}[r, j]$ is 1 and the other 0.

We consider the highest r that satisfies these conditions.

In a first step, if $\text{BM}[r, i] = 1$ and $\text{BM}[r, j] = 0$, then, as $i < j$ and as all columns has been sorted in increasing lexicographical order, there must exist a

row r' higher than r such that $BM[r', i] = 0$ and $BM[r', j] = 1$. We thus consider now w.l.o.g that $BM[r, i] = 0$ and $BM[r, j] = 1$.

Among all pairs of indices i and j such that $BM[r_X, i] = BM[r_X, j] = 1$ and that there exits r such that $BM[r, i] = 0$ and $BM[r, j] = 1$, let us consider one pair i' and j', $1 \leq i' < j' \leq |V|$, that is associated to the highest such r that we denote t.

We now prove that $BM[t, \text{left}(X)] = 0$ and $BM[t, \text{right}(X)] = 1$. If $BM[t, \text{left}(X)] = 1$, thus $i > \text{left}(X)$ and as $BM[t, i] = 0$ and that the columns are sorted in lexicographical order, there should exits an higher row r' such that $BM[r', \text{left}(X)] = 0$ and $BM[r', i] = 1$, which contradicts t to be the highest such row. Thus $BM[t, \text{left}(X)] = 0$. Symmetrically, the same argument holds to prove that $BM[t, \text{right}(X)] = 1$. $\square$

Lemma 5. *Let $X \in \mathcal{F}$. Then $Max(X) \neq \emptyset$ if and only if there exists a row t in BM such that $BM[t, \text{left}(X)] = 0$ and $BM[t, \text{right}(X)] = 1$ corresponding to a set $Y \in \mathcal{F}$ verifying $|Y| \geq |X|$.*

Proof. ($\Leftarrow$) If a set Y corresponds to a row t in BM such that $BM[t, \text{left}(X)] = 0$ and $BM[t, \text{right}(X)] = 1$, Y obviously overlaps X. As $|Y| \geq |X|$, $Max(X) \neq \emptyset$. ($\Rightarrow$) Let us assume that $Max(X) \neq \emptyset$ and let r_M be its row in BM. Then, by lemma 4, there exists a row t in BM such that $BM[t, \text{left}(X)] = 0$ and $BM[t, \text{right}(X)] = 1$ and such that t is higher than or equal to r_M. As $Max(X)$ verifies $|Max(X)| \geq |X|$, the set Y corresponding to r_M is also such that $|Y| \geq |X|$. $\square$

Lemma 6 ([2]). *Let $X \in \mathcal{F}$ such that $Max(X) \neq \emptyset$. Then $Max(X)$ corresponds to the highest row t in BM such that $BM[t, \text{left}(X)] = 0$ and $BM[t, \text{right}(X)] = 1$.*

[Notice that this row might be lower than the row corresponding to X. This is the case for X_8 and X_{10} since $Max(X_{10}) = X_8$ but also $Max(X_8) = X_{10}$. in our example.]

Proof. Let us assume that $Max(X) \neq \emptyset$ and let r_M be its row in BM. Then, by lemma 4, there exists a row t in BM such that $BM[t, \text{left}(X)] = 0$ and $BM[t, \text{right}(X)] = 1$ and such that t is higher than or equal to r_M. However, as such a row t corresponds to a set overlapping X and that $Max(X)$ is the largest of those sets in LF order, $t = r_M$. $\square$

For example, in Figure 2, $Max(X_1) = X_9$ since $\text{left}(X_1) = 1$, $\text{right}(X_1) = 6$ and X_9 (row 2) corresponds to the highest row with 0 on the first column and 1 on the 6th.

Dahlhaus's approach for computing all $Max(X)$ is to identify for each row r corresponding to X the highest row t such that $BM[t, \text{left}(X)] = 0$ and $BM[t, \text{right}(X)] = 1$. To do it efficiently, Dahlhaus reduces the problem to LCA computations. We explain this reduction in the next section 3.1. We then present another approach using class partitions in 3.2. This new approach is much simpler to implement than the LCA algorithm in its real linear worst case complexity. Moreover, it allows an easy computation of the lexicographical order of the columns.

