Enumerating Cliques of Hypergraphs

Marie Pelleau*, Laurent Simon†, Jean-Charles Régin*
*Université Côte d'Azur, CNRS, I3S, France
Email: firstname.lastname@univ-cotedazur.fr
†Bordeaux INP, Université de Bordeaux, CNRS, LaBRI, France
Email: lsimon@labri.fr

***Abstract*—The maximal clique enumeration (MCE) problem consists of computing and listing all maximal cliques in a finite graph. The problem is NP-complete since it subsumes the classic version of the NP-complete clique decision problem. The most well-known algorithm to solve this problem has been proposed by `Bron & Kerbosch` (`BK`).**

Generalizing this algorithm to hypergraphs presents significant challenges and admits multiple solutions. The primary difficulty stems from the fundamental difference in structure: while in standard graphs a vertex has a bounded number of incident edges, in hypergraphs a vertex can participate in an exponential number of hyperedges. Specifically, in a hypergraph with n vertices where hyperedges have arity r, a single vertex may belong to up to $\binom{n-1}{r-1}$ hyperedges. This structural complexity renders the direct generalization of `BK` computationally inefficient.

In this article, we propose three new approaches to solve the MHE problem. First, a relaxation method based on the relaxation of a hypergraph in a graph. Then, a method based on an efficiently computation of the hypercliques containing a set of vertices. Finally, a method combining the previous ones. We conducted an experimental study on a set of benchmarks, showing one or two orders of magnitude performance improvements of our approach with regards to the direct generalization of `BK` to hypergraphs.

***Index Terms*—Hypergraph, Clique Enumeration, Bron & Kerbosch, Experiments**

I. Introduction

In a wide variety of applications such as social network analysis [1], web mining [2], aligning 3D protein sequences [3], detection of protein-protein interaction [4] and entity resolution [5], the detection of *complete subgraphs*, (*i.e.* clique) is an essential component. The problem of determining whether or not a graph has a clique of a size k, called k-CLIQUE is a well-known NP-complete decision problem [6].

The *maximal clique enumeration* (MCE) problem (*i.e.* to compute and to list all the cliques that cannot be augmented by adding additional vertices) subsumes the k-CLIQUE problem and is therefore NP-hard. Hence, a great deal of effort has been spent on building efficient algorithms [7]–[11] to solve this problem. Most of these algorithms are based on the well-known `Bron & Kerbosch` (`BK`) algorithm [7]. In practice, `BK` has been reported as being faster than its alternatives [12], [13]. It uses a backtracking technique based on the vertices neighbourhood to explore the search space. Its efficiency also comes from the fact that it maintains a set of vertices to avoid reporting duplicate and non-maximal cliques.

There is however less research (and practical solutions) on the *maximal hyperclique enumeration* (MHE) problem. This is all the more unfortunate as there are a large number of problems for which this problem cannot be solved by a simple reduction to the MCE problem, such as the detection of cardinality constraints in Boolean formulas [14] or the enumeration of inclusion dependencies (INDs) within and across databases [15], for example. In these applications, for a given r-hypergraph (*i.e.* the arity of each edge is r), a hyperclique is a complete hypersubgraph in which every r-subset of the vertices represents an edge. As stated in [16], it is usually assumed that this problem cannot be solved efficiently in practice. Indeed, generalizing efficient algorithms for graphs to hypergraphs is more complex than it seems and the `BK` algorithm is no exception. This lack of studies is in contrast with the central importance of this problem to many areas in Artificial Intelligence, including constraint satisfaction, knowledge graph analysis, logic-based learning, and multi-relational data mining. Hypergraphs offer a natural and expressive way to model multi-entity relations, which frequently arise in AI applications such as, again, entity resolution, relational reasoning, and complex pattern discovery.

In both cases (graphs and hypergraphs), given a clique $\mathcal{C}$, determining if a vertex x can extend $\mathcal{C}$ is equivalent to checking if x appears with the vertices of $\mathcal{C}$ in a certain number of edges. The main difficulty is that in the case of graphs, this can easily be determined by using only the neighbourhood of x whereas in the case of hypergraphs, the neighbourhood is a necessary condition, but it is not sufficient. Assuming that all the edges have an arity r, it must be ensured that all subsets of vertices in $\mathcal{C}$ of cardinality $(r-1)$ appear with x in an edge. In terms of complexity, the number of mandatory edges to extend the clique is $\binom{n}{r-1}$ where n is the number of vertices in $\mathcal{C}$ and r is the arity of the edges. For graphs (*i.e.* $r=2$), the number of edges is n, while for hypergraphs, with just $r=3$, the number of edges can already be large $\left(\frac{(n-1)n}{2}\right)$.

In this paper, we propose different approaches to solve the MHE problem. First, we propose a relaxation using the clique-expansion conversion [17] to transform a hypergraph into a graph. This conversion replaces each hyperedge with edges forming a clique among the vertices of the hyperedge. This method is a relaxation as it may return bigger cliques than the ones in the hypergraph, but if it returns only the hyperedges (which are the most basic cliques in a hypergraph), then the original hypergraph does not have a larger clique. We then propose a generalization of the `BK` algorithm. Because it is

not sufficient to use the neighbourhood of each vertex to detect a clique, we define the notion of clique neighbourhood, that checks for the $\binom{n}{r-1}$ mandatory edges to expand the clique. We define an incremental way to compute the clique neighbourhood, just by checking the last added vertex to the hyperclique. Finally, we introduce two hybrid versions combining the generalization of the BK algorithm and the relaxation to improve the overall efficiency in practice.

To conclude we experimented the different approaches and variants of this article. The results show that our algorithms improve the performances of a basic approach by a factor of 10 or even 100 on some series of problems. A GitHub repository, allowing to replay most of the experimental part is available[1] and can be easily used as an external library to enumerate maximal cliques in hypergraphs since the source code is open-source.

II. Preliminaries

We reproduce the definitions from Claude Berge [18].

Definition 1 (Hypergraph)**.** *Let $\mathcal{X} = \{x_1, \ldots, x_n\}$ be a finite set. A* hypergraph *on $\mathcal{X}$ is a family $H = (E_1, E_2, \ldots, E_m)$ of subsets of $\mathcal{X}$ such that:*

$$E_i \neq \emptyset \qquad (i = 1, 2, \ldots, m) \tag{1}$$

$$\bigcup_{i=1}^{m} E_i = \mathcal{X} \tag{2}$$

The elements $\{x_1, \ldots, x_n\}$ of $\mathcal{X}$ are called vertices*, and the sets $\{E_1, E_2, \ldots, E_m\}$ are the* edges *of the hypergraph (or* hyperedges*). We will denote by $X(H) = \mathcal{X}$ the set of vertices of a hypergraph.*

Definition 2 (Simple Hypergraph)**.** *A hypergraph is said to be* simple *if no edge contains another that is: $E_i \subseteq E_j \implies i = j$.*

Definition 3 (Order and Rank)**.** *The* order *of H, denoted by $n(H)$, is the number of vertices. The number of edges is denoted by $m(H)$. The* rank *is $r(H) = \max_{1 \leq j \leq m} |E_j|$. The* anti-rank *is $s(H) = \min_{1 \leq j \leq m} |E_j|$.*

Definition 4 (Uniform Hypergraph)**.** *H is a* uniform *hypergraph if $r(H) = s(H)$. A simple uniform hypergraph of rank r is also called r-uniform or r-*hypergraph*.*

Remark 1. *A 2-hypergraph is a graph.*

Definition 5 (Neighbourhood)**.** *The* neighbourhood *$N(x)$ of a vertex x is the set of vertices that appear with x in at least one edge: $N(x) = \{u \in \mathcal{X} \mid \exists E_i \in H \text{ s.t. } \{u, x\} \subseteq E_i\}$.*

Definition 6 (Hypersubgraph [19])**.** *Let $\mathcal{Y}$ be a subset of $\mathcal{X}$. The* hypersubgraph *of H induced by $\mathcal{Y}$, denoted by $H_{\mathcal{Y}}$, is defined as the hypergraph having for vertex set $\mathcal{Y}$ and edge set $E_{\mathcal{Y}} = \{E_i | E_i \subseteq \mathcal{Y}\}$.*

[1] https://github.com/mpelleau/ICTAI-2025

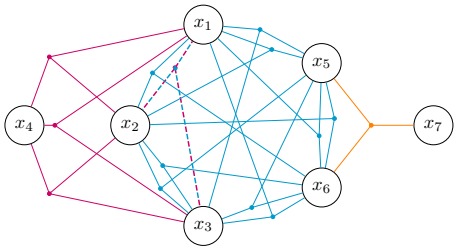

Fig. 1: Hypergraph

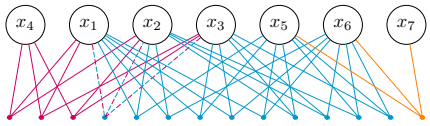

Fig. 2: An illustration of a 3-hypergraph with 7 vertices and 3 maximal hypercliques. Bipartite graph-based representation

Definition 7 (r-Uniform Complete Hypergraph)**.** *The r-*uniform complete hypergraph *or r-*complete hypergraph*, denoted by K_n^r, is the hypergraph containing n vertices in which every r-subset of the vertices represents an edge. It is easily observed that the number of edges in K_n^r is $|K_n^r| = \binom{n}{r} = \frac{n!}{r!(n-r)!}$, and that each vertex x in $X(K_n^r)$ has a degree $d(x) = \binom{n-1}{r-1}$.*

Because we are only considering r-hypergraphs, we propose the following definition to improve readability:

Definition 8 (r-Clique)**.** *In an r-hypergraph H, an r-*clique *$\mathcal{C}^r$ of order n is a hypersubgraph of H isomorphic to K_n^r.*

Remark 2. *In the following, we call* clique *or* hyperclique *$\mathcal{C}$ a complete hypersubgraph of an r-hypergraph. By abuse of notation, $\mathcal{C}$ corresponds to the set of vertices that induces the complete hypersubgraph.*

The following example illustrates the enumeration of maximal hypercliques:

Example 1. *The 3-hypergraph in Fig. 2 has one clique of order 5 ($\{x_1, x_2, x_3, x_5, x_6\}$, in blue), 6 cliques of order 4 ($\{x_1, x_2, x_3, x_4\}$ in pink, and 5 included in the clique of order 5) and 25 cliques of order 3 ($\{x_5, x_6, x_7\}$ in orange, and 24 included in the cliques of order 4). Only 3 cliques are maximal: $\{x_1, x_2, x_3, x_4\}$, $\{x_1, x_2, x_3, x_5, x_6\}$ and $\{x_5, x_6, x_7\}$.*

Finally, since we will use it later, we need to define the Clique-Expansion (CE), which is one way to convert a hypergraph into a graph:

Definition 9 (Clique-Expansion (CE) [17])**.** *Let H be a hypergraph, the* Clique-Expansion graph *(CE-graph) of H is the 2-hypergraph on vertices $X(H)$ and with edge set E such that: $E = \{(u, v) \mid \exists E_i \in H \text{ s.t. } \{u, v\} \subseteq E_i\}$.*

An illustration of the Clique-Expansion conversion is given Fig. 4. Following, we only consider r-hypergraphs, even in the search of maximal hypercliques, since it is "often desirable to study hypergraphs where all the hyperedges have the same

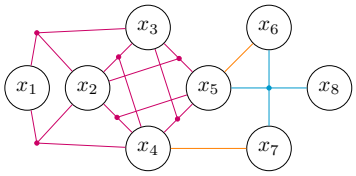

Fig. 3: Initial hypergraph

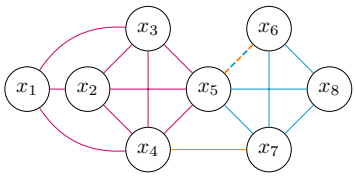

Fig. 4: CE-graph

cardinality" [20], [21]. Therefore a non-uniform hypergraph will be seen as multiple uniform hypergraphs. Indeed, by having a look at Fig. 3, we can split this hypergraph into three different ones: the pink one with 5 vertices and 6 hyperedges, the orange one with 4 vertices and 2 edges and the blue one with 4 vertices and 1 hyperedge. This splitting is always possible. Indeed, as explained in [22], "(A) non-uniform hypergraph is a superposition of m-uniform hypergraphs for $m = 2, \ldots, M$" with M being the biggest hyperedge arity.

III. Detecting all Maximal Cliques

The problem of enumerating maximal cliques is by itself a hard problem. The CLIQUE problem is one of the Karp's 21 NP-complete problems [6] and, moreover, an n-vertex graph can contain up to $3^{\frac{n}{3}}$ maximal cliques [23].

To deal with the intrinsic hardness of our problem, we introduce here four approaches. The first method uses the clique-expansion conversion of hypergraphs and the famous Bron & Kerbosch (BK) algorithm [7] to compute a relaxation of the maximal cliques, then checks if each clique is also a hyperclique. The second method generalizes the BK algorithm in the case of hypergraphs. The last two ones are hybridizations of the relaxation and the generalized BK. One hybrid version applies the relaxation then, if a clique does not correspond to a hyperclique, the generalization is called to find the maximal hypercliques. The other hybrid version uses the relaxation inside the generalized BK. To the best of our knowledge, this is an original work, and after [16] which is too deeply embedded in databases, the second work on detecting maximal hypercliques.

A. Relaxation

The first proposed approach converts the hypergraph into a graph by performing a clique-expansion (CE) conversion [17] where each hyperedge is replaced by a clique. The resulting graph will be referred as CE-graph in the following. It then applies the BK algorithm to find the maximal cliques. This step can find cliques in the graph that are not cliques in the hypergraph. Thus, a checker is applied to verify that the clique found in the CE-graph is also present in the hypergraph.