3.1 Computing all Max(X) using LCA

Let us consider all intermediate columns between all pairs of columns in BM. In those columns, for each row, we place a point • between each motif 01 or 10. This is shown in Figure 3 (left). We link the highest point in each intermediate column, if it exist, in a Dahlhaus's tree (DT) the following way:

1. the root of the tree is the highest point. There can be only one root and there must be one root if one of the set $X \in \mathcal{F}$ differs from V. We assume this below;
2. we recurse the following process: each new point np in the tree (root included) splits the submatrice in two subparts according to the intermediate column it is placed in; the left (resp. right) child of np is the highest point in the left (right) part, if it exits. Note that the lexicographical order of the columns of BM insures that there can be at most one highest point in each part;
3. when a subpart does not contain any new point, a leaf per BM column in this subpart is created and attached as child to the point that created the subpart. If this point is placed to the left (resp. right) of this column, the child is a right (resp. left) child. Each leaf is numbered with the number of the corresponding column in BM.

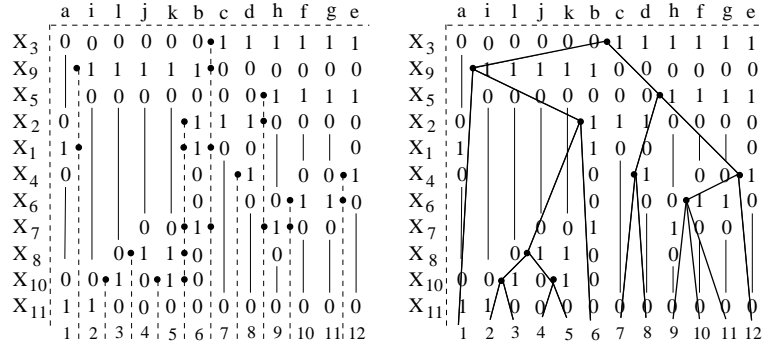An instance of such a tree is given in Figure 3 (right).

Fig. 3. Example continued: Dahlhaus's tree built over a BM matrix.

Proposition 1 ([2]). *Let $X \in \mathcal{F}$. Let $Y \in \mathcal{F}$ be the set corresponding to the row of $LCA(left(X), \, right(X))$ in BM. If $|Y| \geq |X|$, then $Y = Max(X)$. Otherwise $Max(X) = \emptyset$.*

Proof. Let r be the number of the row of $LCA(\text{left}(X), \text{right}(X))$ in BM and let l be the position of the column in BM that is just before the point representing $LCA(\text{left}(X), \text{right}(X))$.

First, $BM[r, l] = 0$ and $BM[r, l+1] = 1$. Suppose a contrario that $BM[r, l] = 1$ and $BM[r, l+1] = 0$. As all columns of BM are sorted in lexicographical order, there must exists an higher row r' such that $BM[r, l] = 0$ and $BM[r, l+1] = 1$. and thus a point in the intermediate column between l an $l+1$ higher than that in row r, which contradicts the construction of DT.

We now prove that $BM[r, \text{left}(X)] = 0$ and $BM[r, \text{right}(X)] = 1$. A contrario, suppose that $BM[r, \text{left}(X)] = 1$. Then, again, as the columns of BM are sorted in lexicographical order, there must exists an higher row r' such that $BM[r', \text{left}(X)] = 0$ and $BM[r', l] = 1$. This again contradicts the construction of DT. A similar argument holds for the right side.

We then prove that r is the highest row with this property. Assume a contrario that there exist an higher row r' such that $BM[r', \text{left}(X)] = 0$ and $BM[r', \text{right}(X)] = 1$. Then there would have been a split 01 somewhere in this row that would have separated $\text{left}(X)$ and $\text{right}(X)$. This implies that there would have been a node in DT in a row higher than or equal to r' that would have split $\text{left}(X)$ and $\text{right}(X)$, which contradicts r to be the number of the row of $LCA(\text{left}(X), \text{right}(X))$.

If $|Y| \geq |X|$, by Lemma 6 $\text{Max}(X) \neq \emptyset$ and the set Y that corresponds to r is such that $Y = \text{Max}(X)$.

If $|Y| < |X|$, since no row r' higher than r can verify $BM[r', \text{left}(X)] = 0$ and $BM[r', \text{right}(X)] = 1$, by Lemma 5 $\text{Max}(X) \neq \emptyset$. □

For example, X_9 corresponds to the row of $LCA(1, 2) = LCA(\text{left}(X_{11}), \text{right}(X_{11}))$. As $|X_9| \geq |X|$, $X_9 = \text{Max}(X_{11})$.