We illustrate this approach with the following example. Fig. 3 represents an initial graph and Fig. 4 its CE-graph. The hypercliques are computed on r-hypergraph. Thus, in the hypergraph in Fig. 3, there are 6 maximal hypercliques: $\{x_5, x_6, x_7, x_8\}$ (in the 4-hypergraph, in blue), $\{x_1, x_2, x_3\}$, $\{x_1, x_2, x_4\}$, $\{x_2, x_3, x_4, x_5\}$ (in the 3-hypergraph, in pink) and $\{x_4, x_7\}, \{x_5, x_6\}$ (in the 2-hypergraph, in orange). By applying BK on each graph in Fig. 4, we obtain 5 maximal cliques $\{x_5, x_6, x_7, x_8\}$ (in the CE-graph of the 4-hypergraph, in blue), $\{x_1, x_2, x_3, x_4\}$, $\{x_2, x_3, x_4, x_5\}$ (in the CE-graph of the 3-hypergraph, in pink) and $\{x_4, x_7\}$, $\{x_5, x_6\}$ (in the CE-graph of the 2-hypergraph, in orange). Using a checker afterwards, we can detect that $\{x_1, x_2, x_3, x_4\}$ is not a hyperclique. In the 3-hypergraph containing only the vertices in the clique, there are less than $\binom{4}{3} = 4$ hyperedges. On the other hand, the other cliques are hypercliques and are necessarily maximal, as proven by the following proposition:

Proposition 1. *Given H an r-hypergraph. Let G be the CE-graph of H. If a clique $\mathcal{C}_G$ is maximal in G and corresponds to a hyperclique $\mathcal{C}_H$ (the two cliques have the same set of vertices)[2] in H, then $\mathcal{C}_H$ is a maximal hyperclique.*

Proof. Let H be an r-hypergraph and G its CE-graph. Let $\mathcal{C}_G$ be a maximal clique in G that corresponds to a hyperclique $\mathcal{C}_H$ in H. For the sake of contradiction, assume that $\mathcal{C}_H$ is not maximal in H. Then there exists at least a vertex v such that it can be added to $\mathcal{C}_H$ so that it forms a new clique. So $\forall \mathcal{V} \subseteq \mathcal{C}_H$ with $|\mathcal{V}| = r - 1$, $\mathcal{V} \cup \{v\} \in H$. In other words, any vertex in $\mathcal{C}_H$ appears with v in at least one hyperedge. Thus, with the CE, this means that there exists an edge between each vertex of $\mathcal{C}_G$ and v in G. Hence, v can be added to $\mathcal{C}_G$ to form a new clique. Contradicting the fact that $\mathcal{C}_G$ is a maximal clique in G. If a clique $\mathcal{C}_G$ is maximal in G and corresponds to a hyperclique $\mathcal{C}_H$ in H, then $\mathcal{C}_H$ is maximal in H. □

Transforming the hypergraph into a simple graph using the CE conversion and then checking if each clique returned by BK is also a hyperclique gives a lower bound on the number of maximal hypercliques.

B. Hyper Bron & Kerbosch

As stated in [16], it is usually assumed that the generalization of BK to hypergraphs cannot be done efficiently. Indeed, the concept of neighbour is not sufficient to decide if a set of vertices is a clique, therefore generalizing BK to hypergraphs is not straightforward. However, by considering the neighbourhood of a clique instead of the neighbourhood of each vertex, BK can be generalized efficiently in practice. We propose the following definition of *clique neighbourhood* to generalize the BK algorithm [7].

Definition 10 (Clique Neighbourhood)**.** *Let H be a r-hypergraph. Given a clique $\mathcal{C}$ and $\mathcal{Y} \subseteq X(H) \setminus \mathcal{C}$, we denote by $\gamma(\mathcal{C})$, the* clique neighbourhood *of $\mathcal{C}$, a set of vertices such that if any vertex is added to $\mathcal{C}$ it forms a new clique:*

[2] $\mathcal{C}_G$ and $\mathcal{C}_H$ correspond to the same set of vertices; the two notations are introduced to improve readability.

Algorithm III.1: Hyper-Bron-Kerbosch

```
Result ← ∅                                   // global variable
Hyper-Bron-Kerbosch(Current, Candidate, Not)
1  if (Candidate = ∅) and (Not = ∅) then
2      Result ← Result ∪ {Current}
3  for each vertex v in Candidate do
4      Current ← Current ∪ {v}
5      CliqueN ← γ(Current)
6      Hyper-Bron-Kerbosch(Current, Candidate ∩
         CliqueN, Not ∩ CliqueN)
7      Current ← Current \ {v}
8      Candidate ← Candidate \ {v}
9      Not ← Not ∪ {v}
```

$$\gamma(\mathcal{C}) = \{y \in \mathcal{Y} \,|\, \forall N \subseteq \mathcal{C} \text{ with } |N| \leq r-1, \exists E_i \in H \text{ s.t. } N \cup \{y\} \subseteq E_i\}$$

From this definition, we propose Algorithm III.1, based on BK, to enumerate all the maximal cliques in a hypergraph.

The advantage of this algorithm is that it enumerates only maximal clique and avoid generating non-optimal ones. Current, Candidate and Not are sets of vertices. On the first call Current and Not are set to $\emptyset$, and Candidate contains all the vertices of the hypergraph.

Current is the temporary result which corresponds to the vertices in a clique, Candidate the set of possible candidates that may extend Current to form a new clique, and Not contains vertices from which cliques have already been computed and is used to avoid reporting duplicates. Once all the cliques for a given vertex v are computed, v is added to Not (line 9). Having the set Not allows the algorithm to avoid generating non-maximal cliques, as it was already the case in the original BK algorithm.

Even though, this algorithm can list all the maximal cliques in a simple graph, it is far from obvious that it can be extended as easily for hypergraphs. To prove it, we need to prove that at each step Current is a clique and that if this algorithm adds a clique to Result, then the clique is maximal. We also need to prove that all the maximal cliques are found.

Proposition 2. *At each step of Algorithm III.1,* Current *is a clique.*

Proof. An empty set or a single vertex is a clique. Let Current be a clique of order $n \geq 2$. Either Candidate is empty and we backtrack to a superior level where the last vertex added is removed, and Current remains a clique. Or Candidate contains at least a vertex v. Candidate is a subset of CliqueN (line 6), then by Definition 10, v can be added to Current such that it forms a new clique. □

Algorithm III.1 terminates and reports only and all the maximal cliques in an r-hypergraph. (Proof in appendix.)

The efficient computation of the clique neighbourhood (Definition 10) depends on the set of considered vertices $\mathcal{Y}$. To test experimentally whether this can speed up the approach, we propose different filtering methods.

1) Filtering: In this section, we present two different filtering techniques. Both are meant to reduce the number of candidates, that is to reduce the size of the set $\mathcal{Y}$, to speed-up the clique neighbourhood computation. These filterings may return false positive matches but do not return false negative matches. In other words, for each possible vertex, the filtering return either "possibly in set" or "definitely not in set". As the combinatorics can be extremely large, it may be more efficient to have some false positives that need to be confirmed than computing the clique neighbourhood for all the vertices.

a) Neighbourhood filtering: One way that we propose to filter the set of candidates uses the neighbourhood of each vertex already in Current.

Proposition 3. *Let $c \in \mathcal{X} \setminus$ Current be a vertex. If there exists a vertex $x \in$ Current such that $c \notin N(x)$, then $c \notin$ Candidate.*

Proof. If a vertex c is not in the neighbourhood of a vertex in Current, then by definition of clique neighbourhood (Definition 10), it cannot belong to Candidate. □

Even though only checking the neighbourhood of each vertex already in Current independently is not sufficient to be sure that a vertex c will expand the current clique, it is an efficient way to filter the list of potential candidates. If the vertex c is not in one neighbourhood, then for sure it cannot expand the current clique.

b) Clique filtering: This filtering is based on Proposition 1: each candidate that is not in a maximal clique in the CE-graph with the vertices in Current, cannot extend the current hyperclique. This is formalized by the following Proposition:

Proposition 4. *Let c be a vertex such that $c \in \mathcal{X} \setminus$ Current and let* Cliques *be the resulting set containing all the cliques in the CE-graph containing all the vertices in* Current*. If $\nexists \mathcal{C} \in$ Cliques, such that $c \in \mathcal{C}$, then $c \notin$ Candidate.*

Proof. From Proposition 1 if a clique is maximal, and is a hyperclique, then it is necessarily a maximal hyperclique. Therefore, if a vertex c is not at least in one maximal clique with all the vertices in Current, then c cannot belong to the set Candidate. □

2) Incremental Computation: Another way to reduce the computation cost of the clique neighbourhood is to incrementally compute it. Indeed, when a new vertex v is added to the clique, then, for all the candidates c, we know that we need to have at least one hyperedge containing v, c and the subsets of all the vertices already in the current clique. For instance, consider a current clique of order 4 in a 3-hypergraph: $\{x_1, x_2, x_3, x_4\}$ with x_4 being the last added element. A vertex c remains a candidate if and only if the following hyperedges exist $\{x_4, x_1, c\}$,$\{x_4, x_2, c\}$,$\{x_4, x_3, c\}$. Note that edges $\{x_1, x_2, c\}$, $\{x_1, x_3, c\}$ and $\{x_2, x_3, c\}$ exist as c is a candidate (these edges were checked on previous vertex additions). We formalize it with the following Proposition:

Proposition 5. *Let x_n be the last vertex added to the clique* Current *and let c a vertex with $c \in \mathcal{X} \setminus$ Current. If there exists a subset* comb $\subseteq$ Current $\setminus \{x_n\}$ *with* $|\texttt{comb}| \leq r-2$*, such that no hyperedge contains* $\{x_n, \texttt{comb}, c\}$*, then* $c \notin$ Candidate.

Proof. Let H be an r-hypergraph, Current be a clique containing the vertex x_n. Let c be a vertex with $c \in X(H) \setminus$ Current. Let comb $\subseteq$ Current $\setminus \{x_n\}$ with $|\texttt{comb}| \leq r-2$ be a subset of Current. As $|\texttt{comb}| \leq r-2$ then $|\{\texttt{comb}, x_n\}| \leq r-1$. By Definition 10, as $\{\texttt{comb}, x_n\} \subseteq$ Current, if $\nexists E_i \in H$ s.t. $\{\texttt{comb}, x_n, c\} \subseteq E_i$ then c cannot be in the clique neighbourhood of Current and is therefore not a candidate. □

By storing each hyperedge in a hashmap, the incremental computation can be performed. It is denoted as "missing edge" in [24, §3.5.1, Lemma 1]. However, one way to implement this computation as filtering is to use a Bloom filter [25]. It is a space-efficient probabilistic data structure that is used to test whether an element is a member of a set. We use this data structure to filter candidates for which at least one hyperedge is missing so that they will not be considered as candidates.

C. Hybridizations

In this section, we propose two hybridizations of the relaxation and the generalization of the BK algorithm.

1) Relaxation as an oracle for Hyper Bron-Kerbosch: As stated in Section III-A, computing the maximal cliques on the CE-graph, may not give the maximal hypercliques. However, if a clique is also a hyperclique then it is maximal (Proposition 1). Based on this proposition, we propose to guide the search for maximal hypercliques using the maximal cliques in the CE-graph. It computes all the maximal cliques, if a clique $\mathcal{C}$ is a hyperclique then it is directly added to the result, otherwise, Hyper-Bron-Kerbosch is called with Candidate $= \mathcal{C}$. The pseudo-code of this method is given Algorithm III.2. To enumerate only the maximal hypercliques two checkers are mandatory. The first verifies if a clique is a hyperclique. The second verifies if a hyperclique is not already contained in another hyperclique found by the call to Bron-Kerbosch or a previous call to Hyper-Bron-Kerbosch.

Example 2. *Applying BK on the CE-graph of the 3-hypergraph in Fig. 4, we obtain 2 maximal cliques $\{x_1, x_2, x_3, x_4\}$ and $\{x_2, x_3, x_4, x_5\}$. The second is a hyperclique while the first one is not. Thus, Hyper-Bron-Kerbosch is called with* Candidate *set $\{x_1, x_2, x_3, x_4\}$ and returns 3 hypercliques $\{x_1, x_2, x_3\}$, $\{x_1, x_2, x_4\}$ and $\{x_2, x_3, x_4\}$. The last one is included in $\{x_2, x_3, x_4, x_5\}$ and is not kept. Finally, this method returns 3 hypercliques $\{x_1, x_2, x_3\}$, $\{x_1, x_2, x_4\}$ and $\{x_2, x_3, x_4, x_5\}$.*

Algorithm III.2 terminates and finds all the maximal cliques in an r-hypergraph. (Proof in appendix.)

2) Hybrid Bron-Kerbosch: In the previous hybrid method, the checker is applied *a posteriori*. We propose here to add it in the original BK algorithm. Algorithm III.3 gives the pseudo-code of this Hybrid BK. In this algorithm, $N(v)$ stands for the neighbourhood of a vertex v (Definition 5).