3.2 Computing all $\text{Max}(X)$ using set partitioning

We present below an alternative approach that permits avoiding LCA queries. Moreover, the lexicographical column order appears as a by-product.

We manipulate sorted partitions of V that we refine by each $X \in \mathcal{F}$ taken in LF order, that is, in decreasing order of their sizes. The initial partition is the whole set V and denoted P_V. For clarity, a set in a partition is called a *part*. In each partition the order of the parts is important, but the order of elements in a same part is not. Let $C = \{v_1, \ldots, v_k\}$ be a part in a partition. Refining C by $X \in \mathcal{F}$ consists in extracting all $v_i \in X$ in C and create a new part C'' with all those v_i. The remaining $v_i \notin X$ in C form a new part C' and C is replaced in the current partition by $C'C''$. If C only contains elements of X as well as if it contains none, C remains unchanged in the partition. Refining a partition P by a set $X \in \mathcal{F}$ consists in refining successively all parts in P. We note this refinement $P|_X$.

For example (continued), if $P = \{a\}\{i, j, k, l\}\{b\}\{c, d\}\{e, f, g, h\}$ and $X = X_4 = \{d, e\}$, $P|_X = \{a\}\{i, j, k, l\}\{b\}\{c\}\{d\}\{f, g, h\}\{e\}$.

Our approach requires 3 steps:

1. refine P_V by all $X \in \mathcal{F}$ taken in LF order;

2. then compute for each $X \in \mathcal{F}$ the values of left(X) and right(X) and sort all $X \in \mathcal{F}$ in a special order in regard with these values;
3. eventually refine P_V again by all $X \in \mathcal{F}$ taken in LF order but using the informations computed in step 2 to compute all Max(X).

We detail below each step.

Step 1 - Refining P_V. Let us consider the final partition we obtain after refining P_V by each $X \in \mathcal{F}$ taken in LF order. We note this partition P_f.

Lemma 7. *The elements of P_f are sorted accordingly to the lexicographical order of the columns of BM.*

Proof. Refining a partition consists in lexicographically sorting a row of BM touching only the 1 in the row but also keeping the global order already defined by the sets in the partition. Thus refining partitions from P_V in LF order consists in lexicographically ordering BM from the top row to the bottom. □

For example (continued), on the data in Figure 1, $P_f = \{a\}\{i\}\{l\}\{j\}\{k\}\{b\}$ $\{c\}\{d\}\{h\}\{f,g\}\{e\}$. Note that equal columns of BM are in the same part of P_f on which we fix an arbitrary order.

Step 2 - Computing all left(X) **and right**(X) **values.** We then compute all left(X) and right(X) values on P_f. This can be done easily in $O(|\mathcal{F}| + n)$ time by scanning each $X \in \mathcal{F}$ and keeping the minimum and maximum position of one of its element in P_f. We also compute a data structure AM that for each position $1 \leq i \leq |V|$ of P_f gives a list of all $X \in \mathcal{F}$ such that $i = $ right(X). All those lists are sorted in increasing order of left(X). The structure also allows an element $X \in \mathcal{F}$ to be removed from the list $AM[\text{right}(X)]$ in $O(1)$ time. This can be insured for instance using doubly linked list to implement each list, and the whole structure can easily be built in $O(n + m)$ time using bucket sorting.

Step 3 - Refining P_V **again and identifying all Max**(X). The main idea is the following. Assume that at a step of the refinement process in LF order we refine a part $C = \{v_1, \ldots, v_k\}$ of a partition P by $Y \in \mathcal{F}$ and that it results two non empty parts $C'C''$.

Lemma 8. *Let $X \in \mathcal{F}$ such that $|X| \leq |Y|$, left$(X) \in C'$ and right$(X) \in C''$. Then $Y = Max(X)$.*

[Note that if $|X| = |Y|$ then X could be before Y in LF order.]
Proof. Let r be the row corresponding to Y in BM. As left$(X) \in C'$ and right$(X) \in C''$, then $BM[r, \text{left}(X)] = 0$ and $BM[r, \text{right}(X)] = 1$, and Y obviously overlaps X. As $|X| \leq |Y|$, Max$(X) \neq \emptyset$. Moreover, the row r is the highest such that $BM[r, \text{left}(X)] = 0$ and $BM[r, \text{right}(X)] = 1$ since otherwise the elements of X would have been split by a set bigger that Y in the LF order. Thus, by Lemma 6, $Y = $ Max(X). □