Algorithm III.2: Relax-HBK(H)

```
1  Result ← ∅
2  cliques ← ∅
3  notcliques ← ∅
4  Bron-Kerbosch(∅, 𝒳, ∅)                    // cliques in Result
5  for each clique c in Result do
6      if Is-hyperclique(c) then
           cliques ← cliques ∪ {c}
7      else notcliques ← notcliques ∪ {c}
8  for each clique c in notcliques do
9      Result ← ∅
10     Hyper-Bron-Kerbosch(∅, c, ∅)  // hypercliques in
         Result
11     for each hyperclique hc in Result do
12         to-add ← true
13         for each clique c' in cliques do
14             if hc ⊆ c' then
                   to-add ← false
15         if to-add then
               cliques ← cliques ∪ {hc}
```

Algorithm III.3: Hybrid-Bron-Kerbosch

```
   Result ← ∅                              // global variable
   Hybrid-Bron-Kerbosch(Current, Candidate, Not)
1      for each vertex v in Candidate do
2          if Is-hyperclique(Current ∪ {v}) then
3              Current ← Current ∪ {v}
4              Hybrid-Bron-
                 Kerbosch(Current, Candidate ∩
                 N(v), Not ∩ N(v))
5              Current ← Current \ {v}
6              Not ← Not ∪ {v}
7          Candidate ← Candidate \ {v}
8      if (Candidate = ∅) and (Not ∩ γ(Current) = ∅)
         then
9          Result ← Result ∪ Current
```

Like in the original algorithm, the clique is kept in Current, Candidate corresponds to the set of possible vertices that can be added and Not to the set of vertices from which a clique has already been computed. As the computation of a clique neighbourhood is time-consuming, we propose to compute it only if a clique is found to ensure it is maximal (line 8). Moreover, each time a new vertex is added the function is recursively called only if Current is a hyperclique (line 2).

Proposition 6. *At each step of Algorithm III.3,* Current *is a clique.*

Proof. An empty set or a single vertex is a clique. Let Current be a clique of order $n \geq 2$. Either Candidate is empty and we backtrack to a superior level where the last vertex added is removed, Current remains a clique. Or Candida-

te contains at least a vertex v. v is added to Current only if by adding it Current forms a new clique (line 2). Hence at each step of Algorithm III.3, Current is a clique. □

Algorithm III.3 terminates and finds all the maximal cliques in an r-hypergraph. (Proof in appendix.)

IV. From graphs to hypergraphs

It has been shown that the vertex ordering can influence the efficiency of BK. We thus decided to check if the same statement can be applied to the case of hypergraphs.

A. Candidate ordering

It has been shown that the best-suited ordering for BK is the degeneracy ordering for sparse graphs [8]. Therefore, we propose to try different orderings to see if this statement is still valid in the case of hypergraphs. We thus experimentally evaluated different orderings: random ordering, according to the min-fill algorithm, ordering by the min-degree of each vertex, the natural ordering and finally the degeneracy ordering of the hypergraph. The random ordering is uniform, the min-fill algorithm is presented in the htd documentation [Algorithm 1] [26], the natural ordering is just the original graph ordering.

B. Preprocessing

As stated in Definition 7, each vertex x in $X(K_n^r)$ has a degree $d(x) = \binom{n-1}{r-1}$. So, for a given rank r, if a vertex x has a degree $d(x) < \binom{r}{r-1} = r$, then the maximal cliques containing x are of order r which corresponds to the hyperedges containing x. We can therefore preprocess the r-hypergraph and remove all the vertices having a degree less than r from the initial Candidate set and add all the hyperedges containing them in Result.

This property can be extended. Given two vertices x, v in $X(K_n^r)$, they appear together in $\binom{n-2}{r-2}$ edges. In other words, x has $n-1$ neighbours with whom it appears in $\binom{n-2}{r-2}$ edges. So, for a given rank r, if a vertex x does not have at least r neighbours with whom it appears in $\binom{r-1}{r-2} = r-1$ edges, then the maximal cliques containing x are of order r which corresponds to the hyperedges containing x. Therefore we can remove all such vertices from the initial Candidate set and add all the hyperedges containing them in Result.

V. Experiments

We implemented the proposed approaches in an open-source solver (written in C++) on top of the htd library [27]. We used a cluster of identical computers with two 16-Cores processors Intel Xeon Gold SKL-6130 with 96GB of memory (two cores were booked for each process to limit memory bandwidth bottlenecks). For each problem, we set a time-out of 1800 seconds. For ensuring maximal reproducibility, we set up a Docker image containing our tool, a database of all our traces, and a tools to easily reproduce them (replaying any entry in the DB, re-computing a table of results, ...). The description for using this Docker image is given in the pre-release of the following GitHub repository https://github.com/mpelleau/ICTAI-2025. In addition, an in-depth interactive visualisation of the exhaustive set of results is accessible at https://mpelleau.shinyapps.io/Hyperclique/.

For comparing the performances of our tool, we have done our best to contact different authors, even for related problems to the MHE problem such as the Maximum Clique Problem, without success. It was, unfortunately, neither possible to get a binary, nor the benchmark sets, nor to get the results of their approach on our set of problems from [28]. We also tried to measure the performances of [24], but this was also not possible (the tool was not designed to be used as a separate tool). To the best of our knowledge, there is no efficient, open-source, solution for enumerating hypercliques. This is also an important contribution of our work.

Therefore, we had to limit our study to our three different approaches. The first one, HyperBK (HBK), is our generalization of the Bron & Kerbosch algorithm (Algorithm III.1). The second one, RelaxHBK (RHBK), applies the relaxation and then calls HyperBK if a clique does not correspond to a maximal hyperclique (Algorithm III.2). The last method, Hybrid (Hyb), provides a hybridization between computing cliques in the CE-graph and computing the hyperclique directly (Algorithm III.3). These three approaches have three parameters to be tested. The first is the ordering of the set Candidate (Section IV-A). The second is the filtering technique (Section III-B1) of the Candidate set before each recursive call to the Hyper-Bron-Kerbosch algorithm. The last parameter is the preprocessing (Section IV-B) of the Candidate set before the first call of the approach.

In the following we denote by:

- o the chosen ordering: random ordering ($o = r$), according to the min-fill algorithm ($o = mF$), ordering by the min-degree of each vertex ($o = mD$), the natural ordering ($o = n$) and finally the degeneracy ordering of the hypergraph ($o = d$).
- f the chosen filtering: the Bloom filtering ($f = b$), the clique filtering ($f = c$), the filtering using the neighbourhood ($f = n$) and no filtering ($f = \emptyset$). We also put in this category the incremental computation ($f = i$).
- p the preprocessing method: based on the vertices degree ($p = d$), based on co-occurrences ($p = c$), both preprocessings ($p = b$), and no preprocessing ($p = \emptyset$).

A. Set of Problems and Configurations

We decided to test all the possible configurations (3 methods, 5 filtering, 5 orders, 4 preprocessings). We have 300 configurations per problem to test, with a timeout of 1,800 seconds, which corresponds to almost one week of CPU time per problem (6.25 CPU Days). One of the challenges in this section is to give a synthetic view of the relative performances of the 300 configurations. In addition to the web-based interactive exploration of our data, we decided to use the Par2 score, as it is used in many competition to rank solvers amongst them. Given a timeout, the Par2 score of a solver on a set of problems is the sum of its CPU time on the finished runs plus the penalty multiplied by the number of unfinished

TABLE I: Some statistical information regarding each subfamilies of the set Hyperbench considered for the experimental evaluation. We computed the average number of vertices and edges for each subfamily. We also computed the average value of the arity for each hyperedge (the standard deviation), the degree of each hypergraph (the standard deviation).

Subfamily	#Ins.	#Vertices	#Edges	Arity (std)	Degree (std)
CQ	175	21.960	5.211	5.049 (2.550)	1.21 (0.192)
CQ-Rand.	500	9.488	1.000	9.488 (4.235)	1.000 (0.000)
CSP-App.	1090	151.709	68.898	5.739 (7.805)	3.341 (1.622)
CSP-Rand.	863	40.736	67.580	4.847 (2.954)	10.438 (10.032)
D-Chrysler	15	592.133	318.600	2.964 (0.099)	1.551 (0.149)
Grid2D	12	1655.333	988.333	3.859 (0.101)	2.294 (0.083)
ISCAS89	24	669.416	428.708	2.871 (0.318)	1.827 (0.240)
MaxSAT	31	120.419	178.000	3.329 (2.086)	4.144 (1.359)
SQLShare	290	29.982	8.075	7.508 (8.714)	1.257 (0.309)

runs. For a ParX ranking, the penality is X times the timeout (our penalty for not solving an instance is thus 3600).

The set of problems we choose comes from *Johannes K. Fichte et al.* [29] (available at http://hyperbench.dbai.tuwien.ac.at/). These benchmarks are usually used for the analysis of hypertree widths, generalized hypertree widths and fractional hypertree widths (some results about the hypertree width of these benchmarks are available in [30]). This set contains 3000 problems that are not too hard, given the number of configurations to test.

We collected 900,000 runs, representing more than 64 days of CPU time. 800,590 jobs were trivial (less than 0.1s) and many other problems were probably too easy to draw any conclusion between the methods. Thus, we decided to select only the 1037 problems where at least one configuration took more than 1s (40,741 jobs took more than 1s). The final set of the 1037 problems we consider from now on contains 328 problems from CSP-Applications, 621 from CSP-Random, 13 from DaimlerChrysler, 11 from Grid2D, 23 from ISCAS 89, 25 from MaxSAT and 16 from SQLShare.

Amongst all the configurations, we identified `HyperBK` with no preprocessing ($p = \emptyset$), no filtering ($f = \emptyset$) and the natural ordering ($o = n$) as the "reference" configuration. It is very encouraging to measure that the average CPU time for this configuration is 13s (on the 1037 problems), without any timeout. That means that scaling the Bron & Kerbosch algorithm to hypercliques may gives good results. The question is now how far we can improve it.

B. Experimental Results

Table II summarizes our results by considering, on each line, a summary of all the configurations matching the line constraint (for instance, the first line considers all the runs, on the 1037 problems, that involves `HyperBK` with any set of parameters). The first observation is the number of timeouts (#TO): clearly, `Hybrid` seems to be the best. There is no timeout, whatever the other parameters are. It is also very surprising to see that the reference solver `HyperBK`, that does not timeout in its native setup (with all the default parameters), shows the worst performances when considering all its variants. That means that some values for the other parameters greatly impact its performances. For a more detailed analysis, we can look at the Par2 column, which gives the average Par2

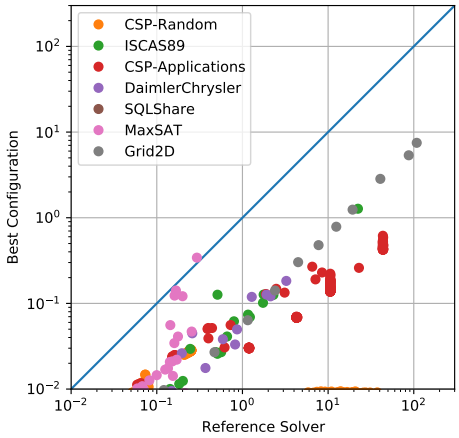

Fig. 5: Scatter plot of the CPU time (s) of the reference solver (see text) against the best configuration.

TABLE II: Synthesis of the Par2 score of all the configurations, aggregated on each parameter. See text for descriptions.

config	#TO	Par2	Par2 Rank
`HBK`	1334	59.34	170 (162, 88)
`Hyb`	0	1.13	112 (112, 66)
`RHBK`	565	25.52	200 (175, 90)
$f = \emptyset$	0	4.16	131 (152, 74)
$f = b$	0	2.90	104 (115, 88)
$f = c$	1719	117.60	180 (192, 83)
$f = n$	0	4.37	133 (156, 76)
$f = i$	180	14.27	114 (131, 91)
$p = \emptyset$	1459	89.64	252 (243, 36)
$p = d$	440	24.24	202 (197, 51)
$p = c$	0	0.38	38 (63, 56)
$p = b$	0	0.38	82 (95, 40)
$o = r$	395	29.56	148 (151, 86)
$o = mD$	379	28.85	158 (152, 87)
$o = mF$	376	28.79	152 (153, 86)
$o = n$	410	29.85	138 (145, 88)
$o = d$	339	26.24	142 (146, 86)

of all the variants of the considered line. This column also offers a number of surprises. The `Hybrid` line shows a very impressive average Par2 value. More importantly, we see that the preprocessing based on co-occurrences ($p = c$) and based on both degree and co-occurrences ($p = b$) leads to great improvements in general. It seems also that the impact of the order choice is very limited. We confirmed this by studying the variance of the same configurations according to the different orders. 95% of the variances are less than 0.1 and 99% less than 10. However, maybe harder problems can demonstrate the advantage of one order above the others.

"Par2 Rank" summarizes the ranks of all the variants matching the considered line, relatively to all the 300 configurations (numbers are noted Median (Average, StD)). This last column shows that, `Hybrid` is generally better than the other solvers (its median ranking is 112). This last column also shows that the preprocessing based on co-occurrences ($p = c$) generally offers better performances than the one based on both degree and co-occurrences ($p = b$) (its median ranking is 38).

Based on the previous analysis, we were able to find the best possible configuration. This configuration is `Hybrid` with the Bloom filtering ($f = b$) and the preprocessing based on co-occurrences ($p = c$). Figure 5 shows the final progress we made during our journey. It shows an impressive improvements on almost all the problems. Our best configuration is almost 10 times faster and, on some families (CSP-Application) it shows an improvement of almost a factor of 100.

VI. Acknowledgment

This work has been supported by the French government, through the 3IA Côte d'Azur Investments in the Future project managed by the National Research Agency (ANR) with the reference number ANR-19-P3IA-0002.

The second author was partially supported by the "Chaire IA Digne de Confiance" (Trustworthy AI) operated by the Fondation Bordeaux Université.

VII. Conclusion

Despite its importance in numerous areas, particularly in Artificial Intelligence (constraint satisfaction, knowledge graph analysis, logic-based learning, multi-relational data mining, etc.), the problem of enumerating all maximal cliques of a hypergraph (MHE) has probably not received all the attention it deserves. In this paper, we present three different methods to solve the MHE problem.