The last phase of the algorithm thus consists in refining P_V again by all $Y \in \mathcal{F}$ taken in LF order. We first initialize all values $\text{Max}(X)$ to $\emptyset$. Each time a new split $C'C''$ appears (say between positions l and $l+1$), for all $v \in C''$ all lists $AM[v]$ are inspected the following way: let X be the top of one of those the list; while $\text{left}(X) \leq l$, X is popped off the list and $\text{Max}(X) \leftarrow Y$. After having refined with Y, if there is no more $Y' <_{LF} Y$ such that $|Y'| = |Y|$, all sets of the same size than Y are removed from the AM structure.

Lemma 9. *Our algorithm correctly computes in 3 steps all $\text{Max}(X)$, $X \in \mathcal{F}$.*

Proof. In step 1 the lexicographical order of the columns of BM is computed as a partition P_f (Lemma 7). In step 2 all values $\text{left}(X)$ and $\text{right}(X)$, $X \in \mathcal{F}$, are computed and the AM structure is built. In step 3, the correctness of the computation relies on the following observation: for each new partition P created after a refinement, all sets X remaining in AM are such that $\text{left}(X)$ and $\text{right}(X)$ belong to the same part in P. This is obviously true since otherwise they would have been split by a previous refinement and removed of AM. This has for consequence that after a split of a set C in $C'C''$ by a set Y, testing if $\text{left}(X) \in C''$ and $\text{right}(X) \in C''$ for all sets in AM is equivalent to test if $\text{right}(X) \in C''$ and $\text{left}(X) \leq l$, where l is the left position in P of the split between C and C''. Moreover, as each set taken in LF order and used for a possible refinement is removed of AM after having processed all the sets of the same size, when a set Y splits a part C in CC'', all sets in AM are such that $|X| \leq |Y|$. We thus fulfill all requirements of Lemma 8 and $Y = \text{Max}(X)$. Thus, if a value $\text{Max}(X)$ is assigned by our algorithm, it is assigned with the right one.

Now, suppose that a set X admits a set Y as $\text{Max}(X)$. It is guaranteed that a certain step of the algorithm Y has been assigned to $\text{Max}(X)$ since that by definition $|X| \leq |Y|$ which implies that X is still in AM when Y is processed and that by Lemma 6 $\text{left}(X) \notin Y$ and $\text{right}(X) \in Y$. The set Y has thus split a part C in a partition in $C'C''$ such that $\text{right}(X) > l$ and $\text{left}(X) \leq l$ where l is the left position in P of the split between C and C''. □

It remains to explain how a partition refinement can be efficiently implemented. We exploit the fact that element's order inside each part of a partition has no importance to obtain a simple implementation: a partition is represented as a table of size n in which each cell contains (a) an element of V and (b) a pointer to the part of the partition in which it is contained. A part is represented by a pair of its bounds on this table. Figure 4 shows such an implementation.
Refining a partition P by a set Y can be done in $O(|Y|)$ the following way. Let $[i, j]$ be the bounds of a part C such that $C \not\subset Y$ (easily testable). Let k be the number of elements of Y that belongs to the subtable $[i, j]$, $1 \leq k \leq j - i$. We swap elements in the subtable $[i, j]$ to place all k elements belonging to Y at the end of this subtable. We then adjust the bounds of C to $[i, j - k]$ and create a new set $[j - k + 1, j]$ on which the k elements of Y now point.

Theorem 1. *The identification of all $\text{Max}(X)$, $X \in \mathcal{F}$, using partition refinement can be done in $\Theta(n + |\mathcal{F}|)$ time.*
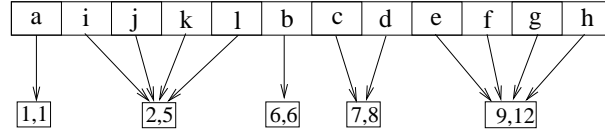
Fig. 4. Example continued: implementation of $P = \{a\}\{i,j,k,l\}\{b\}\{c,d\}\{e,f,g,h\}$.