The first one is a generalization of the Bron & Kerbosch algorithm, to hypergraphs, called `HyperBK`. We demonstrate that our generalization is sound, complete and terminates. We show that the generalization opens the way for several variants. For instance, the size of the candidates set in the main loop of `HyperBK` plays a crucial role in its performances. We have proposed to study three different techniques to efficiently filter out some of the candidates.

The first filtering considers an intersection with the neighbourhood of all the vertices already in the current clique. The second filtering uses the cliques in the Clique-Expansion graph (CE-graph). The third filtering, is a variant of the incremental way to compute the clique neighbourhood using a probabilistic hashmap (Bloom-filter) to identify the vertices for which at least one hyperedge with the last added vertex is missing. We also consider a number of variants for ordering the candidates during the main loop of `HyperBK`.

Then, we propose two hybridizations of our algorithm. The first hybridization considers computing all the maximal cliques in the CE-graph of the input graph and then only checking, for each maximal clique, if it is also a clique in the original hypergraph. Secondly, we consider a more interleaved hybridization, by first checking if the union between the next candidate and the current clique is still a clique before the main recursive call of the `RelaxHBK` algorithm.

At last, we experimentally show, on a set of well established problems, that we were able to improve the basic version of Bron & Kerbosch algorithm for hypergraphs by a factor of 10 and, on some categories of problems, by a factor of 100. All our algorithms (with variants) are open-source, built on top of the widely used library HTD for hypergraph manipulations. We hope that this will allow easy adoption of our algorithms by the community.

References

[1] L. Pan and E. E. Santos, "An anytime-anywhere approach for maximal clique enumeration in social network analysis," in *Proc. of IEEE SMC'08*. IEEE, 2008, pp. 3529–3535.

[2] M. Keller and M. Nussbaumer, "Menuminer: revealing the information architecture of large web sites by analyzing maximal cliques," in *Proc. of WWW'12*. ACM, 2012, pp. 1025–1034.

[3] Y. Chen and G. M. Crippen, "A novel approach to structural alignment using realistic structural and environmental information," *Protein science*, vol. 14, no. 12, pp. 2935–2946, 2005.

[4] B. Zhang, B. Park, T. V. Karpinets, and N. F. Samatova, "From pull-down data to protein interaction networks and complexes with biological relevance," *Bioinformatics*, vol. 24, no. 7, pp. 979–986, 2008.

[5] B. On, E. Elmacioglu, D. Lee, J. Kang, and J. Pei, "Improving grouped-entity resolution using quasi-cliques," in *Proc. of ICDM'06*. IEEE Computer Society, 2006, pp. 1008–1015.

[6] R. M. Karp, "Reducibility Among Combinatorial Problems," in *Proc. of a symposium on the Complexity of Computer Computations*, 1972, pp. 85–103.

[7] C. Bron and J. Kerbosch, "Algorithm 457: Finding All Cliques of an Undirected Graph," *Commun. ACM*, vol. 16, no. 9, pp. 575–577, 1973.

[8] D. Eppstein, M. Löffler, and D. Strash, "Listing all maximal cliques in sparse graphs in near-optimal time," in *Proc. of ISAAC'10*, ser. LNCS, vol. 6506. Springer, 2010, pp. 403–414.

[9] B. Hou, Z. Wang, Q. Chen, B. Suo, C. Fang, Z. Li, and Z. G. Ives, "Efficient maximal clique enumeration over graph data," *Data Science and Engineering*, vol. 1, no. 4, pp. 219–230, 2016.

[10] V. Stix, "Finding all maximal cliques in dynamic graphs," *Comp. Opt. and Appl.*, vol. 27, no. 2, pp. 173–186, 2004.

[11] E. Tomita, A. Tanaka, and H. Takahashi, "The worst-case time complexity for generating all maximal cliques and computational experiments," *Theor. Comput. Sci.*, vol. 363, no. 1, pp. 28–42, 2006.

[12] F. Cazals and C. Karande, "A note on the problem of reporting maximal cliques," *Theor. Comput. Sci.*, vol. 407, no. 1-3, pp. 564–568, 2008.

[13] D. Eppstein, M. Löffler, and D. Strash, "Listing all maximal cliques in large sparse real-world graphs," *ACM Journal of Experimental Algorithmics*, vol. 18, 2013.

[14] A. Biere, D. L. Berre, E. Lonca, and N. Manthey, "Detecting Cardinality Constraints in CNF," in *Proc. of SAT'14*, 2014, pp. 285–301.

[15] N. Shaabani and C. Meinel, "Detecting maximum inclusion dependencies without candidate generation," in *Proc. of DEXA'16*, ser. LNCS, vol. 9828. Springer, 2016, pp. 118–133.

[16] A. Koeller and E. A. Rundensteiner, "Discovery of highdimensional inclusion dependencies," Worcester Polytechnic Institute, Tech. Rep. WPICS-TR-02-15, 2002.
[17] J. A. Frankle, "Circuit placement methods using multiple eigenvectors and linear probe techniques," Ph.D. dissertation, EECS Department, University of California, Berkeley, 1987.
[18] C. Berge, *Hypergraphs: Combinatorics of Finite Sets*, ser. Mathematical Library. Elsevier, 05 1984, vol. 45.
[19] M. A. Bahmanian and M. Sajna, "Connection and separation in hypergraphs," *Theory and Applications of Graphs*, vol. 2, no. 2, p. 5, 2015.
[20] A. Hassan and M. Malik, *Generalized neutrosophic hypergraphs*. Infinite Study, 2016.
[21] M. Predari, "Load balancing for parallel coupled simulations," Ph.D. dissertation, University of Bordeaux, France, 2016. [Online]. Available: https://tel.archives-ouvertes.fr/tel-01518956
[22] D. Ghoshdastidar and A. Dukkipati, "Consistency of spectral partitioning of uniform hypergraphs under planted partition model," in *Proc. of NIPS'14*, 2014, pp. 397–405.
[23] J. W. Moon and L. Moser, "On cliques in graphs," *Israel Journal of Mathematics*, vol. 3, no. 1, pp. 23–28, 1965.
[24] A. Koeller and E. A. Rundensteiner, "Discovery of high-dimensional inclusion dependencies," in *Proc. of ICDE'03*. IEEE, 2003, pp. 683–685.
[25] B. H. Bloom, "Space/Time Trade-offs in Hash Coding with Allowable Errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, 1970.
[26] M. Abseher, N. Musliu, and S. Woltran, "htd–a free, open-source framework for tree decompositions and beyond," Tech. Rep., 2016.
[27] ——, "htd - A free, open-source framework for (customized) tree decompositions and beyond," in *Proc. of CPAIOR'17*, 2017, pp. 376–386.
[28] J. Torres-Jimenez, J. C. Perez-Torres, and G. Maldonado-Martinez, "*hClique*: An exact algorithm for maximum clique problem in uniform hypergraphs," *Discrete Math., Alg. and Appl.*, vol. 9, no. 6, pp. 1–14, 2017.
[29] J. K. Fichte, M. Hecher, N. Lodha, and S. Szeider, "A benchmark collection of hypergraphs," 6 2018.
[30] W. Fischl, G. Gottlob, D. M. Longo, and R. Pichler, "Hyperbench: A benchmark and tool for hypergraphs and empirical findings," in *Proc. of PODS'19*. ACM, 2019, pp. 464–480. [Online]. Available: https://doi.org/10.1145/3294052.3319683

APPENDIX

The proofs of soundness and completeness of the three different Algorithms are in this appendix. We first prove the soundness and the completeness of Algorithm III.1. But before, we recall the Proposition 2

Proposition 2. *At each step of Algorithm III.1,* `Current` *is a clique.*

Proof. An empty set or a single vertex is a clique. Let `Current` be a clique of order $n \geq 2$. Either `Candidate` is empty and we backtrack to a superior level where the last vertex added is removed, and `Current` remains a clique. Or `Candidate` contains at least a vertex v. `Candidate` is a subset of `CliqueN` (line 6), then by Definition 10, v can be added to `Current` such that it forms a new clique. □

Proposition 7. *A clique is added to* `Result` *by Algorithm III.1, if and only if it is maximal.*

Proof. When a vertex is added to `Current`, which from Proposition 2 is a clique, if `Candidate` becomes empty. Then either `Not` is empty and `Current` is maximal as no vertex can be added (line 2). Or `Not` contains at least a vertex v. `Not` is a subset of the clique neighbourhood `CliqueN` (line 6), then by Definition 10, v can be added to `Current` such that it forms a new clique. So if `Not` is not empty, the clique is not maximal and Algorithm III.1 does not add it. The same goes if `Candidate` does not become empty. □

Proposition 8. *When a candidate c is eliminated in Algorithm III.1, it cannot extend the current clique.*

Proof. A candidate c is removed thanks to the intersection between the sets `Candidate` and `CliqueN` (line 6) or after a call to Hyper-Bron-Kerbosch (line 8). In the first case, if c is not in the neighbourhood of `Current`, then by Definition 10, c cannot extend `Current` and thus is eliminated. In the second case, it is removed as all the maximal cliques containing `Current`$\cup\{c\}$ have been computed by the recursive calls to Hyper-Bron-Kerbosch. □

Algorithm III.1 terminates and reports only and all the maximal cliques in an r-hypergraph.

Proof. Soundness comes directly from Proposition 7, Algorithm III.1 only returns maximal cliques. Completeness comes from Proposition 8 it ensures that the set of all the possible candidates is tried as a candidate is eliminated only if it cannot be part of the current clique. Thus all the cliques are tried and reported if they are maximal. Finally, Algorithm III.1 terminates. Otherwise, by Kőnig's lemma, either `Candidate` is infinite or the call stack is infinite. `Candidate` is a subset of `CliqueN` which is a subset of $\mathcal{X}$ which is finite by Definition 1. Thus `Candidate` cannot be infinite. As the call stack depend on the size of `Candidate`, then it cannot be infinite. □

We then prove the soundness, the completeness and the termination of Algorithm III.2.

Proposition 9. *Algorithm III.2 reports maximal cliques.*

Proof. From Proposition 1, if a maximal clique is also a hyperclique then it is reported. Otherwise Hyper-Bron-Kerbosch is called with for candidates only the vertices in the clique. Then by Theorem 9, we know that Algorithm III.2 reports only maximal cliques. □

Proposition 10. *All maximal cliques are found by Algorithm III.2.*

Proof. Toward a contradiction. Assume that a maximal clique $\mathcal{C}$ exists and it is not found. That means that either $\mathcal{C}$ has been returned by Bron-Kerbosch since it is maximal and that Bron-Kerbosch is sound [7]. Therefore, if $\mathcal{C}$ is also a hyperclique, then by the Proposition 1 and line 7, it is returned and therefore contradiction. Or $\mathcal{C}$ has been returned by Hyper-Bron-Kerbosch since this algorithm returns all maximal cliques (Theorem 9). Because we assume that $\mathcal{C}$ is maximal, then at line 16, the variable $to\text{-}add$ is true and thus $\mathcal{C}$ is added, again leading to a contradiction. Because there is no other way to return a clique, this implies that all maximal cliques are found by Algorithm III.2. □

Algorithm III.2 terminates and finds all the maximal cliques in an r-hypergraph.

Proof. Soundness comes directly from Proposition 9, this method only returns maximal cliques. Completeness comes from Proposition 10. Finally, this method terminates, it calls Bron & Kerbosch algorithm which terminates and returns a finite set of cliques. In the worst case, Hyper-Bron-Kerbosch is called for each clique found by Bron & Kerbosch. Hyper-Bron-Kerbosch terminates (Theorem 9) and is called a finite number of times. □

And finally, we prove the soundness and the completeness of the Algorithm III.3. First we recall Proposition 6.

Proposition 6. *At each step of Algorithm III.3,* `Current` *is a clique.*

Proof. An empty set or a single vertex is a clique. Let `Current` be a clique of order $n \geq 2$. Either `Candidate` is empty and we backtrack to a superior level where the last vertex added is removed, `Current` remains a clique. Or `Candidate` contains at least a vertex v. v is added to `Current` only if by adding it `Current` forms a new clique (line 2). Hence at each step of Algorithm III.3, `Current` is a clique. □

Proposition 11. *A clique is added to* `Result` *by Algorithm III.3, if and only if it is maximal.*

Proof. By Proposition 6, `Current` is a clique. If `Candidate` becomes empty, then either $\texttt{Not} \cap \gamma(\texttt{Current})$ is empty and `Current` is maximal as no vertex can be added (line 9). Or $\texttt{Not} \cap \gamma(\texttt{Current})$ contains at least a vertex v. $\texttt{Not} \cap \gamma(\texttt{Current})$ is a subset of $\gamma(\texttt{Current})$, by Definition 10, v can be added to `Current` such that it forms a new a clique. So if $\texttt{Not} \cap \gamma(\texttt{Current})$ is not empty, the clique is not maximal and Algorithm III.3 does not add it to `Result`. The same goes if `Candidate` does not become empty. □

Proposition 12. *When a candidate c is eliminated in Algorithm III.3, it cannot extend the current clique.*

Proof. A candidate c is removed thanks to the intersection between the sets `Candidate` and $N(v)$ (line 6). If c is not in the neighbourhood of `Current`, then by Definition 10, c cannot extend `Current` and thus is eliminated. □

Algorithm III.3 terminates and finds all the maximal cliques in an r-hypergraph.

Proof. Soundness comes directly from Proposition 11, Algorithm III.3 only returns maximal cliques. Completeness comes from Proposition 12. For the termination, it is easy to see that by Kőnig's lemma, either `Candidate` is infinite or the call stack is infinite. `Candidate` is a subset of $\mathcal{X}$ which is finite by Definition 1. Thus Candidate cannot be infinite. As the call stack depend on the size of Candidate, then it cannot be infinite and therefore Algorithm III.3 terminates. □