Proof. By Lemma 9 the algorithm is correct. Steps 1 and 2 are $\Theta(|\mathcal{F}|+n)$ time. In step 3, the fact that all lists in AM are sorted in increasing order of $left()$ values insures that when a set Y splits a part C in $C'C''$, identifying and popping off all sets X such that $\text{left}(X) \in C$ and $\text{right}(X) \in C''$ can be done in $\Theta(|C| + K + 1)$ time, where K is the number of such sets. Removing a set out of AM is $O(1)$ time, thus the total of time managing AM is $\Theta(|\mathcal{F}| + n)$ time. □

The whole algorithm has been implemented in its real worst case time complexity and is freely available in [4].

4   Computing a subgraph of the overlap graph

In some applications like in [3] it is useful to get a spanning tree of all overlap classes of $OG(\mathcal{F}, E)$. The approach of [3] is to first compute Dahlhaus's graph and then compute spanning trees of the connected components of the overlap graph using a quite complex add-on. We thus explain in this section how to simply modify Dahlhaus's approach to compute a subgraph of the overlap graph instead of $D(\mathcal{F}, L)$. The size of the subgraph is linear but it has the same connected components than the overlap graph and it is thus easy from it to compute spanning trees of the overlap graph. The idea of the modification is the following.

Lemma 10. *Let $X, Y \in \mathcal{F}$ such that $X \cap Y \neq \emptyset$, such that $Max(X) \neq \emptyset$ and such that $|X| \leq |Y| \leq |Max(X)|$. Let r_Y be the row of Y in BM. If $BM[r_Y, left(X)] = 0$, Y overlaps X. Otherwise, (a) if $BM[r_Y, right(X)] = 0$, then Y overlaps X, and (b) if $BM[r_Y, right(X)] = 1$, then Y overlaps $Max(X)$.*

Proof. Let r_X be the row of X in BM, and r that of $\text{Max}(X)$. If $BM[r_Y, left(X)] = 0$, as $BM[r_X, left(X)] = 1$, that $X \cap Y \neq \emptyset$ and that $|X| \leq |Y|$, Y overlaps X. Assume now that $BM[r_Y, left(X)] = 1$. Case (a): if $BM[r_Y, right(X)] = 0$, then, as $BM[r_X, right(X)] = 1$, with the same arguments that above Y overlaps X. Case (b): if $BM[r_Y, right(X)] = 1$, then, as by Lemma 6, $BM[r, right(X)] = 1$ and $BM[r, left(X)] = 0$, and that $|Y| \leq |\text{Max}(X)|$, Y overlaps $\text{Max}(X)$. □

We modify the construction of Dahlhaus's graph the following way. We still consider intervals $X..YW..$ on $SL(v)$ lists such that $\text{Max}(X) \neq \emptyset$ and $|W| \leq |\text{Max}(X)|$, but instead of creating a chain $X - ..Y - W - ..$ in $D(\mathcal{F}, L)$, we create an edge $(X, \text{Max}(X))$ (if it does not already exists) and a list of quintuples $(\text{left}(X), \text{right}(X), X, Y, \text{Max}(X))$, $(\text{left}(X), \text{right}(X), X, W, \text{Max}(X))$, .. for
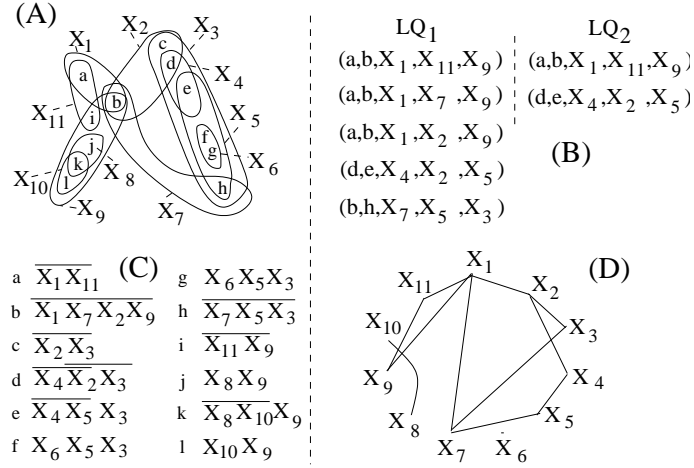
Fig. 5. Global example (continued): (A) input family of 11 sets; (B) LQ_1 and LQ_2 lists in which $\text{right}(X)$ and $\text{left}(X)$ heve been replaced by $Pf_{\text{right}(X)}$ and $Pf_{\text{left}(X)}$; (C) SL lists; (D) the resulting subgraph of the overlap graph.

all the elements in the interval distinct of X and $\text{Max}(X)$. All quintuples for all intervals are placed in the same list LQ_1. Note that if an element belongs to 2 intervals, a unique quintuple is formed with the rightest interval.

To apply Lemma 10, if suffices for each $(\text{left}(X), \text{right}(X), X, Y, \text{Max}(X))$ to test if Y belongs to $SL(Pf_{\text{left}(X)})$. If not, we then create an edge (X, Y). Otherwise, we test if Y belongs to $SL(Pf_{\text{right}(X)})$. If not, we also create an edge (X, Y). However, if it does, we create an edge $(Y, \text{Max}(X))$.

For complexity issues we need to perform those tests at a glance for all quintuples in LQ_1. We do it in two phases. In the first phase we search for all Y in $SL(Pf_{\text{left}(X)})$. If Y does not belong to $SL(Pf_{\text{left}(X)})$, we add the quintuplet $(\text{left}(X), \text{right}(X), X, Y, \text{Max}(X))$ to a second list LQ_2. In the second phase, if LQ_2 is not empty, for all $(\text{left}(X), \text{right}(X), X, Y, \text{Max}(X))$ in LQ_2 we search Y in $SL(Pf_{\text{right}(X)})$.

We assume below that all $SL(v)$ lists are sorted accordingly to the LF order instead of being simply sorted by increasing sizes. To efficiently compare LQ_1 with all $SL(v)$ lists it suffices to sort the list LQ_1 accordingly to $\text{left}(X)$ and then sort all quintuples with the same $\text{left}(X)$ value in the LF order of Y. This can be done in $O(n + |\mathcal{F}|)$ time using bucket sorting. The comparison of LQ_1 and the tables $SL()$ can then be done in $O(n + |\mathcal{F}|)$ time by comparing simultaneously $|V|$ sorted lists. The same approach holds for LQ_2. We thus have:

Theorem 2. *A subgraph of the overlap graph of $\mathcal{F}$ having the same connected components can be computed in $O(n + |\mathcal{F}|)$ time.*

Proof. Lemma 10 insures that the new graph is a subgraph of the overlap graph. To prove that they have the same connected component, it thus suffices to prove that if two sets A and B overlap, there exists a path connecting A and B in

the subgraph. The following observation is the base of the proof: let $X..Y..Z$ sorted by increasing size on the same $SL(v)$ and such that $|Y| \leq |\text{Max}(X)|$, $|\text{Max}(X)| \leq |\text{Max}(Y)|$ and $|Z| \leq |\text{Max}(Y)|$. Then there exists a path between all sets X, Y, Z in the new subgraph since by construction X and $\text{Max}(X)$ are connected, Y is connected to X or $\text{Max}(X)$, Y is connected to $\text{Max}(Y)$ and eventually Z is connected to Y or $\text{Max}(Y)$.

Now let $v \in A \cap B$. Assume w.l.o.g. that $|A| \leq |B|$. Then $\text{Max}(A) \neq \emptyset$ and $|\text{Max}(A)| \geq |B|$. Therefore, in $SL(v)$, there exits a series (potentially empty) of k sets $A..Y_1..Y_2..Y_k..B$ such that $|B| \leq |\text{Max}(Y_k)|$, $|Y_k| \leq |\text{Max}(Y_{k-1})|$, and $|Y_1| \leq |\text{Max}(A)|$. By induction on the series using the previous observation there exits a path from A to B in the subgraph.

The subgraph can obviouly been built in $O(n + |\mathcal{F}|)$ time since all steps can be done in this time. $\square$

An example (continued) of the resulting subgraph is shown in Figure 5.

References

1. A. L. Buchsbaum, H. Kaplan, A. Rogers, and J. R. Westbrook. Linear-time pointer-machine algorithms for least common ancestors, mst verification, and dominators. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing (STOC)*, pages 279–288. ACM Press, 1998.
2. E. Dahlhaus. Parallel algorithms for hierarchical clustering and applications to split decomposition and parity graph recognition. *J. Algorithms*, 36(2):205–240, 2000.
3. R. M. McConnell. A certifying algorithm for the consecutive-ones property. In *SODA*, pages 768–777, 2004.
4. M. Rao. Set overlap classes computation, source code. 2007. Freely available at `http://www.liafa.jussieu.fr/~raffinot/overlap.html`.